Seminarski rad

Tema: *Windows Communication Foundation*

Predmet: *Paralelne i distribuirane arhitekture i jezici*

Mentor: Dinu Dragan

Student: Stefan Aleksić

Novi Sad, decembar 2022.

Sadržaj

Uvod

Distribuirani sistem je sistem čije su komponente locirane na različitim umreženim računarima, koji komuniciraju i koordiniraju svoje akcije prosleđivanjem poruka jedni drugima sa bilo kog sistema. [1] [2]

Komponente distribuiranog sistema komuniciraju jedna sa drugom kako bi se postigao zajednički cilj, međutim, za razliku od paralelnog programiranja, čvorovi u sistemu mogu imati različite uloge, odnosno obavljati različite zadatke, a njihova koordinacija tj. sinhronizacija se obavlja komunikacijom putem mreže. Tri značajna izazova distribuiranih sistema su: održavanje konkurentnosti komponenti, prevazilaženje nedostatka globalnog sata i upravljanje nezavisnim otkazom komponenti. [1] Kada komponenta jednog sistema otkaže, ne sme pasti ceo sistem. [3] Primeri distribuiranih sistema variraju od sistema zasnovanih na SOA-i preko onlajn igara za više igrača do peer-to-peer aplikacija.

Računarski program koji radi u okviru distribuiranog sistema naziva se distribuirani program, [2] a distribuirano programiranje je proces pisanja takvih programa. [4] [5] Postoji mnogo različitih tipova implementacija za mehanizam za prosleđivanje poruka, uključujući čist HTTP, Web sokete, konektore slične RPC-u i redove poruka. [6]

Distribuirano računarstvo se takođe odnosi na upotrebu distribuiranih sistema za rešavanje računarskih problema. U distribuiranom računarstvu, problem je podeljen na mnogo zadataka, od kojih svaki rešava jedan ili više računara, [7] koji međusobno komuniciraju putem slanja poruka. [4]

S obzirom na neverovatnu potrebu za razmenom podataka putem mreže, distribuirano programiranje je sve prostranjenije. Na samoj aplikaciji se stavlja teret da funkcioniše sa nepouzdanim klijentima, nepouzdanim komunikacionim kanalima i da bude otporna na defekte u okviru svojih servisa. Iz tog razloga se sve više razvijaju alati koji programerima olakšavaju razvijanje ovako distribuiranih aplikacija. Jedan primer tih alata jeste WCF o kojem će biti više reči u nastavku.

Windows Communication Foundation

[8]

Osnova za komunikaciju u Windows-u (*eng. Windows Communication Foundation – WCF*) je okvir za izgradnju aplikacija orijentisanih na servise (*eng. service-oriented applications*). WCF omogućava slanje asinhronih poruka sa jedne krajnje tačke (*eng. endpoint*) servisa na drugu. Krajnja tačka usluge može biti deo stalno dostupnog servisa koji se izvršava u okviru IIS-a (*eng. Internet Information Services*) na Windows serveru ili može biti servis pokrenut u okviru neke aplikacije. Krajnja tačka može biti u obliku klijenta koji zahteva podatke od krajnje tačke servisa. Poruke mogu biti jednostavne poput jednog znaka, reči poslate kroz XML, ili složene kao tok binarnih podataka. Nekoliko slučaja korišćenja podrazumeva:

- Sigurna usluga za obradu poslovnih transakcija.
- Usluga koja drugima dostavlja trenutne podatke, kao što je izveštaj o saobraćaju ili drugi servis za praćenje.
- Usluga ćaskanja koja omogućava dvema ljudima da komuniciraju ili razmenjuju podatke u realnom vremenu.
- Aplikacija za kontrolnu tablu koja ispituje jednu ili više usluga za podatke i predstavlja ih u logičnoj prezentaciji.
- Izlaganje toka posla implementiranog pomoću *Windows Workflow Foundation*-a kao *WCF* usluge.

Iako je kreiranje takvih aplikacija bilo moguće pre postojanja *WCF*-a, *WCF* čini razvoj krajnjih tačaka lakšim nego ikada. Ukratko, *WCF* je dizajniran da ponudi pristupačan pristup kreiranju Web servisa i klijenata.

WCF funkcionalnosti

Orijentacija na usluge (*eng. Service orientation*)

Jedna od posledica korišćenja standarda *Web* usluga (*eng. Web service standard – WS*) je to što *WCF* omogućava kreiranje aplikacija orijentisanih na usluge. Servisno-orijentisana arhitektura (*SOA*) je oslanjanje na *Web* usluge za slanje i primanje podataka. Usluge imaju opštu prednost iz razloga što su labavo povezane (*eng. loosely-coupled*) umesto da su veze između njih čvrsto utisnute (*eng. hard-coded*). Slabo povezan odnos podrazumeva da svaki klijent kreiran na bilo kojoj platformi može da se poveže na bilo koju uslugu sve dok su ispunjeni osnovi dogovora.

Interoperabilnost (*eng. Interoperability*)

WCF implementira savremene industrijske standarde za interoperabilnost *Web* usluga. *WCF* klijenti i servisi su kompatibilni na nivou žice (*eng. wire-level*) sa poboljšanjima *Web usluga 3.0*

standarda. Svaki klijent i servis *Microsoft .NET*-a je kompatibilan, kada su *WCF* klijenti i usluge konfigurisani da koriste verziju specifikacije *WS* adresiranje iz avgusta 2004. godine.

Više obrazaca poruka (*eng. Multiple Message Patterns*)

Poruke se razmenjuju na jedan od nekoliko obrazaca. Najčešći obrazac jeste u vidu sinhrone zahtev/odgovor (*eng. request/response*) komunikacije, gde jedna krajnja tačka zahteva podatke od druge krajnje tačke, dok druga krajnja tačka odgovara. Postoje i drugi obrasci kao što je jednosmerna poruka u kojoj jedna krajnja tačka šalje poruku bez ikakvog očekivanja odgovora. Složeniji obrazac je obrazac dupleks razmene gde dve krajnje tačke uspostavljaju vezu i šalju podatke napred-nazad, slično programu za razmenu trenutnih poruka.

Meta-podaci usluga (*eng. Service metadata*)

WCF podržava objavljivanje meta-podataka servisa koristeći formate navedene u industrijskim standardima kao što su *WSDL* (*eng. Web Service Description Language*), *XML* shema i *WS-Policy*. Meta-podaci servisa podrazumevaju kako se ovaj servis poziva i koje vrste odgovora se mogu očekivati, te se mogu koristiti za automatsko generisanje i konfigurisanje klijenata za pristup *WCF* uslugama. Meta-podaci se mogu objaviti preko *HTTP*-a i *HTTPS*-a ili korišćenjem standarda za razmenu meta-podataka *Web* usluga (*eng. Web Service Metadata Exchange standard*).

Ugovori o podacima (*eng. Data contracts*)

Pošto je *WCF* izgrađen pomoću *.NET Framework*-a, on takođe uključuje metode preglednog koda (*eng. code-friendly*) koje omogućavaju kreiranje sopstvenih standarda odnosno ugovora koji su u obavezi da se ispoštuju, kako bi komunikacija između klijenata i servisa bila moguća. Jedna od univerzalnih vrsta ugovora je ugovor o podacima. U suštini, kreiranje usluge koristeći *Visual C#* ili *Visual Basic*, najlakši način za rukovanje podacima je kreiranje klasa koje predstavljaju entitet podataka sa svojstvima koja pripadaju entitetu podataka. *WCF* uključuje sveobuhvatan sistem za rad sa podacima na izuzetno lak način. Kada se kreira klasa koja predstavlja podatke, usluga automatski generiše meta-podatke koji omogućavaju klijentima da budu u skladu sa tipovima podataka koji su dizajnirani.

Bezbednost (*eng. Security*)

Poruke mogu biti šifrirane radi zaštite privatnosti i od korisnika je moguće zahtevati da se autentifikuju pre nego što im bude dozvoljeno da primaju poruke. Bezbednost se može implementirati korišćenjem dobro poznatih standarda kao što su *SSL* ili *WS-SecureConversation*.

Višestruki transporti i kodiranja (*eng. Multiple transports and encodings*)

Poruke se mogu slati na bilo koji od nekoliko ugrađenih transportnih protokola i kodiranja. Najčešći protokol i kodiranje je slanje tekstualno kodiranih *SOAP* poruka koristeći *HTTP*-a za upotrebu na *World Wide Web*-u. Alternativno, *WCF* omogućava slanje poruka preko *TCP*-a, imenovanih cevi (*eng. pipe*) ili redova poruka (*MSMQ – Microsoft Message Queueing*). Ove poruke mogu biti kodirane kao tekst ili korišćenjem optimizovanog binarnog formata. Binarni podaci se mogu efikasno slati

koristeći *MTOM* (*eng. Message Transmission Optimization Mechanism*) standard. Takođe je moguće kreirati i sopstveni prilagođeni transport ili kodiranje.

Pouzdane poruke u redu čekanja (*eng. Reliable and queued messages*)

WCF podržava pouzdanu razmenu poruka korišćenjem pouzdanih sesija implementiranih preko ***WS-Reliable Messaging*** i korišćenjem ***MSMQ***-a. Pouzdane sesije obezbeđuju pouzdan prenos poruka sa jedne krajnje tačke na drugi, odnosno između izvora i odredišta koristeći ***WS-Reliable Messaging*** protokol bez obzira na broj ili tip posrednika koji razdvajaju krajnje tačke za razmenu poruka (izvor i odredište). Ovo uključuje sve transportne posrednike koji ne koriste *SOAP* (na primer, *HTTP* proksije) ili posrednike koji koriste *SOAP* (na primer, rutere ili mostove zasnovane na *SOAP*-u) koji su potrebni za protok poruka između krajnjih tačaka. Pouzdane sesije koriste prozor prenosa u memoriji da maskiraju greške na nivou *SOAP* poruka i ponovo uspostavljaju veze u slučaju neuspeha transporta. Pouzdane sesije pružaju pouzdan prenos poruka sa malim kašnjenjem. Oni obezbeđuju *SOAP* poruke preko bilo kojih proksija ili posrednika, što je ekvivalentno onome što *TCP* obezbeđuje za pakete preko *IP* mostova.

Trajne poruke (eng. Durable messages)

Trajna poruka je ona koja se nikada ne gubi, čak i kada nastupi prekid u komunikaciji. Poruke u trajnom obrascu poruka se uvek čuvaju u bazi podataka. Ako dođe do prekida, baza podataka omogućava da se razmena poruka nastavi kada se veza uspostavi, a neprimljene poruke bivaju ponovo prosleđene odredištima. Takođe je moguće kreiranje trajnih poruka koristeći ***Windovs Workflow Foundation (WF)***.

Transakcije (eng. Transactions)

WCF podržava transakcije koristeći jedan od tri modela transakcija: *WS-AtomicTransactions*, API-je u imenskom prostoru System.Transactions i Mikrosoftovo rešenje koordinatora za distribuirane transakcije (*eng. Microsoft Distributed Transaction Coordinator*).

Podrška za AJAX i REST (eng. AJAX and REST support)

REST je primer ***Web 2.0*** tehnologije koja se razvija. *WCF* se može konfigurisati da obrađuje "obične" *XML* podatke koji nisu umotani *SOAP* zaglavljima. *WCF* se takođe može proširiti da podrži specifične *XML* formate, kao što je *ATOM* (popularni *RSS* standard), pa čak i ne-*XML* formate, kao što je *JSON*.

Proširivost (eng. Extensibility)

WCF arhitektura ima brojne tačke proširivosti. Ako su potrebne dodatne mogućnosti, postoji veliki broj ulaznih tačaka koje omogućavaju proširenje ponašanja neke od usluga. Ove proširivosti se mogu ostvariti u okviru: servisnog modela, komunikacionih varijabli za ostvarivanje konekcije, način razmene poruka kroz kreirane komunikacione kanale, bezbednost pri komunikaciji, kao i meta-podatke servisa.

WCF integracija sa drugim Microsoft tehnologijama

WCF je fleksibilna platforma, koja se takođe koristi u nekoliko drugih *Microsoft* proizvoda. Ako razumete osnove *WCF*-a, imate neposrednu prednost ako takođe koristite bilo koji od ovih proizvoda.

Prva tehnologija koja se uparila sa *WCF*-om bila je osnova za tokove poslova u Windows-u (***eng. Windows Workflow Foundation – WF***). Tokovi posla pojednostavljuju razvoj aplikacija tako što korake u toku posla obuhvataju kao „aktivnosti“. Integracija je omogućila da bilo koji tok posla bude lako pokrenut u okviru WCF-a kao njegov servis.

Microsoft BizTalk Server R2 takođe koristi *WCF* kao komunikacionu tehnologiju. *BizTalk* je dizajniran da prima i transformiše podatke iz jednog standardizovanog formata u drugi. Poruke moraju biti isporučene u njegov centralni okvir za poruke gde se poruka može transformisati korišćenjem strogog mapiranja ili korišćenjem jedne od *BizTalk* funkcija kao što je mehanizam toka posla. *BizTalk* sada može da koristi ***WCF Line of Business (LOB)*** adapter za isporuku poruka u okvir za poruke.

Karakteristike hostinga ***Windows Server AppFabric*** aplikativnog servera su posebno dizajnirane za primenu i upravljanje aplikacijama koje koriste *WCF* za komunikaciju. Karakteristike hostinga uključuju bogat alat i opcije konfiguracije posebno dizajnirane za aplikacije koje podržavaju *WCF*.

Osnove WCF-a

WCF je *runtime* i skup *API*-ja za kreiranje sistema koji šalju poruke između servisa i klijenata. Ista infrastruktura i *API*-ji se koriste za kreiranje aplikacija koje komuniciraju sa drugim aplikacijama na istom računarskom sistemu ili na sistemu koji se nalazi na drugom čvoru, a kojem se može pristupiti uz pomoć mrežne konekcije. Sve što se može modelovati kao poruka (na primer, *HTTP* zahtev ili poruka u redu poruka) može biti predstavljeno na uniforman način u modelu programiranja. Ovo omogućava objedinjeni *API* za različite transportne mehanizme.

Model pravi razliku između klijenata, odnosno aplikacija koje iniciraju komunikaciju, i usluga odnosno servisa, što su zapravo aplikacije koje čekaju da klijenti komuniciraju sa njima i odgovore na tu komunikaciju. Jedna aplikacija može da deluje i kao klijent i kao usluga, ovo je specifično prilikom dupleks komunikacije i *peer-to-peer* mreže.

Poruke se šalju između krajnjih tačaka. Krajnje tačke su mesta gde se poruke šalju ili primaju (ili oboje) i definišu sve informacije potrebne za razmenu poruka. Usluga izlaže jednu ili više krajnjih tačaka aplikacije (kao i nula ili više krajnjih tačaka infrastrukture), a klijent generiše krajnju tačku koja je kompatibilna sa jednom od krajnjih tačaka usluge.

Krajnja tačka na standardno zasnovan način opisuje gde poruke treba da se šalju, kako ih treba poslati i kako poruke treba da izgledaju. Usluga može izložiti ove informacije kao meta-podatke koje klijenti mogu obraditi da bi generisali odgovarajuće *WCF* klijentske i komunikacione stekove.

Komunikacioni protokoli

Jedan od potrebnih elemenata komunikacionog steka je transportni protokol. Poruke se mogu slati preko intraneta i Interneta koristeći uobičajene transporte, kao što su *HTTP* i *TCP*. Uključeni su i drugi transporti koji podržavaju komunikaciju sa aplikacijama koje koriste redove čekanja poruka, kao i onih u okviru *P2P* ravnopravnih mreža (*eng. mesh networks*). Više transportnih mehanizama može se dodati pomoću ugrađenih tačaka proširenja *WCF*-a.

Još jedan obavezan element u komunikacijskom steku je kodiranje koje specificira kako je bilo koja poruka formatirana. *WCF* u svojoj osnovi obezbeđuje sledeća kodiranja:

- Kodiranje teksta, interoperabilno kodiranje.
- Kodiranje mehanizma za optimizaciju prenosa poruka (*MTOM*), koji je interoperabilan način za efikasno slanje nestrukturiranih binarnih podataka na i sa servisa.
- Binarno kodiranje za efikasan prenos.

Više mehanizama za kodiranje (na primer, kompresijsko kodiranje) može se dodati korišćenjem ugrađenih tačaka proširenja *WCF*-a.

WCF Terminologija

- **Poruka (*eng. message*)** – samostalna jedinica podataka koja se može sastojati od nekoliko delova, uključujući telo i zaglavlja.
- **Servis (*eng. service*)** – konstrukcija koja izlaže jednu ili više krajnjih tačaka, pri čemu svaka krajnja tačka izlaže jednu ili više uslužnih operacija.
- **Krajnja tačka (*eng. endpoint*)** – konstrukcija na kojoj se poruke šalju ili primaju (ili oboje). Sadrži lokaciju (adresu) koja definiše gde se poruke mogu slati, specifikaciju mehanizma komunikacije (vezivanje) koja opisuje kako poruke treba da se šalju i definiciju za skup poruka koje se mogu poslati ili primiti (ili oba) na toj lokaciji (ugovor o usluzi) koji opisuje koja poruka se može poslati.
    - **Krajnja tačka aplikacije (*eng. application endpoint*)** – krajnja tačka koju je aplikacija izložila i koja odgovara ugovoru o usluzi koji implementira aplikacija.
    - **Krajnja tačka infrastrukture (*eng. infrastructure endpoint*)** – krajnja tačka koja je izložena infrastrukturi da bi se olakšala funkcionalnost koja je potrebna ili koju pruža usluga koja se ne odnosi na ugovor o usluzi. Na primer, usluga može imati krajnju tačku infrastrukture koja pruža informacije o meta-podacima.
- **Adresa (*eng. address*)** – određuje lokaciju na kojoj se primaju poruke. Navedena je kao jedinstveni identifikator resursa (*eng. Unique Resource Identifier – URI*). Deo *URI* sheme imenuje transportni mehanizam koji će se koristiti za dostizanje adrese, kao što su *HTTP* i *TCP*. Hijerarhijski deo *URI*-ja sadrži jedinstvenu lokaciju čiji format zavisi od transportnog mehanizma. Adresa krajnje tačke omogućava kreiranje jedinstvene adrese za svaku krajnju tačku u usluzi ili, pod određenim uslovima, deljenje adrese preko krajnjih tačaka. Sledeći primer prikazuje adresu koja koristi *HTTPS* protokol sa portom koji nije podrazumevani:
    - https://cohovineri:8005/ServiceModelSamples/CalculatorService

- **Vezivanje (*eng. bidning*)** – definiše kako krajnja tačka komunicira sa svetom. Sastoji se od skupa komponenti koje se nazivaju povezujući elementi (*eng. binding elements*) koji se "slažu" jedan na drugi da bi stvorili komunikacionu infrastrukturu. U najmanju ruku, vezivanje definiše transport (kao što je *HTTP* ili *TCP*) i kodiranje koje se koristi (kao što je tekst ili binarno). Vezivanje može da sadrži elemente vezivanja koji specificiraju detalje kao što su bezbednosni mehanizmi koji se koriste za obezbeđenje poruka ili obrazac poruke koji koristi krajnja tačka.

- **Vezivni element (*eng. binding element*)** – predstavlja određeni deo vezivanja, kao što je transport, kodiranje, implementacija protokola na nivou infrastrukture (kao što je *WS-ReliableMessaging*) ili bilo koja druga komponenta komunikacionog steka.

- **Ponašanja (*eng. behaviours*)** – komponenta koja kontroliše različite aspekte *runtime*-a servisa, krajnje tačke, određene operacije ili klijenta. Ponašanja su grupisana prema obimu:
    - uobičajena ponašanja utiču na sve krajnje tačke globalno,
    - ponašanja usluge utiču samo na aspekte vezane za uslugu,
    - ponašanja krajnje tačke utiču samo na svojstva koja se odnose na krajnju tačku,
    - ponašanja na nivou operacije utiču na određene operacije.

  Na primer, jedno ponašanje usluge je prigušivanje, koje određuje kako usluga reaguje kada višak poruka preti da preplavi njene mogućnosti rukovanja. Ponašanje krajnje tačke, s druge strane, kontroliše samo aspekte koji su relevantni za krajnje tačke, kao što je kako i gde pronaći bezbednosne akreditive.

- **Vezivanja koje obezbeđuje sistem (*eng. system-provided bindings*)** – *WCF* uključuje niz sistemskih vezivanja. Ovo su kolekcije elemenata vezivanja koji su optimizovani za specifične scenarije. Na primer, *WSHttpBinding* je dizajniran za interoperabilnost sa uslugama koje implementiraju različite *WS-** specifikacije. Ova unapred definisana vezivanja štede vreme tako što predstavljaju samo one opcije koje se mogu ispravno primeniti na određeni scenario. Ako unapred definisano vezivanje ne ispunjava zahteve, moguće je kreirati proizvoljno prilagođeno povezivanje.

- **Konfiguracija naspram kodiranja (*eng. configuration versus coding*)** – kontrola aplikacije se može vršiti ili kroz kodiranje, kroz konfiguraciju ili kroz kombinaciju oba. Konfiguracija ima prednost u tome što dozvoljava nekom drugom osim programeru (na primer, administratoru mreže) da postavi parametre klijenta i usluge nakon što je kod napisan i bez potrebe za ponovnim kompajliranjem. Konfiguracija ne samo da omogućava postavljanje vrednosti kao što su adrese krajnjih tačaka, već takođe omogućava dalju kontrolu tako što omogućava dodavanje krajnjih tačaka, vezivanja i ponašanja. Kodiranje omogućava programeru da zadrži striktnu kontrolu nad svim komponentama usluge ili klijenta, a sva podešavanja urađena kroz konfiguraciju mogu se pregledati i po potrebi zameniti kodom.

- **Operacija servisa (*eng. service operation*)** – procedura definisana u kodu servisa koja implementira funkcionalnost za operaciju. Ova operacija je izložena klijentima kao metod na *WCF* klijentu. Metod može da vrati vrednost i može da uzme opcioni broj argumenata, ili da ne uzme argumente i da ne vrati odgovor. Na primer, operacija koja funkcioniše kao jednostavno „Zdravo“ može da se koristi kao obaveštenje o prisustvu klijenta i da započne niz operacija.

- **Ugovor servisa (*eng. service contract*)** – povezuje više kombinovanih operacija u jednu funkcionalnu jedinicu. Ugovor može da definiše podešavanja na nivou usluge, kao što su imenski prostor usluge, odgovarajući ugovor za povratni poziv i slična podešavanja. U većini slučajeva, ugovor se definiše kreiranjem interfejsa na programskom jeziku po izboru i primenom atributa *ServiceContractAttribute* na interfejs. Stvarni kod usluge je rezultat implementacije interfejsa.

- **Ugovor operacije (*eng. operation contract*)** – definiše parametre i povratni tip operacije. Prilikom kreiranja interfejsa koji definiše ugovor o usluzi, označava se i ugovor operacije primenom atributa *OperationContractAttribute* na svaku definiciju metoda koja je deo ugovora. Operacije se mogu modelovati kao uzimanje jedne poruke i vraćanje jedne poruke, ili kao uzimanje skupa tipova i vraćanje tipa. U drugom slučaju, sistem će odrediti format za poruke koje je potrebno razmeniti za tu operaciju.

- **Ugovor poruke (*eng. message contract*)** – opisuje format poruke. Na primer, izjavljuje da li elementi poruke treba da idu u zaglavlju, odnosu u telo, koji nivo bezbednosti treba primeniti na koje elemente poruke, itd.

- **Ugovor defekta (*eng. fault contract*)** – može se povezati sa uslužnom operacijom za označavanje grešaka koje se mogu vratiti pozivaocu. Operacija može imati nula ili više grešaka

povezanih sa njom. Ove greške su SOAP greške koje su modelirane kao izuzeci u modelu programiranja.

- **Ugovor podataka (*eng. data contract*)** – opisi u meta-podacima tipova podataka koje usluga koristi. Ovo omogućava drugima da sarađuju sa uslugom. Tipovi podataka se mogu koristiti u bilo kom delu poruke, na primer, kao parametri ili povratni tipovi. Ako usluga koristi samo jednostavne tipove, nema potrebe za eksplicitnim korišćenjem ugovora o podacima.

- **Hostovanje (*eng. hosting*)** – usluga mora biti hostovana u nekom procesu. Host je aplikacija koja kontroliše životni vek usluge. Usluge se mogu hostovati same od sebe (samostalne usluge) ili njima mogu upravljati postojeći procesi hostovanja.

- **Samostalna usluga (*eng. self-hosted service*)** – usluga koja se pokreće unutar procesne aplikacije koju je kreirao programer. Programer kontroliše njen životni vek, postavlja svojstva usluge, otvara uslugu (što je postavlja u režim slušanja) i zatvara uslugu.

- **Proces hostovanja (*eng. hosting process*)** – aplikacija koja je dizajnirana za hostovanje usluga. To uključuje *Internet Information Services* (*IIS*), *Windovs Activation Services* (*WAS*) i *Windovs Services*. U ovim hostovanim scenarijima, domaćin kontroliše životni vek usluge. Na primer, pomoću *IIS*-a je moguće podesiti virtuelni direktorijum koji sadrži servisni sklop i konfiguracionu datoteku. Kada se primi poruka, *IIS* pokreće uslugu i kontroliše njen životni vek.

- **Instanciranje (*eng. instancing*)** – usluga ima model instanciranja. Postoje tri modela instanciranja:
  - ***Single*** – u kojem jedan *CLR* (*eng. Common Language Runtime*) objekat opslužuje sve klijente,
  - ***Per Call*** – u kojem se kreira novi *CLR* objekat za rukovanje svakim pozivom klijenta,
  - ***Per Session*** – u kojem se kreira skup *CLR* objekata, po jedan za svaku zasebnu sesiju.

  Izbor modela instanciranja zavisi od zahteva aplikacije i očekivanog obrasca korišćenja usluge.

- **Klijentska aplikacija (*eng. client application*)** – program koji razmenjuje poruke sa jednom ili više krajnjih tačaka servisa. Klijentska aplikacija počinje kreiranjem instance *WCF* klijenta i pozivanjem metoda WCF klijenta. Važno je napomenuti da jedna aplikacija može biti i klijent i servis.

- **Kanal (*eng. channel*)** – konkretna implementacija vezivnog elementa. Vezivanje predstavlja konfiguraciju, a kanal je implementacija povezana sa tom konfiguracijom. Prema tome, postoji kanal povezan sa svakim elementom vezivanja. Kanali se slažu jedan na drugi da bi se stvorila konkretna implementacija vezivanja, odnosno stek kanala.

- ***WCF* klijent (*eng. WCF client*)** – konstrukcija klijent-aplikacija koja izlaže operacije usluge kao metode (u programskom jeziku *.NET Framework* po izboru, kao što je *Visual Basic* ili *Visual C#*). Bilo koja aplikacija može da hostuje *WCF* klijenta, uključujući aplikaciju koja hostuje uslugu. Stoga je moguće kreirati uslugu koja uključuje *WCF* klijente drugih usluga. *WCF* klijent se može automatski generisati korišćenjem alata *ServiceModel Metadata Utility Tool* (Svcutil.exe) i usmeravanjem na pokrenutu uslugu koja objavljuje meta-podatke.

- **Meta-podaci (*eng. metadata*)** – u usluzi, opisuje karakteristike usluge koje spoljni entitet treba da razume da bi komunicirao sa uslugom. Meta-podatke može koristiti *ServiceModel Metadata Utility Tool* (Svcutil.exe) za generisanje *WCF* klijenta i prateće konfiguracije koje klijentska aplikacija može da koristi za interakciju sa uslugom. Meta-podaci koje pruža usluga uključuju dokumente *XML* sheme, koji definišu ugovor o podacima usluge, i *WSDL* dokumente, koji opisuju metode usluge. Kada je omogućeno, *WCF* automatski generiše meta-podatke za uslugu proverom usluge i njenih krajnjih tačaka. Kako bi se objavili meta-podaci iz usluge, neophodno je eksplicitno omogućiti ponašanje meta-podataka.

- **Bezbednost (*eng. security*)** – u okviru *WCF*-a, uključuje:
  - poverljivost (*eng. confidentiality*) – šifrovanje poruka radi sprečavanja prisluškivanja,
  - integritet (*eng. integrity)* – sredstvo za otkrivanje neovlašćenog pristupa poruci,
  - autentifikaciju (*eng. authentication)* – sredstvo za validaciju servera i klijenata,
  - autorizaciju (*eng. authorization)* – kontrolu pristupa resursima

  Ove funkcije su obezbeđene ili korišćenjem postojećih bezbednosnih mehanizama, kao što je *TLS* preko *HTTP*-a (poznat i kao *HTTPS*), ili primenom jedne ili više različitih *WS*-* bezbednosnih specifikacija.

- **Sigurnosni režim transporta (*eng. transport security mode*)** – određuje da poverljivost, integritet i autentifikaciju obezbeđuju mehanizmi transportnog sloja (kao što je *HTTPS*). Kada se koristi transport kao što je *HTTPS*, ovaj način rada ima prednost jer je efikasan u svojim performansama i dobro razumljiv zbog svoje rasprostranjenosti na Internetu. Nedostatak je što se ova vrsta bezbednosti primenjuje zasebno na svaki skok u komunikacijskom putu, čineći komunikaciju podložnom napadu „čoveka u sredini“ (*eng. middleman*).

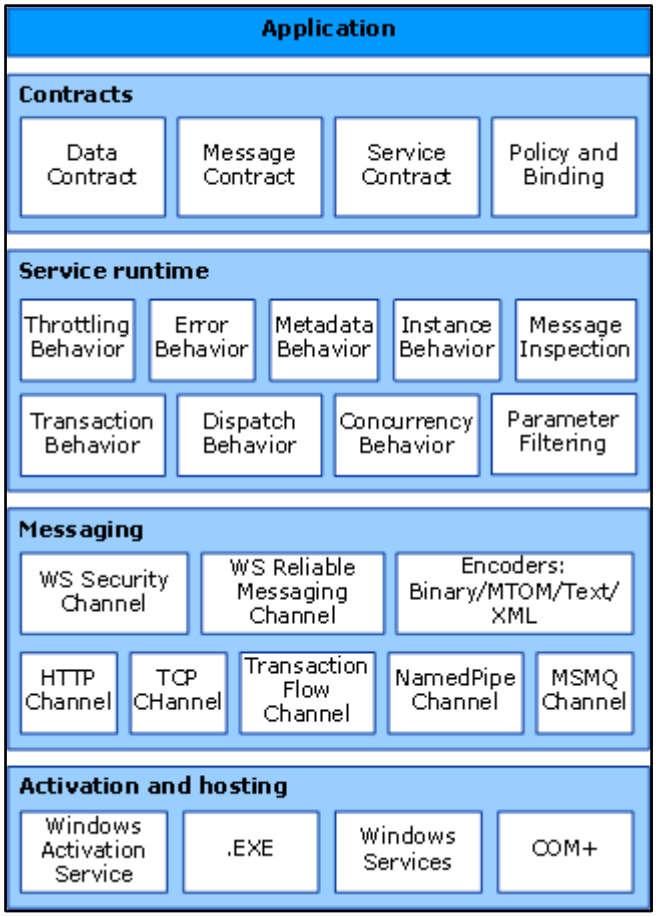
- **Režim bezbednosti poruka (*eng. message security mode*)** – određuje da se bezbednost obezbeđuje implementacijom jedne ili više bezbednosnih specifikacija, kao što je specifikacija pod nazivom *Web Services Security: SOAP Message Secuirty*. Svaka poruka sadrži neophodne mehanizme za obezbeđivanje bezbednosti tokom njenog tranzita, kao i mehanizme koje primaocu omogućuju detektovanje neovlašćenog menjanja poruke i dešifrovanje poruke. U tom smislu, bezbednost je inkapsulirana u svakoj poruci, obezbeđujući sigurnost od kraja do kraja (*eng. end-to-end*) u više skokova. Pošto bezbednosne informacije postaju deo poruke, takođe je moguće uključiti više vrsta akreditiva uz poruku (oni se nazivaju tvrdnje (*eng. claims*)). Ovaj pristup takođe ima prednost u tome što omogućava da poruka putuje bezbedno, nezavisno od načina transporta, uključujući višestruki transport između svog porekla i odredišta. Nedostatak ovog pristupa je složenost upotrebljenih kriptografskih mehanizama, što rezultira smanjivanjem performansi.

- **Transport sa režimom bezbednosti akreditiva za poruke (*eng. transport with message credential security mode*)** – određuje upotrebu transportnog sloja za obezbeđivanje poverljivosti, provere autentičnosti i integriteta poruka, dok svaka od poruka može da sadrži više akreditiva (tvrdnji) koje zahtevaju primaoci poruke.

- ***WS-**** – skraćenica za rastući skup specifikacija *Web* servisa (*WS*), kao što su WS-Security, *WS-ReliableMessaging* i tako dalje, koje su implementirane u *WCF*-u.

Arhitektura WCF-a

Na slici 1 prikazan je grafik koji ilustruje glavne slojeve arhitekture *Windows Communication Foundation (WCF)*-a.



Slika 1 – Glavni slojevi arhitekture WCF-a

Ugovori i opisi

Ugovori definišu različite aspekte sistema za razmenu poruka. Ugovor podataka opisuje svaki parametar koji čini svaku poruku koju usluga može da kreira ili koristi. Parametri poruke su definisani *XML* shemom (*XSD*), omogućavajući svakom sistemu koji razume *XML* da obrađuje dokumente. Ugovor poruke definiše specifične delove poruke koristeći *SOAP* protokole i omogućava finiju kontrolu nad delovima poruke, kada interoperabilnost zahteva takvu preciznost. Ugovor servisa specificira stvarne potpise metoda servisa i distribuira se kao interfejs na jednom od podržanih programskih jezika, kao što su *Visual Basic* ili *Visual C#*.

Vezivanja i polise (*eng. bindings and policies*) predviđaju uslove potrebne za komunikaciju sa uslugom. Na primer, vezivanje mora (najmanje) da navede transport koji se koristi (na primer, *HTTP* ili *TCP*) i kodiranje. Politika obuhvata bezbednosne zahteve i druge uslove koji moraju biti ispunjeni da bi se komuniciralo sa uslugom.

Runtime servisa

Sloj *runtime*-a usluge sadrži ponašanja koja se javljaju samo tokom stvarnog rada usluge, odnosno ponašanja usluge u toku izvršavanja. Prigušivanje kontroliše koliko poruka se obrađuje, što može da varira ako potražnja za uslugom poraste do unapred postavljenog ograničenja. Ponašanje greške određuje šta se dešava kada dođe do interne greške na servisu, na primer, kontrolišući koje informacije se saopštavaju klijentu. (Previše informacija može dati zlonamernom korisniku prednost u pokretanju napada). Ponašanje meta-podataka određuje kako i da li se meta-podaci stavljaju na raspolaganje spoljnom svetu. Ponašanje instance određuje koliko instanci usluge može da se pokrene (na primer, *singleton* specificira samo jednu instancu za obradu svih poruka). Ponašanje transakcije omogućava vraćanje transakcija u prethodno stanje ako dođe do greške. Dispečersko ponašanje je kontrola načina na koji se poruka obrađuje od strane *WCF* infrastrukture.

Proširivost omogućava prilagođavanje *runtime* procesa. Na primer, inspekcija poruke je mogućnost provere delova poruke, a filtriranje parametara omogućava da se dogode unapred podešene radnje na osnovu filtera koji deluju na zaglavlja poruke.

Razmena poruka

Sloj za razmenu poruka se sastoji od kanala. Kanal je komponenta koja obrađuje poruku na neki način, na primer, autentifikacijom poruke. Skup kanala je takođe poznat kao stek kanala. Kanali rade na porukama i zaglavljima poruka. Ovo se razlikuje od sloja *runtime*-a usluge, koji se prvenstveno bavi obradom sadržaja tela poruka.

Postoje dve vrste kanala:

1. **Transportni kanali** – čitaju i pišu poruke iz mreže (ili neke druge tačke komunikacije sa spoljnim svetom). Neki transporti koriste *encoder* da konvertuju poruke (koje su predstavljene kao XML infosetovi) u/iz prikaza toka bajtova koji koristi mreža. Primeri transporta su *HTTP*, imenovane cevi, *TCP* i *MSMQ*. Primeri kodiranja su *XML* i optimizovani binarni.

2. **Kanali protokola** implementiraju protokole za obradu poruka, često čitanjem ili pisanjem dodatnih zaglavlja poruke. Primeri takvih protokola uključuju *WS-Security* i *WS-Reliability*.

Sloj za razmenu poruka ilustruje moguće formate i obrasce razmene podataka. *WS-Security* je implementacija *WS-Security* specifikacije koja omogućava sigurnost na sloju poruka. *WS-Reliable Messaging* kanal omogućava garanciju isporuke poruke. Koderi predstavljaju niz kodova koji se mogu koristiti u skladu sa potrebama poruke. *HTTP* kanal navodi da se *HyperText Transport Protocol* koristi za isporuku poruka. *TCP* kanal na sličan način specificira *TCP* protokol. Kanal toka transakcije upravlja obrascima transakcijskih poruka. Kanal imenovane cevi (*eng. named pipe*) omogućava međuprocesnu komunikaciju. *MSMQ* kanal omogućava interakciju sa *MSMQ* aplikacijama.

Hosting i aktivacija

U svom konačnom obliku, usluga je program. Kao i drugi programi, usluga mora biti pokrenuta u izvršnom fajlu. Ovo je poznato kao samostalna usloga (*eng. self-hosted service*).

Usluge takođe mogu biti hostovane ili pokrenute u izvršnom fajlu kojim upravlja spoljni agent, kao što je *IIS* ili *Windows Activation Service* (*WAS*). *WAS* omogućava da se *WCF* aplikacije automatski aktiviraju kada se primene na računaru koji radi na *WAS*-u. Usluge se takođe mogu ručno pokrenuti kao izvršne datoteke (*.exe* datoteke). Usluga se takođe može automatski pokrenuti kao *Windows* usluga. *COM+* komponente se takođe mogu hostovati kao *WCF* usluge.

Primer korišćenja

Kako bi se demonstriralo korišćenje *WCF framework*-a, napravljena je demonstrativna aplikacija za razmenu poruka između korisnika (*eng. chat application*). Aplikacija je realizovana kroz klijentski i servisni deo, čija je implementacija odrađena u ***Visual C#*** programskom jeziku. U nastavku sledi opis aplikacije.

Servisni deo aplikacije ostvaren je kroz dva interfejsa: ***IMessagingService***, koji predstavlja ugovor servisa i ***iMessageCallback*** interfejs koji se koristi za pozive metoda na klijentskoj strani od strane servisa (*eng. callback*).

U okviru ***IMessagingService*** interfejsa na slici 2, u liniji 13 je dekoratorom ***ServiceContract*** definisano da se zahteva sesija pri ostvarivanju konekcije (***SessionMode*** => **Required**), kao i ***CallbackContract*** koji se koristi pri komunikaciji. Ovo je neophodno kada se radi sa *callback*-om, s obzirom da je nemoguće imati povratni odgovor ukoliko nije ostvarena sesija između servisa i klijenta. Ostatak interfejsa deklariše standardne metode za registrovanje, prijavljivanje na sistem, odjavljivanje sa sistema, pribavljanje i slanje poruka. Bitno je uočiti da ***OperationContract*** dekoratori za operacije koje nemaju povratnu vrednost su definisani kao ***IsOneWay = true***, što znači da se klijent pri pozivu ovih metoda ne blokira, već nastavlja svoje izvršavanje. Metode koje imaju povratnu vrednost su definisane sa ***IsOneWay = false***, što je inače podrazumevano podešavanja, ali je ovde stavljeno radi demonstracije.

```
13      [ServiceContract(SessionMode=SessionMode.Required, CallbackContract = typeof(IMessageCallback))]
        1 reference | Stefan, 8 days ago | 1 author, 2 changes
14      public interface IMessagingService
15      {
16          [OperationContract(IsOneWay = false)]
            1 reference | Stefan, 9 days ago | 1 author, 1 change
17          User Register(string username, string password);
18
19          [OperationContract(IsOneWay = false)]
            1 reference | Stefan, 9 days ago | 1 author, 1 change
20          User Login(string username, string password);
21
22          [OperationContract(IsOneWay = false)]
            1 reference | Stefan, 9 days ago | 1 author, 1 change
23          Chat JoinChat(string username, string chatID);
24
25          [OperationContract(IsOneWay = true)]
            1 reference | 0 changes | 0 authors, 0 changes
26          void Logout(string username);
27
28          [OperationContract(IsOneWay = true)]
            1 reference | Stefan, 8 days ago | 1 author, 2 changes
29          void SendMessage(string username, Message m);
30
31          [OperationContract(IsOneWay = false)]
            1 reference | Stefan, 9 days ago | 1 author, 1 change
32          Dictionary<string, Chat> GetChats(string username);
33
34          [OperationContract(IsOneWay = false)]
            1 reference | Stefan, 9 days ago | 1 author, 1 change
35          List<Message> GetMessages(string username,string chatID);
36      }
```

Slika 2 - IMessagingService

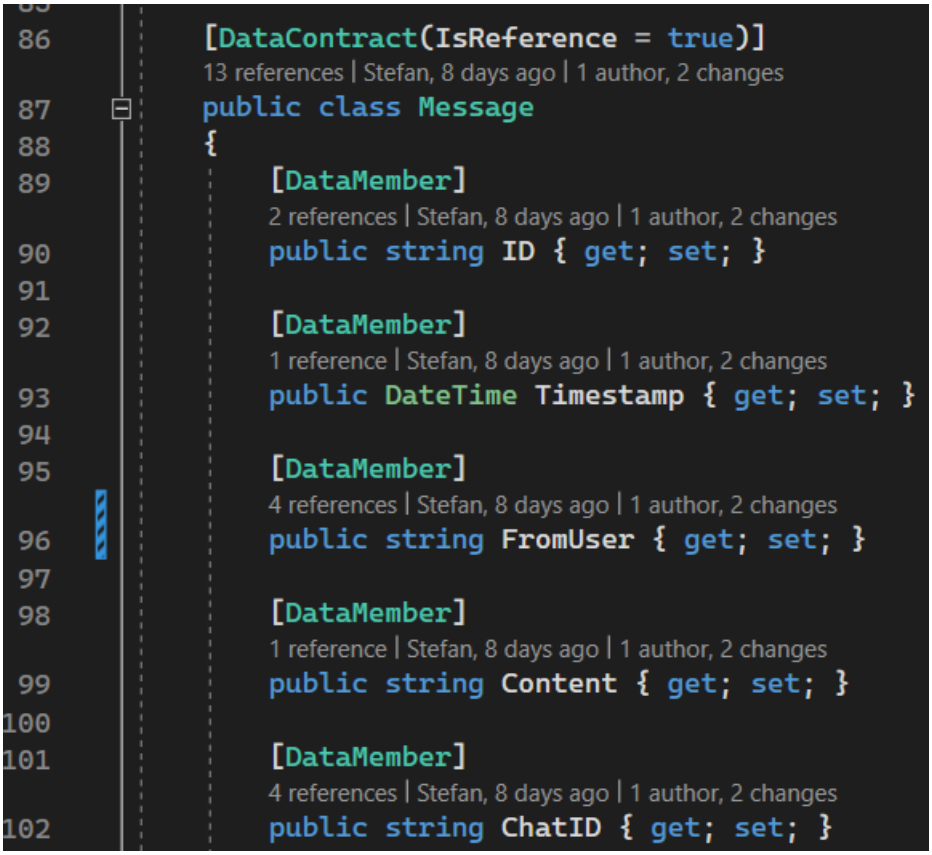
Takođe, ovaj interfejs uvodi i nove tipove, koje je neophodno definisati uz pomoć ***DataContract*** dekoratora, kako bi bili vidljivi klijentima. To su tipovi ***User*** slika 3, ***Chat*** slika 4 i ***Message*** slika 5. U okviru ovih tipova je bitno naznačiti da sve promenljive koje imaju dekorator ***DataMember*** moraju da budu javne, kao i da imaju mogućnost čitanja i ažuriranja vrednosti, odnosno *get()* i *set()* metode. One koje nemaju ***DataMember*** dekorator, neće biti dostupne na klijentskoj strani, te za njih ove restrikcije ne važe. Takođe ***User***, ***Chat*** i ***Message*** imaju uključenu opciju ***IsReference = true*** u okviru dekoratora ***DataContact***, što rešava problem ulančanih referenci pri serijalizaciji.

```
32      [DataContract(IsReference = true)]
        21 references | Stefan, 8 days ago | 1 author, 2 changes
33      public class User
34      {
35          [DataMember]
            3 references | 0 changes | 0 authors, 0 changes
36          public bool IsValid { get; set; }
37          [DataMember]
            3 references | Stefan, 8 days ago | 1 author, 2 changes
38          public string Username { get; set; }
39
40          [DataMember]
            3 references | Stefan, 8 days ago | 1 author, 1 change
41          public string Password { get; set; }
42
            6 references | Stefan, 8 days ago | 1 author, 1 change
43          public IMessageCallback Callback { get; set; }
44
45          [DataMember]
            4 references | Stefan, 9 days ago | 1 author, 1 change
46          public Dictionary<string, Chat> Chats { get; set; }
47
```

Slika 3 – User DataContract

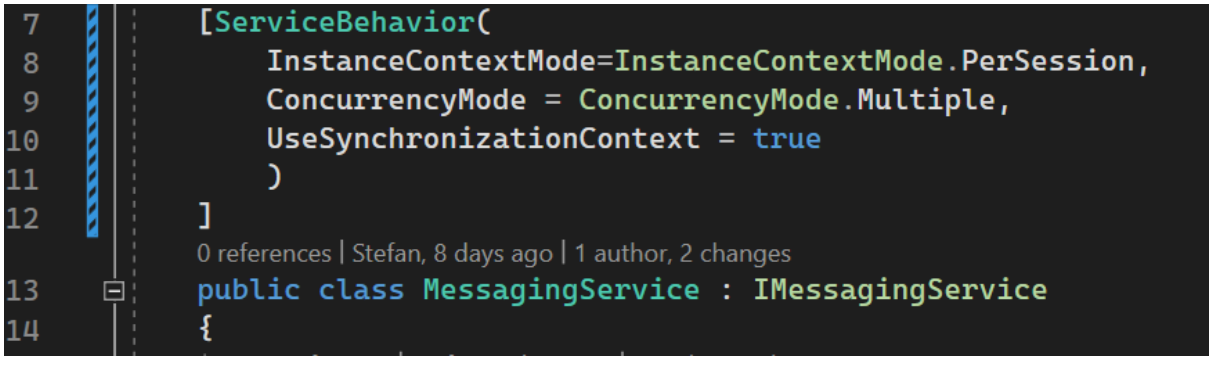
```
69      [DataContract(IsReference = true)]
        16 references | Stefan, 8 days ago | 1 author, 2 changes
70      public class Chat
71      {
72          [DataMember]
            3 references | Stefan, 8 days ago | 1 author, 1 change
73          public string ID { get; set; }
            7 references | Stefan, 8 days ago | 1 author, 1 change
74          public Dictionary<string, User> Users { get; set; }
75          [DataMember]
            4 references | Stefan, 8 days ago | 1 author, 1 change
76          public List<Message> Messages { get; set; }
```

Slika 4 – Chat DataContract



Slika 5 – Message DataContract

Implementacija ***IMessageService*** interfejsa je ostvarena samom klasom servisa ***MessageService*** čije instance se pokreću. Detalji implementacije neće biti prikazani, međutim dostupni su na repozitorijumu [9]. Bitno je napomenuti ***ServiceBehavior*** dekorator slika 5, linija 7, kojim se postavlja ***InstanceContextMode*** na **režim sesije**, što znači da će za svakog klijenta biti kreirana po jedna instanca servisa kada se bude povezao na isti. Te instance će se izvršavati paralelno, što je označeno opcijom ***ConcurrencyModeMultiple*** i između kojih će biti omogućena sinhronizacija kada se pristupa resursima ***UseSynchronizationContext = true***.



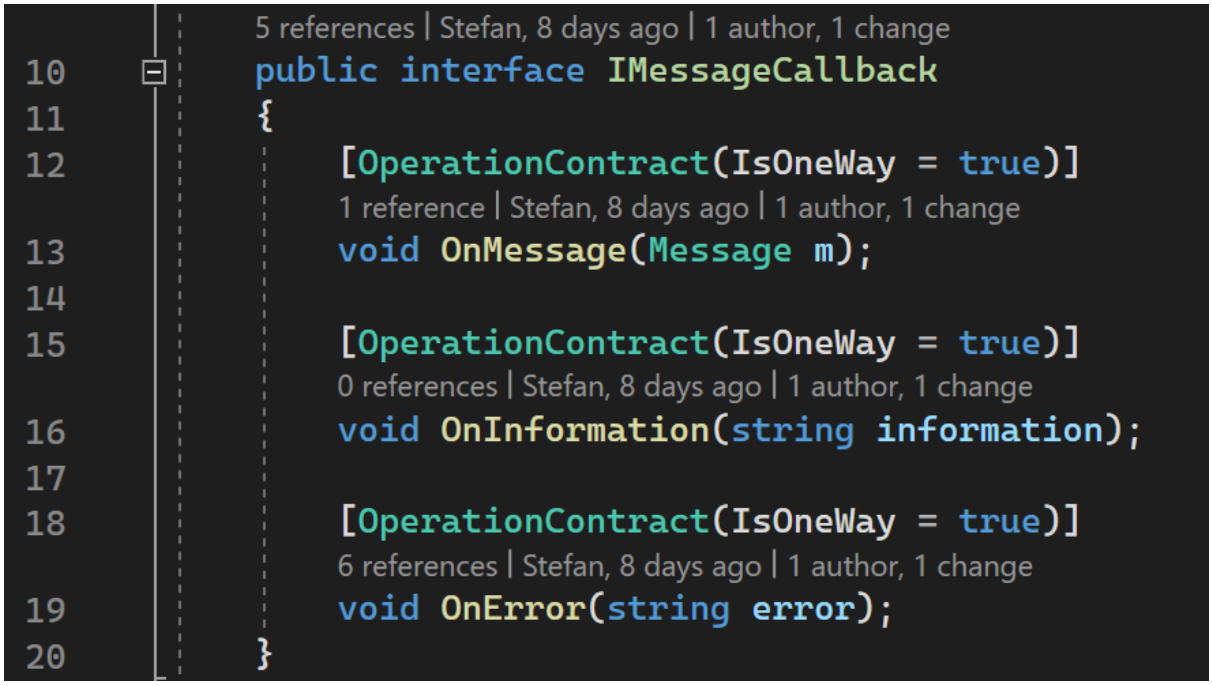
Slika 6 – MessagingService ServiceBehavior

Takođe, s obzirom da se radi sa nekom vrstom ***stateful*** servisa, podaci o korisnicima su čuvani u simuliranoj bazi, koja je ostvarena ***thread-safe singleton*** **obrascem** u okviru klase ***StorageAdapter*** sliak 7.

```
164     public sealed class StorageAdapter
165     {
166         private static StorageAdapter _instance = null;
167         private static readonly object padlock = new object();
168         private Dictionary<string, User> _users;
169         private Dictionary<string, Chat> _chats;
170         private Dictionary<string, Message> _messages;
171
                1 reference | Stefan, 9 days ago | 1 author, 1 change
172         public StorageAdapter(
173             Dictionary<string, User> users,
174             Dictionary<string, Chat> chats,
175             Dictionary<string, Message> messages
176             )
177         {
178             _users = users;
179             _chats = chats;
180             _messages = messages;
181         }
                20 references | Stefan, 9 days ago | 1 author, 1 change
182         public static StorageAdapter Instance
183         {
184             get
185             {
186                 lock (padlock)
187                 {
188                     if (_instance == null)
189                     {
190                         _instance = new StorageAdapter(
191                             new Dictionary<string, User>(),
192                             new Dictionary<string, Chat>(),
193                             new Dictionary<string, Message>());
194                     }
195                     return _instance;
196                 }
197             }
198         }
```

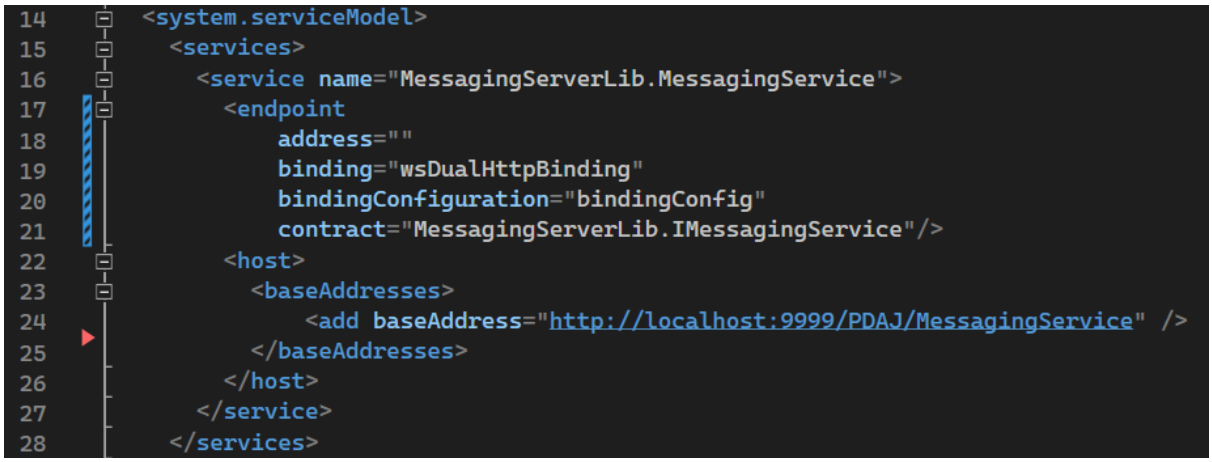
Slika 7 – StorageAdapter

IMessageCallback, slika 8, je interfejs koji je neophodno da implementira klijentska strana pri ostvarivanju konekcije sa servisom. U okviru ovog interfejsa su definisani metodi *OnMessage(...)*, *OnError(...)*, *OnInfo(...)*, koji se pozivaju u odgovarajućim situacijama, a koje klijent obrađuje na adekvatan način. Sve ove metode su definisane ***OperationContract***-om kao ***IsOneWay = true***, što sada znači da se servis neće blokirati kada ih pozove na izvršenje.



Slika 8 - IMessageCallback

Za pokretanje servisa je iskorišćen ***IIS***, te je konfiguracija adrese servisa, način komunikacije, odnosno vezivanja obavljeno u okviru konfiguracionih fajlova, odnosno u okviru fajla ***App.config***. Servis je konfigurisan da koristi *base adresu* (slika 9, linija 24). Vezivanje (*eng. binding*) koje je korišćeno je ***wsDualHttpBinding***, koje omogućava dupleks komunikaciju (slika 9, linija 19). Konfiguracija samog vezivanja je data na slici 10, a ime konfiguracije je ***bindingConfig*** (slika 9, linija 20). Na kraju, kao ugovor servisa je prosleđen ***IMessagingService*** interfejs (slika 9, linija 21).

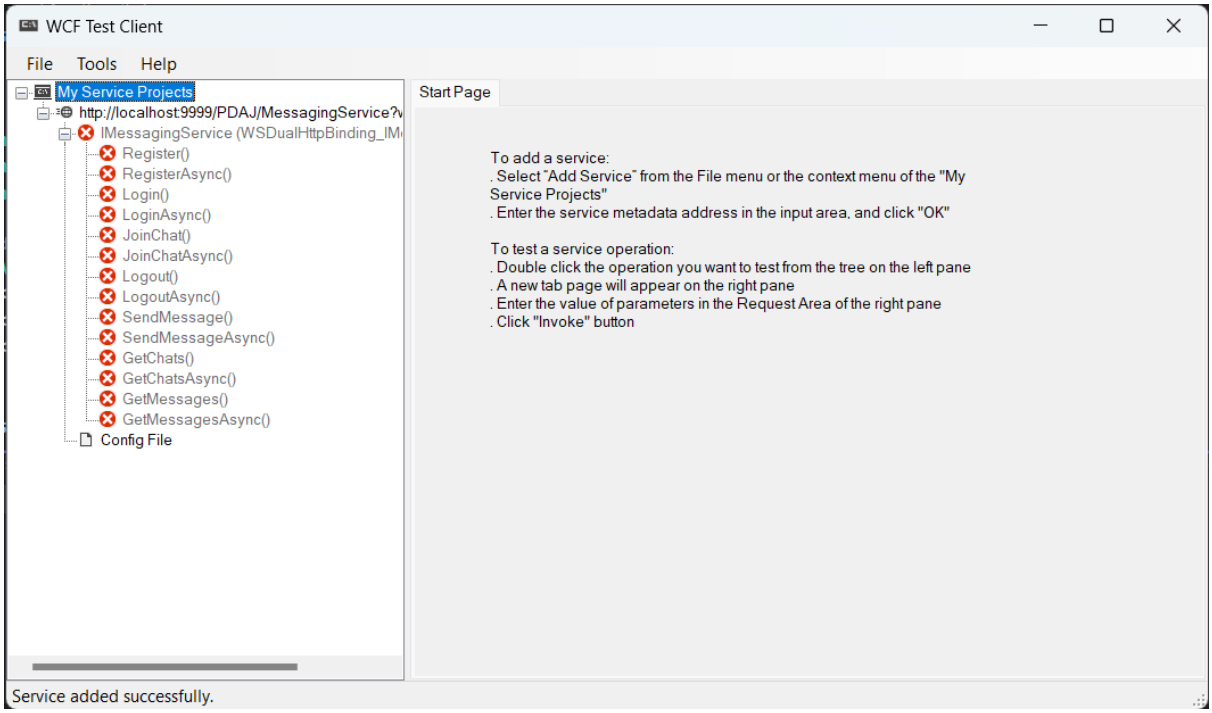


Slika 9 – App.config konfiguracija servisa

Takođe, na slici 10 je pored konfiguracije vezivanja prikazana konfiguracija mapiranja protokola za vezivanja. Za vezivanje su podešeni tajmeri za otvaranje, zatvaranje, slanje i prijem poruke na 10min, a veličina poruke koja se prima na 2.000.000.000 bajtova (linije 33-39).

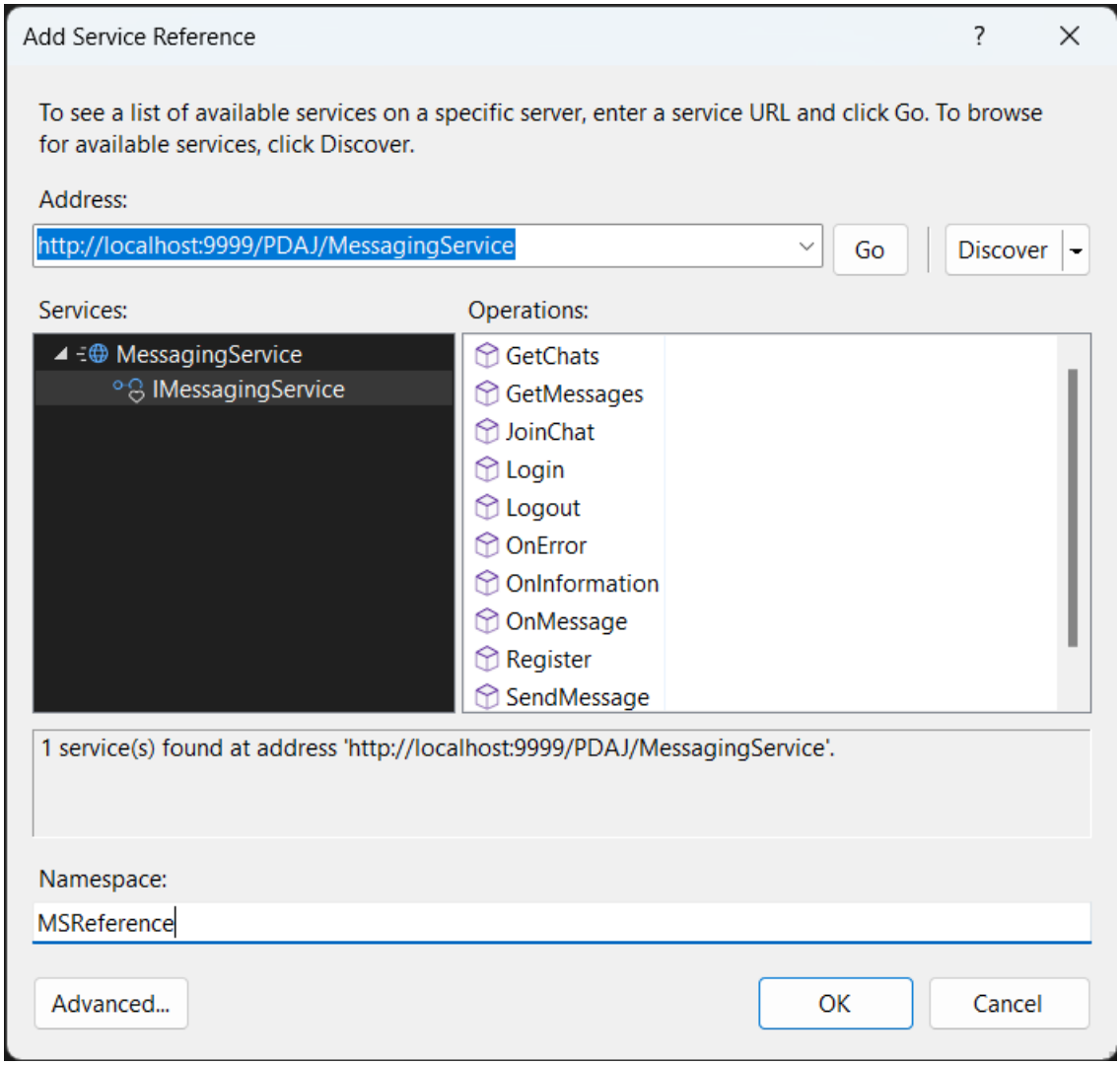
```
    <bindings>
    <!-- Configure a WSDualHttpBinding that supports duplex -->
    <!-- communication. -->
    <wsDualHttpBinding>
        <binding transactionFlow="true"
                 name="bindingConfig"
                 openTimeout="00:10:00"
                 closeTimeout="00:10:00"
                 sendTimeout="00:10:00"
                 receiveTimeout="00:10:00"
                 maxReceivedMessageSize="2000000000">

            <readerQuotas maxDepth="2000000000"
                 maxStringContentLength="2000000000"
                 maxArrayLength="2000000000"
                 maxBytesPerRead="2000000000"
                 maxNameTableCharCount="2000000000" />
        </binding>
    </wsDualHttpBinding>
</bindings>
    <protocolMapping>
            <add
                binding="wsDualHttpBinding"
                scheme="http"
                bindingConfiguration="bindingConfig" />
        </protocolMapping>
```

Slika 10 – App.config konfiguracija vezivanja i protokola



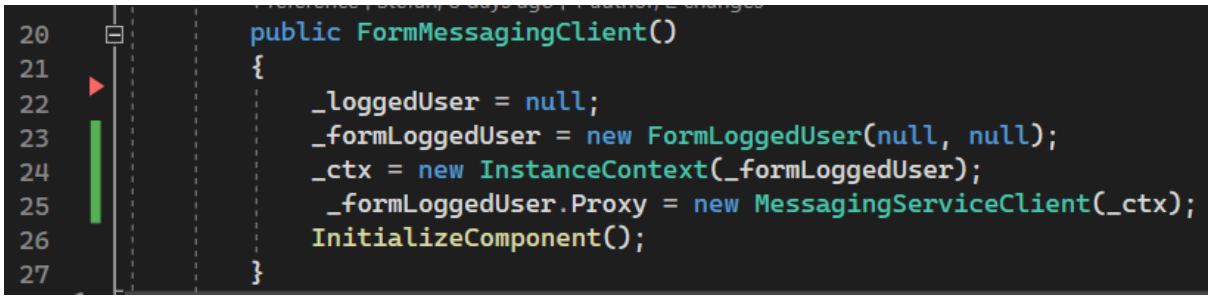
Slika 11 – IIS pokrenut servis

Na kraju, kada se pokrene sam servis, dobije se interfejs prikazan na slici 11. Ovi su meta-podaci servisa dostupni svim klijentima koji mogu putem mreže pristupiti servisu. Kako bi klijent mogao da očita podatke i automatski generiše komunikaciju na svom kraju, neophodno je dodatu referencu na servisu u okviru projekta klijentske aplikacije slika 12.



Slika 12 – Dodavanje reference na servis u okviru klijentske aplikacije

Nakon što se klijent konfiguriše na adekvatan način dodavajući referencu na servis, instanciranje *proxy* objekta je moguće učiniti u jednoj do dve linije koda. Na slici 13, u okviru linije 24 se kreira ***InstanceContext*** objekat, koji predstavlja instancu za komunikaciju sa serverom. S obzirom da je neophodno obezbediti sesiju, odnosno instancu klase koja implementira ***IMessageCallaback*** interfejs, prosleđuje se instanca takve klase, u ovom slučaju je to ***FormLoggedUser*** instanca iz linije 23. U okviru linije 25 se kreira ***proxy*** objekat, odnosno sama klijent instanca koja je tipa ***IMessageService*** interfejs, kroz koju se vrši pozivanje metoda servisa.



Slika 13 – Instanciranje klijentske strane WCF-a

Zaključak

WCF pruža vrlo elegantan pristup distribuiranom programiranju. Veoma je fleksibilan alat za kreiranje aplikacija koje prate arhitekturni model klijent-server. Već je napomenuto da svaki klijent može imati i ulogu servisa, te je moguće kreirati i aplikacije sa arhitekturom *point-to-point*. Konfiguracija komunikacije je takođe dosta fleksibilna, s obzirom da podržava veći broj standarda mrežne komunikacije, bezbednosne modele i slično.

Međutim nije sve toliko bajno. Postoji i dosta restrikcija pri radu sa alatom. Jedan od njih je veličina podataka koji se prenose. Naime, s obzirom da je sve neophodno serijalizovati pre, odnosno deserijalizovati nakon slanja, postavljeno je ograničenje u samoj složenosti objekata koji se prenose. Moguće je izmeniti maksimalnu količinu bajtova koji se prenose u poruci, međutim i dalje je nemoguće očekivati od čvorova da bude u stanju da podrže ogromnu serijalizaciju i deserijalizaciju poruka pri komunikaciji. Ovo predstavlja problem pri projektovanju aplikacije, jer je neophodno ili imati duplirati klase, ili odmah na početku voditi računa da nema previše ulančanih referenci.

Takođe, za programere *.Net Core* verzije, neophodno je koristiti *CoreWCF* [10] verziju, koja je publikovana u javnost u aprilu 2022. godine. Ova verzija je donela neke izmene u odnosu na već postojeći *WCF* za *.Net Framework*, te je neophodno prilagoditi se istim.

Čak i sa pojedinim ograničenjima, *Windows Communication Foundation* je odličan alat za razvoj *REST* aplikacija, gde je poziv metoda vrlo transparentan, odnosno sakriven iza *RPC* poziva, što otvara vrata programerima u svet distribuiranog programiranja.

Reference

[1] A. Tanenbaum i v. S. Maarten, Distributed Systems, New York City: CreateSpace Independent Publishing Platform, 2017.

[2] K. Apt, F. de Boer i E.-R. Olderog, Verification of Sequential and Concurrent Programs (Texts in Computer Science), London: Springer, 2010.

[3] A. Arpaci-Dusseau i R. Arpaci-Dusseau, „Distributed Systems," u *Operating systems: Three Easy Pieces*, Wisconsin–Madison, Arpaci-Dusseau Books, 2016.

[4] G. R. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming, Addison Wesley, 1999.

[5] S. Ghosh, Distributed Systems: An Algorithmic Approach, Second Edition (Chapman & Hall/CRC Computer and Information Science Series), Iowa: Chapman and Hall/CRC, 2014.

[6] L. Magnoni, „Modern Messaging for Distributed Sytems," Journal of Physics, 2015.

[7] B. Godfrey, „A primer on distributed computing," 2006.

[8] Microsoft, „What Is Windows Communication Foundation," 16 12 2021. [Na mreži]. Available: https://learn.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf.

[9] S. Aleksić, *MessagingService implementacija,* Novi Sad, 2022.

[10] *CoreWCF,* 2022.