VISOKA POSLOVNA ŠKOLA
STRUKOVNIH STUDIJA
ČAČAK

DIPLOMSKI RAD

Nadzor i upravljanjelaboratorijsko industrijskom tablom “PROCON 38-714” korižcenjem OPC XML-DA SERVERA

Mentor:_____ Student: _____
Profesor: _____ Br.Indeksa: _____

Zadatak diplomskog rada

Zadatak rada je upoznavanje sa laboratorijsko-industrijskom tablom ”PROCON 38-714”; upoznavanje sa programskim paketom Microsoft Visual Studio.NET i primena .NET tehnologije kroz upotrebu programskog jezika C#, kao i korišćenje XML Web servisa.

Za potpuno razumevanje .NET neophodno je upoznavanje sa XML specijalizovanim jezikom, pomoću koga je napravljen SOAP protokol. Zbog korišćenja OPC XML-DA specifikacije i XML-DA servera neophodno je upoznavanje sa OPC specifikacijama.

Osnovni zadatak diplomskog rada je pravljenje aplikacije u programskom jeziku C#, koja vrši nadzor i upravljanje laboratorijsko-industrijskom tablom”PROCON 38-714”, koristeći pri tome OPC server veličina i XML-DA server za pristup veličinama sa laboratorijske table. Komunikacija između table i aplikacije odvija se preko Interneta uz upotrebu XML Web servisa.

Sadržaj:

1. Uvod

Svedoci smo veoma brzog razvoja industrije u drugoj polovini prethodnog veka, a naročito u poslednjih deset godina. Računarstvo i upravljanje sistemima se razvija u pravcima koji u skorijoj prošlosti nisu mogli ni da se pretpostave.

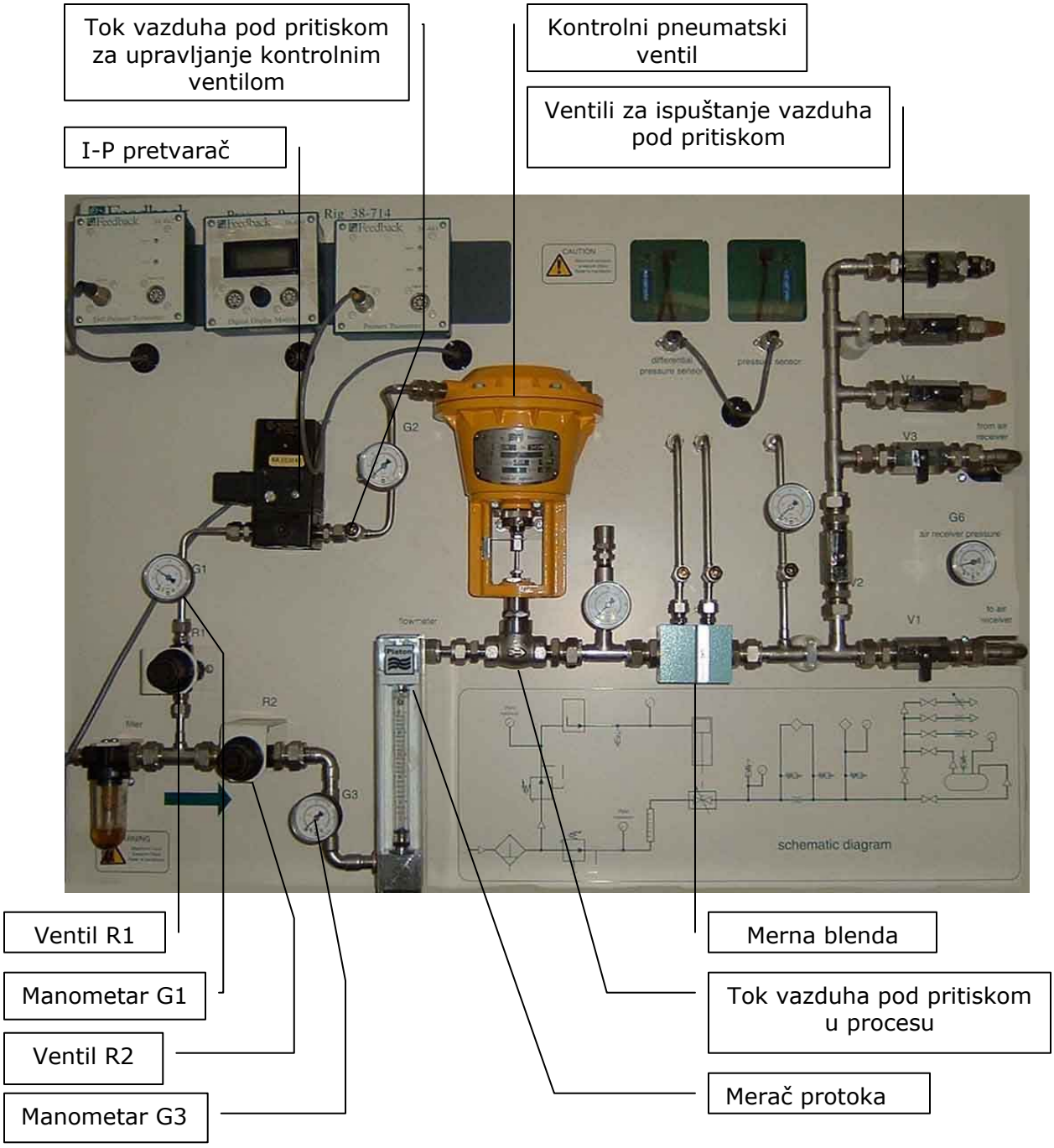
Paralelno sa razvojem samog računarstva formirao se i usavršavao koncept nadzorno – upravljačkih sistema, SCADA (**S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition - Sistemi za kontrolu i nadzor) sistema. SCADA se odlikuje prostornom distribucijom autonomnih računarskih komponenti, koje su komunikacionom mrežom spregnute u jedinstven sistem. Da bi se ostvario cilj SCADA sistema, potrebno je omogućiti razmenu poruka između njegovih komponenti.

U ovom radu biće prikazano jedno rešenje klijentske aplikacije koja vrši nadzor i upravljanje preko Interneta laboratorijsko industrijskom tablom PROCON 38-714, po zahtevima OPC XML-DA specifikacije uz oslonac na OPC XML server veličina.

Do sada je u okviru ove teme urađeno nekoliko radova. Najpre je u jeziku Delphi implementirana aplikacija koja vrši vizuelizaciju termo-procesnog sistema simuliranog u Matlab-u [1]. Nakon toga, uz oslonac na COM tehnologiju, u okruženju Microsoft Visual C++, realizovana je programska podrška za komunikaciju između računara i regulatora COMMANDER 300 [2]. Sledeće rešenje bila je klijent /server aplikacija realizovana u programskom jeziku Delphi. Cilj apliakcije je nadzor rada simuliranog termo procesnog sistema. Nadzor je vršen na klijentskoj strani, a simulacija je izvedena na serverskoj strani [3].

U ovom radu biće prikazano jedno rešenje nadzora i upravljanja laboratorijsko industrijskom tablom PROCON 38-714, uz oslonac na .NET tehnologiju, korišćenjem programskog jezika C# u okruženju Microsoft Visual Studio .NET. Novina u odnosu na prethodna rešenja je komunikacija između klijentske aplikacije i serverske strane preko Interneta.

2. Laboratorijsko industrijska tabla



Slika 2.1 Laboratorijsko industrijska tabla PROCON 38-714

Na slici 2.1 prikazana je laboratorijsko industrijska tabla (u daljem tekstu tabla), sa oznakom PROCON 38-714. Svrha table je upoznavanje i prikazivanje osnovnih principa u upravljanju sistemima i merenjima u industriji. Iako u nazivu stoji pridev laboratorijska, na tabli su upotrebljene realne industrijske komponente, a svi eksperimenti vršeni na njoj izvođeni su u laboratorijskim uslovima. Dotični pridev se odnosi na primenu table, to jest, izvođenje eksperimenata.

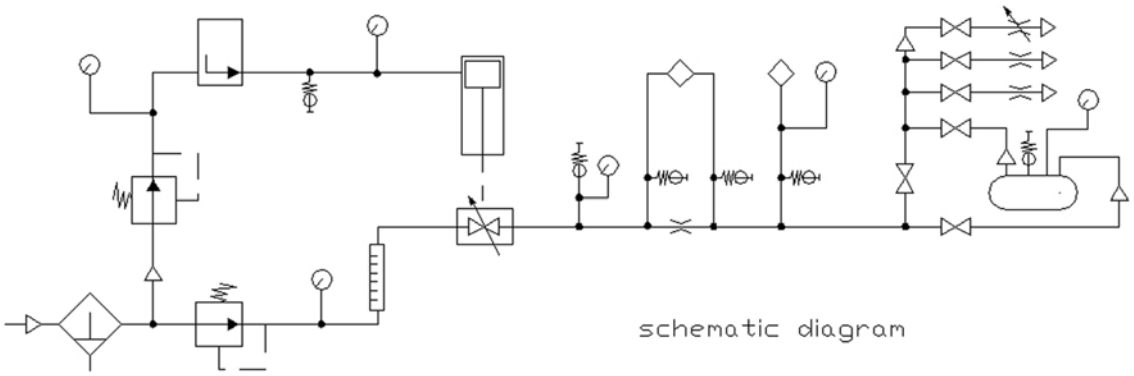
Tabla se sastoji od glavne cevi na koju su postavljeni kontrolni pneumatski ventil, merna blenda, merač protoka i ventili za ispuštanje vazduha pod pritiskom. Tok vazduha pod pritiskom se prazni direktno u atmosferu ili preko primača vazduha, čime se menja odziv u sistemu. Upravljačkim pneumatskim vrentilom upravlja se na osnovu signala iz pretvarača struje u pritisak, i signala sa senzora za apsolutni i diferencijalni pritisak koji pokazuju vrednosti pritiska i protoka, respektivno. Tabla je dizajnirana da radi sa procesnim interfejsom, u oznaci 38-200, i regulatorom, u oznaci 38-300, u zatvorenoj i otvorenoj povratnoj sprezi. Izvor vazduha pod pritiskom je neophodno obezbediti i za tok vazduha i za upravljanje ventilom.

Tabla se sastoji od dva glavna toka: tok za upravljanje kontrolnim ventilom, i tok vazduha pod pritiskom u procesu (glavna cev).

Vazduh pod pritiskom za upravljanje ventilom se određuje ventilom R1, a pritisak je prikazan na manometru G1. Vazduh za sam proces je određen ventilom R2, a pritisak je prikazan na manometru G3. Regulator, u oznaci 38-300, koristi se za formiranje sistema u zatvorenoj povratnoj sprezi. Pretvarač struje u pritisak koristi na svom ulazu standardizovan strujni signal 4-20 mA, sa procesnog interfejsa, a na izlazu daje pritisak u opsegu 3-15 psi.

Tok vazduha pod pritiskom u procesu prolazi kroz upravljački ventil i mernu blendu nakon čega ističe u atmosferu. Moguće je i isticanje vazduha preko primača vazduha, koji može biti povezan serijski ili paralelno sa procesom. Na ovaj način se menja odziv sistema.

Na slici 2.2 prikazana je šema procesa sa table.

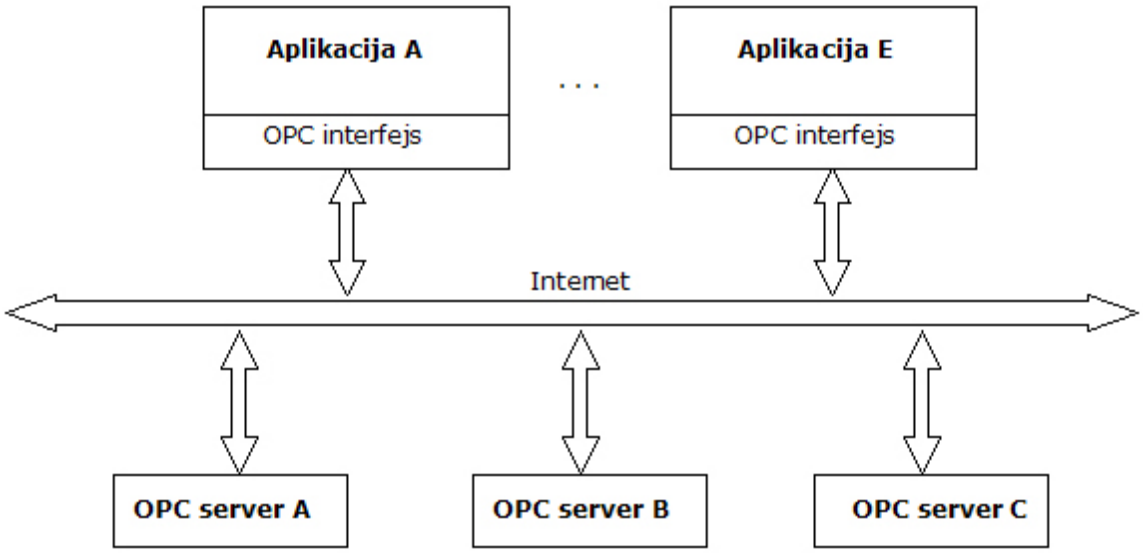


Slika 2.2 Šema procesa na tabli PROCON 38-714

U okviru opreme za tablu nalaze se i dva transmitera: 38-461 transmiter pritiska, i 38-462 transmiter diferencijalnog pritiska. Ovi transmiteri su predviđeni za montiranje na magnetni panel, koji se nalazi na tabli. Transmiteri su opskrbljeni petopinskim DIN priključcima što im omogućuje povezivanje sa procesnim interfejsom. Na taj način je omogućen nadzor i upravljanje sistemom (tokom vazduha pod pritiskom).

3. OPC XML-DA specifikacija

OPC foundation (***O**bject Linking and Embedding for **P**rocess **C**ontrol***)** je neprofitabilna fondacija koja ima u svom članstvu preko 100 vodećih firmi među kojima se nalazi i Microsoft. OPC fondacija radi na standardizaciji razmene raznih vrsta podataka. Njihov glavni cilj je stvaranje otvorene i efikasne komunikacione arhitekture koja se koncentriše na razmenu podataka, a ne na vrstu podataka.



Slika 3.1 Komunikacija između klijenta i servera po OPC specifikaciji

OPC fondacija je definisala interfejse ka serveru podataka (*Data Access Server*), serveru alarma i događaja (*Alarms and Events Server*), serverima za posredno upravljanje proizvodnjom (*Batch Server*) i serveru pristupa podacima iz istorije (*History Data Access Server*). Informacije koje su neophodne okruženju dostupne su aplikacijama u okruženju preko interfejsa razvijenih na OLE/COM tehnologiji.

Korišćenjem XML-a, kao i jezika baziranim na XML-u, na novi način su objašnjeni opis i razmena strukturiranih informacija između dve povezane aplikacije. XML je tehnologija koja je dostupna kroz širok dijapazon platformi. OPC XML Data Access (OPC XML-DA) je rezultat usvajanja seta XML tehnologija sa ciljem da olakšaju razmenu podataka preko Interneta. Svrha ovog rada je nastavak ideje omogućavanja i promovisanja interoperabilnosti među aplikacijama.

XML-DA bazirani interfejsi će pojednostaviti deljenje i razmenu OPC podataka između raznih nivoa hijerarhije, za različite platforme. Cilj ove specifikacije je da omogući podršku za HTTP i SOAP.

OPC XML-DA je razvijan pod uticajem koncepata na kojima je razvijan Simple Object Access Protocol – protokol za pristup jednostavnim objektima (u daljem tekstu SOAP). OPC XML - DA je modelovan na način koji dozvoljava svojim informacijama da budu isporučene u okviru SOAP poruke kao SOAP telo.

Za razliku od prethodnih OPC specifikacija za pristup podacima, OPC XML-DA specifikacija opisuje razmenu podataka u okruženju koje pretpostavlja da nema stalne veze između klijenta i servera[5].

3.1 Microsoft Visual Studio .NET

Microsoft Visual Studio .NET obezbeđuje razvojne alate, radno okruženje, serversku infrastrukturu i omogućava izradu aplikacija za razne platforme i uređaje. Ovo okruženje ima u sebi integrisane razne aplikacije, koristeći standarde kao što su XML i SOAP. .NET uspeva da prevaziđe probleme koji se javljaju prilikom razmene podataka između aplikacija pisanih u različitim programskim jezicima.

.NET Framework je infrastruktura za izradu aplikacija korišćenjem .NET tehnologije. Ovim je obezbeđen objektno orijentisan model koji može da se koristi za kreiranje Windows aplikacija ili Web servisa. Stvaranje aplikacije uslovljeno je kreiranjem klase i definisanjem funkcionalnosti aplikacije preko osobina, događaja i metoda klase. Klase su osnova za programiranje u Web okruženju.

Klase mogu da se formiraju u bilo kom jeziku podržanom od strane .NET Framework. Klasa pisana u jednom .NET programskom jeziku može da se iskoristi u drugom .NET programskom jeziku. Evropsko udruženje proizvođača računara ECMA (**E**uropean **C**omputer **M**anufacturers **A**ssociation) napravilo je standard CLS (**C**ommon **L**anguage **S**pecification) koji definiše pravila programskih jezika za njihovu međusobnu povezanost – interoperabilnost (Interoperabillity).

C# je nov programski jezik napravljen za izradu aplikacija koje se izvršavaju na .NET Framework. Jezik je objektno orijentisan, kompajlira se u okruženju koje podržava Garbage Collection i automatsku dealokaciju objekata čime se programer rastereću jednog ozbiljnog dela posla. C# podržava XML i SOAP, čime se nameće kao kandidat za jezik u kom će biti pisana aplikacija.

Web servis je autonomna, modularna aplikacija koja se može opisati, objaviti, locirati i pozvati preko mreže. Web servis predstavlja crnu kutiju koja omogućava da se interakcija sa njim ostvari preko

njegovog interfejsa, pri čemu implementacija funkcionalnosti ostaje sakrivena. Komunikacija sa Web servisima je ostvariva preko opšte prihvaćenih protokola, kao što su HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) i XML (E**x**tended **M**arkup **L**anguage).

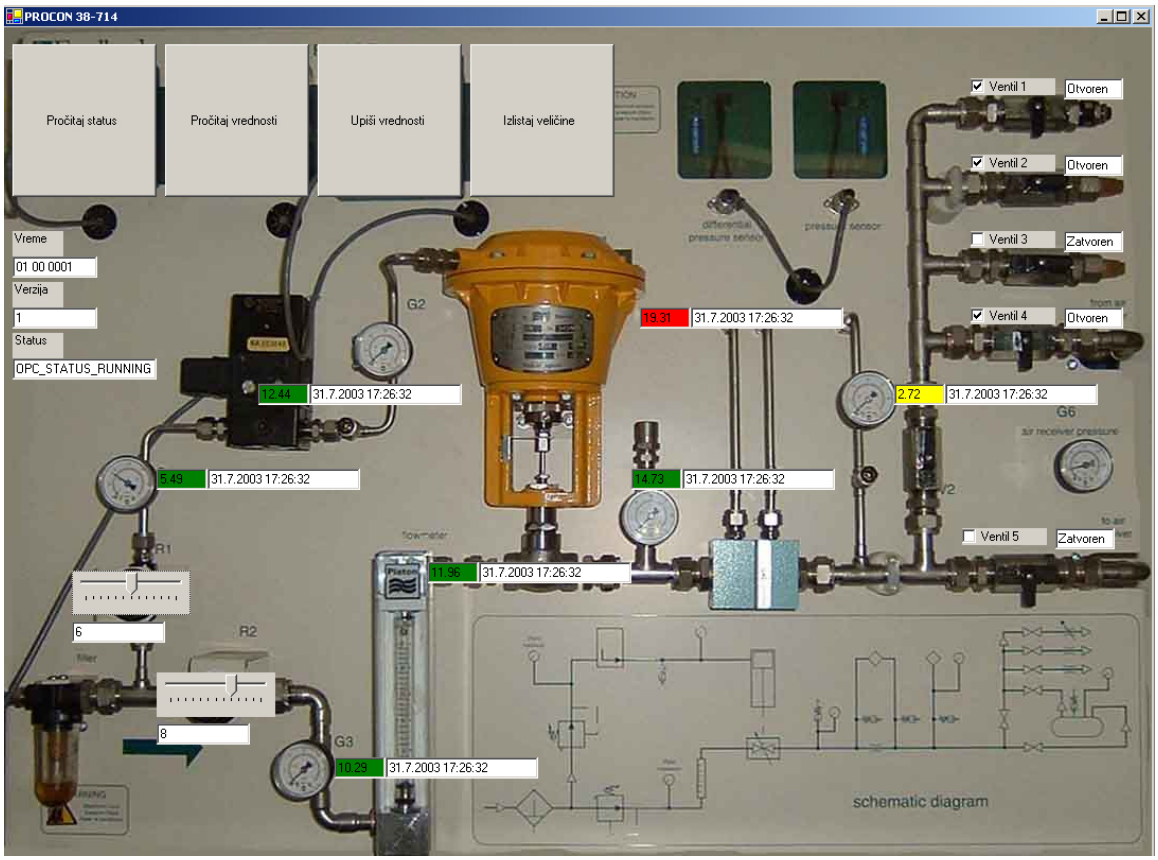
Kao standardizovan način prezentacije podataka prihvaćen je XML. Kao način za razmenu podataka prihvaćen je SOAP (**S**imple **O**bject **A**ccess **P**rotocol*)* koji omogućava dodatnu funkcionalnost svojim zaglavljem, načinom reprezentacije podataka i konvencijom za korišćenje udaljenih poziva. Kao opis servisa koristi se WSDL (**W**eb **S**ervice **D**escription **L**anguage) koji je protokol baziran na XML-u. Za određivanje servisa nekoj lokaciji iskorišćen je protokol DISCO koji omogućava definisanje dokumenata za pronalaženje servisa i prikupljanje tih dokumenata. U slučaju da je adresa željenog servisa nepoznata, koristi se UDDI (**U**niversal **D**escription, **D**iscovery and **I**ntegration) registar koji omogućuje prijavljivanje servisa i njihovo otkrivanje, opis i korišćenje.

XML je nastao sa ciljem da omogući brzo i lako kreiranje opisa podataka, njihov unos i strukturiranje pomoću jednostavnog editora teksta. Struktura XML dokumenta je vrlo pregledna i logična te se ovakvi dokumenti mogu veoma lako kreirati i održavati.

SOAP je protokol za razmenu informacija u decentralizovanom, distribuiranom okruženju. To je XML baziran protokol koji se sastoji od tri dela: koverte koja definiše skelet za opisivanje šta se nalazi u poruci i kako se to može obraditi, pravila kodiranja koja opisuju instance tipova podataka i konvenciju za reprezentovanje udaljenih poziva procedura i odgovora [8].

4. Primer OPC XML-DA klijentske aplikacije

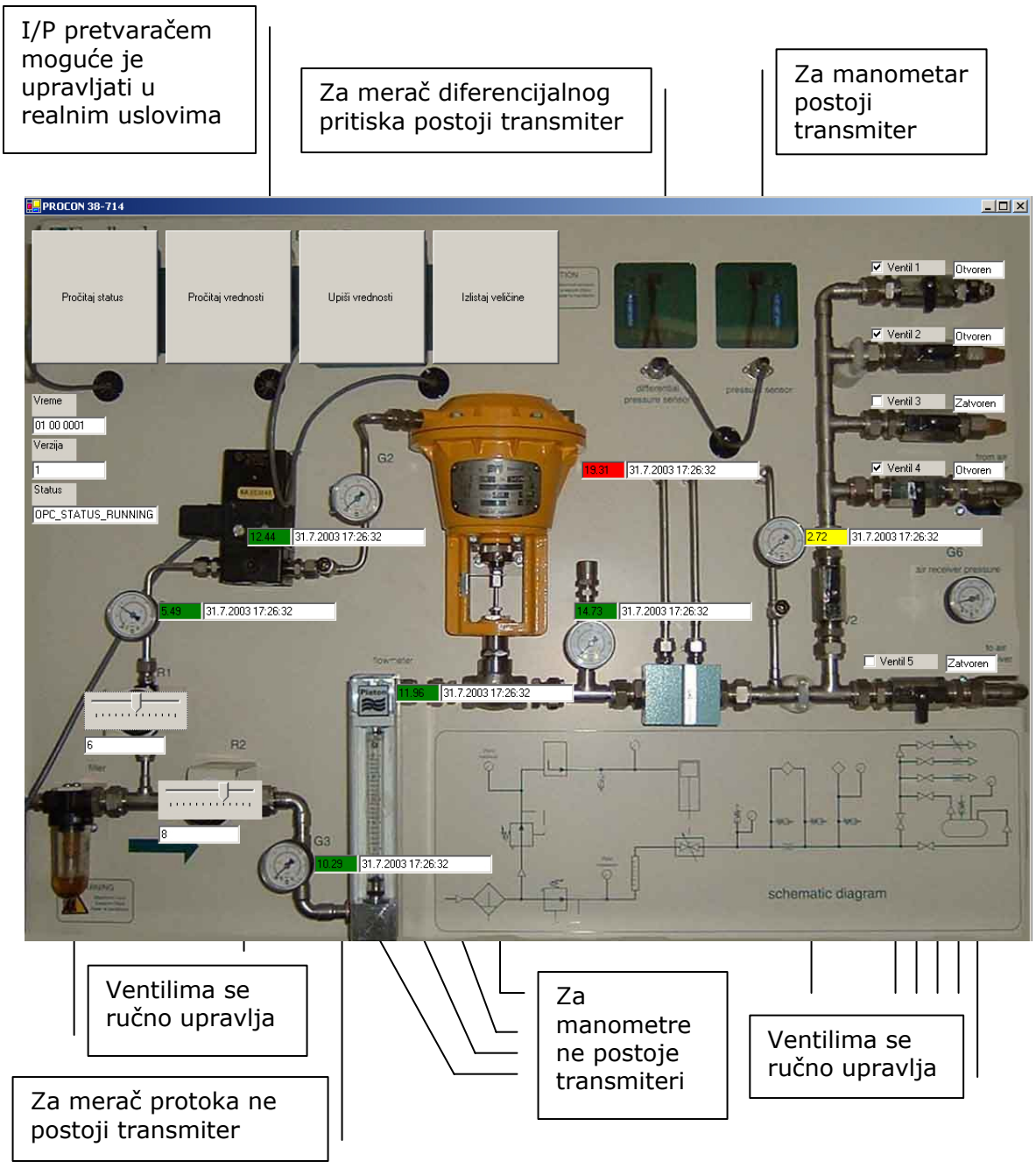
Za izradu aplikacije korišćen je programski jezik C#, programskog paketa Microsoft Visual Studio .NET. Ovaj primer predstavlja samo jedno od mogućih rešenja problema izrade klijentske aplikacije po OPC XML-DA specifikaciji. Pristup rešavanju problema pomoću programskog paketa .NET odabran je zbog podrške .NET za SOAP i XML, koji omogućuju komunkaciju putem Interneta sa serverom veličina.



Slika 4.1 Korisnički interfejs klijentske aplikacije

Na slici 4.1 prikazan je izgled aplikacije – interfejs ka klijentu. U gornjem levom uglu nalaze se tasteri sa natpisima "Pročitaj status", "Pročitaj vrednosti", "Upiši vrednosti" i "Izlistaj veličine", respektivno

sa leva na desno. Aplikacija vrši periodično očitavanje statusa servera i vrednosti i kvaliteta veličina. Klikom na tastere "Pročitaj status" i "Pročitaj vrednosti" klijentu je omogućeno da bez čekanja na istek perioda dobije informacije u vezi sa statusom servera i podatke o vrednosti i kvalitetu veličina. Pomoću tastera "Izlistaj veličine" klijent dobija listu svih veličina koje se nalaze u adresnom prostoru servera.



Slika 4.2 Realne i pretpostavljene veličine na tabli

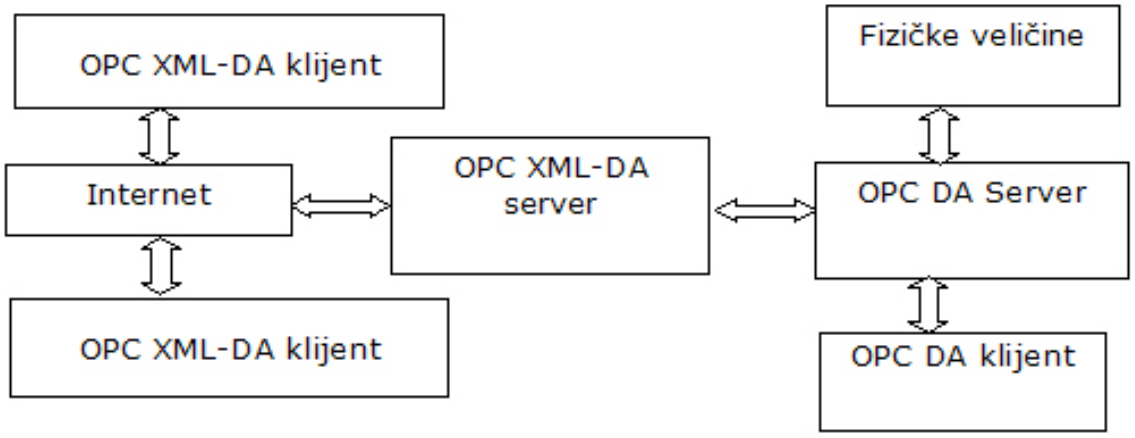
Aplikacija služi za nadzor i upravljanje veličinama sa table. Zbog što jasnijeg opisivanja nadzora i upravljanja veličinama preko Interneta uvedene su promene na tabli. Praktično, moguće je upravljati samo I/P pretvaračem, a podatke je moguće dobiti samo sa merača diferencijalnog pritiska i sa manometra pored merne blende, za koje postoje transmiteri (slika 4.2). Za potrebe ovog rada pretpostavljeno je da se klizačima i ventilima može zadati vrednost za upravljanje, kao i da vrednosti svih manometara mogu biti osmotrene.

Na slici 4.2 prikazane su veličine koje se mogu meriti i kojima se može upravljati u realnim uslovima i veličine za koje je to pretpostavljeno.

Funkcionisanje aplikacije najjednostavnije se može opisati sa samo dva stanja: stanje čekanja na zahtev za čitanjem ili na zahtev za podacima o statusu i stanje komunikacije aplikacije sa serverom.

Aplikacija prelazi iz stanja čekanja u stanje komunikacije u dva slučaja: kada istekne vremenski period nakon čega se pozivaju metode za čitanje vrednosti i kvaliteta veličina ili kada korisnik ne želi da čeka istek perioda, pa klikom na bilo koji taster prevede aplikaciju u stanje komunikacije sa serverom. Iz stanja komunikacije sa serverom, aplikacija prelazi ponovo u stanje čekanja.

5. Izrada OPC XML-DA klijentske aplikacije



Slika 5.1 Pozicija OPC XML-DA klijenta

Na slici 5.1 prikazana je arhitektura sistema u kome se nalazi OPC XML-DA klijent. OPC XML-DA Server je po strukturi web servis, tj. softverski sistem čiji su interfejsi javni i način povezivanja sa klijentom definisani i opisani pomoću XML-a.

Po specifikaciji XML-DA klijent bi trebao da omogući pristup i izmenu vrednosti fizičkih veličina u procesu. Na slici 5.1 prikazana je pozicija i uloga XML-DA klijenta. Pristup korisnika veličinama u procesu preko Interneta odvija se preko XML-DA servera, koji te podatke dobija od servera veličina (DA server). DA server podatke o veličinama dobija iz procesa (najčešće iz PLC-ova).

Za realizaciju klijentske aplikacije, a postupajući po specifikaciji OPC XML-DA, potrebno je implementirati sledeće funkcije: `GetStatus`, `Read`, `Write` i `Browse`.

Sve funkcije koriste isti objekat **`server`** koji je definisan kao instanca klase OPCXML3, na sledeći način:

```
net.neobee.limansoft.OPCXML3 server = new
    net.neobee.limansoft.OPCXML3();
```

U klijentskoj aplikaciji ne postoji klasa OPCXML3. Zbog toga, u klijentsku aplikaciju mora se dodati Web referenca sa koje se može preuzeti ta klasa. Naredbom **`new`** vrši se alokacija memorijskog prostora za objekat `server`. Na ovom mestu je zgodno pomenuti da

se zatvaranjem aplikacije vrši automatsko uklanjanje svih kreiranih objekata. Ovaj proces je poznat kao "Garbage collection" i predstavlja novinu u odnosu na prethodne tehnologije.

Pored objekta **server** koristi se i objekat `opcije` koji predstavlja instancu klase `RequestOptions`. Ova klasa se takođe preuzima na osnovu Web reference. Deklaracija objekta je data sledećim izrazom:

```
net.neobee.limansoft.RequestOptions opcije = new
    net.neobee.limansoft.RequestOptions();
```

Objekat **opcije** sadrži sledeće atribute:

ReturnErrorText
ReturnDiagnosticInfo
ReturnItemTime
ReturnItemName
ReturnItemPath
RequestTimeout
ClientRequestHandle
LocaleID

Kreiranjem ovog objekta atributi se postavljaju na neke unapred definisane vrednosti. Prilikom poziva funkcije sa ovim objektom kao ulaznim parametrom, vrednost atributa objekta ne definiše se posebno, već se koriste podrazumevane (*default*) vrednosti.

5.1 Metoda `GetStatus`

Metodom `GetStatus` klijentu je omogućen mehanizam za proveravanje statusa servera: da li je aktivan, da li postoji greška u komunikaciji, da li je server «pao», suspendovan ili nije konfigurisan. Takođe, ovom metodom je omogućen pristup specifičnim informacijama kao što su verzija i očitavanje vremena kada je server startovan (*UTC – **U**niversal **T**ime **C**onstant*).

Klijent šalje zahtev za informacijama o statusu objektu **`server`**. Na taj način tok kontrole prelazi sa objekta **`klijent`** na objekat **`server`**. Server otvara XML SOAP poruku i prosleđuje je ka serveru veličina. Nakon odgovora servera veličina, tok kontrole se vraća objektu **`klijent`**.

OPC XML-DA specifikacija propisuje sledeći interfejs metode `GetStatus`:

```
GetStatus(string LocaleID, string ClientRequestHandle,
out statusObjekta)
```

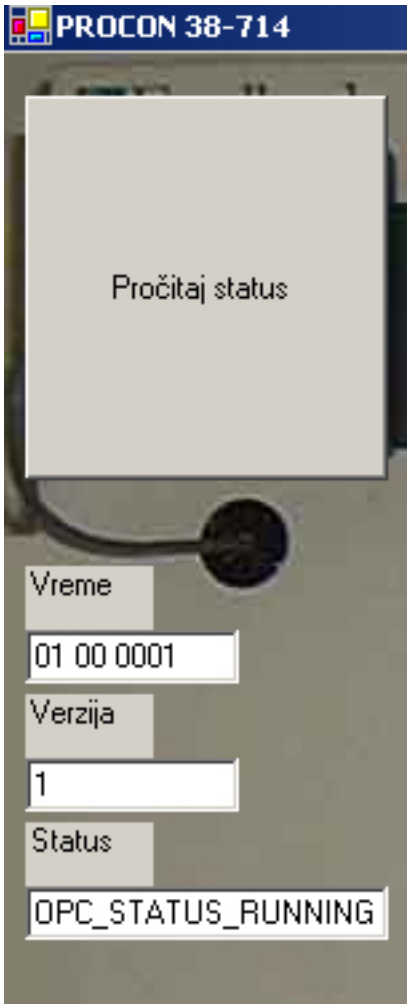
Povratni parametar **`statusObjekta`** jeste objekat koji ima atribute:

`LastUpdateTime`
`LastUpdateTimeSpecified`
`ProductVersion`
`StartTime`
`StartTimeSpecified`
`StatusInfo`
`SupportedLocaleIDs`
`SupportedInterfaceVersions`
`VendorInfo`

Kompletan kod funkcije nalazi se u u delu `kod GetStatus metode`, odeljka Prilog.

Metoda **`GetStatus`** poziva se periodično na svakih 20 sekundi. Red velličine ovog intervala ne bi mogao da bude manji jer zbog komunikacije klijenta i servera preko Interneta postoji kašnjenje. Ukoliko korisnik aplikacije ne želi da čeka na istek vremenskog perioda, on može sam pozvati metodu **`GetStatus`** klikom na dugme **Pročitaj status.**

Jedna od primena ove metode bio bi slučaj kada korisnik očitava uvek iste vrednosti veličina, a zbog prirode procesa ili primenjenog upravljanja očekuje promenljive vrednosti; u tom slučaju, pozivom ove metode korisnik može ustanoviti da je server pao, ili da je greška u komunikaciji, ili da je server u nekom drugom stanju. Na osnovu toga, korisnik zna gde bi mogao biti problem.



Slika 5.1.1 Interfejs aplikacije za metodu `GetStatus`

Na slici 5.1.1 prikazan je deo interfejsa klijentske aplikacije. Polje **Vreme** predstavlja vreme kada je server startovan. Ovaj podatak je konstanta za sve instance servera. **Polje** verzija sadrži podatak o verziji. U polju **Status** prikazan je status servera. Vrednosti ovog polja mogu biti:

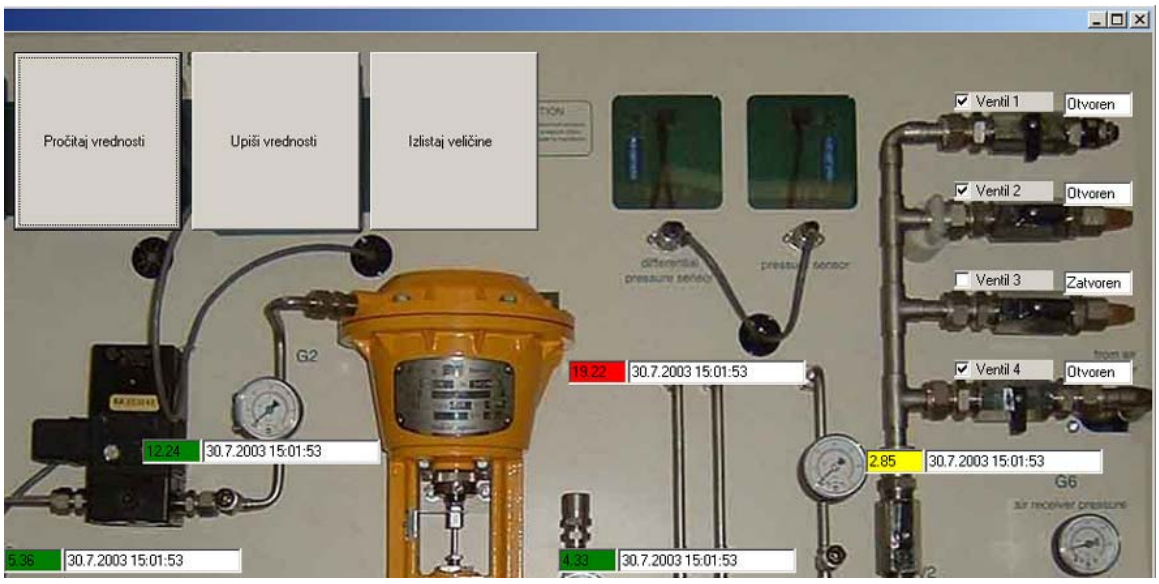
`OPC_STATUS_RUNNING` (slučaj prikazan na slici 5.1.1)
`OPC_STATUS_FAILED`

```
OPC_STATUS_NOCONFIG
OPC_STATUS_SUSPENDED
OPC_STATUS_TEST
OPC_STATUS_COMMFAULT
```

5.2 Metoda `Read`

Metodom **`Read`** omogućeno je čitanje vrednosti veličina. Pored toga, moguće je prikazati podatke o tipu veličine, kvalitetu veličine, o vremenu očitavanja, dijagnostičke informacije, identifikator greške u slučaju pojavljivanja greške, identifikator uspešnog očitavanja vrednosti, bite ograničenja i vendorske bite. U klijentskoj aplikaciji prikazane su vrednost i kvalitet veličine i vremenska značka, odnosno, vreme kada je pristupljeno veličini radi čitanja.

Metoda **`Read`** poziva se, uslovno rečeno, automatski, to jest po isteku vremenskog intervala od 20 sekundi. Izbor ovolikog vremenskog intervala obrazložen je u prethodnom delu ove glave. Ukoliko ne želi da čeka da prođe vremenski interval, korisnik aplikacije može sam pozvati ovu metodu, klikom na taster **Pročitaj vrednosti**. Na slici 5.2.1 prikazan je interfejs aplikacije prilikom poziva metode **`Read`**.



Slika 5.2.1 Interfejs aplikacije prilikom poziva metode **`Read`**

Pored kućica u kojima je prikazana vrednost veličina nalazi se vreme očitavanja veličine. Kvalitet je predstavljen zelenom, crvenom i žutom bojom, u zavisnosti da li je **good**, **bad**, ili **uncertain**, respektivno.

U ovom radu korišćeno je sedam veličina i nije problem da ih po isteku vremenskog intervala pročitamo. Međutim, u slučaju velikog broja veličina (reda veličine 1000 i više) ovakav postupak bio bi neprihvatljiv. Efektivnije bi bilo pročitati samo veličine koje su se promenile od poslednjeg čitanja. Ovakav pristup se naziva ***Subscription***. OPC XML-DA specifikacija propisuje dva načina pretplate: *Basic* i *Advanced* [3]. U *Basic* načinu pretplate klijent započinje pretplatu slanjem zahteva. Server periodično obaveštava klijenta o promenama koje su se desile od poslednjeg prozivanja. Pod promenama se podrazumevaju promene u vrednosti ili promene u kvalitetu veličine. Kada više nije zainteresovan, klijent prekida pretplatu. Server nakon toga oslobađa memoriju zauzetu za klijenta i eventualno obavlja još neke aktivnosti oko raskidanja pretplate. Specifikacija nudi napredniji način pretplate pomoću kojeg je moguće dobiti sofisticiranije rešenje, ali to izlazi iz okvira ovog rada.

OPC XML-DA specifikacija propisuje sledeći interfejs metode **Read**:

```
server.Read(opcije,nizItema, out nizVrednosti,out
                    nizGresaka);
```

Parametar **opcije** je opisan na početku ovog poglavlja. Parametar **NizItema** je niz koji se sastoji od objekata klase **ReadRequestItemList**. Definisan je na sledeći način:

```
net.neobee.limansoft.ReadRequestItemList[] nizItema = new
     net.neobee.limansoft.ReadRequestItemList[1];
```

U ovom slučaju niz se sastoji od samo jednog elementa.
Kreiran je i jedan objekat klase **ReadRequestItemList**:

```
net.neobee.limansoft.ReadRequestItemList
        jedanObjekatItema = new
net.neobee.limansoft.ReadRequestItemList();
```

Ovaj objekat sadrži atribute: **Items**, **ItemPath**, **MaxAge** i **ReqType**.

Za ovu aplikaciju potreban je samo atribut **Items**. Ovaj atribut je niz **potrebniItemi** od osam elemenata tipa **ReadRequestItem**:

```
net.neobee.limansoft.ReadRequestItem[] potrebniItemi = new
         net.neobee.limansoft.ReadRequestItem[8];
```

Atributi elemenata niza koji se koriste u ovoj aplikaciji su **ItemName** i **ItemPath**. Priprema parametra **nizItema** opisana je sledećim delom koda:

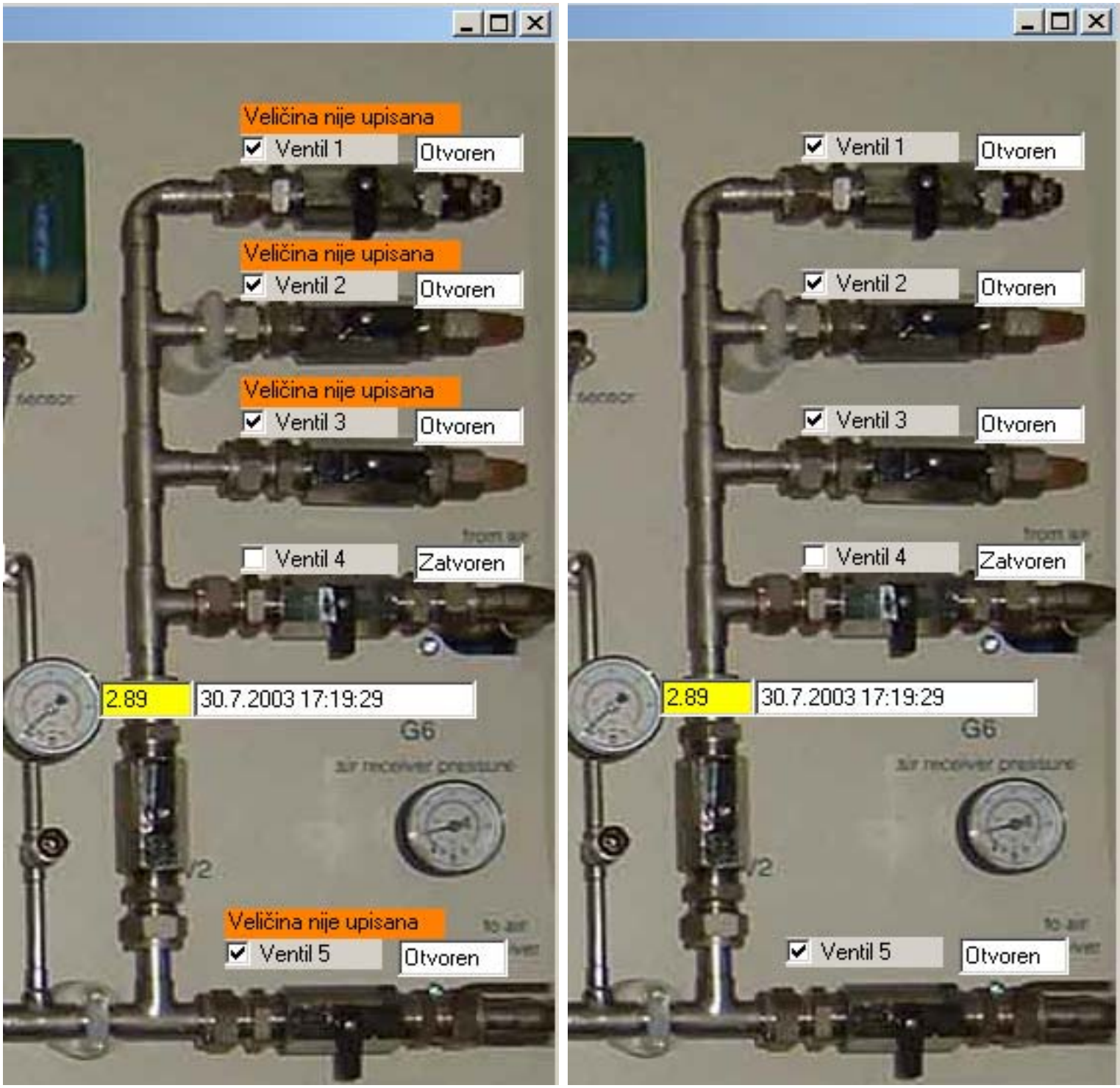
```
for(int i=0;i<7;i++)
{
     potrebniItemi[i] = new
     net.neobee.limansoft.ReadRequestItem();
     potrebniItemi[i].ItemName="dms.sw.S"+(i+1).ToString();
     potrebniItemi[i].ItemPath="dms.sw.S"+(i+1).ToString();
}

jedanObjekatItema.Items=potrebniItemi;
nizItema[0]=jedanObjekatItema;
```

Povratni parametar **nizVrednosti** je niz koji sadrži imena, vrednosti, putanje i kvalitet izmerenih veličina.

Izvorni kod ove funkcije nalazi se u delu kod Read metode u odeljku Prilog.

5.3 Metoda `Write`

Metodom **`Write`** omogućeno je upisivanje vrednosti željenih veličina.



Slika 5.3.1 Izgled klijentske aplikacije pre i posle upisivanja veličina

Na slici 5.3.1 prikazan je interfejs aplikacije prilikom upisivanja veličina. Vrednost veličine koja nije upisana, to jest nije prosleđena serveru, ima labelu narandžaste boje iznad kućice za tekst. Klikom na dugme **Upiši vrednosti,** labela nestaje.

Poziv Write metode ima sledeći oblik:

```
server.Write (opcije, nizItema, vratiVrednosti, out nizVrednosti, out nizGresaka);
```

Objekat **"server"** i parametar "opcije" su opisani na početku ove glave**.** Ulazni parametar "nizItema" ima istu ulogu i osobine kao kod metode **Read**, pa u ovom delu neće biti ponovo razmatran.

Veličina "vratiVrednosti" je tipa **boolean**, a određuje da li će povratni parametar "nizVrednosti" sadržati vrednosti veličina. Povratni parametar "nizGresaka" predstavlja listu grešaka kojom server obaveštava klijenta do kakvih je grešaka došlo prilikom upisa ili čitanja veličina.

Metoda **Write** može se realizovati i na drugi način. Umesto da se markiraju vrednosti veličina koje nisu poslate serveru, moguće je odmah po korisnikovom zadavanju vrednosti veličine slati serveru tu vrednost. Međutim, na ovaj način veza sa serverom se opterećuje, tako da se odustalo od ovakvog pristupa.

5.4 Metoda `Browse`

Pomoću **`Browse`** metode klijent može da pregleda adresni prostor servera veličina (Data Access server). U klijentskoj aplikaciji ova metoda se poziva klikom na taster **Izlistaj veličine.** U posebnoj listi prikazane su sve veličine koje postoje na serveru. Pritiskom na taster **Ukloni listu veličina**, uklanja se lista veličina sa aplikacije. Ponovnim klikom na taster, aplikacija ponovo pristupa serveru i na osnovu njegovog odgovora izlistava sve veličine kojima server raspolaže. Izgled dela aplikacije je prikazan na slici 5.4.1.

Slika 5.4.1 Deo interfejsa aplikacije za upotrebu metode **`Browse`**

Metoda **`Browse`** ima izuzetan značaj za primenu aplikacije. Ukoliko se neka veličina ukloni sa servera, aplikacija će prijaviti grešku prilikom pokušaja pisanja ili čitanja te veličine. U tom slučaju, pozivom metode korisnik može ustanoviti koje veličine postoje, a koje su izbačene.

Poziv metode ima sledeći oblik:

```
server.Browse(nizPropertija, "YU", "", "", ref cp, "",
                System.DateTime.Now, 0,
          net.neobee.limansoft.tristate.either,
net.neobee.limansoft.tristate.either, "", "", false, false,
    out nizElemenata, out nizGresaka, out josElemenata);
```

Parametar **`nizPropertija`** je niz objekata tipa System.Xml.XmlQualifiedName. Parametar "YU" je identifikator lokala. Sledeća dva parametra predstavljaju putanju i naziv početne veličine, respektivno.

U slučaju da je u pitanju drugi poziv metode, moguće je da je u odgovoru na prvi poziv vraćena tačka nastavka (Continuation point) koja predstavlja poziciju odakle treba nastaviti sa čitanjem veličina. U ovoj aplikaciji ne koristi se ova mogućnost. Sledeći parametar koristi se kod komleksnijih sistema, a olakšava klijentu prepoznavanje odgovora na svoj zahtev.

Parametar **`System.DateTime.Now`** je sistemsko vreme poziva metode. Parametar **`0`** govori serveru da ne treba da vraća nijednu veličinu. Sledeća dva parametra govore serveru da veličine mogu, a ne moraju biti veličine, odnosno da li mogu biti čvorovi u stablu. Naredna dva parametra, prazni stringovi, nisu bitni za ovaj rad, a koriste se za sugerisanje serveru o operacijama filtriranja.

Poslednjim ulaznim parametrima serveru se zadaje da ne vraća ni obeležja ni vrednosti obeležja.

Izlazni parametar **`nizElemenata`** predstavlja niz koji sadrži sve veličine koje postoje na serveru. Ukoliko dođe do grešaka prilikom komunikacije, podaci o tome su sadržani u nizu **`nizGresaka`**. Ako server ne uspe da vrati sve veličine u povratnom nizu, obaveštava klijenta o tome preko logičke promenljive **`josElemenata`**.

6. Zaključak

U ovom radu prikazan je jedan način na koji je moguće vršiti nadzor i upravljanje, koristeći XML Web servise i servere definisane po OPC specifikaciji. Pogodnost ovog načina upravljanja je mogućnost njegove primene preko Interneta. Najveću primenu ovaj način upravljanja nalazi za sisteme sa velikim vremenom odabiranja (reda veličine sekunde). Za primenu ovakvog načina upravljanja za sisteme u realnom vremenu, kod kojih su vremena odziva nekoliko redova veličine niža, potrebno je da komunikaciona mreža ima mala kašnjenja.

Ovaj rad zasnovan je na već napravljenim DA i XML-DA serverima po specifikacijama OPC fondacije. Dalji razvoj aplikacije mogao bi da se odvija u sledećim pravcima: razvoj korisničkog interfejsa, grafičkim prikazivanjem merenih veličina; razvoj analize odziva kroz istoriju pomoću smeštanja izmerenih veličina u bazu podataka; za izvođenje "daljinskih" laboratorijskih vežbi na osnovu kojih bi se mogli utvrditi odzivi sistema za primene različitih algoritama. Sledeći korak u unapređenju ovog problema mogla bi biti provera zadatih vrednosti na klijentskoj strani.

Tokom izrade rada proširena su znanja iz oblasti objektno orijentisanog i Internet programiranja, SCADA, prihvaćena je nova tehnologija .NET i savladan programski jezik C#.

7. Literatura

1. Božidar Batančev, Vizuelizacija rada termo-procesnog sistema u programskom jeziku Delphi, Biblioteka Fakulteta tehničkih nauka, Novi Sad, 1999.

2. Slobodan Boljanović, Softver za daljinski nadzor upravljačkog uređaja, Biblioteka Fakulteta tehničkih nauka, Novi Sad, 2000.

3. Branka Šuka, Nadzor rada u kotlarnici realizovan upotrebom client/server apliakcije u prograsmskom okruženju Delphi, Biblioteka Fakulteta tehničkih nauka, Novi Sad, 2001.

4. Feedback instruments Ltd, PROCON - PRESSURE CONTROL TRAINER INSTRUCTION MANUAL 38-714, Crowborrough, Eastern Sussex, UK, 1997.

5. OPC Foundation**,** OPC XML-DA Specification Release candidate version 1.8, June 2002.

6. Srđan Vukmirović, Aleksandar Erdeljan, Jedno rešenje arhitekture OPC XML-DA servera, Etran konferencija, Budva 2003.

7. Jesse Liberty, Naučite C++ za 21 dan, Čačak: Kompjuter biblioteka, 1998.

8. Dragan Vasić, Arhitektura i bezbednost informacionog sistema javne uprave, Biblioteka Fakulteta tehničkih nauka, 2002.

8. Biografija

Marko Radulović rođen je 12.11. 1978. godine u Novom Sadu, gde je završio osnovnu školu "Svetozar Marković Toza" i srednju elektrotehničku školu "Mihajlo Pupin". Školske 1997/98. godine upisao je Fakultet tehničkih nauka u Novom Sadu, odsek Elektrotehnika, smer Računarstvo i upravljanje sistemima, usmerenje Upravljanje sistemima.

9. Prilog

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace Proba4
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	///


	public class Form1 : System.Windows.Forms.Form
	{
	#region Windows forme
.  .  .
	#endregion//Widows forme

	#region Windows Form Designer generated code

.  .  .
	#endregion

	#region Klijentski kod
		net.neobee.limansoft.OPCXML3 server = new
net.neobee.limansoft.OPCXML3();
		net.neobee.limansoft.RequestOptions opcije = new
net.neobee.limansoft.RequestOptions();
		bool ventil1=false, ventil2=false, ventil3=false, ventil4=false,
ventil5=false;
		int klizac1=0, klizac2=0;



		[STAThread]
		static void Main()
		{
			Application.Run(new Form1());
		}
```

9.1 GetStatus

```
		//
		//Opis dogadjaja za dobijanje statusa
		//
	#region Funkcija "Podaci o statusu"
		private void button1_Click(object sender, System.EventArgs e)
		{
			net.neobee.limansoft.ServerStatus statusObjekta = new
net.neobee.limansoft.ServerStatus();
			server.GetStatus("","", out statusObjekta);
			textBox1.Text = statusObjekta.ProductVersion;
			textBox2.Text = statusObjekta.StatusInfo;
			textBox3.Text = statusObjekta.LastUpdateTime.ToString("dd
mm yyyy", null);
		}
	#endregion
```

9.2 Write

```
            //
            //Opis dogadjaja za upisivanje vrednosti
            //
        #region Funkcija "Upisi vrednosti"
            private void button2_Click(object sender, System.EventArgs e)
            {

                    net.neobee.limansoft.OPCXML3 server = new
net.neobee.limansoft.OPCXML3();
                    net.neobee.limansoft.RequestOptions opcije = new
net.neobee.limansoft.RequestOptions();

                    net.neobee.limansoft.WriteRequestItemList[] nizItema = new
net.neobee.limansoft.WriteRequestItemList[1];
                    net.neobee.limansoft.WriteRequestItemList jedanObjekatItema
= new net.neobee.limansoft.WriteRequestItemList();
                    net.neobee.limansoft.ItemValue[] potrebniItemi = new
net.neobee.limansoft.ItemValue[7];

                    net.neobee.limansoft.ReplyItemList[] nizVrednosti = new
net.neobee.limansoft.ReplyItemList[1];
                    net.neobee.limansoft.OPCError[] nizGresaka = new
net.neobee.limansoft.OPCError[2];
                    bool vratiVrednosti=false;

                    for (int i=0;i<7;i++)
                    {
                           potrebniItemi[i] = new
net.neobee.limansoft.ItemValue();
                           potrebniItemi[i].ItemName = "dms.sw.S" +
(i+1).ToString();
                           potrebniItemi[i].ItemPath = "dms.sw.S" +
(i+1).ToString();
                    }

                    //
                    //Ovde se upisuju vrednosti u niz "potrebniItemi", koji
predstavlja jedno od obelezja
                    //objekta "jedaObjekatItema", a koji je opet jedini clan
niza "nizItema" koji se
                    //prosledjuje kao parametar
                    //u pozivu funkcije "Write" objekta "server".
                    //
                    potrebniItemi[0].Value = klizac1;
                    //
                    if (ventil1) potrebniItemi[1].Value = 1;
                    else potrebniItemi[1].Value = 2;
                    //
                    if (ventil2) potrebniItemi[2].Value = 3;
                    else potrebniItemi[2].Value = 4;
                    //
                    if (ventil3) potrebniItemi[3].Value = 5;
                    else potrebniItemi[3].Value = 6;
                    //
                    if (ventil4) potrebniItemi[4].Value = 7;
                    else potrebniItemi[4].Value = 8;
                    //
                    if (ventil5) potrebniItemi[5].Value = 9;
                    else potrebniItemi[5].Value = 10;
                    //
                    potrebniItemi[6].Value=klizac2;
                    //Niz "potrebniItemi" je sada napunjen.
```

```
                    jedanObjekatItema.Items = potrebniItemi;
                    jedanObjekatItema.ItemPath="YU";
                    nizItema[0]=jedanObjekatItema;


                    server.Write(opcije,nizItema,vratiVrednosti,out
nizVrednosti,out nizGresaka);
                }
            #endregion//Od funkcije "Upisi vrednosti".
```

9.3 Read

```
                //
                //Opis dogadjaja za citanje vrednosti
                //
            #region Funkcija "Procitaj vrednosti"
                private void button3_Click(object sender, System.EventArgs e)
                {
                    net.neobee.limansoft.ReadRequestItemList[] nizItema = new
net.neobee.limansoft.ReadRequestItemList[1];
                    net.neobee.limansoft.ReadRequestItemList jedanObjekatItema
= new net.neobee.limansoft.ReadRequestItemList();
                    net.neobee.limansoft.ReadRequestItem[] potrebniItemi = new
net.neobee.limansoft.ReadRequestItem[8];

                    net.neobee.limansoft.ReplyItemList[] nizVrednosti = new
net.neobee.limansoft.ReplyItemList[1];
                    net.neobee.limansoft.OPCError[] nizGresaka = new
net.neobee.limansoft.OPCError[2];

                    for(int i=0;i<7;i++)
                    {
                        potrebniItemi[i] = new
net.neobee.limansoft.ReadRequestItem();
                        potrebniItemi[i].ItemName="dms.sw.S"+
(i+1).ToString();
                        potrebniItemi[i].ItemPath="dms.sw.S"+
(i+1).ToString();
                    }

                    jedanObjekatItema.Items=potrebniItemi;
                    nizItema[0]=jedanObjekatItema;

                    server.Read(opcije, nizItema, out nizVrednosti, out
nizGresaka);

                    textBox4.Text = nizVrednosti[0].Items[0].Value.ToString();
                    progressBar6.Value =
Convert.ToInt32(nizVrednosti[0].Items[0].Value.ToString());

                    textBox6.Text = nizVrednosti[0].Items[1].Value.ToString();
                    progressBar3.Value =
Convert.ToInt32(nizVrednosti[0].Items[1].Value.ToString());

                    textBox7.Text = nizVrednosti[0].Items[2].Value.ToString();
                    progressBar5.Value =
Convert.ToInt32(nizVrednosti[0].Items[2].Value.ToString());

                    textBox8.Text = nizVrednosti[0].Items[3].Value.ToString();
                    progressBar4.Value =
Convert.ToInt32(nizVrednosti[0].Items[3].Value.ToString());

                    textBox9.Text = nizVrednosti[0].Items[4].Value.ToString();
                    progressBar7.Value =
Convert.ToInt32(nizVrednosti[0].Items[4].Value.ToString());
```

```
                    textBox5.Text=nizVrednosti[0].Items[5].Value.ToString();
                    progressBar2.Value =
Convert.ToInt32(nizVrednosti[0].Items[5].Value.ToString());

                    textBox16.Text=nizVrednosti[0].Items[6].Value.ToString();
                    progressBar1.Value =
Convert.ToInt32(nizVrednosti[0].Items[6].Value.ToString());

                }
            #endregion

                //
                //Opis za pojedinacne dogadjaje
        .   .   .
                //
            #region Pojedinacni dogadjaji
        .   .   .
            #endregion //Pojedinacni dogadjaji
        #endregion //Klijentski kod
        }
}
```