UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
Institut za računarstvo, automatiku i merenja
Trg Dositeja Obradovića 6

REALIZACIJA HTTP SPREGE SA GAUS SCADA SISTEMOM

MAGISTARSKI RAD

Mentor:
dr. Branislav Atlagić

Kandidat:
Tomislav Maruna, dipl.ing

Novi Sad, 2006.

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

REDNI BROJ: RBR	
IDENTIFIKACIONI BROJ: IBR	
TIP DOKUMENTACIJE TD	
TIP ZAPISA: TZ	Tekstualni štampani materijal
VRSTA RADA: VR	Magistarski rad
AUTOR: AU	Tomislav Maruna
MENTOR / KOMENTAR: MN	dr Branislav Atlagić
NASLOV RADA: NS	Realizcija HTTP sprege sa GAUS SCADA sistemom
JEZIK PUBLIKACIJE: JZ	Srpski / Latinica
JEZIK IZVODA: JI	Srpski
ZEMLJA PUBLIKOVANJA: ZP	Srbija i Crna Gora
UŽE GEOGRAFSKO PODRUČJE: UGP	Vojvodina
GODINA: GO	2006.
IZDAVAČ: IZ	Autorski reprint
MESTO I ADRESA: MS	21000 Novi Sad, Trg Dositeja Obradovića 6
FIZIČKI OPIS RADA: FO	6 poglavlja / ? strana / ? slika
NAUČNA OBLAST: NO	Elektrotehnika
NAUČNA DISCIPLINA: DI	Računarska tehnika
PREDMET/ODREDNICA/KLJUČNE REČI PO	Nadzor i upravljanje fizičkim procesima, SCADA, Internet
UDK:	
ČUVA SE: ČU	U biblioteci Fakulteta tehničkih Nauka, Novi Sad
VAŽNA NAPOMENA: VN	nema

IZVOD:
IZ

Cilj rada je da ponudi jedno rešenje otvaranja intranet/internet sprege http protokolom ka SCADA informacionom sistemu uzimajući u razmatranje realizaciju sprege web servera ka podacima informacionog sistema. Posebno će biti diskutovane različite tehnologije za realizaciju te veze. Kao poseban aspekt će biti razmatrana i sigurnost tako realizovanih informacionih sistema.

U radu će biti opisano i konkretna realizacija proširenja GAUS SCADA sistema opšte namene radi otvaranja http sprege ka intranet/internet korisnicima. Realizacija će obuhvatiti više vrsta veza web servera i SCADA informacionog sistema.

U prvom delu data je definicija i razmatranje ključnih osobina akviziciono - upravljačkih sistema.

U nastavku su date osobine SCADA sistema u intranet / internet okruženju kao i mogućnosti njihovog sprezanja u opštem slučaju.

Na kraju data su konkretna rešenja realizacije http sprege GAUS SCADA sistema primenom CGI programa

DATUM PRIHVATANJA TEME:
DP
DATUM ODBRANE:
DO
ČLANOVI KOMISIJE
(NAUČNI STEPEN / IME I PREZIME /
ZVANJE / FAKULTET):
KO

Predsednik:	dr. Vladimir Kovačević, red.prof. FTN Novi Sad
Član:	dr. Miroslav Popović, red.prof. FTN Novi Sad
Član:	dr Branislav Atlagić, docent FTN Novi Sad
Član:	dr Borivoj Lazić, red.prof. ETF Beograd
Član:	dr Dragan Kukolj, red.prof. FTN Novi Sad

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES

KEY WORDS DOCUMENTATION

ACCESSION NUMBER:	
ANO	
IDENTIFICATION NUMBER:	
INO	
DOCUMENT TYPE:	
DT	
TYPE OF RECORD:	
TR	Printed text material
CONTEST CODE:	
CC	
AUTHOR:	
AU	Tomislav Maruna
MENTOR /CO-MENTOR:	
MN	Ph.Sc. Branislav Atlagić
TITLE:	
TI	A realization of HTTP interface with GAUS SCADA system
LANGUAGE OF TEXT:	
LT	Serbian
LANGUAGE OF ABSTRACT:	
LS	Serbian / English
COUNTRY OF PUBLICATION:	
CP	Serbia and Montenegro
LOCALITY OF PUBLICATION:	
LP	Vojvodina
PUBLICATION YEAR:	
PY	2006.
PUBLISHER:	
PB	Author reprint
PUBLICATION PLACE:	
PL	21000 Novi Sad, Trg Dositeja Obradovića 6
PHYSICAL DESCRIPTION:	
PD	6 chapters / ? pages / ? figures
SCIENTIFIC FIELD:	
SF	Electrical engineering
SIENTIFIC DISCIPLINE:	
SD	Computer Science
SUBJECT / KEY WORDS:	
SKW	SCADA, data acquisition, physical processes supervision and control, Internet
UC.	
HOLDING DATA:	
HD	The Library of Faculty of Technical Sciences, 21000 Novi Sad, YU.
NOTE:	
N	none

ABSTACT:
AB

The aim of this thesis is realization of intranet / internet connection via http protocol with SCADA information system through realization of web server connection with SCADA values database. Different technologies for connection realization will be discussed. Security overview for this kind of SCADA systems will be also given.

Thesis also contains concrete realization of http interface connection for intranet / internet users with GAUS SCADA system. Realization will discuss several connection types between web server and SCADA information system.

The second chapter of the thesis contains a definition of the SCADA system and it's the most important characteristics.

Following chapters contains characteristics of SCADA systems in intranet / internet environment and possibility for connections in general case.

Final chapters contains concrete solution of http interface connection using CGI script.

ACCEPTED BY SCIENTIFIC
BOARD ON:
AS
DEFENDED ON:
DE
THESIS DEFEND BOARD
(DEGREE / NAME & SURNAME /
TITLE / FACULTY):
DB

President: Ph.Sc. Vladimir Kovačević, professor, FTN Novi Sad
Member: Ph.Sc. Miroslav Popović, professor, FTN Novi Sad
Member: Ph.Sc. Branislav Atlagić, docent, FTN Novi Sad
Member: Ph.Sc. Borivoj Lazić, professor, ETF Beograd
Member: Ph.Sc. Dragan Kukolj, professor, FTN Novi Sad

Najiskrenije se zahvaljujem prof. dr. Vladimiru Kovačeviću i dr. Branislavu Atlagiću na iskrenoj i upornoj podršci u toku realizacije rada. Takođe se zahvaljujem i najbližim saradnicima sa Katedre. Ipak najveću zahvalnost dugujem svojoj porodici.

Zoranu

Sadržaj:

1. UVOD

Problem kvalitetnog nadzora i upravljanja tehnološkim procesima svodi se na prikupljanje i prikaz podataka (merenja i proračuna) o nadziranom sistemu u realnom vremenu. Kao poseban problem izdvaja se mogućnost povezivanja korisnika na takav sistem (SCADA – *Supervisory Control And Data Acquisition*) i preuzimanje podataka u realnom vremenu. Sa druge strane trend razvoja računarskih mreža (lokalnih *intranet* i globalnih *internet*) učinio je razmenu podataka između računara jednostavnijom nego ikad. Jedan od najpopularnijih pristupa globalnoj mreži je putem HTTP protokola (WEB). U okviru rada razmatraće se mogućnost povezivanja na SCADA sistem putem HTTP protokola u realnom vremenu.

Oblast istraživanja obuhvata povezivanje informacionih sistema sa distribuiranim akviziciono-upravljačkim sistemima sa radom u realnom vremenu, i kao poseban problem, prezentaciju podataka iz takvog jedinstvenog informacionog sistema. Zahtevi koji se stavljaju pred SCADA sisteme su različiti, a nekad i suprotni, zahtevima koji se stavljaju pred informacione sisteme.

Osnovni problem koji se u radu razmatra je realizacija sprege između komercijalnog web servera i SCADA informacionog sistema radi preuzimanja procesnih veličina. Posmatrana računarska platforma je svakako PC platforma i Windows okruženje (operativni sistem). Mogućnost realizacije SCADA sistema odnosno akvizicije u intranet / internet okruženju nije razmatran.

Cilj rada je da ponudi jedno rešenje otvaranja intranet/internet sprege http protokolom ka SCADA informacionom sistemu, uzimajući u razmatranje realizaciju sprege web servera ka podacima informacionog sistema. Posebno su diskutovane različite tehnologije za realizaciju te veze, kao i sigurnost tako realizovanih informacionih sistema.

U radu je opisano jedno rešenje proširenja SCADA sistema opšte namene radi otvaranja http sprege ka intranet/internet korisnicima. Realizacija obuhvata više vrsta veza web servera i SCADA informacionog sistema. Na kraju, data je uporedna analiza sa drugim relevantnim komercijalnim paketima.

Rad je organizovan u šest poglavlja.

U sledećem, drugom poglavlju izneta je definicija i razmatranje ključnih osobina akviziciono - upravljačkih sistema.

U trećem poglavlju su date osobine SCADA sistema u intranet / internet okruženju kao i mogućnosti njihovog sprezanja u opštem slučaju. Pojedinačno su izloženi primeri povezivanja od datotečnog (file) servera do server / klijent veze sa transakcionim i aplikacionim serverom. Problem sigurnosti tako realizovanih veza je takođe izložen.

Četvrto poglavlje sadrži konkretna rešenja realizacije http sprege GAUS SCADA sistema. Izložena rešenja predstavljaju realizaciju raspoloživih veza GAUS SCADA sistema sa rezultatima prezentovanim u alfanumeričkom i grafičkom obliku.

Zaključak, rezultati i rezime istraživanja dati su u petom poglavlju.

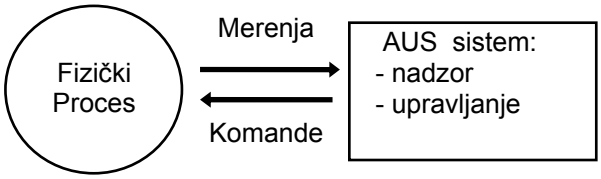
Šesto poglavlje sadrži listu korišćene literature.

2. AKVIZICIONO UPRAVLJAČKI SISTEMI - SCADA SISTEMI

2.1 Definicija i funkcionalni zahtevi SCADA sistema

Pod akviziciono - upravljačkim sistemom (AUS) podrazumeva se skup namenskih, prostorno distribuiranih, međusobno povezanih računarskih modula, čiji je zajednički cilj ostvarenje funkcija nadzora i/ili upravljanja fizičkim procesom u realnom vremenu[1][2].

Osnovna funkcija AUS sistema je ciklična akvizicija digitalizovanih odmeraka različitih fizičkih veličina koje određuju stanje proizvoljnog tehnološkog (fizičkog) procesa. Formiranjem baze merenih podataka u računaru, stvara se osnova za proveru i prikaz trenutnog stanja fizičkog procesa, odnosno za obavljanje efikasnog nadzora nad njim. Istovremeno, izvršenjem aplikativne upravljačke podrške moguće je odrediti i zahtevati korektivne (kontrolne) aktivnosti, tj. izvršiti upravljanje nad fizičkim sistemom. Slika 2.1 prikazuje vezu AUS sistema sa fizičkim procesom, kao i tok mernih i upravljačkih signala.



Slika 2.1. Povezivanje akviziciono - upravljačkog sistema i fizičkog procesa

Očigledo je da proizvodni pogoni upravljani pomoću ovako definisanog AUS, predstavljaju tipičan primer računarski utemeljenih sistema, te da projektovanje i eksploatacija AUS podrazumeva primenu ECBS metoda. Specifična namena AUS sistema nameće određene zahteve koji su temelj na kojem projektant gradi fizičku i programsku arhitekturu samog akviziciono - upravljačkog sistema. Njihovim poštovanjem u svakoj fazi razvoja, obezbeđuje se funkcionalnost i fleksibilnost ukupnog proizvodnog sistema. Fundamentalni zahtevi koje AUS sistem opšte namene mora ispunititi su:

- rad u realnom vremenu
- distribucija računarskih resursa u okviru AUS-a
- postizanje maksimalne pouzdanosti i raspoloživosti.

Rad u realnom vremenu proističe iz zahteva da AUS obezbedi adekvatno reagovanje na poremećaje u fizičkom procesu kojim se upravlja. U određenim slučajevima, vreme reakcije je vrlo malo i zahteva posebne elemente fizičke arhitekture u okviru računarskih modula AUS sistema, kao i primenu specifičnih programskih metoda.

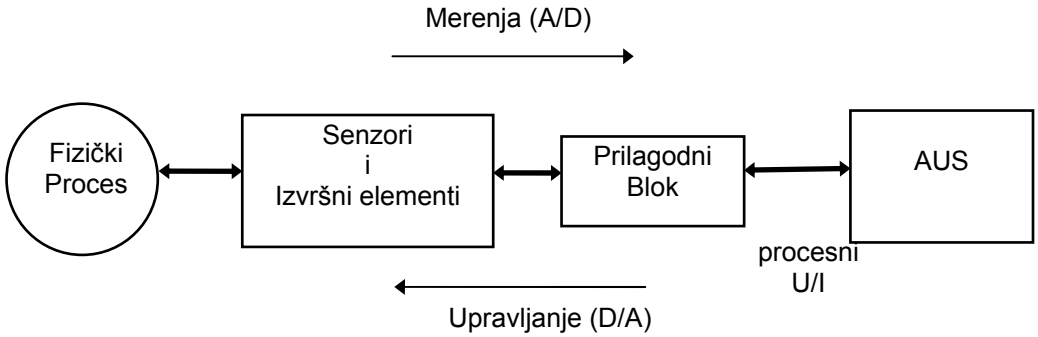
Proizvodni procesi, kao objekt nadzora i upravljanja AUS sistema, sami po sebi su prostorno dislocirani u okviru industrijskog postrojenja, ili i u mnogo širem geografskom području. Stoga se struktura AUS sistema odlikuje prostornom distribucijom autonomnih računarskih komponenti, koje su komunikacionom mrežom spregnute u jedinstven sistem. Razmenom poruka između njih ostvaruje se međusobna kooperacija u ostvarenju zajedničkog cilja. Zato se AUS sistemi svrstavaju u distribuirane računarske sisteme.

S obzirom na visoke troškove prekida rada procesnog sistema usled eventualnog otkaza računarskog podsistema, važan zahtev koji AUS mora zadovoljiti je postizanje

maksimalne pouzdanosti i raspoloživosti. Pouzdanost AUS sistema vezana je za kvalitet ugrađene fizičke i programske opreme, i najčešće se definiše srednjim vremenom između ispada. Raspoloživost se definiše procentom neispravnih komponenti koji još uvek ne onemogućuju funkcionisanje AUS sistema u celini. Drugim rečima, raspoloživost je mera sposobnosti akviziciono-upravljačkog sistema da nastavi rad i u slučaju otkaza pojedinih komponenti.

2.2 Sprega SCADA sistema sa fizičkim procesom

Strukturna šema ukupnog procesno-upravljačkog sistema (slika 4.2) prikazuje način povezivanja AUS-a sa fizičkim procesom. Senzori i izvršni elementi obezbeđuju spregu sa procesom kojim se upravlja, i stoga čine osnovu merno-regulacionog dela akviziciono-upravljačkog sistema. Funkcije fizičke komunikacije AUS sa njima vrše se posredstvom podsistema procesnih ulaza/izlaza (*procesni U/I*). U literaturi je uočena dilema u pogledu granice između procesnog i akviziciono - upravljačkog dela postrojenja. U ovom radu je usvojen stav da *AUS počinje na ulazima/izlazima prve računarske komponente u sistemu*, odnosno na nivou procesnog U/I. U široj definiciji, akviziciono - upravljački sistem obuhvata sve električne komponente upravljačkog sistema, počev od mernih signala senzora, završno sa pobudnim kolom izvršnih elemenata.



Slika 2.2. Strukturna šema procesnog sistema

Prilagodni blok vrši uobličavanje procesnih U/I signala i galvansko razdvajanje između procesnih uređaja i akviziciono-upravljačkog sistema. Kvalitet analognih komponenti u okviru prilagodnog bloka direktno utiče na kvalitet merno/upravljačkih signala, kao i na pouzdanost ukupnog sistema.

2.2.1 Procesni ulazi

Najniže u hijerarhiji nadzorno-upravljačkog sistema su *senzori*, tj. pretvarači neke fizičke veličine u ekvivalentni električni signal. Digitalizacijom ovakvih signala u okviru AUS-a, vrši se obuhvat merenih podataka (fizička akvizicija). Fizička akvizicija podataka se izvršava posredstvom tzv. *procesnih ulaza*. Standarni tipovi procesnih ulaza u akviziciono upravljačkom sistemu su:

- *Analogni ulazi*. Nivo kontinualnog električnog signala (strujnog ili naponskog) na analognim ulazima AUS sistema u svakom trenutku je proporcionalan trenutnoj vrednosti merene fizičke veličine. Tipični opsezi ulaznih električnih signala su: 4-20 mA, ±5V, 0-10 V, 0-100 mV i sl. Izvorne fizičke veličine koje se na ovaj način mere su pritisak, temperatura, masa i sl. Digitalizacija analognih

ulaznih signala vrši se korišćenjem A/D konvertora, a rezolucija i tačnost konverzije određena je brojem bita digitalizovanog odmerka. U praksi je očigledna tendencija da se digitalizacija električnog signala vrši već u sklopu samog pretvarača ("smart" transmiteri), a da se odmerna vrednost do AUS sistema prenosi posredstvom serijskog kanala i adekvatnog komunikacionog protokola.

- *Brojački (impulsni) ulazi*. Za brojačke ulaze je karakteristično da učestanost električnih impulsa koji se prihvataju predstavlja meru trenutne vrednosti fizičke veličine. Impulsni pretvarači se najčešće koriste za merenja protoka tečnosti ili gasova, merenje brzine rotacije (taho - generator), ugla zakretanja i sl.. Frekvencija generisanih impulsa u praksi je najčešće manja od 1 KHz, a njihov naponski nivo ne veći od 24 Vdc. Pošto se u okviru AUS-a impulsni signali prihvataju posredstvom digitalnih brojačkih kola, ovi procesni ulazi se nazivaju *brojačkim ulazima*.
- *Digitalni ulazi*. Digitalni ulazi predstavljaju fizičke veličine koje su diskretne već po svojoj prirodi. Najčešće se preko njih prati stanje izvršnih elemenata u postrojenju, kao što su ventil (*otvoren / zatvoren*) ili sklopka (*uključena / isključena*). Izvor ovakvih signala su i razni graničnici (*pun / prazan*), sigurnosni prekidači poput presostata (*natpritisak / normalno*), i sl. Ulazni električni signal je naponski, nivoa 24 Vdc ili 220 Vac. Digitalni ulazni signali se očitavaju posredstvom ulaznih registara.

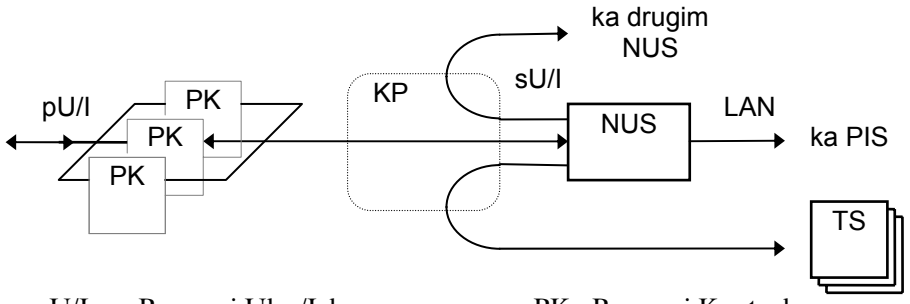
2.2.2 Procesni izlazi

Preko izvršnih elemenata (uređaja) utiče se na sam fizički proces, tj. ostvaruju upravljačke funkcije akviziciono-upravljačkog sistema. U opštem slučaju, to su elektromehanički uređaji direktno uključeni u samo procesno postrojenje. Posredstvom *procesnih izlaza*, akviziciono - upravljački sistem kontroliše izvršne elemente generisanjem pobudnih električnih signala. Standardni su sledeći tipovi procesnih izlaza:

- *Digitalni izlazi*. Posredstvom digitalnih izlaza ostvaruje se upravljanje tipa *uključi/isključi* ili *otvori/zatvori*, za uređaje kao što su sklopke, kontaktori, aktuatori ventila i sl. Digitalni izlazi se najčešće opisuju radnim naponom i maksimalnom strujom koju mogu da propuste. Tipične vrednosti su 24Vdc/1A, 220Vac/3A, i sl. U okviru AUS-a digitalni izlazi se pobuđuju posredstvom izlaznih registara (PIO)].
- *Analogni izlazi*. Analognim (kontinualnim) pobudnim signalom zadaje se radna tačka izvršnog elementa (pozicija regulacionog ventila, referentna vrednost eksternog PID kontrolera i sl). Analogni procesni izlaz opisuje se električnim opsegom izlaznog signala: 0-20 mA, 0-5 V, itd. Generisanje analognog izlaznog signala vrši se korišćenjem D/A konvertora rezolucije 8-12 bita. Poput analognih transmitera, sve je veći broj tzv. "pametnih" (*smart*) izvršnih elemenata koji se, zahvaljujući ugrađenom mikroprocesorskom bloku, serijski povezuju sa akviziciono-upravljačkim sistemom. Ovakvi uređaji često preuzimaju izvršenje složenijeg lokalnog upravljanja (PID kontrola npr.), čime značajno pojednostavljuju izvedbu AUS, uz istovremeno podizanje ukupne raspoloživosti upravljačkog sistema.

2.3 Arhitektura akviziciono-upravljačkih sistema

U toku razvoja koncepta akviziciono-upravljačkih sistema, u pogledu podele posla između pojedinih elemenata i ukupne arhitekture, kao industrijski "*de facto*" standard je usvojena hijerarhijska struktura prikazana na slici 2.3. U njoj dominantne komponente *procesni kontroler* i *nadzorno upravljačka stanica*, međusobno spregnute komunikacionim podsistemom, ostvaruju funkcije operativnog nadzora i upravljanja nad fizičkim procesom. Dodatna povezivanje stanice NUS, prikazana na slici, su u funkciji kooperacije i razmene podataka sa poslovnim ili širim tehnološkim okruženjem proizvodnog segmenta u kome je akviziciono - upravljački sistem implementiran.



pU/I : Procesni Ulaz/Izlaz
sU/I : Serijski Ulaz/Izlaz
NUS : Nadzorno Upravljačka Stanica
PIS : Poslovni Informacioni Sistem
PK : Procesni Kontroler
KP : Komunikacioni Podsistem
TS : Terminalska Stanica

Slika 2.3. Konfiguracija tipičnog AUS sistema

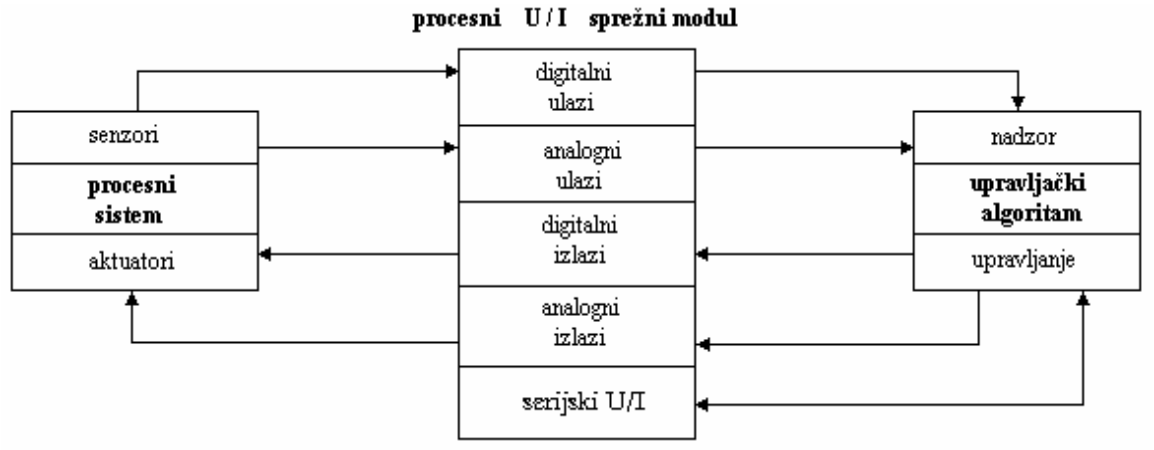
2.3.1 Procesni kontroler

Procesni kontroler (*PK*) je mikroprocesorska stanica koja služi prvenstveno za obuhvat mernih signala sa senzora, kao i za generisanje pobude izvršnih elemenata. Standardna definicija PLC uređaja sasvim je adekvatna i za procesni kontroler u širem smislu.

Po NEMA (*National Electrical Manufacturers Associations*) definiciji, prgramabilni logički kontroler je rešenje zasnovano na mikroprocesoru, koje koristi ulazne module povezane sa senzorima radi očitavanja stanja kontrolisanog sistema, programsku podršku radi analiziranja stanja sistema i rešavanja značajnih akcija, i na kraju, koristi izlazne module za delovanje na sistem preko aktuatora.

Procesni U/I moduli prikupljaju signale koji dolaze sa senzora, skladište ih u memoriju u digitalnom obliku, i proizvode električne izlazne signale proporcionalne digitalnim veličinama izlaznih promenljivih uskladištenih u memoriji PK. Za ostvarenje ovih funkcija, označenih kao *procesni U/I* (pU/I), PK je dodatno opremljen posebnim elementima fizičke arhitekture. Oni prilagođavaju i pretvaraju procesne U/I signale u formu pogodnu za obradu na digitalnom mikroprocesoru. Njihovu osnovu čine A/D i D/A konvertori, brojačka i PIO kola. Galvansko razdvajanje digitalnog dela PK od okoline se vrši mernim transformatorima i opto-elektronskim komponentama. Programska podrška PK obezbeđuje očitavanje svih procesnih ulaza u diskretnim vremenskim trenucima, čija učestanost mora biti adekvatna dinamici fizičkog procesa kojim se upravlja. Povezivanje PK sa kontrolisanim fizičkim sistemom prikazan je na slici 2.4.

Važan deo PK je podsistem *serijskog U/I* (sU/I) koji ostvaruje razmenu poruka sa centralnom tačkom akviziciono upravljačkog sistema - stanicom NUS. Režim komunikacije je najčešće višeutični ("*multi-drop*"), pri čemu NUS u celosti kontroliše pristup komunikacionom prenosnom kanalu. Pojedini PK na zajedničkom komunikacionom kanalu međusobno se razlikuju po *adresi*. Međusobnom razmenom poruka NUS i PK koordiniraju svoje aktivnosti. Minimalne funkcije koje se na PK iniciraju prijemom poruke su slanje prikupljenih podataka ka NUS, kao i izvršavanje primljenih komandi.



Slika 2.4. Povezivanje PK sa kontrolisanim sistemom

Osnovni zahtev koji se postavlja pred PK je pouzdanost u radu. U pogledu fizičke arhitekture, pouzdanost se obezbeđuje izborom najpouzdanijih elektronskih komponenti, pažljivom montažom i rigoroznim proverama pre puštanja u rad. Istovremeno, primenjuju se posebne tehnike i elementi fizičke podrške kojima se postiže maksimalna pouzdanost u izvršenju programske podrške PK. Standardno su prisutni vremenska kontrola ("*watch-dog timer*"), baterijsko napajanje RAM memorije, detekcija nestanka napajanja uz mogućnost kontrolisanog završetka rada, i sl. Za bezbednosno kritične procese, gde je preko potreban visok stepen pouzdanosti, koriste se redundantna rešenja. Pored redundancije fizičke arhitekture, zasnovane na udvajanju fizičkih resursa, postoje metode programske redundancije koje se baziraju na dodatnoj nezavisnoj verifikaciji postignutih rezultata.

Deo obrade merenih podataka i upravljačkih funkcija se mogu izvršavati lokalno, na samom PK. Smanjenjem uticaja komunikacionih grešaka i otkaza komunikacione mreže, postiže se povećanje raspoloživosti ukupnog AUS sistema, a pogotovo konzistentnost merenih podataka.

2.3.2 Komunikacioni podsistem

Komunikacioni podsistem u okviru akviziciono-upravljačkog sistema čine komunikacioni prenosni kanali i komunikacioni kontroler (server) stanice NUS.

Komunikacioni kanali povezuju stanicu NUS sa mrežom PK ili drugim serijski spregnutim komponentama ukupnog akviziciono-upravljačkog sistema. Komunikacioni kanal obuhvata svu opremu potrebnu za kvalitetnu razmenu poruka (modemi, koncentratori, prilagodni elementi, i sl.). U cilju postizanja efikasnosti AUS-a, brzina i propusnost komunikacionog kanala moraju biti na odgovarajućem nivou.

U industrijskim AUS, gde su razdaljine između pojedinih komponenti relativno male, komunikacija se najčešće realizuje korišćenjem RS-485 ili RS-422 fizičkih protokola, preko namenski instaliranih parica. Ovakva rešenja obezbeđuju vrlo

pouzdan, brz i jevtin prenosni kanal. Prethodno navedeni fizički protokoli su najčešće i osnova za realizaciju *procesne magistrale*, uz dodatak specijalizovanih komunikacionih kontrolera koji obezbeđuju protokole višeg nivoa i mrežne usluge. Raspoloživa su različita rešenja procesnih magistrala u pogledu topologije, brzine prenosa i logičkog linijskog protokola, ali im je zajednička osobina visok nivo standardizacije i raspoloživost međusobno kompatibilne opreme različitih proizvođača. Zahvaljujući tome, "fieldbus" pristup doživljava ekspanziju u oblasti periferne procesne i PC opreme, šireći se van granica PLC domena u kome je inicijalno razvijen. Naime, fleksibilnost "fieldbus" rešenja došla je do punog izražaja zahvaljujući otvorenosti PC arhitekture, i visokom kvalitetu programske podrške i raspoloživih programskih razvojnih alata. Zato su *procesne magistrale* osnova savremenih distribuiranih upravljačkih sistema razvijenih na IT platformi.

U novije vreme, sve prisutnije je korišćenje brzih LAN kanala (Ethernet IEEE 802.3) i protokola TCP/IP, koji pružaju izuzetnu brzinu i propusnost, kao i proširenje opcija u pogledu protokola razmene poruka. Zahvaljujući karakteristikama pristupnih LAN protokola, svaka od stanica na LAN kanalu može inicirati slanje poruke (bez čekanja na upit od NUS), i to ka bilo kojoj od stanica u mreži. Time su otvorene nove mogućnosti međusobne komunikacije između stanica, kao i izmenjena organizacija ukupnog AUS sistema.

Geografski distribuirani telemetrijski sistemi koriste složen, a često i vrlo nepouzdan sistem komunikacionih veza. Najčešće se koriste privatni ili iznajmljeni telefonski kanali, kao i sopstvena radio mreža. U određenim slučajevima neophodna je upotreba satelitskih veza. Kvalitet komunikacionih veza u ovakvim sistemima u najvećoj meri određuje pouzdanost i raspoloživost ukupnog akviziciono - upravljačkog sistema.

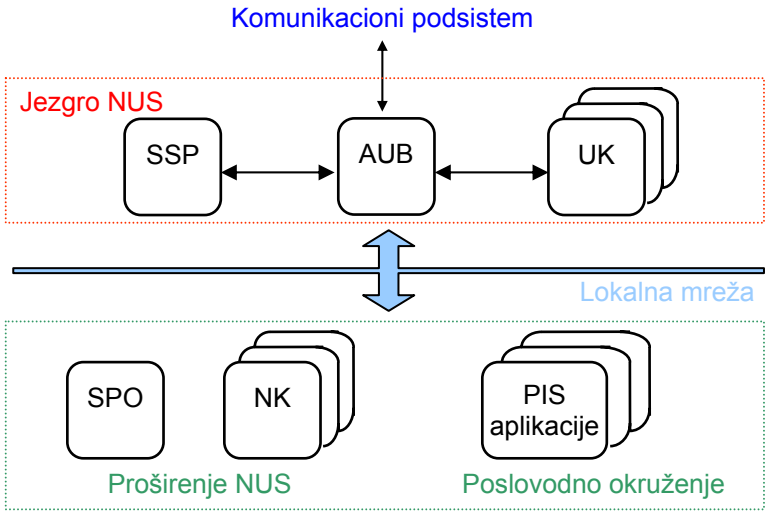
Komunikacioni kontroler je posvećen obavljanju i kontroli svih komunikacionih aktivnosti stanice NUS. Na ovaj način se centralni računar oslobađa komunikacionih prekida koji opterećuju izvršavanje primarnih funkcija NUS [3]. U principu, komunikacioni kontroler može se smatrati komponentom NUS. Osnovni zahtevi koje mora zadovoljiti vezani su za visok stepen efikasnosti i pouzdanost u radu.

2.3.3 Nadzorno upravljačka stanica

Nadzorno - upravljačka stanica (NUS) je centralni čvor akviziciono-upravljačkog sistema, sa primarnim funkcijama objedinjavanje i obrada svih procesnih podataka, i komunikacije sa ljudskom posadom (operaterima). Objedinjavanje procesnih podataka ostvaruje se komunikacijom sa mrežom instaliranih procesnih kontrolera. Obrada prikupljenih podataka podrazumeva proveru ispravnosti i vrednosti merenja, uz eksplicitno izveštavanje operatera o otkrivenim neregularnostima. Svi podaci se pretvaraju u oblik pogodan za prikaz, kao i za generisanje upravljačkih akcija. Operaterska sprega ostvaruje se posredstvom dinamički osvežavanih, posebno definisanih grafičkih i alfanumeričkih prikaza. Operaterski podsistem mora obezbediti pregledno i kvalitetno izveštavanje o promenama u sistemu, kao i brzo i tačno prihvatanje operaterskih komandi. Važan deo aktivnosti NUS vezan je za formiranje istorijata akviziciono - upravljačkog sistema, trajnog zapisa stanja i događaja u fizičkom sistemu, kao i praćenje operaterskih upravljačkih akcija.

Jedna od osnovnih karakteristika AUS sistema je centralizacija najprioritenijih upravljačkih funkcija na stanici NUS. Izvršenje upravljačkog algoritma podeljeno je po nivoima sistema, i odvija se prvenstveno na nivou procesnih kontrolera. Ipak, zbog prisustva ljudske posade, iniciranje najznačajnijih kontrolnih funkcija i krajnja verifikacija njihovog izvršenja vrši se isključivo na stanici NUS.

Fizički, stanica NUS obuhvata jedan ili više računara opremljenih monitorima, štampačima i drugim periferijama koje obezbeđuju prikaz i skladištenje prikupljenih podataka. Logička organizacija komponenti u okviru NUS prikazana je na slici 2.5.



Slika 2.5. Struktura *Nadzorno Upravljačke Stanice*

Osnovne komponente, neophodne za izvršenje navedenih zadataka, čine samo jezgro nadzorno upravljačke stanice. To su:

- *Akviziciono Upravljački Blok* (*AUB*) posevećen je funkcijama akvizicije, obrade podataka, iniciranja i izvršenja upravljačkih akcija. Istovremeno, AUB komunicira i usklađuje rad sa svim ostalim komponentama NUS. U cilju povećanja pouzdanosti AUB, kao kritične komponente za rad ukupnog AUS sistema, često se primenjuje udvajanje centralnih procesora u tzv. "*tandem*" konfiguraciju.
- *Upravljačka Konzola (UK)* je operaterska radna stanica sa punim pravima upravljanja i rukovanja akviziciono - upravljačkim sistemom. Zato je UK u najtešnjoj vezi sa AUB blokom, najčešće u formi permanentne komunikacione sesije. Važan deo operaterske podrške su procedure provere prava pristupa i izdatih komandnih direktiva.
- *Server SCADA Podataka* (*SSP*) čuva bazu podataka, koja čuva sve konfiguracione podatke i istorijat akviziciono - upravljačkog sistema. Savremena rešenja podrazumevaju upotrebu mrežnog servera i relacione baze podataka. Na taj način, server SSP postaje i spona AUS sa okruženjem.

Kod izvođenja jednostavnih aplikacija, sve tri komponente se često realizuju upotrebom samo jednog računara. Složenije primene zahtevaju udvajanje AUB i veći broj operaterskih radnih mesta, te stoga broj ugrađenih računara može biti i vrlo velik. Iz bezbednosnih razloga, prihvaćena je praksa njihove prostorne i mrežne izolacije, kao i ograničenog fizičkog pristupa.

U proizvodnim preduzećima postoji širok skup korisnika podataka proisteklih iz akviziciono upravljačkog sistema. Korporacijska lokalna mreža i poslovni informacioni sistem su okvir u kome se ostvaruje veza stanice NUS sa njima.

Najbrojnija kategorija takvih korisnika zainteresovana je za pristup podacima u realnom vremenu, tačnije za uvid u trenutno stanje kontrolisanog fizičkog sistema. To se zadatak programske podrške *nadzorne konzole* (*NK*), koja se pokreće sa radnih

stanica unutar poslovnog segmenta lokalne mreže. Suština nadzorne konzole je prikaz AUS podataka u formatu sličnom ili identičnom onom na upravljačkoj konzoli.

U cilju ostvarenja potpunije kontrole nad kompleksnim tehnološkim procesima, osnovni AUS sistem se proširuje *stanicom za podršku u odlučivanju*. Cilj simulacionog modela ili ekspertnog sistema koji se izvršava na njoj, je predviđanje kritičnih događaja u fizičkom procesu, kao i upućivanje na moguće korektivne akcije. Osnova za ove funkcije su podaci koje obezbeđuje akviziciono-upravljački sistem, uključujući i trend kritičnih procesnih veličina. Uvođenjem metoda veštačke inteligencije u okruženje sistema u realnom vremenu, teži se postizanju sledećih prednosti:

- Brže rešavanje kritičnih situacija i smanjenje vremena odziva operatera, pa i celog sistema AUS;
- Obezbeđivanje mogućnosti da se problemi otkriju bez prisustva eksperata, analitički ili na osnovu baze znanja ekspertnog sistema, pre nego što postanu opasni ili skupi za rešavanje;
- Unapređenje u delu održavanja skupe procesne opreme.

Podaci prikupljeni akviziciono-upravljačkim sistemom su često od velikog interesa i u drugim segmentima poslovnog sistema, kao što su službe održavanja, kontrole kvaliteta, komercijale, planiranja i sl. Stoga je potrebno obezbediti spregu AUS sistema sa poslovno - tehnološkim informacionim sistemom. U osnovi ovog zahteva je prenos AUS podataka do baze podataka dostupne iz poslovnog segmenta IS, koji se ostvaruje posredstvom lokalne mreže.

2.3.4 Nadzorno - upravljačka stanica sistema GAUS

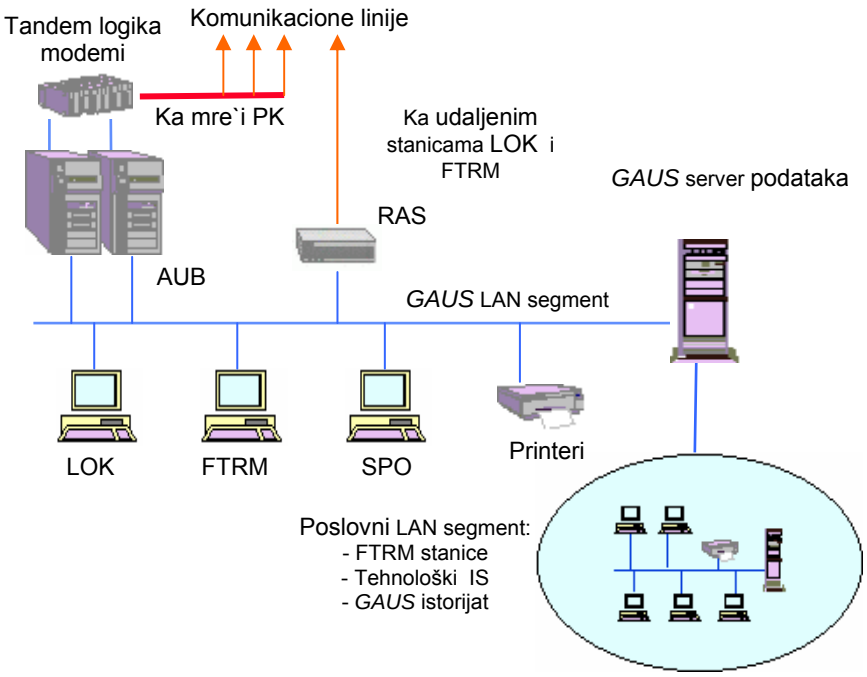
U okviru sistema *GAUS* razvijena je stanica NUS vrlo bogate strukture, primerena i najzahtevnijim primenama u industriji. Posebno je prilagođena primenama u telemetrijskim sistemima, gde je u proteklih deset godina u neprekidnoj eksploataciji, kao i u procesu kontinualnog unapređivanja.

U pogledu fizičke arhitekture, NUS realizuju PC radne stanice standardne konfiguracije. One su međusobno povezane LAN magistralom, koja obezbeđuje komunikacioni kanal velike brzine prenosa poruka i visoke propusnosti. Slika 2.6 prikazuje arhitekturu stanice NUS u sistemu *GAUS* [4-8].

Akviziciono upravljački blok je realizovan pouzdanim industrijskim računarima u udvojenoj "tandem" konfiguraciji, sa obezbeđenom vrućom rezervom. Pored standardnih serijskih komunikacionih kanala (modemska i RS-232/ 485/ 422 veza), podržano je i povezivanje sa procesnom magistralom. Pored podrške za povezivanje sa procesnim kontrolerom sistema *GAUS*, podržano je povezivanje sa kontrolerima Nuovo-Pignone, Omron, Modicon i ADAM. AUB obavlja funkcije komunikacionog servera akviziciono - upravljačkog sistema:

- Komunikacija sa mrežom procesnih kontrolera.
- Sekundarna obrada prikupljenih podataka.
- Alarmiranje i evidencija događaja.
- Zapis istorijata.
- Distribuiranje podataka o stanju sistema ka ostalim komponenatama NUS-a.

Lokalne operaterske konzole (*LOK*) imaju ulogu upravljačke konzole sistema *GAUS*. LOK konzole dobijaju sliku o stanju sistema direktno od AUB-a, a osim funkcije nadzora, LOK omogućuje i generisanje upravljačkih akcija. Posredstvom LOK konzole operater može kontrolisati sistem u potpunosti ili parcijalno (po delovima tehnološkog procesa) u zavisnosti od dodeljenih prava. Moguće je povezivanje do 30 operaterskih konzola u *GAUS* sistem.



Slika 2.6. Puna konfiguracija *Nadzorno Upravljačke Stanice* sistema *GAUS*

Komunikacija između računarskih komponenti AUB i LOK vrši se preko LAN magistrale posredstvom NETBIOS usluga. Ovaj komunikacioni protokol je veoma rasprostranjen i većina proizvođača mrežne opreme obezbeđuje odgovarajuću programsku podršku za njegovu upotrebu. Osim što je jednostavan za rukovanje, NETBIOS obezbeđuje podršku za komunikaciju putem sesija i putem datagrama. Komunikacija putem NETBIOS sesija obezbeđuje spregu ka protokolu transportnog sloja ISO OSI referentnog modela, čime se garantuje pravilan prijem poruka i njihovo pristizanje na odredište u odgovarajućem redosledu.

Oslanjanjem na usluge NETBIOS-a postignuta je nezavisnost komunikacionih programskih funkcija od mrežnih protokola, s obzirom da većina proizvođača mrežnih operativnih sistema (*Novell Netware, Windows NT*) nudi NETBIOS usluge za različite komunikacione protokole (IPX/SPX, TCP/IP, NETBEUI).

Realna brzina prenosa između dve stanice na 10Mb *Ethernet* magistrali u režimu NETBIOS sesije iznosi oko 1-3 Mbit/sec. Obzirom na raspoloživost velikih komunikacionih bafera, očigledno je da aplikacija NUS obezbeđuje efikasan prenos poruka uz minimalna kašnjenja u obradi.

Posebnu pogodnost za razvoj programske podrške NUS pruža LAN okruženje mogućnošću višestrukog, transparentnog pristupa podacima na *GAUS* serveru podataka. Sve konfiguracione datoteke *GAUS*, formirane posebnim alatima, čuvaju se na jednom mestu, kao jedna radna kopija. Ipak, obezbeđen je zapis rezervnih kopija na lokalni disk PC-stanica, kao i mogućnost samostalnog rada AUB u slučaju ispada servera podataka. Centralizacija konfiguracionih podataka za sve komponente NUS

ima izuzetan značaj u toku eksploatacije sistema (održavanje ažurnog stanja konfiguracije za sve učesnike).

Osnovne programske funkcije stanice NUS su sledeće:

- *Opis nadziranog fizičkog procesa i sistema telemetrije*. Opis fizičkog procesa podrazumeva definisanje svih potrebnih parametara procesnih veličina i uređaja koji se prate, ili se njima upravlja. Deo opisa se odnosi i na ustanovljavanje logičkih relacija među njima, uslovljenih strukturom tehnološkog procesa. Konfiguracionim direktivama opisuju se i svi parametri elemenata telemetrijskog podsistema (komunikaciona mreža i procesni kontroleri). Na ovoj osnovi, u operativnoj memoriji računara uspostavlja se trajno prisutna centralna baza podataka za koju je vezano izvršavanje svih ostalih programskih aktivnosti NUS. Njena organizacija mora obezbediti efikasan pristup svakoj od procesnih veličina korišćenjem jedinstvene identifikacije.
- *Komunikacija sa mrežom PK*. Primarna funkcija NUS je komuniciranje sa mrežom PK u cilju pribavljanja merenih podataka. Primljeni podaci predstavljaju stanje fizičkog procesa u diskretnim kvazi - uniformnim trenucima.
- *Sekundarna obrada prikupljenih podataka*. Sekundarna obrada podataka podrazumeva proveru primljenih podataka i njihovo pretvaranje u oblik koji se pamti u bazi podataka.
- *Alarmiranje i evidencija događaja*. Događajem u sistemu AUS se smatra detekcija određene okolnosti u procesnom sistemu označene eksplicitnom porukom (po ugrađenom kriterijumu). Ova poruka se eksplicitno prikazuje operateru, a sam događaj trajno memoriše. Alarm je događaj koji prati detekciju kritičnih događaja u sistemu, i stoga ga prati posebna procedura (potvrda i brisanje od strane operatera). Događajima se obično pridružuje nivo prioriteta, tako da je u toku rada akviziciono-upravljačkog sistema moguće zabraniti ili dozvoliti ispis određenih događaja.
- *Iniciranje upravljačkih akcija*. Upravljačke akcije mogu se inicirati ručno ili automatski. Ručno se upravlja posredstvom operaterskog sprežnog podsistema (grafičke šeme procesa i sl.). U drugom slučaju, radi se o automatskoj upravljačkoj proceduri koja je ugrađena u aplikativnu programsku podršku akviziciono-upravljačkog sistema.
- *Prikaz stanja i funkcije komunikacije sa operaterom*. Prikaz stanja obično ima dva nivoa. Prvi nivo je u principu potpun prikaz svih procesnih parametara u fiksnom formatu, najčešće u formi tabela. Postupkom listanja i pregleda niza ekranskih izveštaja, korisnik može ostvariti uvid u trenutno stanje, izmenu određenih procesnih parametara, kao i generisanje upravljačkih zahteva. Drugi nivo je grafički prikaz u formatu koji definiše korisnik, najčešće u formi tehnološke šeme postrojenja, sa dinamički osvežavanim simbolima itd. Na kraju, ove funkcije obuhvataju evidenciju dežurnog operatera uz kontrolu pristupa pomoću unosa lozinke.
- *Zapis istorijata*. Očigledna je potreba i značaj trajnog arhiviranja podataka poteklih iz akviziciono-upravljačkog sistema. Ovi podaci su neophodna osnova za analizu rada samog postrojenja, ostvarenih učinaka, zastoja i sl., kao i za odgovarajući simulacioni model ili ekspertni sistem za podršku u odlučivanju.

Aplikacije Nadzorno Upravljačke Stanice karakteriše bogata operaterska sprega. Tabelarni (alfanumerički) prikaz u predefinisanoj formi prikazuje stanja procesnih promenljivih, tok komunikacije, alarme, događaje u sistemu, itd. Prikaz procesnih veličina može se organizovati kroz korisničke tabele, u proizvoljno određenom redosledu, sve u cilju jednostavnijeg i preglednijeg prikaza.

Formiranje grafičkih procesnih šema vrši se iz posebnog alata *SEDIT*. Grafičke procesne šeme karakteriše:

- mogućnost kombinovanje vektorskih i rasterskih formata,
- skalabilnost prikaza ("zumiranje"),
- hijerarhijska (višenivovska) organizacija šema,
- prikaz šema u više prozora,
- prikaz trenutnih vrednosti i stanja procesnih veličina,
- izdavanje ručnih komandi preko elemenata šeme,
- animacija objekata na šemi, stalna ili u zavisnosti od trenutnog stanja procesne veličine,
- formiranje "bar" i "scroll" grafikona za prikaz procesnih veličina sa mogućnošću periodičnog zapisa merenih vrednosti,
- stalno praćenje sistemskih događaja uz obavezno potvrđivanje kritičnih,
- mogućnost dodele različitih prioriteta procesnim veličinama.

Komunikacije između *GAUS* i korisnika van njega (tehnološko-poslovni IS) se vrši posredstvom lokalne mreže, mrežnog file sistema i servera podataka.

File Terminal (*FTRM*) je aplikacija za distribuciju funkcija nadzora na stanice u poslovnom segmentu. Predmet prikaza su podaci zapisani na *GAUS* serveru, a koje AUB periodično osvežava u intervalu reda 1 - 5 min. Pristup FTRM korisnika centralnim podacima je zaštićen, tj. postoji kontrola pristupa.

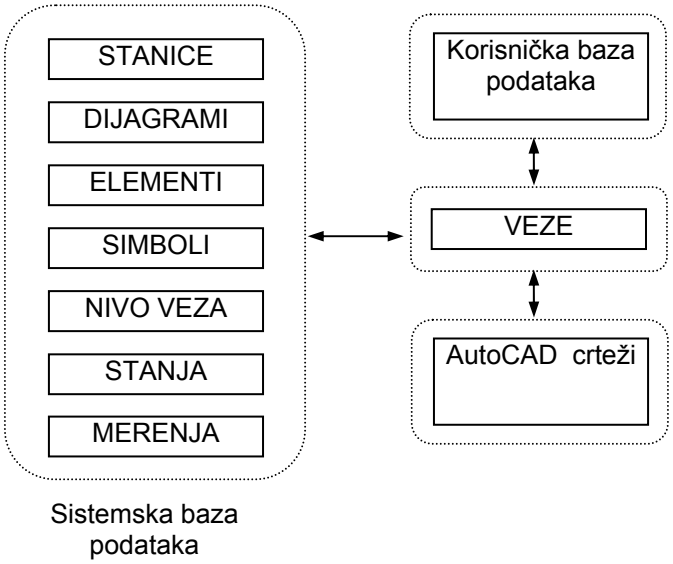
Iako razvijena na *Katedri za računarsku tehniku*, podrška za pomoć u odlučivanju nije predmet razmatranja u ovom radu, jer je direktno vezana za fizičku prirodu procesa kojim se upravlja. Konkretno, na bazi metoda veštačke inteligencije, razvijen je estimator i prediktor stanja gasovodnog transportnog sistema [9].

Kao spona između AUS i poslovnog sistema, na Katedri je razvijen programski paket nazvan *TIS - Tehnološki Informacioni Sistem*, čiji je koncept orijentisan ka širokoj oblasti primene u industriji [10-11]. Paket TIS je grafički informacioni sistem baziran na AutoCAD-u, razvijen u nastojanju da se pruži podrška u proceni i analizi stanja proizvodnog sistema. U praksi, to stanje zavisi od brojnih elemenata koji iz različitih razloga nisu uključeni u *GAUS* sistem. Njihovo stanje nije pod kontinualnom kontrolom, nego se utvrđuje periodičnim obilaskom i ručnim evidentiranjem promena. Deo tih podataka od najvećeg je značaja za uspešno održavanje opreme i uređaja u sistemu. Povezivanjem ovih podataka sa *GAUS* merenjima ostvaruje se potpun uvid u strukturu i ponašanje kontrolisanog sistema.

Na slici 2.7 je prikazana organizacija TIS podataka, i način povezivanja AutoCAD crteža sa bazom podataka.

Korisnički podaci su proizvoljnog formata (sem ključa za vezu sa ostatkom TIS-a), i sadrže najrazličitije informacije o tipu, stanju i održavanju komponenti proizvodnog sistema.

AutoCAD crteži pokazuju tehnološku strukturu proizvodnih postrojenja i instalacija. Zahvaljujuću posebnom načinu njihove organizacije, ovi crteži predstavljaju za prikaz svih TIS informacija, uključujući i dinamičku animaciju transportnih režima (protoka gasa ili električne energije npr.).



Slika 2.7 Baza podataka Tehnološkog Informacionog Sistema

Sistemski deo baze podataka obezbeđuje međusobno povezivanje crteža i korisničkih podataka, ali istovremeno i vezu sa SCADA merenjima iz baze podataka *GAUS* aplikacije.Grafička korisnička prega, kao i set alfanumeričkih alata razvijeni su za lak pristup svim podacima i njihovu analizu.

Osnovne karakteristike nadzorno upravljačke stanice sistema *GAUS* su bogata i fleksibilna konfiguracija, otvorena LAN arhitektura sa kontrolisanim pristupom procesnim podacima - trenutnim vrednostima ili istorijatu sistema zapisanom u relacionoj bazi podataka. Pored visoke pouzdanosti u radu, ostvarene redundantnom tandem konfiguracijom akvizicionog bloka, sačuvana je optimalna cena i obezbeđena usaglašenost sa savremenim trendovima informacionih tehnologija.

Razvijena programska podrška izvršava se pod operativnim sistemom Windows NT. Standardna relaciona baza je SyBase sa PowerBuilder razvojnim alatima, ali je po potrebi moguća laka adaptacija *GAUS* aplikacije na drugo rešenje.

3. SCADA SISTEMI U INTRANET / INTERNET OKRUŽENJU

3.1 Uvod

Trideset godina posle njihovog uvođenja internet protokoli su postali prihvaćeni kao standard od strane kompanija kao Microsoft i sastavni deo najvećeg broja SCADA sistema. Danas se oni koriste u SCADA sistemima kako u lokalnom (intranet) tako i spoljnom (internet) mrežnom okruženju. Masovna upotreba i širenje internet tehnologija dovelo je do značajnog sniženja cene i povećanja dostupnosti ovog vida komunikacije.

Do skora je povezivanje SCADA sistema sa drugim informacionim sistemima bilo moguće, ali ne i jednostavno. Napori da se ova situacija izmeni doveli su do tehnologija koje standardizuju i maksimalno pojednostavljuju integraciju SCADA sistema i informacionih sistema u intranet okruženju. *Browser* tehnologijom i web prezentacijom informacije mogu biti dostupne svugde u okviru proizvodnog postrojenja pa i dalje. Primenom interneta omogućen je pristup informacijama koje su ranije bile dostupne samo u okviru SCADA sistema, i ´otkriva´ ih drugim slojevima koji učestvuju u vođenju i organizovanju proizvodnog procesa.

Prelazak na zajedničke standarde značajno umanjuje napore neophodne da se povežu i održavaju različiti delovi informacionih sistema. Takođe je omogućena i upotreba moćnih alata, koji su razvijeni za internet okruženje, za nadzor i upravljanje procesa.

Standardi primenjeni u takvim alatima uključuju TCP/IP, HTML, DCOM, CORBA, OLE, ActiveX, Java i dr. Tehnologije koje koriste te standarde su:

- web browser-i
- mrežni PC računari i Windows CE (lokalno izvršenje programa)
- *thin* klijenti (izvršenje programa na serveru i terminalski prikaz)
- *push* tehnologija (server formira informacije za slanje ka klijentu)
- OLE for Process Control (OPC)

Najveći broj SCADA paketa danas omogućava metode za pregled informacija o nadziranom procesu sa udaljenog mesta pomoću intranet i/ili internet veze. Neki paketi koriste pomoćne programe za formiranje HTML stranica od grafičkih operaterskih maski, koje mogu biti prikazane bilo kojim standardnim web klijentom (statički prikaz). Drugi koriste specijalizovane klijent aplikacije za pregled informacija koje su dostupne od strane specijalizovane server aplikacije (dinamički prikaz). Neki sistemi omogućavaju i izmenu informacija klijent aplikacijom, ali se u tom slučaju značajna pažnja mora posvetiti sigurnosti takvog sistema.

Primarna korist od intranet / internet tehnologija u SCADA aplikacijama je jednostavna i uniformna sprega ka korisniku, browser.

Međutim i dalje postoji nekoliko nedostataka primeni internet tehnologija u SCADA aplikacijama. Prvi je svakako brzina. Svako ko je koristio internet tehnologiju sigurno se upoznao sa sporom grafikom i dugotrajnim preuzimanjem podataka (download). To je glavni razlog zašto internet aplikacije nisu praktične za veoma kritične sisteme u realnom vremenu.

Drugi značajan problem je svakako u sigurnosti takvih sistema. Svedoci smo čestih slučajeva neautorizovanog pristupa PC računarima putem intranet / internet veza.

Nasuprot tim nedostacima i ograni~enjima velik broj glavnih proizvođača SCADA aplikacija ili već ima ili najavljuje verzije programa koje će koristiti prednosti internet tehnologija [12]:

- Intellution- podrška za web browser (FIX Web Server) i *push* klijente (FIX Broadcast Network).
- Wonderware - podrška za web browser i *push* klijente (primenom Intenet Explorer *channels* tehnologije).
- Citect - podrška za web browser (HTML strane eksportovane na web server).
- GenSym G2 - podrška za web browser (CORBA and DCOM, Java, ActiveX).
- PCSoft WizBrowz - prikaz Wizcon ekrana bilo kojim standardnim web browser-om koji podržava Java tehnologiju.

3.2 Sprega web servera i SCADA sistema

Kako je razmatranje ograničeno na realizaciju http sprege primenom komercijalnih web servera biće izložene sprege web servera sa SCADA informacionim sistemom.

Sprega web servera i SCADA informacionog sistema se pre svega odnosi na preuzimanje podataka o stanju i vrednosti procesnih veličina nadziranog procesa od strane web servera.

U nastavku će biti razmotrene tehnologije preuzimanja podataka putem CGI skripta i putem aplikativnog servera.

3.2.1 Osnovni principi web tehnologije

Web ili globalna mreža podatke sadrži u obliku prezentacija. Pristup informacijama na web prezentaciji je omogućen web serverom, a korisnici im pristupaju putem web klijenta i uglavnom je to *browser*.

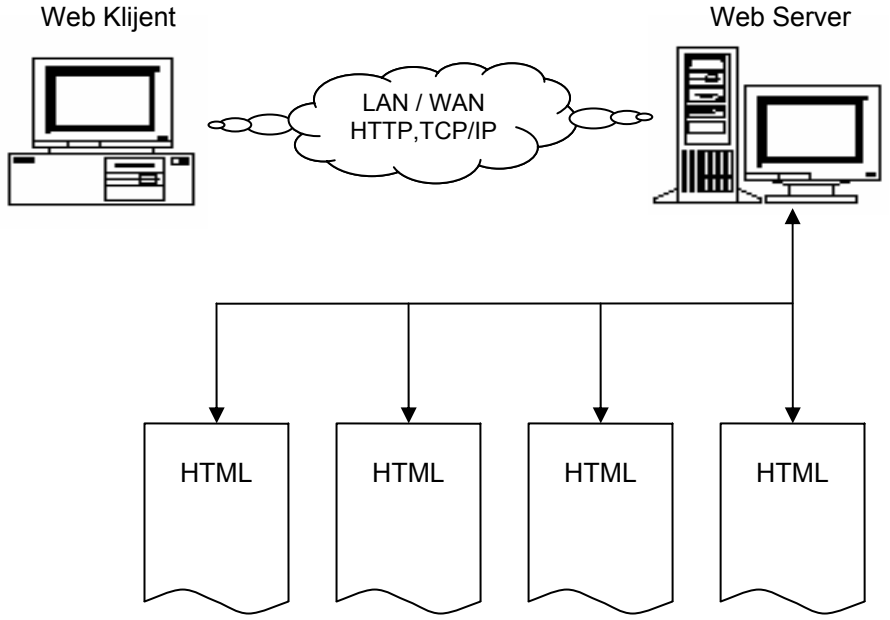
Informacije na prezentacijama su sadržane u dokumentima (resursima) primenom HTML (HyperText Markup Language) jezika. Web klijenti moraju razumeti i znati interpretirati HTML da bi mogli prikazati dokumente. Protokol razmene informacija između web servera i web klijenta je HTTP (HyperText Transfer Protocol).

Dokumenti i lokacije u dokumentu su određeni adresama definisanim kao URL (Uniform Resource Locator). URL adrese određuju o kojem web serveru je reč i o kojem dokumentu unutar web servera (http://www.micronasnit.com/index.html). URL adresa takođe može sadržati i dodatne informacije koje se prosleđuju od strane web klijenta ka serveru (World Wide Web Consortium - http://www.w3.org/pub/WWW/Addressing/).

Kada korisnik aktivira (klikne) određenu vezu (link) u okviru dokumenta u okviru web klijenta, URL adresa linka se pošalje odgovarajućem web serveru. Web server nalazi traženi dokument i salje ga kao HTML web klijentu.

HTML dokumenti mogu sadržati slike ili druge tipove podataka referencom na spoljnu datoteku (na primer GIF ili JPEG datoteke sa slikama). Nisu svi spoljni formati podžani od strane web klijenta. Ukoliko web klijent ne podržava format podataka sadržanih u spoljnoj datoteci on ih ni ne zahteva od web servera.

Ukoliko web server ima datotečnu organizciju (file based) svaki resurs je sadržan u posebnoj datoteci. Održavanje i manipulacija takvim prezentacijama predstavlja ozbiljan problem. Struktura povezivanja web klijenta i web servera data je na slici 3.1.



Slika 3.1. Struktura povezivanja web klijenta i web servera

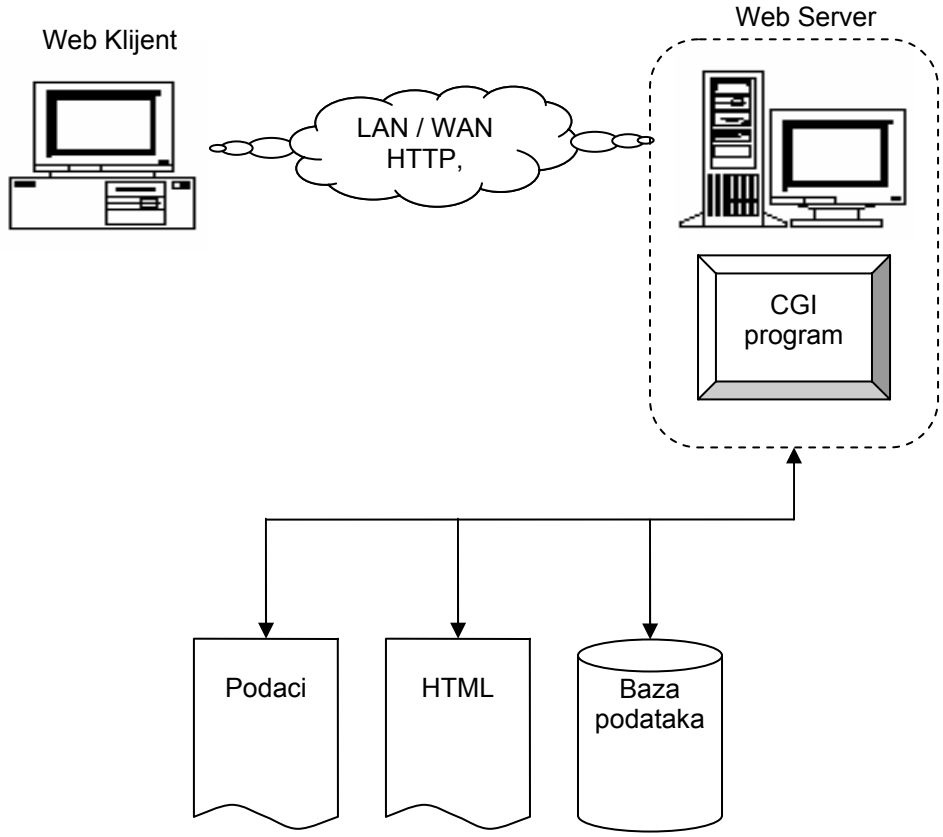
Zbog toga povezivanje baza podataka sa web tehnologijom predstavlja značajan korak unapred ka održavanju velikih web prezentacija i njihovog dinamičkog sadržaja. Takođe, organizacija web prezentacije u obliku baza podataka može zameniti organizaciju resursa po datotekama.

3.2.2 CGI skript

CGI [14][15] (Common Gateway Interface) je standard koji je phihvaćen za uspostavljanje veze između eksternih aplikacija i web servera.

Web serveri su na početku razvoja interneta podržavali samo statičke HTML dokumente i pridružene statičke datoteke. To je značilo da programi koji su služili za pregledanje HTML dokumenata, od strane web servera, mogli su da prikažu samo one strane čiji se sadržaj ne menja. Međutim sa razvojem interneta i proširivanjem njegovih mogućnosti pojavila se potreba i za dinamičkim sadržajem u HTML dokumentima. Za to je trebala da postoji određena veza između web servera i programa, koji bi obezbeđivao dinamički sadržaj web strane. To je rešeno uvođenjem CGI standarda.

CGI standard ne određuje kako će web server da radi ili kako će eksterni program da generiše rezultat, već određuje niz pravila koji moraju da poštuju i server i programi kako bi ispravno razmenjivali podatke. Definisan je i način kako će se razmenjivati informacije između programa za pregledanje web strana i web servera. Spoljni program je taj koji obezbeđuje dinamički sadržaj u HTML dokumentu. Ovi programi su poznati pod nazivom CGI programi ili CGI skriptovi. Rezultati njihovog izvršavanja, šalju se web serveru koji ih dalje prosleđuje programu za pregledanje HTML dokumenata (klijent strani) koji obezbeđuje njihov prikaz na ekranu korisnika. CGI skriptovi su dizajnirani za izvršenje na web serveru u okviru postojećeg operativnog sistema. Izvršavaju se u realnom vremenu, po upućenom zahtevu korisnika. Struktura realizacije web servera sa CGI skriptom prikazana je na slici 3.2.



Slika 3.2. Realizacija web servera sa CGI programom

CGI skriptovi mogu biti napisani u bilo kom programskom jeziku koji može da se izvršava na ciljnoj web platformi. Programski jezik se bira u zavisnosti od toga šta je zadatak skripta. Tako je npr. Perl pogodniji za pisanje skriptova koji su namenjeni manipulaciji stringovima ili tekstulnim datotekama, dok je za složenije programe pogodnije koristiti više programske jezike (C ili C++ programski jezik). Ostali jezici koji su takođe česti korišćeni su: PHP, TCL, Python, Visual Basic itd.

3.2.3 Aplikativni i Transakcioni server

Kako je već rečeno u prethodnom poglavlju, razvojem interneta pokazala se potreba za dinamičkim web prezentacijama. Za razliku od CGI skripta koji ne predstavlja univerzalno i objektno orijentisano rešenje dalji razvoj intenet tehnologija na server strani doveo je do nove organizacije web prezentacija.

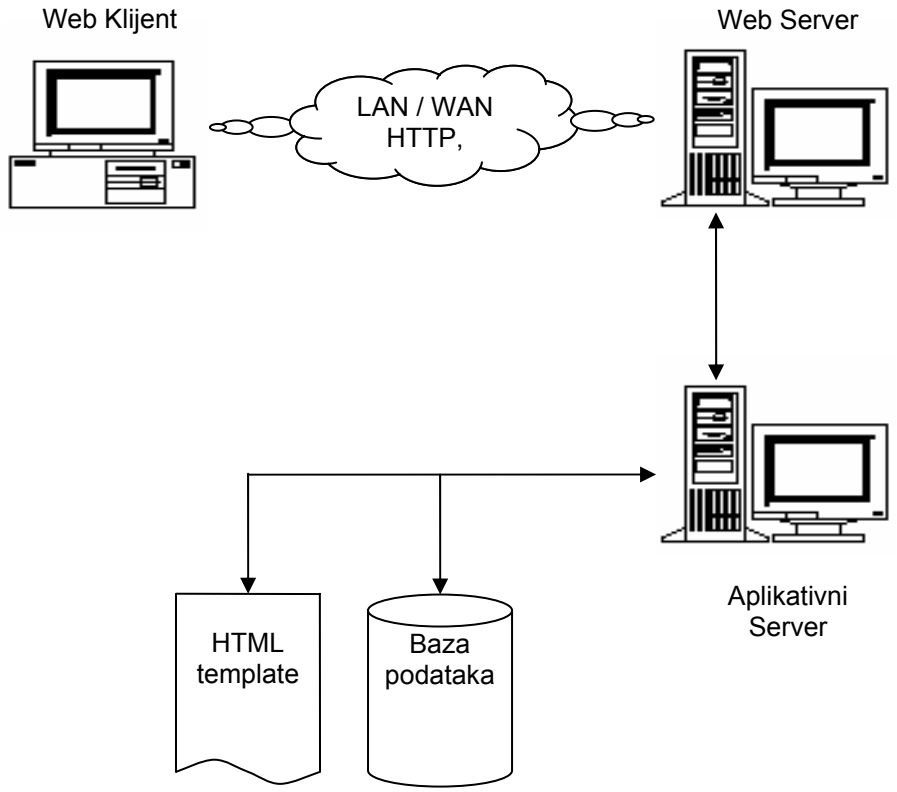
Prvobitna organizacija web prezentacije u okviru datoteka i direktorijuma u okviru operativnog sistema web servera zamenjena je organizacijom resursa u bazi podataka. Menadžer takve baze podataka predstavlja aplikativni server. Povezivanje web servera i aplikativnog servera se svodi na instalaciju aplikativnog servera na web serveru i konfigurisanje parametara web servera.

Protokol razmene podataka između web servera i aplikativnog servera je najčešće CGI standard, ali može biti i neki drugi (NSAPI - Netscape Application Programming Interface ili Microsoft ISAPI - Internet Server Application Programming Interface).

Pored statičkih resursa aplikativni server dozvoljava i dinamički sadržaj web stranica. Dinamička web stranica sadrži sledeće informacije:

- statički HTML deo,
- ugnježdene upite i programske instrukcije i
- podatke iz baze podataka koji su umetnuti kada se stranica formira.

Statički deo se zajedno sa upitima i instrukcijama naziva model ili *template*. Prilikom formiranja stranice upiti i instrukcije se izvršavaju od strane aplikativnog servera koji kombinuje rezultat upita sa statičkim HTML delom u konačni oblik koji se šalje web klijentu. Struktura realizacije web servera sa aplikativnim serverom data je na slici 3.3.



Slika 3.3. Realizacija web servera sa aplikativnim serverom

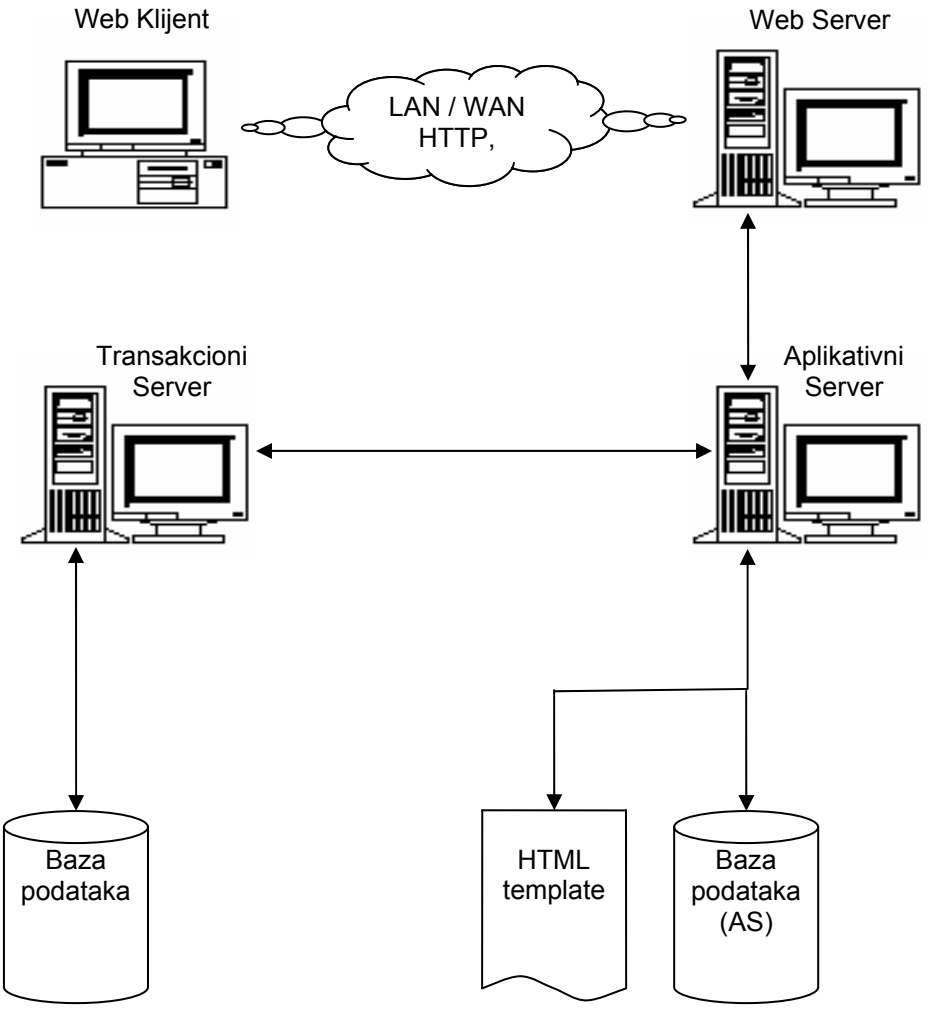
Ukoliko je potrebna veća funkcionalnost od prostog izdvajanja podataka dolazi se do potrebe za transakcionim serverom. Transakcioni serveri su programi koji izvršavaju *komponente* unapred definisanog oblika i ulazno / izlaznih sprega i preuzimaju rezultat njihovog izvršavanja i prosleđuju ga klijentu.

Transakcioni server sadrži komponente (objekte) realizovane različitim tehnologijama (Java, ActiveX, C++,...). Kada klijent pozove metodu određene komponente, transakcioni server nalazi jednu instancu komponente (ili je kreira), prosleđuje parametre komponenti i vraća klijentu rezultat. Dozvoljena je i definicija novih komponenti od strane korisnika. Pravila za pisanje novih komponenti i dozvoljeni programski jezici i alati su definisani od strane transakcionog servera.

Metodologija rada transakcionih servera i pojedinih komponenti se može pojednostavljeno uporediti sa respektivno web serverom i CGI skriptom.

Tako organizovana web prezentacija ima sledeći tok podataka od klijent upita do HTML stranice kao odgovora. Na HTTP upit od strane klijenta web server pokušava da pronađe odgovarajuću HTML stranicu od aplikativnog servera. Ukoliko HTML stranica sadrži neku dinamičku komponentu aplikativni server poziva *izvršenje* komponente od strane transakcionog servera. Rezultat izvršenja komponente

transakcioni server vraća aplikacionom serveru koji kombinuje rezultat sa statičkim delom u konačni oblik HTML stranice. Tako formirana HTML stranica se vraća web serveru koji je šalje web klijentu kao odgovor na HTTP upit. Struktura realizacije web servera sa aplikacionim i transakcionim serverom data je na slici 3.4.



Slika 3.4. Realizacija web servera sa aplikativnim i transakcionim serverom

3.3 Tipovi sprege

Osnovni problem povezivanja web servera i SCADA informacionog sistema se, primenom komercijalnih web servera, sveo na preuzimanje podataka od SCADA informacionog sistema putem CGI ili CTS. U zavisnosti od tipa veza otvorenih na SCADA serveru odnosno od vrste zapisa vrednosti procesnih veličina određene su i moguće veze sa SCADA serverom odnosno SCADA informacionim sistemom.

Generalno tipove veza ka SCADA sistemima možemo podeliti najjednostavnije u zavisnosti od izvora podataka. Izvorom podataka određena je i mogućnost izmene podataka ili samo pregleda kao i mogućnost preuzimanja vrednosti u realnom vremenu ili sa određenim kašnjenjem.

3.3.1 SCADA server - specijalizovani programi (TCP port ili protokol)

Najčešći oblik povezivanja sa SCADA sistemima je specijalizovanim programima koji se povezuju sa SCADA sistemom na određeni TCP port ili po nekom standardnom protokolu. Po pravilu se razmena i oblik podataka odvijaju protokolom koji je definisan od strane proizvođača SCADA sistema. Jedan od primera se može naći na GAUS SCADA sistemu gde LOK stanice sa SCADA serverom komuniciraju putem Netbios protokola sa korisnički definisanim datagramima. Ovim sistemom povezivanja obezbeđeno je preuzimanje vrednosti procesnih veličina u realnom vremenu kao i njihova promena (u zavisnosti od aplikativnog protokola).

3.3.2 Datotečni ili file server

File server kao sprega ka SCADA sistemu takođe često primenjena opcija. Razlog za to su pre svega dugogodišnja primena mrežnih operativnih sistema (npr. Novell) i njihova dokazana sigurnost u radu. Razmena podataka se odvija putem datoteke ili sistema datoteka (baze podataka) čiji sadržaj odgovara vrednostima procesnih veličina. Ovakva veza obezbeđuje pristup vrednostima procesnih veličina sa određenim kašnjenjem (potrebnim za zapis datoteka sa vrednostima). Izmena vrednosti procesnih veličina uglavnom nije podržana.

3.3.3 Baza podataka

Savremeniji oblik razmene putem file servera predstavljaju SQL serveri relacionih baza podataka. Zapis vrednosti procesnih veličina je u obliku relacionih baza podataka, dok su menadžeri baza podataka uglavnom SQL serveri. Kašnjenje zapisa vrednosti procesnih veličina je znatno manje uzimajući u obzir konkurentnost pristupa, sigurnost i validnost podataka kao i mrežno okruženje. Sa druge strane pored pregleda vrednosti procesnih veličina retko je dozvoljena i njihova izmena.

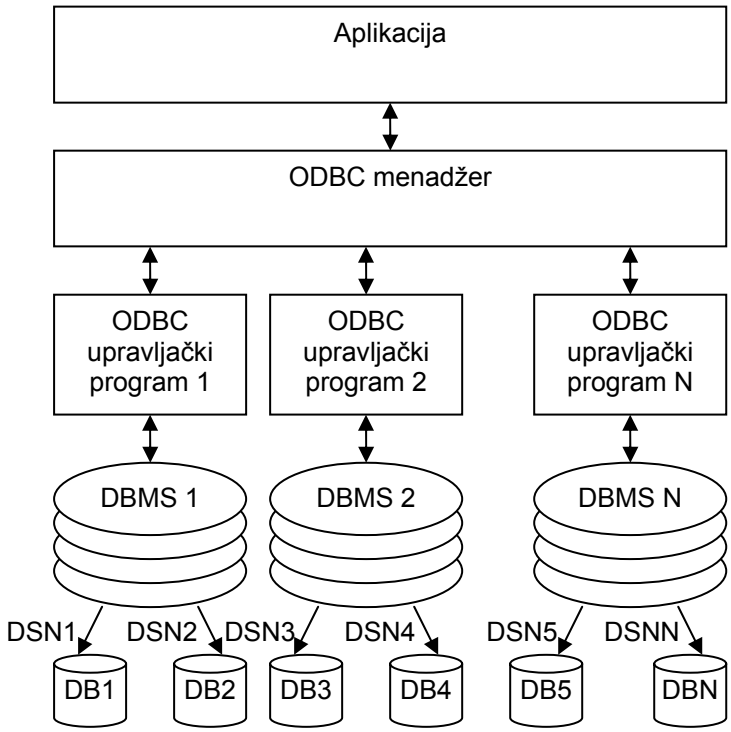
Standard primenjen za povezivanje na baze podataka pod Windows operativnim sistemom je Open Database Conectivity (ODBC) [16].

- ODBC pruža mogućnost komunikacije sa menadžerom baze podataka (DataBase managment system - DBMS) korišćenjem standardne programske sprege i SQL sintakse. Ukoliko DBMS podržava SQL, ODBC dozvoljava pristup DBMS-u preko SQL sprege. Ukoliko DBMS ne podržava SQL, opet je moguć rad sa njim, ako za njega postoji odgovarajući ODBC upravljački program (driver). ODBC se može koristiti u programu, ukoliko postoji ODBC upravljački program za DBMS, sa kojim se želi raditi. Ukoliko je program napisan, a javila se potreba da se koristi neki drugi DBMS, jednostavno se zameni ODBC upravljački program kao i sam DBMS.

Ceo koncept primene ODBC-a zasniva se na jedinstvenom načinu rada sa DBMS-om, bez obzira na njegovu konkretnu implementaciju.

Da bi se obezbedio API koji će raditi sa svim DBMS-ovima, definisan je određeni minimum karakteristika koje svaki ODBC upravljački program mora da poseduje (osnovni nivo - Core level). Kako bi se pored minimalnog nivoa saglasnosti, obezbijedila i podrška za složenije DBMS-ove, ODBC upravljački programi su definisani ili kategorizovani po nivoima saglasnosti (Conformance levels). Osnovni nivo je definisan specifikacijom CLI (Call Level Interface) od strane X/Open SQL Access grupe. Kada neki ODBC upravljački program odgovara nekom nivou saglasnosti, taj program mora imati svu funkcionalnost koja je definisana na tom nivou. Različiti nivoi podrške doveli su do toga da se u vreme rada programa može pozvati

ODBC funkcija za ispitivanje mogućnosti upravljačkog programa (postojanje podrške za određenu funkcionalnost ili ne). Komponente ODBC standarda i njihov odnos dati su na slici 3.5.



Slika 3.5. Struktura komponenti ODBC standarda

Osnovne komponente ODBC-a su:

- program - to je program koji je napisan u bilo kom jeziku visokog nivoa koji podržava DLL-ove, tj. biblioteke sa dinamičkim povezivanjem (Dynamic Link Libraries), u ovom slučaju program napisan u Visual C++.
- program za rad sa ODBC upravljačkim programima - to je Microsoftov DLL koji predstavlja vezu za komunikaciju sa DBMS-om. Sve funkcije ODBC API-ja koje se pozivaju su iz ovog DLL-a.
- ODBC upravljački program - to je komponenta koja radi sa DBMS-om, kada je program za rad sa upravljačkim programima uputio zahtev DBMS-u. Svaka informacija ili odgovor je vraćen programeru opet preko programa za rad sa ODBC upravljačkim programima. Da bi se moglo raditi sa konkretnim DBMS-om, potrebno je pre toga imati ODBC upravljački program
- izvor podataka (Data Source Name - DSN) - da bi ODBC mogao da radi sa bazom podataka, ona mora biti poznata programu za rad sa upravljačkim programima. To se obezbeđuje definisanjem imena izvora podataka. Pomoćni Microsoftov program za rad sa DSN-om je ODBC Data Source Administrator.

GAUS SCADA sistem podržava pristup relacionoj bazi podataka vrednosti procesnih veličina. Omogućena je ODBC konekcija na komercijalni Sybase SQL server.

3.3.4 OPC server

OPC (OLE for Process Control) [17] definiše industrijski standard baziran na Active X i OLE (Object Linking and Embedding) tehnologijama. Spomenute tehnologije omogućavaju interakciju između fizičkih komponenti različitog tipa koje mogu biti prostorno udaljene, ali i delovi istog sistema, između delova sistema zaduženih za realizaciju automatizacije (na primjer procesni kontroler - PK) i nadzorno - kontrolnog dela sistema (na primjer nadzorno-upravljačke stanice - NUS) kao i između celog akviziciono-upravljačkog sistema - AUS i poslovnog informacionog sistema.

U bliskoj prošlosti proizvođači opreme iz oblasti automatike kao i sami projektanti ovakvih sistema su razvili brojne načine sprezanja (interface) koji su bili specifični za dati proizvod odnosno uređaj. U jednom trenutku sprega kompanije Microsoft - DDE (Dynamic Data Exchange) je postala prihvaćena od strane velikog broja proizvođača razne opreme. Ova sprega je postala standard u mnogim granama automatizacije i upravljanja procesima. Možda najveću primenu dostiže u oblasti komunikacije PC upravljačkih aplikacija sa programabilnim kontrolerima (Programmable Logic Controllers - PLC). Ovako razvijeni uređaji, zasnovani na DDE-u ili sličnim spregama kao što je NetDDE koji su služili kao mehanizam za prosleđivanje podataka između različitih aplikacija, su imali dosta uspeha. Međutim, mnogi krajnji korisnici ovih uređaja i sistema su imali zamerke koje su se odnosile na ograničene performanse i mogućnost proširenja primene. Posebne zamerke su se odnosile na brzinu prenošenja podataka između fizičkih uređaja i upravljačkih aplikativnih programa.

Nakon nekoliko godina DDE sprega je zamenjena novom spregom - OLE koja je po svojim osobinama pružala više performanse, veću robusnost (mera osetljivosti sistema na značajnije promene parametara) i što je najvažnije mnogo sigurniji protok i razmenu podataka (data exchange). Bazirana na COM (Component Object Model) tehnologiji, OLE sprega je deo programske platforme koja je poznata pod nazivom Active X. Active X i DCOM (Distributed Component Object Model) su važni delovi strategije razvijene od strane kompanije Microsoft koja se zasniva na distribuiranoj klijent / server saradnji. Zahvaljajući ovako uspostavljenim pravilima definisanim u okviru spomenutih tehnologija, omogućena je međukooperativnost i komunikacija modula rasprostranjenih na različitim lokacijama računarske mreže.

Međutim, da bi prednosti Active X tehnologije bile iskorišćene na pravi način potrebno je da se od strane proizvođača opreme i projektanata sistema automatskog upravljanja usvoje standardna pravila i preporuke korišćenja Active X platforme. Shodno tome, nekoliko vodećih svetskih proizvođača u saradnji sa Microsoft-om su kreirali i ustanovili okvir koji definiše način primene Active X tehnologije u oblasti industrijske primene računara za upravljanje i kontrolisanje fizičkih procesa. Standard je nazvan OPC (OLE for Process Control) što u prevodu znači primena OLE sprege u procesnom upravljanju.

COM (Component Object Model) tehnologija predstavlja standard u komunikaciji između različitih komponenti. Uz pomoć COM-a omogućeno je da jedan programski modul koristi određena svojstva ili spregu (interface) druge aplikacije ili operativnog sistema da bi dobio informacije ili podatke. COM predstavlja osnovni standard na kome se zasnivaju DCOM, ActiveX i OLE technologija.

OLE (Object Linking and Embedding) omogućuje integraciju kroz aplikacije tako što obezbeđuje visok stepen kompatibilnosti među aplikacijama. Primenjuje se takođe i u razvoju objektne i ponovo korišćene (reusable) programske podrške gde pojedine komponente mogu biti pisane na različitim programskim jezicima

DCOM (Distributed Component Object Model) predstavlja proširenje COM tehnologije na mrežnu arhitekturu. To je visoko optimizovan protokol u kome se udaljene komponente ponašaju kao lokalne pa je ovakvim protokolom omogućeno da veliki programski i fizički sistemi kao što su SCADA ili HMI (Human-Machine Interface) prikupljaju i obrađuju podatke prikupljene od OPC servera koji se izvršavaju na bilo kom računaru u računarskoj mreži.

Standardom su definisane i mnoge druge prednosti kao što je na primer pristup procesnoj veličini sa OPC servera preko imena itd. Prilikom razvoja programske podrške kao i fizičkih komponenti inženjerima i programerima je potrebno da realizuju postavljene zahteve tako što će na osnovu specifikacije standarda realizovati OPC servere i OPC klijente. Nakon toga tako realizovani serveri i klijenti će međusobno razmenjivati podatke u realnom vermenu i to bez obzira o kakvim se sistemima radi (SCADA sistemi, PLC kontroleri, distribuirani ulazno - izlazni uređaji, distribuirani upravljački sistemi DCS itd)

Ovakav pristup eliminiše dugotrajni proces razvoja programske podrške za svaki tip fizičke komponente tako što definiše pravila na osnovu kojih se se ovakvi poslovi urade jednom (realizuju se OPC serveri i OPC klijenti) a zatim se lako koriste od strane, na primjer, SCADA ili bilo koje korisničke aplikacije koja upravlja odgovarajućim procesom.

GAUS SCADA sistem u okviru NUS-a ima i integrisani OPC server koji omogućava konekciju i preuzimanje vrednosti i statusa svih procesnih veličina (u realnom vremenu) preko imena procesne veličine. Realizovani OPC server ne dozvoljava izmenu podataka već samo njihov pregled.

3.4 Sigurnost SCADA sistema u intranet / internet okruženju

Otvorena priroda interneta zahteva ozbiljno razmatranje sigurnosti podataka prilikom realizacije SCADA sistema koji je dostupan putem interneta. Sa današnjim stepenom distribucije znanja i iskustava veoma je lako doći do specifičnih informacija vezanih za komercijalne SCADA sisteme.Preko 90% proizvođača SCADA sistema dokumentaciju i specifikaciju za svoje proizvode objavljuje na slobodan uvid putem internet prezentacija.

U prošlosti su ´oni´ sa destruktivnim idejama i namerama u vezi rušenja SCADA sistema bili bez znanja da to ostvare, a oni koji su to znali nisu imali nisu imali želje ni motivacije da ´atakuju´ na SCADA sisteme. Danas smo u situaciji da se motivacija i znanje sreću i predstavljaju realnu opasnost po SCADA informacione sisteme. Takođe, internet omogućava veoma brzo pronalaženje informacija i mogućnosti lakog pokušaja povezivanje na SCADA informacione sisteme (vidljivih na web-u).

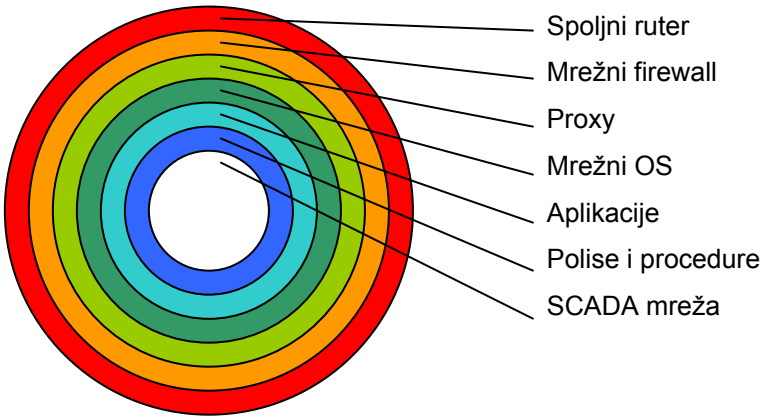
Sa druge strane SCADA sistemi koji su u prošlosti bili nestandardizovani u komunikacionim protokolima vidljivim ´spolja´ (poznati samo proizvođaču SCADA sistema) danas su u značajnoj meri standardizovani po pitanju komunikacionih protokola za povezivanje sa ´spoljnim´ aplikacijama.

Najveći broj proizvođača SCADA sistema podrazumevaju TCP/IP i Ethernet sprege i mnogi koriste TCP pakete za razmenu specifičnih podataka. Primena tehnologija kao što su VisualBasic, ODBC, OPC, ActiveX je takođe veoma prisutna. Koliko god da je primena standarda olakšala primenu i povezivanje SCADA sistema isto toliko je povećala problem bezbednosti i sigurnosti takvih sistema.

Zaštita ovakvih SCADA sistema se može podeliti na dva dela:

- tajnost podataka (neautorizovano čitanje podataka)
- bezbednost sistema (neautorizovana izmena podataka)

Pod zaštitom tajnosti podataka podrazumeva se zaštita od neautorizovanog pristupa SCADA informacionom sistemu odnosno prikupljenim vrednostima procesnih veličina, dok se pod zaštitom bezbednosti sistema podrazumeva zaštita pristupu rsursima i procedurama (programima) za upravljanje sistemom odnsno izmenama vrednosti procesnih veličina. Model zaštite [12] koji se najčešće može prepoznati izložen je u nastavku.



Slika 3.1. Struktura poslovnog (spoljnog) dela mreže

Jedan od najčešće korišćenih modela je, pre svega, razdvajanje poslovnog dela intranet mreže od dela SCADA intranet mreže. Time se problem zaštite deli na zaštitu poslovnog dela mreže (kao prvi sloj) i zaštitu SCADA mreže (kao drugi sloj).

Arhitektura ovakve mreže koja je dostupna internetom ima sledeću strukturu (slika 3.1.):

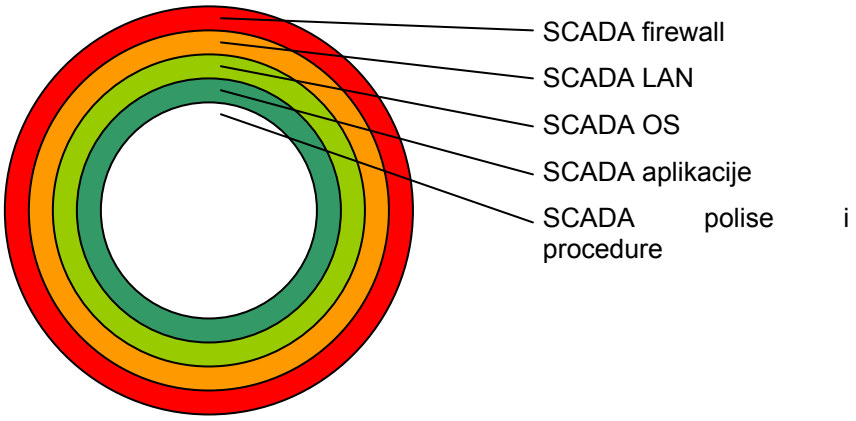
- Spoljni ruter (router)
- Mrežni firewall
- Proxy
- Mrežni operativni sistem
- Aplikacije
- Polise i procedure
- SCADA mreža

Spoljni ruteri i firewall računari, ukoliko su pravilno konfigurisani, uspešno mogu zaštititi lozinke, IP adrese, datoteke i sl. Međutim bez ozbiljnijeg (po pitanju organizacije korisnika - *multiuser*) operativnog sistema moguće je ostvariti pristup privatnim lokalnim mrežama što ih čini beskorisnim.

Proxy serveri su krucijalni za filtriranje i formiranje TCP paketa ka i od aplikativnog sloja (npr. HTTP ili SMTP servisi). Međutim uporeba proxy servera ne isključuje opasnost od ugrožavanja aplikativnog sloja.

Mrežni operativni sistemi mogu omogućiti, čak i prilikom pravilnog konfigurisanja i održavanja (*upgrade*), pristup mreži odmah po njenom aktiviranju. Izbor operativnog sistema mora težiti ka efikasnom i sigurnom i dokazanom na tržištu.

Primena efikasnih polisa i procedura obezbeđuje osnovu za dobru zaštitu. Međutim razvoj, dokumentacija i primena efikasnih zaštitnih polisa je jedna od najteže primenjivih mera. Jedino aktivan i konstantan razvoj sa praćenjem trendova na ovom polju ima mogućnosti da bude uspešan dugoročno gledano. Ovo, između ostalog, uključuje i izbor sigurnih lozinki, koje nisu uobičajene reči, imena ili poznati datumi, i obavezu periodičnih izmena lozinki.



Slika 3.2. Struktura SCADA (unutrašnjeg) dela mreže

Kako svaki od ovih slojeva ima svoje nedostatke (više ili manje) po pitanju sigurnosti, organizacija SCADA mreže takođe je slojevita (slika 3.2.):

- SCADA firewall
- SCADA lokalna mreža
- SCADA operativni sistem
- SCADA aplikacije
- SCADA polise i procedure

SCADA firewall serveri su prvenstveno namenjeni zaštiti od nenamrenog intranet pristupa podacima i eventualnog narušavanja sistema. Nisu retki primeri otkaza rada SCADA sistema usled slučajno obrisane datoteke ili pokrenutog programa u okviru SCADA mreže, i to sa računara iz poslovnog dela mreže. Sa druge strane to će obezbediti bar dva sloja firewall zaštite između SCADA mrže i interneta. Iz tih razloga čvrsto se preporučuje postojanje ovakve zaštite između poslovnog i SCADA informacionog sistema.

SCADA lokalna mreža treba da bude izdvojena u posebni IP segment. Takođe se preporučuje upotreba aktivnih dinamičkih mrežnih komponenti (*smart switches*), a ne statičkih (*hub*). Potrebno je obezbediti razdvajanje mrežnim maskama (*subnet mask*) akviziciono okruženje od uobičajenih korisničkih servisa (npr. pristup datoteci ili štampaču). Ukoliko se koristi bežični (*wireless*) pristup posebnu pažnju treba posvetiti izmeni podrazumevanih imena konekcija i korisnika i uključenju enkripcije (ukoliko je podržana upotrebljenom opremom).

SCADA operativni sistemi i dalje mogu biti meta neautorizovanog pristupa. Uz dosta veštine moguće je odrediti MAC adresu mrežne kartice i iskoristiti je za prolaz kroz mrežno okruženje do SCADA operativnog sistema. Stoga je potebno pravilno i stalno održavanje i pravilno konfigurisanje SCADA operativnog sistema. Za Windows NT, 2000 i XP to podrazumeva preuzimanje i instalaciju popravki (*patch, upgrade*) za operativni sistem kao i brisanje svih podrazumevanih korisničkih naloga. Slično administriranje je neophodno i za druge operativne sisteme (UNIX, LINUX, Novell, itd).

SCADA aplikacije takođe moraju sadržati interne metode zaštite. Virus programi poput *Trojan Horses* and *worms* mogu biti importovani u SCADA segment mreže i iskoristiti se za manipulaciju podacima ili izdavanje komandi. Takvi programi se mogu instalisati putem email-a ili web browser-om. To su razlozi da se strogo ograniči upotreba računara iz SCADA dela mreže na specijalizovane i proverene programa čija je svrha u okviru SCADA sistema.

SCADA polise i procedure su poslednja, ali ne i najmanje važna stavka u modelu zaštite od neautorizovanog pristupa SCADA sistemima. Pravilno administriranje, dodela lozinki i disciplina njihove tajnosti je bitan segment ovog sloja zaštite. Čest je slučaj da su lozinke zalepljene na monitoru ili da su identične za različite operatere.

Očigledno je da se zaštita SCADA sistema više ne završava na nivou mrežnih informacionih sistema već se proteže i u dubinu SCADA mrežnih segmeneta. Ovakva organizacija zaštite treba da omogući lakšu analizu sigurnosti sistema, a zatim i pravce razvoja svakog sloja pojedinačno. Svaki od slojeva treba detaljno projektovati i u potpunosti realizovati od strane odgovarajućih stručnjaka za informacione sisteme, mreže i operativne sisteme uz potpunu međusobnu kooperaciju. Takav pristup garantuje kvalitetnu zaštitu kako podataka tako i pouzdanog rada samog SCADA sistema.

4. REALIZACIJA HTTP SPREGE SA GAUS SCADA SISTEMOM

4.1 Koncept

U ovom rešenju http sprega je realizovana web aplikacijom sa CGI skriptom. Kao izvori podataka o stanju i vrednosti procesnih veličina realizovane su sprege sa file serverom, SQL serverom i OPC serverom koji predstavljaju sve dostupne sprege GAUS SCADA sistema. Izlazni podsistem web aplikacije realizovan je u obliku http stranica sa kombinacijom teksta i slike u GIF formatu koje su identične slikama prikazanim u grafičkim prikazima GAUS SCADA programa.

CGI skript je realizovan korišćenjem MS Developer Studio 6.0 programskog okruženja u programskom jeziku MS C/C++.

4.2 Preuzimanje podataka sa file servera

Preuzimanje podataka sa file server-a vrši se učitavanjem trenutnih vrednosti procesnih veličina iz datoteka namenjenih file terminalima. Prilikom preuzimanja vrednosti, iz datoteka namenjenih FTRM stanicama (IME.VAL), koje opisuju stanje sistema potrebno je da se one preuzimaju u skladu sa principima koji važe u programskoj podršci GAUS što znači da se prilikom pristupanja datotekama sa vrednostima mora čekati da datoteka bude slobodna za čitanje i kada se dobije dozvola datoteka se zaključava sa informacijama o preduzetoj akciji. Klasa ***CScadaValues*** sadrži metode neophodne za pristup datoteci sa vrednostima koje opisuju stanje sistema.

4.2.1 Klasa CScadaValues

Ova klasa sadrži nekoliko osnovnih metoda koje omogućuju pristup i učitavanje vrednosti procesnih pomenljivih (vrednosti koje opisuju stanje elemenata sistema). Metode ralizovane u ovoj klasi obezbeđuju pristup i učitavanje procesnih promenljivih u skladu sa pravilima preuzimanja istih. Kako realizacija programske podrške GAUS sistema podrazumeva da se procesne promenjlive ne učitavaju u posebnu strukturu nego su deo programske realizacije pojedinačnih elemenata pri realizaciji ove klase nema potrebe da se definišu posebni elementi (atributi) za čuvanje vrednosti procesnih promenljivih. Deklaracija klase ***CScadaValues*** data je ispod.

Osim konstruktora i destuktora parametar svih ostalih metoda predstavlja naziv datoteke koja sadrži vrednosti procesnih promenljivih. Atribut *CTRL_REC* koristi se pri zaključavanju datoteke sa vrenostima procesnih promenljivih.

Prilikom realizacije klase ***CScadaValues*** realizovane su metode za pristup i preuzimanje vrednosti procesnih promeljivih elemenata sistema iz datoteka koje sadrže vrednosti opisa stanja sistema i koje se koriste kao izvor podataka FTRM stanicama. Algoritam pristupa podacima (traži dozvolu, zaključaj datoteku, obradi podatke i oslobodi datotku) realizovan je u metodi ***ReadValues*** ove klase. Traženje dozvole i zaključavanje obavlja metoda ***Lock*** dok se obrada, u ovom slučaju čitanje datoteke, obavlja pozivanjem metode ***GetValues***. Da bi se onemogućio nepravilan pristup datotekama sa vrednostima procesnih promenljivih metode ***Lock*** i ***GetValues*** su privatne metode ove klase što vaši i za metodu ***UnlockCtrl*** koja vrši oslobađanje datoteke. Za čitanje vrednosti procesnih promenljivih iz datoteke mogu se koristiti metode ***ReadValues*** ili ***Load*** koje su realizovane tako da se datoteci sa vrednostima

procesnih promenljivih pristupa prema opisanom algoritmu. Metoda *Load* će formirati pun naziv datoteke sa vrednostima procesnih promenljivih na osnovu konfiguracionih podataka dok se metodi *ReadValues* preko parametra *pcFileName* mora dati potpuno ime datoteke iz koje se čitaju vrednosti procesnih promenljivih.

```
typedef struct ctrl_rec
{
BYTE sts;
WORD FTRMno;
} CTRL_REC;

class CScadaValues
{
public:
struct ctrl_rec CTRL_REC

CScadaValues();
~ CScadaValues ();

int GetValues ( char *pcFileName);
int UnlockCtrl (char *FileName);
int LockCtrl(char *FileName);
int ReadValues (char *pcFileName);
int Load (char *FileName);
};
```

Kontrola pristupa datoteci sa vrednostima procesnih promenljivih obavlje se korišćenjem kontrolne datoteke koja ima isto ime kao i datoteka kojoj se pristupa sa ekstenzijom .CTL. U kontrolnu datoteku se upisuje status tj. akcija koja se preduzima i brojač započetih akcija (odgovara polju *FTRMno*) se uvećava za 1. Da bi se sprečile greške pri upisivanju ovih vrednosti u kontrolnu datoteku pri pristupanju datoteci koristi se mehanizam zaključavanja datoteke tako da u jednom trenutku izmenu kontrolne datoteke može vršiti samo jedna programska nit i/ili program. Metoda *LockCtrl* vrši upisivanje započete akcije u kontrolnu datoteku što podrazumeva i zaključavanje kontrolne datoteke.

Pravila pristupa datoteci sa vrednostima procesnih promenljivih opisana u prethodnom pasusu primenjena su u realizaciji metode *LockCtrl*. Na početku ova metoda generiše ime kontrolne datoteke u skladu sa imenom datoteke sa vrednostima procesnih promenljivih i ekstenzijom .CTL. Zatim pokušava da otvori datoteku koja mora postojati u istom direktorijumu kao i datoteka sa vrednostima procesnih promenljivih (ekstenzija .VAL) i ako otvaranje nije uspelo onda se prijavljuje greška sa simboličkim nazivom *IDS_ERR_UNABLE_OPEN_FILE*. Ova greška može značiti da ne postoji kontrolna datoteka što bi moglo biti posledica loše konfiguracije sistema. U slučaju uspešnog otvaranja kontrolne datoteke zaključava se sadržaj otvorene datoteke i dalje se pristupa ispitivanju stanja datoteke. Ako je sadržaj kontrolne datoteke zaključan nije ga moguće menjati i povratna vrednost metode ima simboličku vrednost *CTRL_USED*. Izmena kontrolne datoteke u ovakvoj situaciji mogla bi uzrokovati nepravilnosti u rukovanju datotekom sa vrednostima procesnih promenljivih. Naime može se desiti da kontrolna datoteka u ovoj situaciji ne sadrži prave informacije o trenutnom stanju zahteva za korišćenjem datoteke sa vrednostima procesnih promenljivih. Uzmimo za primer da se u bliskim vremenskim trenucima pojave dva zahteva za obradu datoteke procesnih promenljivih i da oba zahteva preuzmu isto stanje broja zahteva čime bi došli u situaciju da sadržaj polja sa brojem zahteva u kontrolnoj datoteci pokazuje za jedan manju vrednost od stvarnog stanja. Druga nepravilnost koja se tiče tipa zahteva je ta da će u ovoj situaciji biti poništeno stanje koje upiše kao prvi zahtev i da bi se dozvolilo rukovanje zahtevu koji se

poslednji završi bez obzira na tip akcije koja će biti preduzeta po prvom zahtevu. Ako kontrolna datoteka nije zaključana tada se nastavlja sa algoritmom i učitava se sadržaj kontrolne datoteke i poredi se veličina učitanog sadržaja sa veličinom kontrolne strukture *CTRL_REC*. Takođe i u slučaju da veličina učitanog sadržaja ne odgovara veličini kontrolne strukture, treba prekinuti sa daljim izvršavanjem metode i vratiti kod greške koji u ovom slučaju ima simboličko ime *IDS_ERR_CTRL_REC*. Sledeća situacija koja se proverava je pozicioniranje na početak datoteke radi upisa novog sadržaja u kontrolnu datoteku. Ukoliko pozicija upisivanja nije početak datoteke upis u kontrolnu datoteku će narušiti strukturu kontrolne datoteke te ne treba preduzimati dalje akcije. U prethodne dve situacije pošto je kontrolna datoteka zaključana potrebno je datoteku otključati i tek onda prekinuti započetu akciju. Ako je sve u redu i nisu detektovane greške može se početi sa obradom kontrolne datoteke što podrazumeva sledeće: ako je datoteka vrednosti procesnih promenlivih slobodna (polje *CtrlRec.sts* ima vrednost *VALUEDAT_FREE*) potrebno je postaviti status. Kako je pri realizaciji odlučeno da CGI skript vrednostima procesnih promenljivih pristupa kao FTRM stanica to se onda za status postavlja vrednost *VALUEDAT_FTRM* i broj zahteva *CtrlRec.FTRMno* dobija vrednost 1 označavajući tako da samo jedna FTRM stanica zahteva vrednosti procesnih promenljivih. Ako već ima zahteva od FTRM stanice onda je potrebno samo uvećati brojač *CtrlRec.FTRMno* i kontrolni slog treba upisati u datoteku. Prilikom neuspešnog upisavanja metoda će otključati kontrolnu datoteku i prijaviti grešku sa simboličkom oznakom *IDS_ERR_CTRL_WRITE*. Ukoliko se akcije specificirane ovom metodom uspešno izvrše onda metoda ima povratnu vrednost simbolički označenu sa *OK*. Za sve ostale vrednosti statusnog polje kontrolnog sloga metoda vreća vrednost *CTRL_USED*.

Greške opisane u prethodnom pasusu mogu izazavati nepravilno rukovanje datotekom vrednosti procesnih promenljivih te na svaku od grešaka nastalih u ovoj metodi treba obratiti pažnju.

Metoda *UnlockCtrl* je dualna sa metodom *LockCtrl* i jedan deo njene realizacije je u potpunosti isti kao reaizacija metode *LockCtlr*. Razlika u odnosu na metodu *LockCtrl* je u obradi kontrolne datoteke, a sve provere prilikom otvaranja, učitavanja, zaključavanja i pozicioniranja su realizovane na isti način i po istom algoritmu. Razlika je u obradi i formiranju kontrolne datoteke i to u sledećem. Kada je učitan kontrolni slog i postavljen pokazivač na početak kontrolne datoteke ispituje se vrednost polja statusa *CtrlRec.sts* kontrolnog sloga.

Ako je vrednost ovog polja *VALUEDAT_FREE* onda polje *CtrlRec.FTRMno* dobija vrednost 0. Ovo nije neophodna akcija jer bi u ovom slučaju trebalo da ovo polje već ima vrednost 0, ako to nije slučaj onda ova akcija ispravlja grešku u kontrolnoj datoteci.

Ako je vrednost polja *CtrlRec.sts VALUEDAT_FTRM* ispituje se tip stanice koja pristupa datoteci vrednosti procesnih promenljivih i ako polje *stanica.tip* ima vrednost *FILETERM* onda se u slučaju da skript nije jedini koji pristupa datoteci vrednosti procesnih promenljivih i tada se vrednost polja *FTRMno* kontrolnog sloga umanjuje za jedan. Ako je posle ovog umanjena vrednost polja *FTRMno* 0 onda se mora promeniti i vrednost polja *sts* kontrolnog sloga na vrednost *VALUEDAT_FREE*. Kada vrednost polja *stanica* nije *FILETERM* oslobađa se datoteka vrednosti procesnih promenljivih postavljanjem sledećih vrednosti u kontrolni slog: *CtrlRec.sts* na *VALUEDAT_FREE* i *CtrlRec.FTRMno* na 0.

Kada je vrednost polja *status* jednako *VALUEDAT_TND* datoteka procesnih promenljivih se oslobađa (*CtrlRec.sts* dobija vrednost *VALUEDAT_FREE* i *CtrlRec.FTRMno* dobija vrednost 0).

Za vrednost *sts* polja jednaku simboličkoj vrednosti *VALUEDAT_TND* i stanicu tipa *SCADA* metoda oslobađa datoteku sa vrednostima procesnih promenljivih.

Ukoliko je vrednost polja *sts* kontrolnog sloga jedanka simboličkoj vrednosti *VALUEDAT_REQ* tada se primenjuje procedura kao i za vrednost *VALUEDAT_FTRM* jer je u ovom slučaju potrebno osloboditi datoteku simboličkih vrednosti procesnih promenljivih tj. potrebno je okončati sve zahteve tipa FTRM.

Greške koje mogu nastati u toku izvršavanja ove metode su identične greškama koje su opisane u opisu metode *LockCtrl* pa nije potrebno ponovo komentarisati uzroke i tipove nastalih grešaka. Parametar ove motode *pcFileName* predstavlja ime datoteke sa vrednostima procesnih promenljivih i ime kontrolne datoteke bez ekstenzije.

Za učitavanje vrednosti procesnih promenljivih koje opisuju stanje elemenata sistema pa time i samog sistema realizovana je metoda *GetValues*. Procesne promenljive se učitavanju iz datoteke vrednosti procesnih promenljivih sa imenom koje metoda dobija kao parametar *pcFileName* i ekstenzijom ".VAL" koja se podrazumeva. Rukovanje datotekom i greške koje mogu nastati prilikom otvaranja i čitanja datoteke su već opisivane pa nema potrebe da se ponovo komentarišu. Nakon uspešnog otvaranja i učitavanja datoteke sa vrednostima procesnih promenljivih proverava se da li su učitane odgovarajuće vrednosti procesnih promenljivih. Ova provera vrši se na osnovu identifikatora sistema koji se upoređuje sa identifikatorom iz datoteke koji je smešten na početku datoteke, ako su identifikatori različiti postoji greška u konfiguraciji sistema.

Datoteka vrednosti procesnih promenljivih je dalje organizovana na sledeći način: sledeći podatak u datoteci predstavlja veličinu sloga, u bajtima, koji sledi odmah iza veličine. Ovakva organizacija omogućava smeštanje slogova različite veličine u datoteku i kontrolu ispravnosti podataka smeštenih u datoteku. U slučaju greške koja može nastati pri čitanju slogova iz datoteke vrednosti procesnih promenljivih metoda vraća kod greške sa simboličkom oznakom *IDS_ERR_VAL* koja označava grešku u datoteci vrednosti procesnih promenljivih i tada treba prekinuti učitavanje vrednosti i proveriti konfiguraciju sistema. Ako je vrednost procesne promenljive za neki element uspešno učitana tj. učitan je slog iz datoteke, vrednost procesne promenljive smešta se u memoriju kao deo objekta koji predstavlja elemenat sistema. Svaki slog u datoteci vrednosti ima i kontrolnu reč koja se nalazi na kraju sloga pa se ova kontrolna reč upoređuje sa reči koja se dobija primenom algoritma za kreiranje kontrolne reči nad učitanim slogom. Ako ove reči nisu iste onda postoji greška u datoteci vrednosti procesnih promenljivih i važi napomena da treba prekinuti čitanje vrednosti i proveriti konfiguraciju sistema. Operacija učitavanja vrednosti se obavlja na sledeći način: pošto je iz sloga učitana struktura *pvid* koja sadrži podatke za identifikaciju procesnih promenljivih (tip, pripadnost i sekvenca) pozivaju se rutine iz biblioteke **Scadacom.dll** za formiranje pokazivača na vrednost procesne promenljive i na njen privremeni deo, a zatim se određuje dužina vrednosti i dužina privremenog dela procesne promenljive kako bi se iz sloga učitale dobre vrednosti za ove elemente, i na kraju se vrednost i privremeni deo upisuju na adresu dobijenu pozivom rutine *pid_pok*.

Metoda *ReadValues* realizuje algoritam za pristup i učitavanje vrednosti procesnih promenljivih koji podrazumeva čekanje na dozvolu za pristup vrednostima, zaključavanje podataka pri čitanju i na kraju oslobađanje datoteke sa vrednostima procesnih promenljivih. Metoda je realizovana tako da na svakih *RETRY_PERIOD* milisekundi pokušava da dobije pravo na korišćenja kontrolne datoteke pozivom metode *LockCtrl* ove klase. Podrazumeva se da je uvek moguće dobiti datoteku na korišćenje ili u suprotnom da postoji greška. Pozivanje metode *LockCtrl* se ponavlja

sve dok je rezultat poziva ove metode jednak vrednosti *CTRL_USED*. Kada je rezultat pomenutog poziva druga vrednost prestaje se sa pokušajima da se dobije datoteka na korišćenje jer je datoteka ili dobijena ili je došlo do greške u toku izvršavanja metode *LockCtrl*. Metoda dalje ispituje koji slučaj se desio ispitivanjem vraćene vrednosti. Ako je vraćena vrednost jednaka vrednosti *OK*, dobijena je dozvola pristupa datoteci i može se pozvati metoda *GetValues*. Posle uspešnog ili neuspešnog završetka metode *GetValues* treba osloboditi datoteku sa vrednostima procesnih promenljivih pozivanjem metode *UnlockCtrl* na isti način kako je pozivana metoda *LockCtrl*. Greška nastala u toku izvršavanja metode *UnlockCtrl* ima veći prioritet u odnosu na greške koje nastanu u toku čitanja tako da će greška koja nastane u toku čitanja biti vraćena jedino ako je datoteka uspešno otključana.

Umesto metode *ReadValues* preporučuje se korišćenje metode *Load* za čitanje vrednosti procesnih promenljivih. Ova metoda se u potpunosti oslanja na metodu *ReadValues*. Osim toga ova metoda formira pun naziv datoteke sa vrednostima procesnih promenljivih, a time i kontrolne datoteke na osnovu informacija iz konfiguracije sistema. Ako nije zadato ime datoteke onda se koristi podrazumevano ime datoteke vrednosti procesnih promenljivih i odgovarajuće kontrolne datoteke. Kada je formirano ime datoteka (bez ekstenzije) ova metoda poziva metodu *ReadValues*.

Dakle čitanje vrednosti procesnih procesnih promenljivih moguće je izvršiti i pozivom metode *ReadValues* s tim da se u tom slučaju mora zadati potpuno ime datoteke vrednosti procesnih promenljivih (bez ekstenzije).

Greške koje nastanu u toku izvršavanja metode *ReadValues* se samo prosleđuju kao povratna vrednost metode. Nakon uspešno izvršenog učitavanja procesnih promenljivih svaki predstavnik elementa sistema sadrži informacije o trenutnom stanju odgovarajućeg elementa sistema.

4.3 Preuzimanje podataka sa SQL servera

Radi preuzimanja podataka sa SQL servera realizovane su dve klase. *ODBCdatabase* za realizaciju veze sa upravljačkim programom baze podataka i *SQLResult* za preuzimanje eventualnih rezultata izvršene SQL komande.

4.3.1 Klasa ODBCdatabase

Ova klasa je realizovana tako da sadrži sve metode koje su neophodne za uspostavljanje i prekidanje veze sa izvorom podataka (DSN), kao i metodu za prijavljivanje grešaka koje su se pojavile u toku rada sa ODBC upravljačkim programom. Deklaracija klase *ODBCdatabase* data je ispod.

Konstruktor ove klase *ODBCdatabase*, alocira memoriju za atribut klase *pcErrorMsg* u kojoj se čuva poruka o grešci. Promenljivoj *bConnected*, koja predstavlja indikator da li je uspostavljena veza sa izvorom podataka ili ne, konstruktor dodeljuje vrednost FALSE, koja govori da veza sa izvorom podataka još nije ostvarena.

Metoda koja se po logičkom redosledu poziva prva iza konstruktora je *ConnectToDataSource*. Ulazni parametri ove metode su: *lpsDataSourceName* (ime izvora podataka na koji se treba povezati), *UserName* (korisničko ime koje treba navesti prilikom poveyivanja na izvor podataka) i *Password* (šifra koju treba navesti prilikom povezivanja).

Prva funkcija u okviru ove metode koja se poziva je ODBC funkcija *SQLAllocHandle*, kojom se u ovom slučaju kreira jedinstveni identifikator okruženja *henv*.

Nakon uspešno izvršenog kreiranja identifikatora okruženja, poziva se ODBC funkcija *SQLSetEnvAttr* kako bi se naznačila verzija ODBC sa kojim je aplikacija saglasna.

```
class ODBCdatabase
{
private:
SQLHENV henv;
SQLHDBC hdbc;
SQLRETURN  SQLreturn;

char acSQLState[MAX_STATE_LENGTH];
char* pcErrorMsg;
bool bConnected;

public:
ODBCdatabase();
~ODBCdatabase();
bool ConnectToDataSource (LPCSTR lpsDataSourceName, LPCSTR UserName, LPCSTR Password);
SQLResult* ExecuteSQLCommand(LPCSTR lpsSQLCommand);
bool HandlerRemover();
bool ErrorMessage(SQLHSTMT hstmt);

SQLHENV Gethenv()   {return henv;}
SQLHDBC Gethdbc()   {return hdbc;}
char* GetpcErrorMsg(){return pcErrorMsg;}
bool GetbConnected() {return bConnected;}
};
```

Ponovnim pozivanjem funkcije *SQLAllocHandle* zatim se kreira jedinstveni identifikator veze *hdbc*.

Ukoliko je uspešno kreiran i identifikator veze poziva se ODBC funkcija *SQLConnect*, koja uspostavlja vezu sa željenim izvorom podataka. Nakon uspostavljanja veze, promenljiva *bConnected* dobija vrednost TRUE. U ovom slučaju i izlazna vrednost funkcije *ConnectToDataSource* je TRUE.

Ukoliko je došlo do greške prilikom izvršavanja ODBC funkcija, dealociraju se do tad svi uspešno kreirani identifikatori. Dealokacija identifikatora se obavlja pozivom metode *HandlerRemover* koja je članica klase **ODBCdatabase**. Zatim se poziva metoda *ErrorMessage*, takođe članica klase, koja određuje tip greške koja se pojavila. Nakon toga prekida se izvršavanje metode *ConnectToDataSource*, čija izlazna vrednost u ovom slučaju je FALSE.

Sledeća metoda koja ide po nekom logičkom redosledu je *ExecuteSQLCommand*. Ulazni parametar, *lpsSQLCommand*, ove metode je SQL naredba koju treba izvršiti nad povezanom bazom podataka.

U ovoj funkciji prvo se kreira instanca klase *SQLResult*. Zatim se atributu te klase, *pODBCdatabase*, dodeljuje adresa objekta klase *ODBCdatabase* iz koga su kreirani identifikatori okruženja i veze. Nakon toga se pozivaju metode članice klase *SQLResult*. Prva metoda te klase koja se poziva je *AllocateStatementHandle* koja kreira identifikator naredbe. Ukoliko je ona uspešno izvršena poziva se metoda *ExecuteSQL* koja izvršava SQL naredbu. Ako je i ona uspešno izvršena, onda se proverava da li se radi o SELECT SQL naredbi. To proverava metoda *IsSQLCommandSelect*. Ukoliko se radi o SELECT naredbi poziva se funkcija *BindBafersAndColumns* koja rezultate upita, razvrstane po kolonama, povezuje sa

odgovarajućim baferima, atributima klase *SQLResult*. Nakon toga čita se prvi zapis seta rezultata postavljenog upita pozivanjem metode *ReadResults*. Pod čitanjem zapisa, podrazumeva se smeštanje zapisa seta rezultata u odgovarajuće bafere, atribute klase **SQLResult,** koji su već povezani sa kolonama rezultata. Sve metode navedene u ovom pasusu su članice klase **SQLResult**. Metoda *ExecuteSQLCommand* vraća pokazivač na kreiranu instancu klase **SQLResult**.

Metoda koja je zadužena za raskid veze sa izvorom podataka, kao i za uklanjanje identifikatora okruženja i veze je članica klase *ODBCdatabase*, *HandlerRemover*.

Prvo se vrši provera da li identifikatori okruženja i veze uopšte postoje. Ako postoje onda se oslobađaju pozivanjem ODBC funkcije *SQLFreeHandle*, za svaki identifikator posebno. Raskid veze sa izvorom podataka ostvaruje se pomoću ODBC funkcije *SQLDisconnect*. Ukoliko je raskid veze uspešno obavljen atribut klase, *bConnected* dobija vrednost FALSE.

Ukoliko nije došlo do greške prilikom pozivanja ODBC funkcija, funkcija *HandlerRemover* vraća vrednost TRUE. U protivnom se poziva meoda *ErrorMessage* koja utvrđuje tip greške, a izlazna vrednost tada je FALSE.

Metoda koja određuje tip greške koji se pojavio u programu prilikom izvršavanja ODBC funkcija je *ErrorMessage*. Ulazni parametar, *hstmt*, ove metode je identifikator naredbe.

ODBC funkcija koja se poziva, u cilju određivanja tipa greške koja se pojavila, je *SQLError*.

Atribut klase *acSQLState* dobija vrednost šifre SQL stanja, a u *pcErrorMsg* kratak opis greške koja se pojavila u programu. Funkcija *ErrorMessage* ne vraća vrednost.

Metode *Gethenv*, *Gethdbc*, *GetpcErrorMsg* i *GetbConnected* su realizovane kao *inline* funkcije koje obezbeđuju pristup članicama klase, *henv*, *hdbc*, *pcErrorMsg* i *bConnected* respektivno. Inline funkcije obezbeđuju pristup članicama klase iz privatne sekcije, ukoliko se njima pristupa iz neke druge klase. Ove funkcije ne omogućavaju promenu sadržaja promenljivih iz privatne sekcije.

Destruktor klase *ODBCdatabase* poziva metodu *HandlerRemover* koji raskida vezu sa izvorom podataka ukoliko ona još postoji i dealocira identifikatore okruženja i veze. U destruktoru se takođe dealocira i memorija koja je zauzeta za atribut klase *pcErrorMsg*.

4.3.2 Klasa SQLResult

Klasa *SQLResult* sadrži metodu za izvršavanje SQL naredbe, metodu za proveru da li se radi o SELECT SQL naredbi, kao i metode za manipulisanje rezultatima upita ukoliko ih ima. Deklaracija klase *SQLResult* data je ispod.

U deklaraciji klase *SQLResult*, navedena je kao *friend* klasa *ODBCdatabase*. Ovim je omogućeno da u klasi *SQLResult* kao atribut bude i objekat klase *ODBCdatabase* koja još nije definisana.

U klasi su definisana dva konstruktora. Jedan konstruktor ima, kao ulazni podatak, *lpsSQL*, SQL naredbu koja treba da se izvrši. Drugi konstruktor nema ulaznih podataka i on se koristi za kreiranje objekta klase *SQLResult* kada SQL naredba nije unapred poznata. U tome je i jedina razlika između ova dva konstruktora. I u jednom i drugom vrši se inicijalizacija istih atributa klase.

Atribut klase, *bNoData* je indikator da nema podataka za čitanje, pa se zato u konstruktoru njegova vrednost postavlja na TRUE.

Indikator *bBound* signalizira da li je uspostavljena veza između kolona podataka i odgovarajućih bafera u koji će se smeštati rezultati upita. Njegova inicijalna vrednost je FALSE.

Indikator koji signalizira da nema greške u programu je *bNoError*. Njegova inicijalna vrednost je TRUE.

lpsSQLCommand je bafer u koji se smešta SQL upit. Ukoliko SQL upit nije unapred poznat, u konstruktoru on dobija vrednost NULL. Međutim ukoliko je poznat, on se kao ulazni parametar prosleđuje konstruktoru i njegova vrednost se dodeljuje članici klase *lpsSQLCommand*.

```
class SQLResult
{
friend class ODBCdatabase;

private:
bool bBound;
bool bNoData;
bool bNoError;

SQLRETURN  SQLreturn;
LPCSTR lpsSQLCommand;
SQLHSTMT hstmt;
short int* psiNumberOfColumns;
unsigned char** ppcColumnsNames;
char** ppcColumnsDataTypes;
char** ppcColumnsData;
DWORD** ppDColumnsDataSize;
ODBCdatabase* pODBCdatabase;
char* pcErrorMsg;

char DetermineType(SQLSMALLINT* pDataType);
bool SQLStatementFree();

public:
SQLResult();
SQLResult(LPCSTR lpsSQL);
~SQLResult();
bool AllocateStatementHandle();
bool ExecuteSQL();
bool IsSQLCommandSelect();
bool BindBuffersAndColumns();
bool ReadResults();
void ErrorMessage();

bool GetbNoData()                    {return bNoData;}
bool GetbNoError()                   {return bNoError;}
SQLHSTMT Gethstmt()                  {return hstmt;}
short int *GetpsiNumberOfColumns()       {return psiNumberOfColumns;}
unsigned char** GetppcColumnsNames()         {return ppcColumnsNames;}
char** GetppcColumnsDataTypes()      {return ppcColumnsDataTypes;}
char** GetppcColumnsData()           {return ppcColumnsData;}
char* GetpcErrorMsg()                {return pcErrorMsg;}
DWORD** GetppDColumnsDataSize()      {return ppDColumnsDataSize;}
};
```

U konstruktoru se inicijalizuju i sledeći atributi klase:

ppcColumnsNames - pokazivač na niz bafera u kojima će se čuvati imena kolona podataka.

ppcColumnsDataTypes - pokazivač na niz bafera u kojima će se čuvati tipovi podataka kolona.

ppcColumnsData - pokazivač na niz bafera u kojima će se čuvati podaci iz kolona podataka.

ppDColumnsDataSize - pokazivač na niz bafera u kojima će se čuvati veličina podataka kolone.

U konstruktoru se definiše i veličina bafera u koji će se smeštati poruke o grešci. U ovom slučaju veličina bafera je MAX_ERROR_MESSAGE_LENGTH bajta, gde je MAX_ERROR_MESSAGE_LENGTH konstanta i njena vriednost je 256. Ime ovog bafera, koji je članica klase, jeste *pcErrorMsg*.

Posle konstruktora prva metoda koja se poziva jeste *AllocateStatementHandle*. Ova metoda nema ulaznih parametara. Njena uloga je da pozove ODBC funkciju *SQLAllocHandle* i prosledi joj identifikator veze, kako bi se kreirao identifikator naredbe, *hstmt*.

Ukoliko je identifikator uspešno kreiran, metoda *AllocateStatementHandle* vraća vrednost TRUE. Ukoliko je došlo do greške prilikom izvršavanja ODBC funkcije poziva se metoda *ErrorMessage*, koja je takođe članica klase *SQLResult*, i određuje se tip greške. U ovom slučaju povratna vrednost je FALSE.

Metoda *ExecuteSQL* je zadužena za izvršavanje SQL upita. Ova metoda nema ulaznih parametara. Ona poziva ODBC funkciju *SQLExecDirect* i prosleđuje joj, indikator naredbe, *hstmt,* i samu SQL naredbu, *lpsSQLCommand*, kako bi se izvršio SQL upit. Ukoliko je SQL upit uspešno izvršen povratna vrednost funkcije je TRUE. U protivnom je FALSE.

Metoda *IsSQLCommandSelect* proverava da li se radi o SELECT SQL upitu koji vraća rezultate ili ne. Ova metoda nema ulaznih parametara. Pozivanjem ODBC funkcije *SQLNumResultCols*, sa prosleđenim identifikatorom okruženja *hstmt* i atributom klase *psiNumberOfColumns*, određuje se broj kolona podataka, rezultata.

Broj kolona podataka se zapisuje u atribut klase *psiNumberOfColumns*. Ukoliko je broj veći od nule, to je znak da se radi o SELECT SQL upitu i izlazna vrednost metode je TRUE. U protivnom radi se o SQL naredbi koja menja sadržaj baze podataka i u tom slučaju je povratna vrednost FALSE. Povratna vrednost je FALSE i ukoliko je došlo do greške prilikom izvršavanja ODBC funkcije *SQLNumResultCols*.

Metoda *BindBafersAndColumns* nema ulaznih podataka i njen zadatak je da poveže kolone podataka rezultata SQL upita, sa odgovarajućim članicama klase u koje će se upisivati ti podaci.

Prvo se proverava da li je broj kolona podataka, *psiNumberOfColumns*, koji je određen metodom *IsSQLCommandSelect*, veći od nule.

Ukoliko je broj kolona jednak nuli, postavljeni upit ne vraća nikakve rezultate, pa indikator, *bNoData* koji ukazuje na to da nema podataka za čitanje, ima vrednost TRUE. U ovom slučaju povratna vrednost metode je FALSE.

Međutim, ukoliko je broj kolona veći od nule, postavljeni upit vraća rezultate. U ovom slučaju prvo se inicijalizuju lokalne promenljive koje će kao parametri biti prosleđeni ODBC funkciji *SQLDescribeCol*. Ova funkcija određuje ime kolone podataka, SQL tip podataka koje podržava, veličinu kolone podataka, broj decimalnih mesta koja kolona koristi za predstavljanje realnih brojeva i indikator da li kolona podataka dozvoljava mogućnost da njena vrednost bude NULL. Promenljive koje se inicijalizuju i prosleđuju *SQLDescribeCol* funkciji su:

siBaferLength tipa short int, čija inicijalna vrednost je 512. Ova vrednost određuje veličinu bafera u kome se čuva ime kolone podataka.

pcColumnName je pokazivač na bafer tipa unsigned char u koji se smešta ime kolone podataka.

psiNameLengthPtr je pokazivač na bafer, tipa short int, u koji se smešta maksimalan broj bajtova koji se dostupni za smeštanje u **pcColumnName*.

psiDataTypePtr je pokazivač na bafer, tipa short int, u koji se smeštaju SQL tipovi podataka u kolonama.

pdColumnSizePtr je pokazivač na bafer, tipa DWORD, u koji se upisuje veličina kolone podataka.

psiDecimalDigitsPtr je pokazivač na bafer, tipa short int, u koji se smešta broj decimalnih mjesta koja kolona koristi za predstavljanje realnih brojeva.

psiNullablePtr je pokazivač na bafer, tipa short int, u koji se smešta indikator da li kolona dozvoljava NULL vrednost ili ne.

Funkcija *SQLDescribeCol* se poziva onoliko puta koliko ima kolona podataka. Ukoliko je funkcija uspešno izvršena, vrednosti gore navedenih promenljivih se dodeljuju odgovarajućim atributima klase.

Atributi klase *ppcColumnsNames*, *ppcColumnsDataTypes*, *ppDColumnsDataSize* i *ppcColumnsData* imaju onoliko pokazivača na bafere koliko ima kolona podataka. U baferima na koje pokazuje *ppcColumnNames* čuvaju se imena kolona podataka, a u baferima na koje pokazuju, *ppcColumnsDataTypes* *ppDColumnsDataSize* i *ppcColumnsData* čuvaju se tipovi podataka, veličina podataka i podaci, respektivno.

Ukoliko je došlo do greške prilikom izvršavanja funkcije *SQLDescribeCol*, poziva se metoda *ErrorMessage*, koja određuje tip greške i prekida se izvršavanje funkcije *BindBafersAndColumns*, čija je povratna vrednost u ovom slučaju FALSE.

Povezivanje bafera, u koji se smeštaju podaci iz kolona i njihove veličine, sa kolonama podataka obavlja se pomoću ODBC funkcije *SQLBindCol*.

I ova funkcija se poziva onoliko puta koliko ima kolona, kako bi se svaka kolona uspešno vezala za nju predviđen bafer.

Ukoliko je povezivanje uspelo, indikator uspostavljanja veze između kolona podataka i odgovarajućih bafera, *bBound* ima vrednost TRUE. Indikator koji ukazuje na to da nema podataka za čitanje, *bNoData*, sad ima vrednost FALSE, a indikator nepostojanja greške, *bNoError*, ima vrednost TRUE. Povratna vrednost funkcije *BindBafersAndColumns* u ovom slučaju je TRUE.

Ako je došlo do greške prilikom izvršavanja funkcije *SQLBindCol*, *bBound* ima vrednost FALSE, a *bNoData* TRUE. Povratna vrednost *BindBafersAndColumns*, tada je FALSE.

Prilikom izlaska iz metode *BindBafersAndColumns*, svejedno da li je izvršena uspešno ili neuspešno, vrši se oslobađanje svih lokalnih promenljivih koje predstavljaju pokazivače.

Metoda *ReadResults* čita jedan zapis seta rezultata, koji se nalazi u kolonama podataka i upisuje ga u bafere koji su u metodi *BindBafersAndColumns*, povezani sa tim kolonama.

ODBC funkcija koja upisuje podatke u bafere je *SQLFetch*. Parametar koji se prosleđuje ovoj funkciji je indikator naredbe, *hstmt*.

Ukoliko je *SQLFetch* funkcija uspešno izvršena, prekida se izvršavanje metode *ReadResult*. U tom slučaju povratna vrednost je TRUE.

Ako je došlo do greške prilikom izvršavanja *SQLFetch* funkcije, onda se utvrđuje razlog njene pojave. Prvo se proverava vrednost atributa, *SQLReturn* u koju ODBC funkcije zapisuju svoje povratne vrednosti. Ukoliko je njena vrednost SQL_NO_DATA, to znači da više nema zapisa rezultata za čitanje. Ako *SQLReturn* ima neku drugu vrednost poziva se metoda *ErrorMessage* i izlazi se iz metode sa FALSE povratnom vrednošću.

Ukoliko je problem u nedostatku podataka za čitanje, proverava se da li je indikator nepostojanja podataka za čitanje, ***bNoData*** aktiviran. Ukoliko nije, tj. njegova vrednost je FALSE, onda se ona postavlja na TRUE. U tom slučaju indikator pojave greške, ***bNoError*** postavlja se na FALSE, a u bafer, koji čuva poruke o grešci, ***pcErrorMsg*** zapisuje se poruka o nedostatku podataka za čitanje. Izvršavanje metode ***ReadResult*** se prekida i njena povratna vrednost je FALSE.

Ukoliko je indikator nepostojanja podataka već aktiviran, to znači da je korisnik već primio obaveštenje o tome da nema podataka za čitanje i da želi, ponovno izvršavanje istog SQL upita. Izvršavanjem ponovljenog SQL upita omogućava se ponovni pregled zapisa seta rezultata i to iz početka. Da bi se omogućilo ponovno izvršavanje SQL upita, neophodno je poništiti identifikator naredbe, ***hstmt***, pozivanjem ODBC funkcije ***SQLCancel***.

Ako je ova funkcija uspešno izvršena onda se poziva metoda klase ***ExecuteSQL*** koja je zadužena za izvršavanje SQL upita. Ukoliko je ona uspešno izvršena, u ***bNoData*** se upisuje FALSE vrednost, a u ***bNoError*** TRUE. Zatim metoda ***ReadResult*** poziva samu sebe kako bi se odgovarajući baferi napunili podacima. U ovom slučaju povratna vrednost ***ReadResult*** metode je TRUE.

Metoda ***DetermineType*** je zadužena za utvrđivanje tipa podataka kolone. Ulazni parametar je ***pDataType*** koji predstavlja SQL tip podataka kolone. SQL tipovi podataka koji su podržani u ovoj klasi su: SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_DATE, SQL_INTEGER i SQL_SMALLINT. Na osnovu utvrđenog tipa podataka kolone, ova metoda vraća odgovarajući karakter koji predstavlja skraćenicu SQL tipa podataka. Za SQL_CHAR, SQL_VARCHAR i SQL_LONGVARCHAR skraćenica je C. Za SQL_DATE je D, a za SQL_INTEGER i SQL_SMALLINT je I. Ukoliko se ne radi ni o jednom od navedenih tipova onda je povratna vrednost N.

ErrorMessage je metoda koja je zadužena za utvrđivanje tipa greške koji se pojavio prilikom izvršavanja metoda ove klase. Nema ulaznih parametara.

Metoda poziva istoimenu metodu iz klase ***ODBCdatabase*** koja je već obrađena. Ona se poziva preko atributa, ***pODBCdatabase*** koja je objekat klase ***ODBCdatabase***. Rezultati metode se smeštaju u bafer, ***pcErrorMsg*** koji je predviđen za čuvanje poruka o greškama. Indikator ***bNoError*** tada dobija vrednost FALSE.

Metoda ***SQLStatementFree*** je zadužena za oslobađanje identifikatora naredbe. Nema ulaznih parametara.

Prvo se proverava postojanje veze sa izvorom podataka. Ukoliko postoji onda se proverava i to da li su baferi povezani sa kolonama podataka. Ukoliko su povezani, poziva se ODBC funkcija ***SQLFreeStmt***, sa prosleđenim identifikatorom naredbe i zahtevom za prekidanje veza između bafera i kolona.

Ako je ova funkcija uspešno izvršena indikator povezanosti kolona i bafera, ***bBound*** poprima vrednost FALSE. Zatim se poziva ODBC funkcija *SQLFreeHandle* kako bi se oslobodio identifikator naredbe, ***hstmt***.

Ova metoda se poziva i ukoliko baferi i kolone podataka nisu bili povezani.

Ukoliko su pozvane ODBC funkcije uspešno izvršene onda je povratna vrednost metode ***SQLStatementFree*** TRUE. U protivnom poziva se metoda za utvrđivanje greške i izlazna vrednost je FALSE.

Destruktor klase ***SQLResult*** poziva metodu ***SQLStatementFree***, kako bi se oslobodio identifikator naredbe, ***hstmt***. Zatim se iz memorije uklanjaju i sledeći atributi klase: ***psiNumberOfColumns***, ***ppcColumnsNames***, ***ppcColumnsDataTypes***, ***ppcColumnsData***, ***ppDColumnsDataSize*** i ***pcErrorMsg***.

4.4 Preuzimanje podataka sa OPC servera

Povezivane sa OPC serverom i preuzimanje podataka je realizovano primenom komercijalno dostupne DLL biblioteke *wtclient.dll*. Realizovana je klasa ***COPCClient*** koja objedinjuje povezivanje sa pomenutom bibliotekom.

4.4.1 Klasa COPCClient

Ova klasa sadrži metode koje određuju rad OPC klijenta. Deklaracija klase ***COPCClient*** data je ispod.

```
class COPCClient
{
public:
COPCClient();
~COPCClient();

HANDLE hConnection;
HANDLE hGroup;
CObArray TagList;

BOOL Init_OPC_Clt();
void Uninit_OPC_Clt();

int Number_Of_OPC_Servers(BOOL UseOPCNUM, LPCSTR MachineName);
BOOL Get_OPC_Server_Name(int index, char* Buffer, int BuffSize);
BOOL Get_OPC_Item_Name(HANDLE hConnect,int index,char*pBuf,int BuffSize);
int Number_Of_OPC_Items(HANDLE hConnect);

HANDLE Connect_OPC(LPCSTR MachineName, LPCSTR ServerName, BOOL EnableDLLBuffering);

void Disconnect_OPC_Clt(HANDLE hConnect);
HANDLE Add_OPC_Item(HANDLE hConnect,HANDLE hGroup,LPCSTR ItemName);

HANDLE Add_OPC_Group(HANDLE hConnect, LPCSTR Name, DWORD *pRate, float *pDeadBand);

BOOL Enable_OPC_Notification(HANDLE hConnect, NOTIFYPROCAPI lpCallback);

BOOL Read_OPC_Item(HANDLE hConnect,  HANDLE hGroup, HANDLE hItem, VARIANT *pVar,
FILETIME *pTimeStamp, DWORD *pQuality);

Number_Of_Item_Properties(HANDLE hConnect, char* pcItemName);

BOOL Get_Item_Property_Description(HANDLE hConnect, int PropertyIndex, DWORD *pPropertyID,
VARTYPE *pVT, BYTE *pDescr, int BufSize);

BOOL Read_Property_Value(HANDLE hConnect, char* ItemName, DWORD ulPropertyID, VARIANT
*pValue);
};
```

Sve funkcije koje se pozivaju u okviru metoda ove klase su funkcije dinamičke biblioteke Wtclient.dll.

U konstruktoru klase ***COPCClient*** poziva se metoda klase ***Init_OPC_Clt*** koja služi za inicijalizaciju klijenta.

Metoda ***Init_OPC_Clt*** koja inicijalizuje OPC klijenta, ustvari inicijalizuje DCOM i OPC spregu na način koji obezbeđuje siguran rad. Pri tom poziva funkciju dinamičke biblioteke Wtclient.dll, ***WTclientCoIni***.

Povratna vrednost metode ***Init_OPC_Clt*** je tipa BOOL i ona predstavlja povratnu vrednost funkcije *WTclientCoIni*.

Connect_OPC poziva funkciju *ConnectOPC* kojoj prosleđuje svoje ulazne parametre, *MachineName* (ime računara na kome se OPC server izvršava), *ServerName* (ime OPC servera) i *EnableDLLBafering*, kako bi se izvršilo povezivanje sa OPC serverom. Metoda vraća identifikator veze sa serverom.

Number_Of_OPC_Items određuje broj procesnih veličina na datom OPC serveru. Ulazna vrednost ove metode je *hConnect* (identifikator veze), a povratna vrednost je broj raspoloživih procesnih veličina. U okviru metode *Number_Of_OPC_Items* poziva se funkcija *NumberOfOPCItems* koja i određuje broj procesnih veličina.

Metoda *Get_OPC_Item_Name* sa ulaznim parametrima *hConnect* (identifikator veze), *index* (redni broj procesne veličine čija vrednost je zahtevana), *pBuf* (bafer u koji se smešta ime procesne veličine) i *BuffSize* (veličina bafera), nakon poziva funkcije *GetOPCItemName*, određuje ime procesne veličine.

Number_Of_Item_Properties je metoda, čiji su ulazni parametri *hConnect* (identifikator naredbe) i *pcItemName* (ime procesne veličine) poziva funkciju *NumberOfItemProperties*, kako bi odredila broj karakteristika željene procesne veličine. Povratna vrednost ove metode je broj karakteristika.

Metoda *Get_Item_Property_Description* utvrđuje karakteristike željene procesne veličine. Ulazni parametri su: *hConnect* (identifikator veze), *PropertyIndex* (redni broj parametra procesne veličine), *pPropertyID* (ID parametra), *pVT*, *pDescr* (bafer u koji se smešta opis parametra) i *BufSize* (veličina bafera). U okviru ove metode poziva se funkcija *GetItemPropertyDescription*. Povratna vrednost ove metode je tipa BOOL.

Read_Property_Value sa ulaznim parametrima, *hConnect* (identifikator veze), *ItemName* (ime procesne veličine), *ulPropertyID* (ID parametra procesne veličine) i *pValue* (bafer u koji se smešta vrednost procesne veličine) poziva funkciju *ReadPropertyValue* kako bi se odredila trenutna vrednost procesne veličine. Povratna vrednost je tipa BOOL.

Disconnect_OPC_Clt je metoda koja služi za raskidanje veze sa OPC serverom. U okviru nje poziva se funkcija *DisconnectOPC* sa parametrom *hConnect* (identifikator veze).

Metoda *Uninit_OPC_Clt*, deinicijalizuje OPC klijent, a pri tom poziva funkciju *coUninitialize*. Ova metoda nema povratnu vrednost.

U destruktoru klase *COPCClient* se vrši raskidanje veze sa OPC servera, pozivanjem metode *Disconnect_OPC_Clt*, i dealokacija OPC klijenta pozivanjem metode *Uninit_OPC_Clt*.

4.5 Formiranje GIF slike SCADA grafičkog prikaza

Pristup grafičkom prikazu sistema, njegovo iscrtavanje i na kraju preuzimanje stanja elemenata sistema realizovano je pomoću dve klase. Učitavanje konfiguracije (grafičke i alfanumeričke) i iscrtavanje šema je realizovano pomoću klase *CShemaWeb*, a konverzija iscrtane šeme u GIF sliku pomoću klase *CGIF*. Radi dobijanja grafičkog prikaza sistema koji je isti ili sličan grafičkom prikazu dobijenom alatima realizovanim u GAUS sistemu, polazi se od ideje da se funkcije za iscrtavanje elemenata pa i funkcije za preuzimanje stanja elemenata ne realizuju ponovo već da se koriste postojeće, realizovane u okviru GAUS sistema. Sledeći zadatak koji se nameće zbog prirode i mesta izvršavanja CGI skripta je da se funkcije za iscrtavanje elemenata sistema koje su realizovane za jedno okruženje (Windows grafička aplikacija) sada upotrebe u drugačijem okruženju (Windows konzolna aplikacija). Zbog ovih zahteva

neke od funkcija nisu mogle biti direktno primenjene već je bilo potrebno izvršiti njihovu izmenu kako bi se uskladile sa novim okruženjem.

4.5.1 Klasa CGIF

Klasa sadrži metode i atribute potrebne za kompresiju grafike LZW algoritmom koji je podrazumenvani algoritam za GIF format grafike i realizovana je tako da omogući stvaranje GIF grafike na osnovu bitmape. Takođe jedan od zadataka je da se GIF grafika prosledi u odgovarajućem formatu na željeni uređaj. Deklaracija klase *CGIF* data je ispod.

```
class CGIF
{
private:
Byte *pPixelBytes;
Byte byBitsPerPixel;
Byte byBytePixelsLeft;
DWord dwCurrentByte;
DWord dwBytesNum;
Word wNewCode;
Byte Buffer[256];
Word wBufIndex;
Byte byLeftToFit;
Byte byToDWORD;
Word x,yByteOff, CurrentPix;
DWord HashTab [HSIZE];
Word CodeTab [HSIZE];
CFile *pFile;
GIFScreenDescriptor ScreenD;
GIFImageDescriptor ImageD;

Word AddCode(Word wPrefix, Word c);
Word WriteRest(char *pcFile);
Word WriteCode(Word wCode, Byte byNumBits, char *pcFile);
void InitBuffer();
Word Compress (Byte byCodeSize, char *pcFile);
Word ReadPixel ();
Word FindString(Word wPrefix, Word c);

public:
CGIF();
~CGIF();

Word Save (CDC *pDC, LPSTR lpFileName);
};
```

Atribut *pPixelBytes* se koristi kao pokazivač na prvi bajt bitmapirane slike i koristi se za pristip pikselima bitmape kao bazna adresa slike. Ovaj atribut zajedno sa atributima *dwCurrentByte*, *dwBytesNum*, *CurrentPix*, *byBytePixelsLeft*, *byBitsPerPixel*, *x* i *yByteOff* omogućava čitanje tekućeg piksela iz bitmape korišćenjem metode *ReadPixel*. O načinu korišćenja ovih atributa i nijhovom značenju govoriće se u opisu metode *ReadPixel*.

Za specificiranje broja bitova koji se koriste za jedan piksel a time i broja boja koje se mogu prikazati u grafici koristi se atribut *byBitsPerPixel*.

Sledeći kod koji će biti korišćen pri GIF kodiranju grafike u toku kodiranja sadržan je u atributu *wNewCode*. Ovaj atribut zajedno sa atributom *byLeftToFit* koji predstavlja broj bitova koji su ostali nepopunjeni u tekućem bajtu pri popunjavanju bajtova i koristi se u toku izvršavanja LZW algoritma.

Atribut ***Bafer*** predstavlja bafer za GIF blok podataka koji može imati najviše 256 bajta. Ovaj bafer se prazni tako što se podaci upisuju u datoteku ili na standardni izlaz koji predstavljaju sliku u GIF formatu.

Indeks tekućeg bajta u baferu podataka je vrednost sadržana u atributu ***wBufIndex***. Vrednost ovog atributa se menja u toku izvršavanja LZW algoritma u skladu sa popunjavanjem bafera podataka. Kada se bafer podataka isprazni, tj. kada se njegov sadržaj upiše u datoteku ili na standardni izlaz ovaj atribut se resetuje na vrednost 0.

Kako su linije u bitmapi organizovane tako da svaka linija ima celi broj duplih reči (DWORD), da bi se prilikom kodiranja izbeglo horizontalno smicanje linija, atribut ***byToDWORD*** koristi se radi određivanja broja bajtova na kraju linije koji ne sadrže vrednosti piksela nego su prisutni radi poravnanja.

Za izvršavanje LZW algoritma koristi se heš tabela koja je sadržana u atributu ***HešTab***, i tabela sa kodovima stringova ***CodeTab***.

Naziv izlazne datoteke, ako se izlaz upisuje u datoteku, čuva se u atributu ***pFile*** koji predstavlja instancu C++ klase ***CFile***.

Zbog jednostavnijeg rukovanja izlaznim podacima definisana su još dva atributa koji predstavljaju GIF blokove. Ovi atributi odgovaraju po njihovoj definiciji i sadržaju odgovarajućim GIF blokovima i nakon popunjavanja mogu se direktno zapisati u datoteku ili na standardni izlaz.

Klasa ***CGIF*** realizovana je sa jednom jedinom namenom, a to je da grafiku iz bitmape isporuči u GIF formatu. Zbog ovakve namene jedine metode koje su napisane kao *public* su metoda ***Save*** i konstruktor i destruktor klase. Ostale metode koriste se unutar klase za realizaciju zadataka koji ova klasa ispunjava.

Namena metode ***Save*** je da za bitmapiranu grafiku na koju pokazuje pokazivač ***pDC*** kreira datoteku koja sadrži grafiku u GIF formatu sa imenom sadržanim u parametru ***lpFileName*** ili ako je vrednost ovog parametra NULL, da grafiku u GIF formatu ispiše na standardni izlaz. Za potrebe CGI skripta koristi se ova druga opcija tj. ispisivanje GIF podataka na standardni izalz. Parametar ***pDC*** je pokazivač na *device context* koji sadrži bitmapu nezavisnu od uređaja.

Realizacija LZW algoritma je izvršena u metodi ***Compress*** u kojoj se takođe vrši i podela podataka u blokove podataka veličine 256 bajta kao i pakovanje kodova u bajtove.

Priprema GIF blokova, opis ekrana i bloka i opis grafike vrši se popunjavanjem odgovarajućih polja ovih blokova na osnovu informacija koje su sadržane u instanci ***pDC*** klase ***CDC*** (*device context*).

Osim nelinearnosti realizovanih ovom metodom koje se odnose na ispitivanje grešaka koje mogu nastati u toku izvršavanja algoritma postoji i nelinearnost koja se odnosi na ispitivanje zahtevanog izlaza. U zavisnosti od vrednosti parametra ***lpFileName*** upisivanje blokova podataka vrši se ili u datoteku ili na standardni izlaz.

Rukovanje datotekom izvedeno je korišćenjem klase ***CFile*** i njenih metoda dok je deo koji se koristi za rukovanje standardnim izlazom realizovan pomoću funkcija za rukovanje tokom podataka.

Bitmapa u memoriji je predstavljena tako da pikseli su u memoriji smešteni obrnutim redom u odnosu na njihovo prikazivanje na ekranu. Takođe veličina memorije koja se koristi za bitmapu nije prost umnožak broja piksela nego se koristi celobrajan umnožak veličine duple reči (32 bita) i broja linija. Ovo znači da svaka linija ima ceo broj duplih reči i da u svakoj liniji ima određen broj bajtova koji ne sadrže informacije o pikselu. Iz iznetih razloga potrebno je odrediti koliko se bajtova na kraju svake linije ne koristi za memorisanje piksela nego samo za poravnanje.

U trenutku kada su upisani svi GIF blokovi koji prethode blokovima podataka tj. kada je potrebno upisati blokova podataka poziva se metoda *Compress* koja vrši upisivanje blokova podataka. Kada su svi blokovi podataka upisani ostaje da se upiše blok terminator i GIF terminator (Trailer) koji označavaju kraj bloka i kraj GIF toka podataka respektivno.

Povratna vrednost ove metode je kod greške koja može nastati u toku njenog izvršavanja. Greške koje mogu nastati su: IDS_ERR_TOO_MANY_COLORS – koja označava da ulazna bitmapa ima više od 256 boja tj. da je jedan piksel predstavljen sa više od 8 bita i IDS_ERR_CREATE_FILE – greška pri kreiranju datoteke koja ne daje informacije o razlogu neuspeha.

Metoda *WordCompress* realizuje LZW algoritam za kompresiju podataka i formiranje GIF blokova podataka kao i njihovo zapisivanje na odgovarajući izlaz. Parametar *byCodeSize* predstavlja početnu dužinu LZW koda dok se parametar *pcFile* koristi za određivanje izlaza (datoteka ili standardni izlaz). U toku izvršavanja ove metode inicijalizuju se osnovni kodovi u vezi sa LZW algoritmom, inicijalizuje se heš tabela koja se u paru sa tabelom kodova (*CodeTab*) koristi kao tabela kodova – stringova. Jedan elemenat u heš tabeli je predstavnik stringa koji je predstavljen kombinacijom **wPrefix<<8 + c** gde je LZW kod stringa koji postoji u tabeli kodova a **c** karakter koji se dodaje na string da bi se ispitalo postojanje novog stringa. Elementi u *CodeTab* sadrže LZW kodove stringa čiji se predstavnik nalazi u heš tabeli tako što su parovi povezani istim indeksom. Ovakvom realizacijom smanjuje se količina memorije koja se koristi za čuvanje stringova koji se dodaju u tabelu kodova.

Za realizaciju pakovanja kodova u bajte i dalje u GIF blokove podataka koristi se bafer u koji se privremeno smeštaju podaci dok se ne formira jedan GIF blok podataka. Inicijalizacija bafera obavlja se metodom *InitBafer*.

Metoda takođe vrši inicijalizaciju parametara vezanih za kodiranje LZW algoritmom i to *wClearCode* koji reprezentuje *Clear Code (CC)*, *wEOFCode* koji ima značenje *End Of Information (EOI)* koda.

Zbog realizacije LZW algoritma uz korišćenje bafera potrebno je nakon završetka kodiranja u bafer upisati kod *wEOFCode* a preostali sadržaj bafera upisati na odgovarajući izlaz što je realizovano metodom *WriteRest*.

Nakon inicijalizacije parametara koji definišu broj bajtova koji se ne čitaju pri čitanju piksela, za očitavanje vrednosti pojedinog piksela može se koristiti metoda *ReadPixel*. Ova metoda nema nikakvih ulaznih parametara i očitava vrednost tekućeg piksela. Piksel koji će biti očitan zavisi od vrednosti atributa *dwCurrentByte*, *byBytePixelLeft*, *CurrentPix* i to na sledeći način: *dwCurrentByte* sadrži vrednost koja predstavlja indeks prvog bajta linije koja se čita, u bitmapi; ***x*** je indeks bajta u liniji koja se čita (*dwCurrentByte* + ***x*** = bajt iz koga se čita piksel); *CurrentPix* se koristi kao brojač očitanih piksela u tekućoj liniji radi provere da li je pročitana tekuća linija; *byBytePixelLeft* predstavlja broj neočitanih piksela u tekućem bajtu.

Zbog načina na koji je organizovana bitmapa (obrnuti red linija u odnosu na GIF format) čitanje piksela se vrši od poslednje linije pa se inicijalizacija atributa *dwCurrentByte* vrši u skladu sa napomenom na sledeći način:

dwCurrentByte = *dwBytesNum* – *yByteOff*

gde *dwBytesNum* kako je već rečeno sadrži ukupan broj bajtova koje zauzima bitmapa, a atribut *yByteOff* predstavlja broj bajtova koji je zauzela jedna linija. Jedna linija zauzima ceo broj duplih reči pa je potrebno uskladiti vrednost atributa sa ovom činjenicom.

Samo očitavanje piksela realizovano je korišćenjem maske *wMask* koja se formira na osnovu atributa *byBytePixelLeft* i broja piksela u jednom bajtu. Zatim se

ova maska pomera za *byBytePixelLeft * byBitsPerPixes* u levo, tj. za proizvod preostalih piksela u bajtu i bitova koje zauzima jedan piksel. Maska sadrži onoliko jedinica koliko bitova sadrži jedan piksel pa se vršenjem operacije logičkog množenja dobija vrednost piksela koji se čita. Kako je ovako očitana vrednost na neodgovarajućem mestu u reči potrebno je rezultat pomeriti u desno za istu vrednost za koju je maska pomerana u levo.

Vrednost očitanog piksela vraća se kao povratna vrednost metode u manje značajnom bajtu. Ako je u toku čitanja bajta došlo do greške povratna vrednost ove metode je 0xFFFF. Greška koja se ispituje u ovoj metodi je čitanje van opsega tj. pogrešna inicijalizacija gore opisanih atributa.

Metoda *FindString* služi za pronalaženje stringa u tabeli kodova ako on postoji u ovoj tabeli. Kako je već opisano u tabelu kodova se ne upisuje kompletan string nego kombinacija koda i karaktera na ulazu. Ulazni karakter koji se dodaje na string koji postoji u tabeli je predstavljen parametrom *c* koji zajedno sa parametrom *wPrefix*, koji označava kod stringa, čini ulazne parametre ove metode.

Ako u tabeli nije pronađen traženi string metoda vraća vrednost 0xFFFF, a ako je pronađen traženi string povratna vrednost ove metode predstavlja LZW kod ovog stringa koji se dalje može koristiti za novo pretraživanje u skladu se gore pomenutim principom.

Metoda *AddCode* je komplementarna sa prethodno opisanom metodom. Koristi isti algoritam za pronalaženje mesta za upis koda, koji se upisuje u heš tabelu, (prazna tabela je popunjena vrednostima 0xFFFFFFFF) tražeći prvo slobodno mesto za upis. U tabelu kodova upisuje se vrednost *wNewCode* +1. Zbog veličine tabele i veličine LZW koda kao i zbog činjenice da se kodovi ne ponavljaju nije moguće prepunjavanje tabele. Zato pri pravilnom rukovanju tabelom, pražnjenu nakon dostizanja maksimalne dužine koda, ne može nastati greška. Povratna vrednost ove metode je vrednost LZW koda stringa koji je upisan u tabelu.

Kako je upisivanje kodova na izlaz (standardni izlaz ili datoteka na disku) realizovano uz pomoć bafera veličine jednog GIF bloka podataka tj. 256 bajtova to je bilo potrebno realizovati prikladnu metodu za rukovanje izlazom. Za ovu namenu relizovana je motoda *WriteCode* koja osim upisivanja kodova na izlaz vrši i „pakovanje" LZW kodova promenljive dužine u bajtove. Ulazni parametri ove metode su *wCode* kao LZW kod koji se upisuje na izlaz, *byNumBits* kao dužina LZW koda i *pcFile* koji sadrži naziv izlazne datoteke ili vrednost NULLza standardni izlaz.

Metoda prvo proverava da li u baferu postoji mesta za upis koda i ako ne postoji mesto za upis koda vrši se pražnjenje bafera tako što se njegov sadržaj upisuje na odgovarajući izlaz, zatim se vrši inicijalizacija bafera metodom *InitBafer* i na kraju se u inicijalizovan bafer upisuje LZW kod koji je ulazni parametar metode. U slučaju da bafer nije popunjen preskače se prvi korak (pražnjenje bafera i njegova inicijalizacija) i vrši se upisivanje koda u bafer. Atribut *byLeftToFit* uvek sadrži broj bitova koji se još mogu upisati u tekući bajt bafera, *wBuffIndex* je indeks tekućeg bajta u baferu.

Samo upisivanje koda u bafer ima dva slučaja. Jedan kada je broj bitova koji se upisuju manji od broja bitova koji su slobodni u tekućem bajtu kada je potrebno samo pomeriti kod na dogovarajuće mesto (8 – *byLeftToFit*) u levo i upisati kod. Drugi slučaj je slučaj kada u tekućem bajtu nema dovoljno mesta da se čitav kod upiše u njega, pa se u ovom slučaju kod upisuje delom u tekući a delom u sledeći bajt. U oba slučaja potrebno je ažurirati atribute koji opisuju stanje bafera a to su: *byLeftToFit* i *wBaferIndex*. Upisivanje u bafer vrši se dok se ne upišu svi bitovi iz ulaznog parametra ove metode.

Povratna vrednost ove metode predstavlja broj bitova upisanih u bafer i ova vrednost se može koristiti kao indikator greške tj. situacije da iz nekog razloga nisu upisani svi ulazni bitovi. Maksimalna dužina LZW koda je 12 bita pa je ulazni parametar dužine reči (16 bita) dovoljan da se prenese maksimalni LZW kod.

Nakon što su obrađeni svi podaci bitmape bafer u koji se upisuju izlazni kodovi ne mora biti prazan. Kako metoda ***WriteCode*** vrši pražnjenje izlaznog bafera samo u slučaju da je ovaj popunjen bilo je potrebno realizovati metodu koja će sadržaj bafera upisati na odgovarajući izlaz i u slučaju kada izlazni bafer nije popunjen. Upravo za ovu namenu realizovana je metoda ***WriteRest***.

Jedina namena ove metode je dakle da se sadržaj izlaznog bafera bez obzira na stanje bafera upiše na odgovarajući izlaz. Kao jedini parametar ove metode javlja se naziv izlazne dadoteke ili vrednost NULL koji specificira standardni izlaz. Ova metoda se poziva po okončanju LZW algoritma. Ova metoda kao i metoda ***WriteCode*** vrši inicijalizaciju (pražnjenje) izlaznog bafera.

Metoda ***InitBafer*** koristi se za inicijalizaciju izlaznog bafera što podrazumeva inicijalizaciju indeksa tekućeg bajta u koji se upisuje sledeći LZW kod, inicijalizaciju tekućeg bajta i inicijalizaciju atributa ***byLeftToFit***.

4.5.2 Klasa CSeditWeb

Kako je ova klasa realizovana sa jasnom namenom i za tačno određeni zadatak to je i struktura klase takva da su kao javne metode definisane samo konstruktor, destruktor i metoda sa imenom ***Output*** sa samo jednim parametrom koji predstavlja ime datoteke koja je izlaz CGI skripta. Ako je vrednost ulaznog parametra NULL onda se izlaz CGI skrpita ispisuje na standardni izlaz. Ostale metode i svi atributi ove klase su privatni (private). Za potrebe obrade *SEditWeb.ini* datoteke definisane su strukture koje se koriste za formiranje liste sa imenima i vrednostima parametara koji se koriste kao parametri za pristup podacima (putanje i imena datoteka koje sadrže grafički prikaz elemenata sistema, parametre za kreiranje bitmape, identifikaciju osnovne šeme sistema i sl.). Ove dve strukture definisane su ispod.

```
struct _IniLines
{
char *pcName;
char *pcValue;
_IniLines *pNext;
};
struct _IniSection
{
char *pcSection;
_IniLines *pFirstLine;
_IniSection *pNextSection;
};
```

Lista generisana na osnovu prethodnih struktura hijerarhijski preslikava strukturu konfiguracione datodeke u memoriji i koristi se za pristup parovima ime – vrednost parametra. Deklaracija klase ***CSeditWeb*** data je ispod.

Konstruktor ove klase obavlja osnovnu inicijalizaciju atributa klase postavljajući atribute klase na vrednost NULL.

Destruktor poziva nekoliko metoda koje vrše oslobađanje memorije koju instanca klase zauzima. Metode koje destruktor poziva vezane su za određene segmente realizacije i to na sledeći način: metoda ***DelIniLines*** briše listu parova naziv – vrednost parametara za inicijalizaciju CGI skripta, dok metoda ***UnloadShema*** vrši oslobađanje memorije koja je korišćena za smeštanje šeme (objekata koji predstavljaju

odgovarajuće elemente realnog sistema kojim se upravlja). Na kraju se oslobađa memorija koju zauzima atribut u kome se čuva naziv datoteke sa inicijalnim vrednostima parametara CGI skripta.

```
class CSeditWeb
{
private:

LPSTR lpszIniFileName;
struct _IniSection *pHeadSection;
CSeditFile *pSeditFile;
CSeditShema *pCurrentShema;
int iOutType, iZoom, iRefresh;

UINT Initialize(LPSTR FileName);
UINT SetLines();
UINT GetIniValue(char *Value,char *Section,char* Name);
void DelIniLines();
void ClearLine(char *Line);
UINT LoadShema(char *Err);
int SetCurrentShema (BYTE byShemaId);
int PrintHTML (char *pcFileName, int iShemaID, char *pErr);
int ShemaToHTML (BYTE bShemaId,char *pcOutFile, char *pErr);
UINT DrawWebSema(char *pcShemaFileName, char *pErr);
UINT DrawShema(CDC *pDC,SIZE *pSize,_ViewData VData);
int UnloadShema();
void PrintErr (int ErrCode, char *pErr);
void CreateFileName (char *pcFileName);

public:
CSeditWeb ();
~ CSeditWeb ();
int Output (char *pcOutFile);

};
```

Atributi *iOutType*, *iZoom* i *iRefresh* tj. njihove vrednosti određuju tip izlaza realizovanog CGI skripta (datoteka ili standardni zlaz), faktor uveličanja – umanjenja izlazne grafike u odnosu na njenu originalnu veličinu i interval (u sekundama) osvežavanja grafičkog prikaza sistema respektivno. Osvežavanje grafike vrši web klijent inicirajući komunikaciju sa web serverom na svakih *iRefresh* sekundi. Podaci o elementima sistema (stanju elemenata) i načinu njihove grafičke prezentacije, predstavljeni su odgovarajućim klasama, a atribut klase sa nazivom *pSeditFile*. Ovi podaci obuhvataju čitav sistem kojim se upravlja i sadrže kako informacije o stanju sistema tako i informacije koje su neophodne za grafički prikaz sistema. Sistem kojim se upravlja podeljen je u logičke ili fizičke celine koje u terminologiji grafičkog pikaza nazivamo šemama. Šema koja će biti iscrtana po pozivu CGI skripta određuje pokazivač na podatke koji reprezentuju odgovarajuću celinu sistema i ovaj se smešta u atribut *pCurrentShema*. Poslednja dva atributa predstavljaju instance odgovarajućih objekata koji reprezentuju sistem.

Klasa *CSeditWeb* čini osnovu realizovanog skripta radi dobijanja grafičkog prikaza i sadrži pored informacija o stanju sistema i njegovoj grafičkoj reprezentaciji i metode za pristup ovim podacima, metode za formiranje grafičkog prikaza i druge sporedne metode. Prikaz grafike realizovan ovom klasom obavlja se u dva prolaza (izvršavanja skripta). U prvom prolazu (poziv skripta bez parametara) preuzima se i obrađuje sadržaj datoteke *SeditHtml.htm* i u HTML kod se dodaju, na predviđena mesta, elementi neophodni za prikaz HTML stranice, dok se u drugom prolazu (poziv skripta sa parametrima) vrši generisanje grafičkog prikaza sistema i njegovog stanja.

Elementi koji se dodaju u html kod su: lista postojećih šema u sistemu koja se ubacuje na mesto posebne labele <sh_options>, naslov koji sadrži ime trenutno prikazane šeme i ubacuje se na mesto teksta SeditHtml i naziv datoteke koja sadrži grafički prikaz sistema ili naziv CGI skripta sa parametrima koji određuju šemu koja će se prikazati i ostalim parametrima koji određuju način prikaza (faktor uveličanja/umanjenja) i tip izlaza CGI skripta koji zamenjuje tekst pic_place.gif u HTML kodu. U drugom pozivu CGI skripta koji se aktivira kada klijent preuzima HTML stranicu koja na mestu pic_place.gif sadrži naziv CGI skripta sa odgovarajućim parametrima.

Sa ciljem da se na jedinstven način rukuje nastalim greškama u toku izvršavanja CGI skripta realizovana je metoda za ***PrintErr*** koja za određeni kod greške ***ErrCode*** na standardni izlaz ispisuje poruku o grešci koja odgovara kodu. Kako neke greške mogu biti i dodatno objašnjene to se dodatna informacija koja se tiče uzroka greške prenosi pokazivačem na string koji sadrži dodatni opis. Prilikom realizacije ove metode kao mehanizam kojim se za određeni kod greške pronalazi odgovarajuća poruka iskorišćena je tabela stringova. Ova tabela sadrži sledeće elemente: simbolički kod greške (ID), kod greške (Value) i opis greške (Caption). Kodovi grešaka koje mogu nastati u toku izvršavanja skripta imaju vrednosti od 1001 do 1024. Tabela stringova je korišćena radi kasnijeg eventualnog prevođenja poruka koje generiše CGI skript na druge jezike.

Već je objašnjeno da se za konfigurisanje realizovanog CGI skripta koristi inicijalna datoteka *SEditWeb.ini*. Za potrebe mogućih proširenja CGI skripta metoda je realizovana tako da njen ulazni parametar ***FileName*** predstavlja naziv inicijalne datoteke te je moguće CGI skript inicijalizovati na različite načine u zavisnosti od načina pozivanja CGI skripta. Metoda vrši inicijalizaciju atributa ***lpszIniFileName*** koji se koristi u metodi ***SetLines*** kao podatak koji opisuje naziv konfiguracione datoteke.

Na kraju da bi se izvršila dalja inicijalizacija CGI skripta ova metoda poziva metodu ***SetLines*** koja preuzima sadržaj konfiguracione datoteke i inicijalizuje atribute klase koji sadrže listu parova naziv i vrednost parametara.

U toku izvršavanja metode vrši se provera zadatog imena konfiguracione datoteke i u zavisnosti od preuzetog parametra u atribut ***lpszIniFileName*** se upisuje odgovarajuća vrednost.

Za potrebe preuzimanja konfiguracionih parametara realizovana je metoda ***SetLines*** koja vrši obradu konfiguracionoe datoteke i tekst sadržan u ovoj datoteci razdvaja na parove naziv parametra i vrednost parametra. Ovi parovi su organizovani po grupama (poglavljima) koja predstavljaju logičke celine određenih parametara, kako je opisano u uvodnom izlaganju ovog poglavlja. Konfiguraciona datoteka ima strukturu u skladu sa standardnim Windows konfiguracionim datotekama.

Da bi se obezbedio brži pristup vrednostima parametara parametri zajedno sa njihovim vrednostima čine atribut klase ***CSeditWeb***. Kao atribut klase ovi parametri su organizovani u jednostruko spregnutu listu. Prilikom preuzimanja vrednosti nekog parametra potrebno je pretražiti sve elemente liste tražeći element sa odgovarajućim imenom parametra u slučaju kada se ne zna kojoj sekciji pripada traženi parametar ili pretražiti sve elemente odgovarajuće sekcije ako je poznato kojoj sekciji pripada parametar.

Osnovni zadatak ove metode je da na osnovu sadržaja konfiguracione datoteke kreira listu parametara i njihovih vrednosti. Prilikom realizacije ovog problema korišćena je pomoćna metoda koja iz proizvoljne linije konfiguracine datoteke uklanja karaktere koji ne čine ime parametra i koji nisu deo sintakse konfiguracione datoteke. Naime u konfiguracionoj datoteci se mogu pojaviti komentari odvojeni od ostatka sadržaja stringom "//" iza kojeg sledi komentar ili blanko karakteri. Metoda ***SetLines***

prolazi kroz konfiguracionu datoteku obrađujući liniju po liniju tako što se u prvom koraku iz tekuće linije koja se obrađuje uklanjaju pomenuti elemeni da bi se dobila linija koja predstavlja sekciju parametara ili par naziv parametra i njegovu vrednost razdvojen znakom jednakosti. Nakon ovog koraka utvrđuje se da li linija konfiguracione datoteke sadrži naziv sekcije parametara ili par naziv i vrednost parametra. Ako je u pitanju sekcija kreira se novi elemenat liste i to elemenat tipa _IniSecton i isti se ulančava u listu tako da se kreira struktua opisana u prethodnom pasusu. Ako linija konfiguracione datoteke predstavlja par naziv i vrednost parametra kreira se novi elemenat tipa _IniLines i popunjavaju se odgovarajuća polja ovog elementa te se on ulančava u listu kao što je objašnjeno za elmenat koji sadrži naziv sekcije.

Kao kriterijum za određivanje o kojoj vrsti linije se radi ispituje se da li linija počinje karakterom ´[´ (simbolički označen kao BEGIN_SECTION) a završava karakterom ´]´ (simboličkii označen kao END_SECTION) kada linija predstavlja naziv sekcije parametara a svi ostali slučajevi se tretiraju kao parovi naziva i vrednosti parametara. Prilikom obrade linije koja je prepoznata kao par naziva i vrednosti parametra kriterijum za obradu i ispitivanje sintakse linije je postojanje karaktera ´=´ u liniji, koji razdvaja naziv od vrednosti parametra.

U slučaju nastanka greške u sintaksi ona se identifikuje kao greška nepostojanja sekcije simbolički označena kao IDS_ERR_NO_SECTION ili kao greška liniji sekcije simbolički IDS_ERR_SECTION_LINE. U oba slučaja čitava lista već kreiranih elemenata se briše i kao izlazna vrednost metode prosleđuju se pomenuti kodovi greške. Druge greške koje su obrađene prilikom realizacije ove metode su: IDS_ERR_UNABLE_OPEN_FILE koja označava da je iz nekog razloga nemoguće otvaranje konfiguracione datoteke (nepostojanje datoteke, oštećena datoteka isl.) i IDS_ERR_INSUFFICIENT_MEMORY koja signalizira nemogućnost alociranja memorije za potrebe baferovanja konfiguracione datoteke.

Po uspešnoj obradi konfiguracione datoteke metoda vraća vrednost OK, a atribut klase ***pHeadSection*** pokazije na prvu sekciju elemenata sa parovima naziva i vrednosti parametara. Polje ***pNextSection*** pokazuje na sledeću sekciju, a vrednost pokazivača poslednjeg elementa u ovom nizu je NULL. Pokazivač ***pFirstLine*** određenog elementa liste sekcija, predstavlja glavu liste parova naziva i vednosti parametara a elementi ove liste osim pokazivača na sledeći elemenat (***pNext***) liste sadrže polja ***pcName*** koje sadrži pokazivač na string sa nazivom parametra i ***pcValue*** pokazivač na string sa vrednošću parametra.

Metoda ***ClearLine*** izvršiće uklanjanje komentara i blanko znakova iz linije konfiguracione datoteke. Uklanjanje se vrši po sledćem algoritmu: prvo se uklanjaju komentari koji čiji početak se identifikuje kombinacijom karaktera "//". Sadržaj linije iza ove kombinacije se smatra se komentarom i odbacuje se, zatim se sa kraja linije uklanjaju oznake novog reda i oznaka povratka na prvu kolonu (*Carrige Return*) i na kraju se sa početka i kraja linije uklanjaju blanko znaci. Primećuje se da se ne izbacuju blanko znaci koji se mogu naći ispred ili iza znaka jednakosti u konfiguracionoj liniji koja sadrži par naziv i vrednost parametra. Zbog toga se pri obradi ovakve linije u metodi ***SetLines*** metoda ***ClearLine*** poziva i nakon što se par parametar – vrednost razdvoji na naziv parametra i njegovu vrednost, čime se uklanjaju blanko znaci sa kraja i početka nekog od ovih elemenata. Blanko znaci unutar naziva parametra ili njegove vrednosti su dozvoljeni i oni se ne uklanjaju. Takođe se ne uklanjaju blanko znaci iz naziva sekcije parametara.

Obrada se vrši direktno na prosleđenom parametru koji u ovom slučaju predstavlja rezultat obrade. Kako se u toku izvršavanja ove metode vrši samo

manipulacija stringovima to se ne očekuje da može doći do greške. Metoda ne vrši nikakvu kontrolu ulaznog stringa već samo, kako je rečeno, iz njega uklanja komentar i blanko znakove ne kontrolišući bilo kakav sintaksni smisao ulaznog parametra.

Za dobijanje vrednosti odgovarajućeg parametra iz konfiguracione datoteke realizovana je metoda *GetIniValue* koja se može koristiti u dva oblika: jedan prikazan u naslovu gde se traži vrednost parametra koji je deo tražene sekcije i traženog naziva i drugi način pozivanja gde je poznat samo naziv parametra ali ne i sekcija kojoj pripada. Zbog načina realizacije ove metode tj. zbog primenjenog načina pretraživanja liste sa parametara iz konfiguracione datoteke i zbog organizacije ove liste prvi način preuzimanja vrednosti parametra zahteva ispitvanje manje elemenata liste utoliko što se ne pretražuju sve sekcije već samo sekcija definisana parametom Section. Drugi način pozivanja zahteva pretraživanje svih sekcija dok se ne nađe traženi parametar.

Metoda u slučaju da nije pronađena vrednost traženog parametra vraća jedan od kodova greške IDS_ERR_NO_SEC_NAME ili IDS_ERR_NO_NAME u zavisnosti od toga da li nije pronađeno ime sekcije ili ime parametra. Prvi način pozivanja podrazumeva da se vrednost traži u tačno određenoj sekciji i ako sekcija ne postoji onda ne postoji ni parametar. Drugi način pozivanja (traženje vrednosti prema imenu parametra) podrazumeva da nije bitna sekcija u kojoj se nalazi parametar te se ne može javiti greška IDS_ERR_NO_SEC_NAME. U ovom slučaju vraća se vrednost prvog parametra koji zadovoljava uslov odgovarajućeg naziva parametra.

Za učitavanje konfiguracije sistema i opisa njegove grafičke reprezentacije koriste se rutine koje su realizovane u okviru ScadaSedit.dll biblioteke čime se postiže jedinstven način kreiranja strukture koja sadrži konfiguraciju i opis grafičke reprezentacije. Na ovakav način izmene koje se izvrše u rutinama za učitavanje konfiguracije sistema i opisa grafičkog prikaza (u daljem tekstu učitavanje šeme) ne zahtevaju intervenciju u CGI skriptu.

Da bi se koristile rutine realizovane u ScadaSedit.dll biblioteci potrebno je pre poziva rutine za učitavanje šeme potrebno je izvršiti inicijalizaciju parametara potrebnih za učitavanje (lokacije datoteka koje sadrže šemu i njihova imena). Osim namene da se učita šema ova metoda nema drugih zadataka pa se nakon inicijalizacije poziva rutina iz ScadaSedit.dll biblioteke *CreateSeditFile* koja poziva konstruktor klase i inicijalizuje potrebne parametre. Zatim se poziva rutina za učitavanje šeme sa nazivom datoteke koja sadrži šemu *LoadShemaFile*. Nakon uspešnog izvršavanja ove metode kreirana je instanca klase *CSeditFile* definisane u biblioteci ScadaSedit.dll i atribut *pSeditFile* sadrži pokazivač na instancu klase.

Povratna vrednost metode predstavlja kod nastale greške. Greške koje mogu nastati u toku izvršavanja metode je nedostatak memorije za kreiranje instance klase *CSeditFile* sa simboličkom oznakom IDS_ERR_INSUFFICIENT_MEMORY i greška pri učitavanju šeme koja rezultira neučitavanjem sa simboličkom oznakom IDS_ERR_NO_SHEMA. Kod analize poslednje greške nije moguće utvrditi uzrok neučitavanja šeme jer se za učitavanje koristi rutina iz biblioteke koja ne vraća nikakvu povratnu informaciju o nastaloj grešci pa se za detektovanje ove greške ispituje pokazivač *pSeditFile* i ako je njegova vrednost NULL šema nije učitana. Ostali uzroci grešaka se ne mogu proveriti.

Sistem kojim se upravlja može biti fizički i/ili logički podeljen u delove kao što je već govoreno u ovom izlaganju. Fizičke i/ili logičke celine se u SCADA sistemu predstavljaju pojedinačnim šemama dela sistema. Svaka šema se identifikuje idnentifikacionim kodom šeme koji je jedinstven za svaku šemu. Dakle moguće je jednoznačno identifikovati svaku od šema.

Da bi se odredila šema čija grafička reprezentacija će biti prikazana pozivom CGI skripta potrebno je da se postavi attibut *pCurrentShema* tako da pokazuje na šemu koja se prikazuje. Metoda *SetCurrentShema* prolazi kroz listu šema upoređujući identifikator šeme sa zadatim identifikatorom *byShemaId* i kada pronađe odgovarajuću šemu postavlja atribut *pCurrentShema*. Kako su identifikatori šema njihovi redni brojevi to se provera postojanja tražene šeme vrši i na početku izvršavanja metode kako bi se uštedelo vreme koje bi se utrošilo na nepotrebno pretraživanje.

U slučaju da nije pronađena tražena šema povratna vrednost ove metode je kod greške sa simboličkim imenom IDS_ERR_NO_SHEMA. Ova greška se može detektovati i pre pretrage ako je identifikator šeme veći od broja šema u sistemu. Ukoliko dođe do ove greške metoda dodeljuje vrednost NULL atributu *pCurrentShema*.

Ukoliko se metoda uspešno izvrši pokazivač *pCurrentShema* se postavlja da pokazuje na šemu sa identifikatorom *byShemaId*.

Za generisanje HTML koda koji predstavlja izlaz CGI skripta realizovana je metoda *PrintHTML*. Rešenje koje je u ovom slučaju realizovano polazi od postojanja delimično kompletnog koda stranice koji se smešta u datoteku *SeditHTML.htm*. Specifičnost ovakvog rešenja je u tome da se izmenom *SeditHTML.htm* datoteke može menjati konačnu izgled realizovanog rešenja. Datoteka se može generisati iz proizvoljnog HTML editora sa tim da po genetisanju osnovnog izgleda treba odgovarajuće elemente zameniti posebnim labelema koje CGI skript prepoznaje kao mesta za ubacivanje odgovarajućih elemenata sprege. Ovde će biti objašnjeno koji su to posebni elementi koji predstavljaju delove koji nisu fiksni i zavise od sistema za koji se generiše grafički prikaz.

Realizacija i algoritam po kome je realizovana ova metoda ne odstupa mnogo od postavljenih zaheva za generisanje HTML koda. Metoda za obradu HTML koda iz datoteke koristi bafer u koji se učitava HTML datoteka i sve obrade se vrše nad baferom. Prvo se postavljaju pokazivači na elemente kojtreba da se zamene. Zatim se na standardni izlaz štampaju delovi HTML datoteke koji se ne menjaju i kad se naiđe na jednu od pomenutih labela izvrši se izmena ili dodavanje potrebnog sadržaja tako što se labele preskaču i na njihovo mesto se ubacuju potrebni delovi HTML koda.

Metodi se prosleđuju parametri koji označavaju HTML datoteku i odabranu tj. šemu koja se iscrtava, prvi preko parametra *pcFileName*, a drugi preko parametra *iShemaID*, dok se parametar *pErr* u kombinaciji sa povratnom vrednošću (kodom greške) koristi za detaljnije opisivanje nastale greške.

Osim grešaka koje nastaju pri čitanju datoteke ostale koje se mogu javiti u toku izvršavanja metode tiču se pravilno (u skladu sa opisanim načinom) napisane HTML datoteke. Hronološki prva greška koja se moše pojaviti je nemogućnost da se otvori datoteka sa HTML kodom i njena simbolička oznaka je IDS_ERR_UNABLE_OPEN_FILE. Kao dopunska informacija o ovoj grešci daje se preko parametra *pErr* i ime datoteke o kojoj se radi. Druga greška (simbolički opisana sa IDS_ERR_FILE_READ) opisuje situaciju da iz nekih razlog nije moguće čitanje HTML datoteke. Ostale greške simbolički opisane sa IDS_ERR_HTML nastaju u slučaju da ne može da se pronađe neki od gore pomenutih obaveznih elemenata HTML datoteke. U ovom slučaju potrebno je proveriti ispravnost HTML koda i da li on zadovoljava postavljene uslove u skladu sa gore pomenutim.

Iscrtavanje tekuće šeme obavlja metoda *DrawShema*. Ova metoda vrši iscrtavanje šeme prolazeći kroz listu elemenata šeme te u ovom prolazu vrši i učitavanje trenutnih vrednosti parametara pojedinačnih elemenata. Dimenzije i faktor

uvećanja grafičke reprezentacije metodi se prosleđuju preko parametara *pSize* i *VData*. Prvi parametar sadrži dimenzije bitmape na koju se iscrtava šema dok je drugi vezan za iscrtavanje šeme u rutinama za crtanje iz biblioteke *ScadaSedit.dll*.

Struktura *VData* sadrži polja za opis trenutnog pogleda. Polja *ZoomX* i *ZoomY* određuju faktor uveličanja po odgovarajućoj osi i ova polja se koriste u metodi *DrawShema*. Polje *Shema* sadrži pokazivač na tekuću šemu. Ostala polja ove strukture vezana su za prikazivanje šeme u prozoru Windows okruženja i nemaju značenje za CGI skript ali se inicijalizuju zbog toga što se koriste metode za iscrtavanje iz biblioteke koja se koristi i za Windows okruženje.

Metoda na početku preuzima parametre potrebne za iscrtavanje šeme kao što su: broj boja koje će se koristiti za iscrtavanje, i parametri koji definišu dimenzije šeme. Prvi parametar je definisan u konfiguracionoj dsatoteci *SeditWebi.ini* dok se ostali prosleđuju metodi putem parametara u pozivu metode. Broj boja koje se mogu koristiti za crtanje šeme je ograničen na 256 boja a definisan je kao broj bitova po pikselu. Ograničenje broja boja proizilazi iz ograničenja veličine palete GIF formata. Ostali parametri koa što su faktor uveličanja i dimenzije generisane grafičke reprezentacije zavise jedan od drugog na sledeći način: ako su zadate dimenzije grafike (nijedna od dimenzija nije 0) onda se zanemaruje faktor uveličanja i grafika se formira tako da se ne naruši odnos visine i širine šeme tj. da ne dolazi do izobličavanja šeme. Ovo očuvanje odnosa dimenzija šeme vrši se tako što se kao faktor uveličanja po osama bira manji od faktora uveličanja (ZoomX i ZoomY) pa se oni izjednačavaju tako što veći od njih dobija vrednost manjeg. Na ovaj način postavlja se veličina šeme tako da elementi na šemi ne budu izobličeni.

Sledeći zadatak ove metode je da formira grafički prikaz sistema. Ovo je bio i jedan od ključnih problema ovog dela zadatka. Naime kako je biblioteka *ScadaSedit.dll* razvijena i prilagođena korišćenju u Windows grafičkom okruženju bilo je potrebno realizovati način da se ista koristi u CGI skriptu gde naravno ne treba kreirati prozore niti je potrebno bilo kakvo vidljivo grafičko okruženje. Opet sa druge strane trebalo je kreirati okruženje u kome bi funkcije za iscrtavanje grafike korišćene u pomenutoj biblioteci funkcionisale. Jedno od rešenja koje je realizovano u ovoj metodi je korišćenje virtuelnog konteksta uređaja (u ovom slučaju ekrana) u kome su funkcije vršile iscrtavanje. Naime metoda dobija pokazivač na kompatibilan kontekst uređaja koji u velikoj meri preslikava osobine odgovarajućeg uređaja (u našem slučaju ekrana) i dozoljava korišćenje većine funkcija za iscrtavanje. Da bi se koristile funkcije za iscrtavanje elemenata šeme iz biblioteke potrebno je da kompatibilni kontekst uređaja sadrži bitmapu. Ova metoda kreira bitmapu nezavisnu od uređaja i zatim inicijalizuje zaglavlje ovog elementa konteksta uređaja postavljajući dimenzije, paletu boja i ostale elemente zaglavlja bitmape. Zatim je za kontekst uređaja potrebno postaviti grafički mod i mod mapiranja konteksta uređaja na prozor aplikacija, što je potrebno učiniti radi kompatibilnosti sa funkcijama za iscrtavanje realizovanim u biblioteci.

Pošto je izvršena inicijalizacija neophodnih parametara metoda prolazi kroz strukturu koja predstavlja elemente tekuće šeme i za svaki elemenat preuzima opis (vrednosti parametara) trenutnog stanja elementa. Kada je uspešno preuzeto stanje elementa poziva se metoda za iscrtavanje objekta kojim je elemenat predstavljen. Ove dve operacije ponavljaju se za svaki elemenat u šemi čime se svi elementi iscrtavaju u generisanoj bitmapi.

Osim parametara koji su opisani na početku ova metoda dobija kao parametar pri pozivu pokazivač na kontekst uređaja (u našem slučaju kontekst kompatibilan uređaju) koji će se koristiti za iscrtavanje grafike.

Za kreiranje konteksta kompatibilnog uređaja i postavljanje parametara za iscrtavanje šeme relaizovana je metoda *DrawWebSema*. Ova metoda inicijalizuje strukturu *ViewData* i postavlja veličinu grafičkog prikaza. Radi jasnijeg definisanja veličine parametri koji se odnose na dimenzije grafike postavljaju se na vrednosti 0 te se u ovom slučaju veličina grafičkog prikaza određuje na osnovu faktora uveličanja.

Metoda dalje kreira kontekst kompatibilan uređaju Windows sistemskim pozivom *CreateCompatibleDC* i pokazivač na njega zajedno sa ostalim parametrima prosleđuje metodi *DrawShema* koja će kreirati grafički prikaz tekuće šeme. Na kraju se formirana grafička reprezentacija zapisuje u GIF datoteku ili se ispisuje na standardni izlaz što zavisi od vrednosti parametra *pcShemaFileName*.

Greške nastale u pozvanim metodama se prosleđuju preko povratne vrednosti metode i njenog parametra *pErr* za kojeg važi napomena data u opisu realizacije metode *DrawShema*.

Učitavanje konfiguracije CGI skripta i inicijalizacija parametara neophodnih za korišćenje funkcija za preuzimanje i iscrtavanje šeme i funkcija za očitavanje vrednosti koje opisuju stanje elemenata sistema objedinjena je u metodi *ShemaToHTML*. Osim namene da se inicijalizuju parametri za izvršavanje CGI skripta ova metoda vrši i pozivanje odgovarajućih metoda u zavisnosti od toga šta je traženi izlaz CGI skripta.

Prvo se inicijalizuje lista parametara iz konfiguracione datoteke CGI skripta (učitavanje linija konfiguracione datoteke). Zatim se učitava konfiguraciona datoteka sistema iz odgovarajućeg direktorijuma pozivom funkcije *ReadASCIIInfo* koja učitava datoteku *INFO.INI*. Datoteka iz koje se učitava konfiguracija sistema može biti definisana u konfiguracionoj datoteci CGI skripta (*SeditWeb.ini*) ili se koristi datoteka sa konfiguracijom sistema iz tekućeg direktorijuma, ako direktorijum nije definisan u konfiguracionoj datoteci CGI skripta. Zatim se proverava da li je definisan identifikator šeme koji treba da se iscrta i ako nije postavlja se identifikator šeme koji je definisan u konfiguracionoj datoteci CGI skripta. Parametar metode *bShemaId* ima vrednost 0 ako treba učitati šemu definisanu u konfiguracionoj datoteci ili vrednost koja je identifikator šeme iz sistema.

Nakon odabiranja šeme koja će se crtati metoda vrši inicijalizaciju atributa *iZoom* na sličan način kao što se odabira i šema koja se crta tj. ako nije definisan faktor uveličanja onda se on inicijalizuje prema vrednosti odgovarajućeg parametra u konfiguracionoj datoteci CGI skripta. Pri kreiranju konfiguracione datoteke treba obratiti pažnju da parametar koji određuje faktor uveličanja bude u saglasnosti sa HTML datotekom tj. da njegova vrednost bude jedna od vrednosti koje su definisane u HTML datoteci. Pošto su inicijalizovani osnovni elementi ptrebno je učitati šemu pozivom metode *LoadShema*, a da bi se uspešno koristile funkcije iz biblioteka programske podrške GAUS sistema, ostaje da se izvrše još neke inicijalizacije koje se tiču učitavanja konfiguracije sistema. Nakon uspešne inicijalizacije može se početi sa kreiranjem grafičkog prikaza sistema. U ovom trenutku se u zavisnosti od tipa izlaza pozivaju metode za iscrtavanje grafičkog prikaza ili metode za generisanje HTML koda. Kada je zahtevani tip izlaza CGI skripta grafički prikaz šeme vrši se i učitavanje vrednosti koje opisuju stanje elemenata sistema.

Za parametar *pErr* važi ista napomena koja je data u prethodnom opisima realizacije metoda. Povratna vrednost metode predstavlja kod greške nastale u toku izvršavanja metode i može imati sledeće simboličke vrednosti:

IDS_ERR_UNABLE_OPEN_FILE koja simbolizuje grešku pri otvaranju datoteke. Parametar *pErr* u ovom slučaju sadrži ime datoteke čije otvaranje nije uspelo

IDS_ERR_INFO_FILE greška pri učitavanju ili obradi *INFO.INI* datoteke

IDS_ERR_NO_SEC_NAME greška u koja označava nepostojanje sekcije i u ovom slučaju parametar *pErr* sadži ime sekcije koja nije pronađena u konfiguracionoj datoteci CGI skripta.

IDS_ERR_NO_NAME greška u koja označava nepostojanje parametra i u ovom slučaju parametar *pErr* sadži ime parametra koji nije pronađen u konfiguracionoj datoteci CGI skripta.

IDS_ERR_BIN_FILE greška pri učitavanju konfiguracije sistema.

Sve dosadašnje inicijalizacije parametara su lokalnog karaktera u smislu da nema nikakve komunikacije sa klijentskom aplikacijom koja se obavlja putem CGI protokola. Spregu sa klijentom obezbeđuje metoda *Output* koja obezbeđuje preuzimanje parametara i pozivanje odgovarajućih metoda radi formiranja izlaza.

Algoritam realizovan ovom metodom može se opisati na sledeći način: ako nema CGI ulaza postavi podrazumevane vrednosti promenljivih parametara, inače preuzmi parametre sa ulaza i inicijalizuj odgovarajuće atribute; pozovi metodu *ShemaToHTML*.

Ako ne postoji CGI ulaz atributi *iOutType*, *byShemaID*, *iZoom* i *iRefresh* tako da izlaz iz CGI skripta bude HTML kod, da se šema i faktor uveličanja učita iz konfiguracione datoteke CGI skripta i da nema osvežavanja prikazanog sadržaja. U ovom slučaju definisani su svi atributi te se može preduzeti sledeća akcija (poziva se metoda *ShemaToHTML*). Ako postoji CGI ulaz onda se vrednosti pomenutih atributa preuzimaju sa ulaza čime se obezbeđuje veza sa korisnikom. Takođe nakon poziva metode *ShemaToHTML* ova metoda vrši i ispisivanje poruke o nastaloj grešci na standardni izlaz.

Greške čiji se opis daje su greške koje nastaju u toku izvršavanja prethodno opisanih metoda u ovom poglavlju, i one su takođe opisane u opisima metoda u kojima nastaju. Ova metoda dakle vrši samo ispisivanje poruka o tim greškama.

4.6 Formiranje HTML stranice

4.6.1 Klasa CHTML

Klasa CHTML sadrži metode koje se koriste prilikom kreiranja HTML dokumenta. Deklaracija klase HTML data je dole.

Konstruktor klase *CHTML* poziva funkciju *InitDefaultParameters*, koja inicijalizuje određene članice klase. U konstruktoru se u bafer za greške, *pcErrorMessage*, upisuje poruka ´´No Errors´´. U metodi *InitDefaultParameters* se inicijalizuju sljedeće članice klase:

pcDefaultBgColor – podrazumevana boja pozadine web strane je plava
pcDefaultFont – podrazumevani font je Arial
pcDefaultHeader – podrazumevana veličina zaglavlja je H2
pcDefaultTableBorder – podrazumevani tip ruba tabele je 2
pcDefaultTableFieldHeight – podrazumevana visina ćelije tabele je 30
pcDefaultTableFieldWidth – podrazumevana širina ćelije tabele je 100
pcDefaultTableBorderColor – podrazumevana boja tabele je siva
pcDefaultTextColor – podrazumevana boja teksta je žuta
*pcDefaultLinkColo*r – podrazumevana boja linka je plava
pcDefaultVisitedLinkColor – podrazumevana boja posećenog linka je zelena
pcDefaultActivLinkColor – podrazumevana boja aktivnog linka je crvena

Metoda *DetermineParameter* je zadužena da na osnovu ulaznog para parametar=vrednost, prepozna parametru i da njegovu vrednost smesti u odgovarajući atribut klase.

```
class CHTML
{
public:
char* pcDefaultBgColor, pcDefaultTextColor, pcDefaultFont, pcDefaultHeader,
  pcDefaultTableBorder, pcDefaultTableFieldHeight, pcDefaultTableFieldWidth,
      pcDefaultTableBorderColor, pcDefaultLinkColor,    pcDefaultVisitedLinkColor,
      pcDefaultActivLinkColor, pcTitle_1, pcBackgroundColor_1, pcBackgroundColor_2,
      pcBackgroundColor_3, pcBackgroundColor_4, pcBackgroundColor_5, pcTextColor_1,
      pcTextColor_2, pcTextColor_3, pcTextColor_4, pcTextColor_5, pcFont_1,
  pcFont_2, pcFont_3, pcFont_4, pcHeader_1, pcHeader_2, pcHeader_3, pcHeader_4,
      pcHeader_5, pcTableBorder_1, pcTableBorder_2, pcTableBorder_3, pcTableBorder_4,
      pcTableBorder_5, pcTableFieldHeight_1 pcTableFieldHeight_2, pcTableFieldHeight_3,
      pcTableFieldWidth_1, pcTableBorderColor_2, pcTableBorderColor_3,
      pcLinkColor_1, pcLinkColor_2, pcLinkColor_3, pcLinkColor_4, pcLinkColor_5;
      pcVisitedLinkColor_1, pcVisitedLinkColor_2, pcVisitedLinkColor_3, pcVisitedLinkColor_4,
      pcVisitedLinkColor_5, pcActivLinkColor_1, pcActivLinkColor_2, pcActivLinkColor_3,
      pcActivLinkColor_4, pcActivLinkColor_5, pcErrorMessage;
CHTML();
~CHTML();
void InitDefaultParameters();
bool DetermineParameter(char* pcString);
void PrintText(char* pcText);
void GoToNextLine();
void InsertHorizontalLine();
void InsertItemToList();
void InsertLink(char* pcLink, char*pcLinkName, char* pcTarget);
void StartHtml(char* pcTitle);                    void EndHtml();
void StartBody(char* pcBackgroundColor, char*pcTextColor, char* pcLinkColor,
                char* pcVisitedLinkColor, char* pcActivLinkColor);
void EndBody();
void StartParagraph();                            void EndParagraph();
void StartHeader(char* pcHeader);                 void EndHeader(char* pcHeader);
void StartTable(char* pcBorder, char* pcColor); void EndTable();
void StartRow();                                  void EndRow();
void StartField(char* pcHeight, char* pcWidth); void EndField();
void StartUnnumberedList();                       void EndUnnumberedList();
void StartNumberedList();                         void EndNumberedList();
void StartFont(char* pcFont);                     void EndFont();
void StartBold();                                 void EndBold();
void StartItalic();                               void EndItalic();
void StartCenter();                               void EndCenter();
};
```

Parametri koji se inicijalizuju na ovakav način su: naslov HTML strane (*pcTitle*), boja pozadine web strane (*pcBackgroundColor*_X), boja teksta *(pcTextColor*_X), font (*pcFont*_X), tip zaglavlja (*pcHeader*_X), tip ruba tabele (*pcTableBorder*_X), visina ćelije tabele (*pcTableFieldHeight*_X), širina ćelije tabele *(pcTableFieldWidth*_X), boja ruba tabele (*pcTableBorderColor*_X), boja linka (*pcLinkColor*_X), boja posećenog linka (*pcVisitedLinkColor*_X) i boja aktivnog linka (*pcActivLinkColor*_X).

Metoda *StartHtml* sa ulaznim parametrom *pcTitle* kreira zaglavlje HTML dokumenta. Ulazni parametar *pcTitle* predstavlja ime HTML strane.

Funkcija *StartBody* ispisuje <body> tag. Ulazni parametri su boja pozadine web strane, boja teksta, boja linka, boja posećenog linka i boja aktivnog linka.

Ostale metode klase *CHTML* obezbeđuju ispisivanje HTML tagova prilikom kreiranja rezultantne web strane. Ove funkcije zbog svoje jednostavnosti nema potrebe dodatno opisivati.

4.7 Realizacija CGI skripta

Sam CGI skript je realizovan pomoću dve klase. Prva klasa ***CCGI*** realizuje komunikaciju sa web klijentom putem CGI protokola. Druga klasa ***CWeb*** povezuje sve dosad pomenute klase u jednu celinu i vrši obradu ulaznih podataka i na osnovu toga formiranje izlaznih podataka pomoću jedne od dosad pomenutih klasa.

4.7.1 Klasa CGI

Prilikom realizacije ove klase trebalo je realizovati nekoliko jednostavnih algoritama kako bi se izvela obrada URL kodiranog stringa i ispoštovali zahtevi koje postavlja CGI protokol u smislu prenošenja parametara tj. obrade ulaza i načina njegovog dekodiranja.

Klasa ***CCGI*** sadrži metode za ostvarivanje komunikacije između CGI skripta i Web klijenta po CGI protokolu. Da bi se ostvarila ova komunikacija potrebno je, bez obzira na metod kojim se podaci razmenjuju, preuzeti parametre od klijenta i iste protumačiti na odgovarajući način poštujući pravila koja postavlja CGI protokol. Deklaracija klase je data ispod.

Atributi ***iLength*** i ***strVarHead*** su atributi koji se koriste u interne svrhe i predstavljaju privatne promenljive ove klase. Dužina ulaza, u bajtima, pri POST metodu prenosa parametara smešta se u atribut ***iLength***, dok je u slučaju prenosa parametara GET metodom vrednost ovog atributa 0. Pošto je veličina podataka koji se pri prenosu podataka POST metodom ograničena na 1024 bajta, vrednosti koje može imati ovaj atribut su u opsegu od 0 do 1023. Atribut ***strVarHead*** je pokazivač na promenljivu tipa ***_strVar*** koja predstavlja definiciju pojedinog elementa u listi koja se interno koristi za smeštanje parova naziv promenljive i vrednosti te promenljive. Struktura ***_strVar*** data je zajedno sa deklaracijom same klase.

Pokazivač ***pcName*** je pokazivač na string koji sadrži naziv promenljive, a ***pcValue*** pokazivač na string koji sadrži vrednost te promenljive. Pokazivač ***pNext*** je pokazivač na sledeći element liste. Adresa prvog elementa liste nalazi se u atributu ***strVarHead***.

Ulazni podaci, preuzeti po CGI protokolu u nepromenjenom obliku smeštaju se u memoriju na adresi koja je smeštena u atribut ***pcInputBuff***. Atribut ***pcMethod*** sadrži string koji označava metod kojim su preneti podaci dakle vrednosti "GET" ili "POST".

Metoda ***GetInput*** preuzima parametre prosleđene CGI skriptu i neizmenjen URL kodiran string smešta u atribut ***pcVarBuff***, bez obzira na metod koji se koristi dok metoda ***ParseInput*** obrađuje URL string smešten u ***pcVarBuff*** i formira listu sa parovima naziv promenljive i vrednost koju koristi metoda ***GetValue*** da preuzme vrednost za odgovarajući parametar. Metode ***HexToChar*** i ***RemoveTokens*** su pomoćne i interno se koriste pri obradi URL kodiranog stringa. Prva da se se ASCII kod karaktera pretvori u odgovarajući karakter a druga da se iz stringa uklone posebne URL oznake "%hh" , "&" i "+" sa odgovarajućim karakterima.

Metoda ***GetInput*** realizuje preuzimanje parametara u obliku URL kodiranog stringa i samo smešta ovaj string u atrribut ***pcVarBuf*** čime je ulazni string stavljen na raspolaganje ostalim metodama ove klase. Metoda preuzima string u skladu sa CGI metodom prenosa parametara. Metod prenosa parametara preko komandne linije nije podržan jer prilikom rešavanja problema realizacije CGI skripta nije bilo potrebe da se isti koristi. Korišćeni metod preuzima se iz promenljive okruženja sa imenom REQUEST_METHOD i u zavisnosti od vrednosti ove promenljive realizovano je punjenje bafera ***pcVarBuff*** iz odgovarajućeg izvora. Vrednost promenljive okruženja

REQUEST_METHOD smešta se u atribut *pcMethod* koji je javan i može mu se neposredno pristupiti. Ova metoda nema nikakvih ulaznih parametara a podaci koje obrađuje preuzima u zavisnosti od korišćenog metoda prenosa podataka preuzimaju se preko standardnog ulaza ili iz promenljive okruženja sa imenom QUERY_STRING.

```
struct _strVar
{
char *pcName;
char *pcValue;
struct _strVar *pNext;
};

class CCGI
{
private:
char* pcRequestMethod;
int iContentLength;
int iSizeOfInputBuffer;
char* pcInputBuffer;
bool bQueryType;
bool bFieldName;
bool bName;
char* pcQueryType;
char* pcFieldName;
char* pcName;
char* pcErrorMessage;
bool DetermineUsedVariable(char*);
int FromHexToInt(char, char);

public:
CCGI();
~CCGI();
int DetermineMethod();
bool ReadingSTDIN();
bool ReadingQueryString();
ReadingCommandLine(char* pcString);
bool ParsingTheInput();
void DecodeURL();
char* GetpcQueryType()          {return pcQueryType;}
char* GetpcFieldName()          {return pcFieldName;}
char* GetpcName()               {return pcName;}
char* GetpcErrorMessage()       {return pcErrorMessage;}
};
```

Kao izlaz ove funkcije pojavljuje se kod greške koja je nastala ili informacija o korektno izvršenom zadatku. Ove informacije pretstavljene su konstantama. Vrednosti koje metoda može vretiti su: OK kada je URL kodirani string preuzet i nije bilo nikakvih grešaka, NO_INPUT kada je detektovana situacija da ne postoji ulazni string, IDS_ERR_RQ_METHOD kada nije prepoznat ni jedan od dva CGI metoda prenosa parametara GET ili POST.

Metoda *ParseInput* vrši obradu URL kodiranog stringa, a kao ulaz koristi string iz atributa *pcVarBuff*. Obrada se vrši po sledećim pravilima:

1. pronaći karakter (&) koji označava kraj jednog para naziv promenljive – vrednost promenljive u URL kodiranom stringu,
2. razdvojiti parove naziv promenljive – vrednost promenljive na naziv i vrednost promenljive,
3. zameniti karaktere (+) sa „blanko” znakom u nazivu promenljive i u vrednosti promenljive,
4. pretvoriti posebnu kombinaciju (%hh) u odgovarajući karakter gde hh predstavlja heksadecimalni ASCII kod karaktera

Da bi se parovi naziv promenljive – vrednost promenljive učinili dostupnim oni se smeštaju u listu zasnovanu na strukturi *_strVar* koja definiše jedan elemenat liste. Kao „bočni efekat" ova funkcija menja sadržaj atributa *pcVarBuff* na sledeći način: karakteri „&" i „=" zamenjuju se /0 karakterom, karakter + zamenjuje se „blanko" karakterom a kombinacija „%hh" zamenjuje se odgovarajućim karakterom. Dakle nakon pozivanja ove metode sadržaj atributa *pcVarBuff* ne predstavlja URL kodiran ulaz.

Algoritam realizovan ovom metodom u potpunosti odgovara ponavljanju tačaka 1, 2, 3, 4 sve dok u ulaznom stringu postoji karakter „&" s tim da se u svakom koraku ovako obrađeni elementi dodaju u listu koja sadrži dekodirane parove sa nazivom promenljive i njenom vrednošću. Kako poslednji par nema karakter „&" na kraju potrebno je ovaj deo ulaznog stringa takođe obraditi tako što treba ponoviti tačke 2, 3 i 4 još jednom i dobijeni par takođe smestiti u listu.

Realizovana lista predstavlja jednostruko spregnutu listu, a pokazivač na početak liste smešta se u atribut *strVarHead*. Elementi ove liste sadrže parove promenljivih i njihovih vrednosti sve dok postoji instanca klase CCGI.

Izlaz ove metode predstavlja kod greške a kodovi koje ova metoda može vratiti i njihovo značenje su: OK kada je generisana lista i nije bilo grešaka prilikom dekodiranja URL kodiranog stringa, NO_INPUT kada ne postoji URL kodiran string u atributu *pcVarBuff* (npr. nije pozivana metoda *GetInput*) i IDS_ERR_NAME_VALUE kada postoji greška u ulaznom stringu (ulazni string nije URL kodiran). Kao „bočni efekat" metoda generiše listu koja predstavlja izlaz ove metode.

Metoda *GetValue* realizovana je da obezbedi pristup vrednosti određenog parametra. Vrednost parametra uzima se iz liste sa parovima naziv promenljive – vrednost promenljive. Metoda upoređuje dobijenu vrednost *pcName* sa vrednostima *pcName* sve dok ne pronađe naziv parametra u listi i kao povratnu vrednost vraća vrednost iz liste na koju pokazuje pokazivač *pcValue*. Ako u listi ne postoji traženi parametar metoda kao povratnu vrednost vraća konstantnu VARIABLE_NOT_FOUND, a ako je pronađena vrednost konstantu OK.

Metoda ne rezerviše memoriju za vrednost promenljive i pre poziva potrebno je rezervisati memoriju u koju će se prihvatiti vrednost promenljive i pri pozivu metodi preneti pokazivač na rezervisanu memoriju. Veličina memorije koja se mora rezervisati ograničena je maksimalnom dužinom koju može imati vrednost promenljive i iznosi VALUE_LENGTH (256) bajtova. Pre pozivanja ove metode potrebno je preuzeti ulaz (metoda *GetInput*) i isti razdvojiti na parove naziv promenljive – vrednost promenljive (metoda *ParseInput*).

Pronalaženje vrednosti za određenu promenljivu realizovano je tako što se prolazi kroz listu (glava *strVarHead*) i upoređuje naziv traženog parametra sa vrednošću naziva promenljive u listi. Kada se pronađe par u listi sa odgovarajućim nazivom promenljive, vrednost promenljive se kopira u memoriju na koju pokazuje pokazivač *pcVal*. Ovakvo rešenje, sa kopiranjem vrednosti u odnosu na prosleđivanje pokazivača na vrednost promenljive, je realizovano da bi se izbeglo nenamerno menjanje vrednosti u listi sa parovima.

Metoda *RemoveTokens* je pomoćna metoda koja je realizovana da bi se deo posla koji obavlja metoda *ParseInput* podelio na logički povezane celine. Metoda realizuje dekodiranje URL kodiranih naziva i vrednosti promenljivih i to deo koji se odnosi na zamenu karaktera "+" "blanko" karakterom i zamenu kombinacije keraktera "%hh" odgovarajućima karakterom.

Ulaz ove funkcije je string koji sadrži naziv promenljive ili vrednost promenljive koji ne ispunjavaju u potpunosti pravila URL kodiranja što znači da su već razdvojeni parovi naziv promenljive – vrednost promenljive.

Obrada ulaznog stringa vrši se nad ulaznim stringom tako da je nakon poziva ove metode rezultat njenog delovanja obrađen ulazni string koji se nalazi na adresi *pcString*. Osim ovog efekta metoda ima i povratnu vrednost (konstanta OK) koja označava da je string obrađen. Kako je sama funkcija ove metode da iz ulaznog stringa ukloni određene specijalne karaktere i zameni ih odgovarajućim tj. tačno je precizirana namena ove metode i kako metoda nije namenjena korišćenju van klase kojoj pripada to u ovom slučaju nisu analizirane greške koje mogu nastati u toku procesiranja ulaza. Metoda će izvršiti uklanjanje i zamenu posebnih karaktera bez obzira na to da li ulazni string zadovoljava uslove URL kodiranja. Metoda ne vrši nikakvu logičku kontrolu ulaznog stringa te zbog toga i nije previđeno da se koristi izvan pripadajuće klase. Zamena kombinacije karaktera "%hh" vrši se pozivanjem metode *HexToChar*.

Metoda *RemoveTokens* pri dekodiranju URL kodiranog stringa koristi pomoćnu metodu *HexToChar*. Namena ove metode je da sa string *pcHex* koji sadrži heksadecimalni ASCII kod karaktera kao izlaz vrati karakter čiji je to ASCII kod. Metoda se koristi za dekodiranje kombinacije karaktera "%hh" u URL kodiranom stringu. Ulazni parametar (*pcHex*) ove metode sadrži dvocifreni ASCII kod karaktera i ova situacija predstavlja podrazumevano stanje za ovu metodu. To znači da će u slučaju da *pcHex* pokazuje na duži string, metoda uzeti u obzir samo prva dva bajta (karaktera) prosleđenog stringa.

Rezultat koji metoda vraća je karakter čiji je ASCII kod prosleđen kao ulazni parametar metode.

4.7.2 Klasa CWeb

Klasa *CWEB* čini centralnu klasu realizovanog CGI skripta koja povezuje sve ostale klase. Njene članice klase su i objekti svih gore navedenih klasa. U njenim funkcijama pozivaju se metode ostalih klasa, kako bi se ostvarila puna funkcionalnost CGI skripta.

U konstruktoru klase *CWEB* indikatori koji pokazuju da li su učitani podaci iz inicijalizacionih datoteka postavljaju se na vrednost FALSE. Indikatori odgovarajućih datoteka su *bHTMLini*, *bDatabaseini* i *bOPCini*, respektivno. U bafer za greške, *pcErrorMessage*, upisuje se poruka ´´No Errors´´.

Ukoliko se CGI skript poziva iz programa za pregledanje web strana onda se za prijem ulaznih podataka poziva metoda *GetDataInput*.

U njoj se prvo proverava da li se radi o GET ili POST načinu prosleđivanja podataka. Ta provera se vrši pozivanjem metode *DetermineMethod* koja je članica klase *CGI*. Na osnovu utvrđenog metoda pozivaju se metode za smeštanje podataka u ulazni bafer. Ako se radi o POST metodi onda se poziva metoda *ReadingSTDIN*, a ako se radi o GET metodi onda *ReadingQueryString*. Obe ove metode su članica klase *CGI*.

Nakon prijema ulaznih podataka pristupa se njihovom dekodiranju. Za to je zadužena metoda *URLDecode*. I ova metoda je članica *CGI* klase.

Nakon obavljenog dekodiranja vrši se izvlačenje parova parametar=vrednost iz ulaznog bafera. To se obavlja metodom *ParsingTheInput*, koja je takođe članica klase *CGI*.

Sledi provera da li je zahtevan grafički ili tekstualni prikaz. U slučaju grafičkog prikaza poziva se metoda *Output*, članica klase *CSeditWeb*. Izlaz iz ove metode je formirana

HTML strana sa grafičkim izgledom u skladu sa ulaznim parametrima. U suprotnom nastavlja se formiranje izlazne HTML datoteke.

```
class CWeb
{
private:
CGI cgi;
HTML html;
COPCClt OPCclient;
CSeditWeb SeditWeb;
SQLResult* pResultOfQuery;
ODBCdatabase ConnectToODBC;
bool bHTMLini, bDatabaseini, bOPCini, bError;
char* pcSqlQuery, pcDataSourceName, pcUserName, pcPassword, pcHeaderTitle, pcMachineName;
char* pcServerName, pcIniBuffer, pcErrorMessage, pcItemName, pcVariableType, pcFieldName;
char* pcVariableName, pcDigitalVariableType, pcItemValue;
int iNumberOfItems, iNumberOfServers;

public:
CWeb();
~CWeb();
bool GetDataInput();
bool GetDataFromCommandLine(char* pcString[]);
bool ScriptInit();
bool OpenIniFiles(char* pcFileToOpen);
bool ReadSettingsFromBuffer(char* pcBuffer);
bool DetermineParameter(char* pcString);
char* DetermineTypeOfQuery();
void GetItemName();
bool CreateSqlQuery();
bool ExecuteSqlQuery();
bool CheckForError();
void EncodeString(char* pcString);
bool ConnectOnOPCServer();
void ParseTheNameOfItem(char* pcString);
void DetermineItemValue(char* pcItem);
void FormatVariant (VARIANT *pValue);
void LinkCreate(char* pcLinkName);
void InsertColumnsNamesIntoTable();
void InsertDataIntoTable();
void InsertDataIntoList();
void CreateHTMLPage_1();
void CreateHTMLPage_2();
void CreateHTMLPage_3();
void CreateHTMLPage_4();
void CreateHTMLPage_5();
void CreateHTMLPage_Error();
CGI Getcgi()                    {return cgi;}
char* GetpcItemName()           {return pcItemName;}
};
```

Zatim se proverava da li je zahtevana vrednost procesne veličine sa OPC servera. U tom slučaju poziva se metoda *GetItemName*, koja upisuje ime procesne veličine u bafer *pcItemName*.

Ukoliko su sve ove metode uspešno izvršene povratna vrednost metode *GetDataInput* je TRUE, u protivnom je FALSE. Povratna vrednost je FALSE i ako je CGI skriptu pristupano na neki drugi način koji nije ni POST ni GET.

Svim funkcijama koje su članica klase *CGI* je pristupano preko instance klase *CGI* - *cgi* koja je atribut klase *CWeb*.

Ukoliko se ovom programu podaci prosleđuju preko komandne linije a ne preko programa za pregledanje web strana onda se ne radi o CGI skriptu već o običnom

programu. U ovom slučaju za prijem podataka iz komandne linije zadužena je metoda *GetDataFromCommandLine*.

Ulazni parametar ove metode je skup podataka unetih iz komande linije. Ti podaci se prvo kodiraju metodom *EncodeString* da bi se metode *CGI* klase mogle iskoristiti u potpunosti, bez ikakvih promena, kao da su podaci prosleđeni iz programa za pregledanje web strana metodom GET ili POST.

Nakon toga poziva se metoda klase *CGI*, *ReadingCommandLine*, kojoj se prosleđuju kodirani ulazni podaci.

Zatim se, kao i u prošloj metodi *GetDataInput*, pozivaju metode *DecodeURL* i *ParsingTheInput*.

Ukoliko se od programa zahteva vrednost određene procesne veličine onda se poziva i metoda *GetItemName*, koja upisuje ime te procesne veličine u bafer *pcItemName*.

Ako su sve ove operacije uspešno završena povratna vrednost metode *GetDataFromCommandLine* je TRUE. U protivnom je FALSE.

Metoda *EncodeString* je zadužena za URL kodiranje podataka koji su programu prosleđeni iz komandne linije.

Ulazni podatak ove metode je string, *pcString* koji treba da se kodira. Kodiranje se vrši na taj nači što se znakovi + zamijene sa znakovima &. Povratna vrednost je kodirani string, *pcString*.

Metoda *ScriptInit* poziva metode klase koje su zadužene za otvaranje datoteka, za čitanje sadržaja tih datoteka i za kreiranje SQL upita.

Prvo se učitava html.ini datoteka koja sadrži parametre koji se koriste u HTML sintaksi, a zatim database.ini ili opc.ini u zavisnosti od tipa zahteva. Ova metoda nema ulaznih podataka.

Za otvaranje datoteka i upisivanje njenog sadržaja u bafer *pcIniBafer* zadužena je metoda *OpenIniFiles* kojoj se prosleđuje ime datoteke koja treba da se pripremi za čitanje.

Ukoliko je metoda *OpenIniFiles* uspešno izvršena onda se pristupa čitanju i tumačenju sadržaja datoteke koji je upisan u bafer *pcIniBafer*. Za čitanje sadržaja bafera zadužena je metoda *ReadSettingsFromBafer*. Parametar koji se prosleđuje ovoj metodi je *pcIniBafer*.

Ako su sve metode uspešno izvršene povratna vrednost metode *ScriptInit* je TRUE. U protivnom je FALSE.

Kao što je već rečeno metoda *OpenIniFiles* je zadužena za otvaranje datoteke i upisivanje njenog sadržaja u bafer *pcIniBafer*. Ulazni parametar ove metode je ime datoteke koju treba učitati.

Prilikom učitavanja neke od inicijalizacionih datoteka (ini) onda se odgovarajući indikator postavlja na vrednost TRUE, a ostali indikatori na vrednost FALSE.

Ako željena datoteka nije uspešno otvorena ili pročitana onda se u bafer *pcErrorMessage* upisuje obaveštenje o neuspehu, a povratna vrednost metode *OpenIniFiles* tada je FALSE.

Metoda *ReadSettingsFromBafer* je članica klase koja podatke smeštene u baferu *pcIniBafer* razlaže na parove parametar=vrednost, a zatim zavisno od datoteke iz koje su učitani ti podaci poziva odgovarajuću metodu, koja će učitane vrednosti dodeliti odgovarajućim članicama klasa.

Tako ako se u baferu *pcIniBafer* nalaze podaci iz datoteke html.ini, onda se poziva metoda *DetermineParameter* koja je članica klase *HTML*. Njoj se prosleđuje par parametar=vrednost, a onda, na osnovu imena parametra dodeljuje se vrednost odgovarajućem atributu klase *HTML*. Ukoliko su podaci učitani iz datoteka

database.ini i opc.ini onda se poziva metoda *DetermineParameter* koja je članica klase *CWeb*. I njoj se prosleđuje par parametar=vrednost, i na osnovu imena parametra vrednost se dodeljuje odgovarajućem atributu klase *CWeb*.

Povratna vrednost metode *ReadSettingsFromBafer* je tipa *bool* i ima vrednost TRUE ukoliko su sve metode koje su pozvane u okviru nje uspešno izvršene. U protivnom povratna vrednost je FALSE.

DetermineParameter je metoda koja na osnovu ulaznog parametra *pcString*, koji čuva par parametar=vrednost, određuje o kom se parametru CGI skripta radi i koja je njegova vrednost.

Prvo se proverava da li je naziv parametra koji je učitan iz datoteke jedan od sledećih: DataSourceName, UserName, Password, OPCServerName ili MachineName. Ukoliko jeste onda se vrednost tog parametra upisuje u odgovarajući atribut klase *CWeb*. Tako se za parametar DataSourceName učitana vrednost upisuje u atribut *pcDataSourceName*, a za parametre UserName, Password, OPCServerName i MachineName u atribute *pcUserName*, *pcPassword*, *pcServerName* i *pcMachineName*, respektivno.

Ukoliko za učitani par naziv parametra ne odgovara ni jednom od gore navedenih onda se prekida izvršavanje metode, a u bafer *pcErrorMessage* se upisuje obaveštenje o grešci. U ovom slučaju povratna vrednost metode *DetermineParameter* je FALSE.

Ako je parametar prepoznat i njegova vrednost upisana u predviđeni atribut klase povratna vrednost funkcije je TRUE.

Metoda *DetermineTypeOfQuery* samo poziva *inline* funkciju *CGI* klase *GetpcQueryType* i vraća njenu vrednost.

Metoda *CreateSqlQuery* služi za kreiranje SQL upita koji će se izvršiti nad povezanom bazom podataka. Ova metoda nema ulaznih parametara.

Ime procesne veličine se čuva u baferu *pcName* koji je atribut klase *CGI*. Kreirani SQL upiti se upisuju u bafer *pcSqlQuery*.

U ovoj metodi se u zavisnosti od kreiranog SQL upita, određuje i tekst koji će biti ispisan u naslovu web strane. Taj tekst se zapisuje u atribut *pcHeaderName*.

Ukoliko par parametar=vrednost nije ni jednog oblika od gore navedenih onda se ne kreira SQL upit, već se u *pcErrorMessage* upisuje poruka da je skriptu prosleđen pogrešna vrednost. U ovom slučaju povratna vrednost je FALSE.

Ako je SQL iskaz uspešno kreiran onda je povratna vrednost metode TRUE.

ExecuteSqlQuery je metoda koja je zadužena za povezivanje sa bazom podataka i za izvršavanje SQL upita nad bazom. Ova metoda nema ulaznih parametara.

Za povezivanje sa bazom podataka poziva se metoda *ConnectToDataSource* koja je članica klase *ODBCdatabase*. Parametri koji se prosleđuju ovoj funkciji su *pcDataSourceName* (ime izvora podataka), *pcUserName* (korisničko ime) i *pcPassword* (šifra korisnika). Ova metoda se poziva metodom klase *ODBCdatabase*, *ConnectToODBC*.

Nakon povezivanja sa bazom podataka poziva se metoda, takođe članica klase *ODBCdatabase*, *ExecuteSQLCommand* kojoj se prosleđuje parametar *pcSqlQuery* (SQL upit). I ova metoda se poziva preko *ConnectToODBC*. Povratna vrednost ove funkcije je pokazivač na objekat klase *SQLResult*. Ta vrednost se upisuje u atribut *pResultOfQuery*.

Povratna vrednost funkcije *ExecuteSqlQuery*, je bool tipa, pa ako su sve metode koje se pozivaju unutar nje uspešno izvršene povratna vrednost je TRUE. U protivnom je FALSE.

Metoda *CheckForError* proverava da li je prilikom izvršavanja skripta došlo do greške u skriptu. Ova funkcija nema ulaznih vrednosti, a povratna vrednost je tipa bool.

Ukoliko je došlo do greške povratna vrednost je TRUE, a ako nije došlo do greške onda je FALSE.

ConnectOnOPCServer je metoda koja je zadužena za povezivanje sa OPC serverom i za određivanje broja procesnih veličina na njemu. Takođe se inicijalizuje i vrednost bafera *pcHeaderTitle*, koja će biti ispisana u naslovu web strane. Ova funkcija nema ulaznih parametara.

U ovoj funkciji se preko objekta klase *COPCClt*, *OPCclient* poziva metoda *Connect_OPC* kojoj su prosleđeni parametri *pcMachineName* (ime računara na kome se izvršava OPC server) i *pcServerName* (ime OPC servera). Ukoliko je ova metoda uspešno izvršena, ona vraća identifikator veze. Identifikator veze se čuva u atributu klase *COPCClt*, *hConnection*.

Broj procesnih veličina se određuje metodom *Number_Of_OPC_Items* kojoj je kao ulazni parametar prosleđen identifikator veze sa OPC serverom. Povratna vrednost ove metode predstavlja broj procesnih veličina koji se upisuje u *iNumberOfItems*.

Ukoliko je uspešno izvršeno povezivanje sa OPC serverom povratna vrednost je TRUE. U protivnom je FALSE.

Metodom *ParseTheNameOfItem* se ime procesne veličine koje je oblika tip_promenljive.ime_polja.ime_promenljive razlaže na delove koji se upisuju u odgovarajuće članice klase.

Tip promenljive se upisuje u atribut *pcVariableType*. Ime polja se upisuje u *pcFieldName*. Ime procesne veličine se čuva u *pcVariableNam*e. Ukoliko je tip promenljive digital, onda se u *pcDigitalVariableType* upisuje tip digitalne promenljive.

Ova metoda nema ulaznih parametara i ne vraća vrednost.

Metoda *DetermineItemValue* utvrđuje vrednosti parametara procesne veličine. Ova funkcije nema ulaznih parametara ni povratnu vrednost.

U okviru ove metode prvo se poziva metoda *Number_Of_Item_Properties*, koja je članica klase *COPCClt*, kako bi se odredio broj karakteristika procesne veličine. Zatim se poziva metoda *Get_Item_Property_Description* onoliko puta koliko ima karakteristika procesne veličine. I ova metoda je članica klase *COPCClt*.

Ukoliko je opis karakteristike procesne veličine Item Value onda se pomoću još jedne članice klase *COPCClt*, *Read_Property_Value*, čita vrednost te karakteristike. Ta vrednost se zapisuje u promenljivu *var*, koja je VARIANT tipa. Zatim se metodom *FormatVariant* ta vrednost dodeljuje baferu *pcItemValue*.

FormatVariant je metoda koja utvrđuje vrednosti promenljive VARIANT tipa i upisuje njene vrednosti u bafer *pcItemValue*. Atribut *pcItemValue* je zadužen za čuvanje vrednosti procesne veličine.

Ova metoda nema izlaznu vrednost. Ulazni parametar ove funkcije je promenljiva VARIANT tipa.

Metoda *LinkCreate* je zadužena za kreiranje linkova ka CGI skriptu. Ulazni parametar ove metode je bafer *pcLinkName*. Preko njega se funkciji prosleđuje zahtevano ime polja ili ime procesne veličine. Zavisno od toga šta je zahtevano kreira se link ka CGI skriptu sa odgovarajućim parametrima.

Kreirani link se upisuje u bafer *pcLink*. Zavisno od tipa podataka koji žele da se vide određuje se i okvir web strane na kome će se ti podaci prikazati. Informacije o kom okviru se radi upisuju se u *pcTarget*. Ulazni parametar, *pcLinkName*, ove metode je ujedno i ime kreiranog linka.

Kreirani link, zajedno sa njegovim imenom i imenom okvira u kome će biti prikazan prosleđuju se preko instance klase *HTML*, *html*, metodi te klase *InsertLink*.

Metoda klase *InsertColumnsNamesIntoTable* je zadužena za ispisivanje naziva kolona tabele u kojima će biti prikazani rezultati. Ova metoda poziva metode klase HTML koje su zadužene za kreiranje polja tabele i za ispisivanje teksta.

InsertDataIntoTable je metoda koja zapisuje željene podatke u tabelu rezultata koja će biti prikazana na kreiranoj web strani. I ova metoda poziva metode klase *HTML* koje su zadužene za kreiranje polja tabele i za ispisivanje teksta.

Metoda *InsertDataIntoList* unosi podatke u listu koju treba prikazati na web strani. Ova metoda poziva metode klase HTML koje su zadužene za ispisivanje podataka u listi i za ispisivanje teksta.

Metode *CreateHTMLPage_1, CreateHTMLPage_2, CreateHTMLPage_3, CreateHTMLPage_4* i *CreateHTMLPage_5* su metode koje predstavljaju formate za kreiranje web strana. Zavisno od toga koji podaci se prikazuju i zavisno od želje kako web strana da izgleda poziva se odgovarajuća funkcija od gore navedenih.

Metoda *CreateHTMLPage_Error* kreira HTML dokument ukoliko je došlo do greške prilikom izvršavanja skripta. U ovoj metodi pozivaju se isključivo metode klase *HTML*. Ova metoda nema ulaznih podataka ni povratnu vrednost.

5. ZAKLJUČAK

U radu je izložen koncept realizacije http sprege SCADA sistema. Koncept obuhvata primenu komercijalno dostupnih web servera kao sprege ka klijent strani i razmatra realizaciju veze web servera sa SCADA informacionim sistemom u cilju preuzimanja vrednosti i stanja procesnih veličina.

Izložene su osobine SCADA sistema u takvom okruženju sa osvrtom na sigurnost podataka i samog SCADA sistema. Diskutovane su različite vrste sprega sa osvrtom na mogućnost izmene podataka kao i mogućnosti preuzimanja podataka u realnom vremenu.

Na kraju su data i konkretna rešenja realizacije http sprege sa GAUS SCADA sistemom. Realizovane su sprege sa svim tipovima podataka koji su obezbeđeni od strane GAUS SCADA sistema (OPC server, SQL server i file server). Takođe je realizovana sprega sa SCADA grafičkim podsistemom za iscrtavanje grafičkih prikaza u cilju formiranja istih i na strani web klijenta. Time je realizovana kompletna sprega u smislu preuzimanja raspoloživih podataka od GAUS SCADA kako alfanumeričkih tako i grafičkih. Najvažniji rezultat rada je značajno proširenje mogućnosti GAUS realizacijom http sprege, čime je svakako dokazana i ispravnost koncepta višegodišnjeg razvoja.

Rezultujući sistem svojom otvorenom, slojevitom i standardnom arhitekturom omogućava laku zamenu pojedinih komponenti u smislu zamene primenjenih tehnologija radi lakšeg održavanja i unapređivanja.

Mogućnosti daljeg razvoja ovog aspekta GAUS mogu se sažeti u sledećem:

- Puna integracija realizovanih rešenja i njihovo uključenje u standardnu instalaciju GAUS.
- Realizacija sprega koje bi omogućile interaktivnost na klijent strani kao i mogućnost upravljanja.
- Proširenje podržanih GAUS komunikacionih protokola na TCP/IP i podrška za akviziciju u intranet / internet okruženju.
- Realizacija funkcionalnosti CGI skripta u okviru nekog od standardnih transakcionih servera i time SCADA sistem povezati sa standardnim web aplikacijama

Predloženo rešenje obezbeđuje jednostavan i efikasan pristup podacima SCADA informacionog sistema. Takođe, analizom je obuhvaćeno više različitih tehnologija za preuzimanje podataka, a sa druge strane uniformni pristup sa strane korisnika sistema - web browser kao klijent aplikacija.

6. LITERATURA

[1.] B.Atlagić, "*Jedan pristup realizaciji akviziciono-upravljačkog sistema opšte namene*", magistarski rad, Fakultet Tehničkih Nauka, Novi Sad 1996.

[2.] B.Atlagić, "*Jedan pristup realizaciji generalizovanog akviziciono-upravljačkog sistema opšte namene*", doktorska disertacija, Fakultet Tehničkih Nauka, Novi Sad 2001.

[3.] R.Merriam, "SCADA satellite communications - Four case histories", Pipe Line Industry, April 1988.

[4.] B.Atlagić, Z.Krajačević, V.Mihić, "Rešenje univerzalnog komunikacionog koprocesora GASINT za IBM PC kompatibilan računar ", ETRAN, Zlatibor 1997.

[5.] B.Atlagić, Ž.Jurca, Z.Galović, "*Realizacija akvizicionog sistema za gasovodnu mrežu*", XXXIV ETAN, Zagreb, juni 1990.

[6.] V.Duvnjak, B.Atlagić, Z.Galović, "*Sekundarna obrada podataka u sistemu za nadzor i upravljanje gasovodnom mrežom*", XV simpozijum o informacionim tehnologijama, Jahorina, 1991.

[7.] N.Pešić, B.Atlagić, V.Duvnjak, V.Kovačević, "*Grafička prezentacija podataka u akviziciono - upravljačkom sistemu",* XXXV ETAN, Ohrid, 1991.

[8.] B.Atlagić, V.Maruna, V.Mihić, "Realizacija nadzorno-upravljačkog čvora SCADA sistema u ambijentu lokalne mreže", SAUM, Novi Sad 1995.

[9.] V.Mihić, B.Atlagić, V.Maruna, B. Ađanski, B. Kaćanski, "*Rešenje lokalne mreže za akviziciono - upravljački sistem NIS-GAS*", Telfor, Beograd 1995.

[10.] V.Maruna, V. Kovačević, B. Atlagić, B. Kaćanski, S. Nikolić, T. Maruna, "*Introduction of enhanced RTU for Yugoslav main gas pipeline",* International Pipeline Conference IPC-96, Calgary 1996

[11.] V.Mihić, B.Atlagic, T.Maruna, "*GAUS: An integrated SCADA/DCS control system*", XIV International Conference on Systems Science, Wroclav 2001.

[12.] B.Atlagic, V.Mihić, T.Maruna, "*A methodology for specification and development of control code in industrial DCS application*", XIV International Conference on Systems Science, Wroclav 2001.

[13.] T. Maruna, V. Mihić, V. Duvnjak , "Jedno rešenje razmene poruka između radnih stanica pod OS UNIX i DOS", ETRAN 1995

[14.] I. Kuprešanin, V. Kovačević, Z. Konjević, T. Maruna, "Implementacija FUZZY samopodešavajućeg PID regulatora u RTU", SAUM 1995

[15.] T. Maruna, B. Kaćanski, V. Mihić, B. Atlagić, " Implementacija i održavanje grafičke baze podataka u AutoCAD okruženju", CAD FORUM 1996.

[16.] V. Maruna, B. Kaćanski, T. Maruna, V. Mihić, V. Duvnjak, "Jedno rešenje grafičkog informacionog sistema gasovodne mreže", CAD FORUM 1996.

[17.] T. Maruna, V. Duvnjak, Z. Kaprocki, S. Nikolić, "Realizacija grafičkog informacionog sistema za praćenje tehnološkog stanja gasovodne mreže", ETRAN 1996.

[18.] I. Kuprešanin, T. Maruna, B. Lovre, "Stacionarni simulator kao osnova za inteligentno upravljanje gasovodnom mrežom ", ETRAN 1996.

[19.] T.Maruna, V.Duvnjak, Z.Kaprocki, S.Nikolić, "Realizacija gra grafičkog informacionog sistema za praćenje tehnološkog stanja gasovodne mreže", Naučno-stručni skup o gasu i gasnoj tehnici sa međunarodnim učešćem GAS 96.

[20.] T.Maruna, Z.Kaprocki, G.Tatić, "Razvoj programske podrške tehnološkog informacionog sistema za gasovodnu mrežu", ETRAN 1997.

[21.] V.Mihić, B.Atlagić, T.Maruna, "Koncept I programska podrška generalizovango akviziciono upravljačkog sistema GAUS", ETRAN 1997.

[22.] T.Maruna, V.Mihić, V.Maruna, *"Alati za konfiguraciju SCADA sistema NIS-GAS"*, Naučno-stručni skup o gasu i gasnoj tehnici sa međunarodnim učešćem GAS 97.

[23.] Z. Kaprocki, T. Maruna, S. Nikolić, B. Kaćanski, *"Uvođenje tehnološkog informacionog sistema u NIS-GAS"*, Naučno-stručni skup o gasu i gasnoj tehnici sa međunarodnim učešćem GAS 97.

[24.] V. Mihić, T. Maruna, T. Aleksić, "Razvoj i primena komunikacionih usluga i protokola u akviziciono-upravljačkom sistemu GAUS", ETRAN 1998.

[25.] T.Maruna, V.Mihić, Z.Kaprocki, V.Maruna, "*Realizacija tehnološkog informacionog sistema u mrežnom okruženju*", YUNG 1999.

[26.] V.Mihić, T.Maruna, V.Maruna, U.Grbić "*Konfiguracija nadzorno upravljačke stanice dispečerskog centra NIS-GAS*", YUNG 1999.

[27.] B.Atlagić, V.Mihić, T.Maruna, "*GAUS: Integrisani SCADA/DCS upravljački sistem*", JUNG 2001.

[28.] U.Grbić, B.Atlagić, T.Maruna, "Realizacija programskog alata za automatizovanu izradu projektne dokumentacije", JUNG 2001.

[29.] B.Atlagić, S.Jovanović, T.Maruna, "Realizacija CGI skripta za prikaz procesnih veličina akviziciono upravljačkog sistema", ETRAN 2003.

[30.] Jonathan Pollet, "*SCADA Security Strategy*", PlantData Technologies, 2002.

[31.] Jeffry Dwight and Michael Erwin, "*Special Edition Using CGI*"

[32.] John December and Mark Ginsburg, "*HTML 3.2 and CGI Unleashed*"

[33.] msdn.microsoft.com, "*The Microsoft Open Database Connectivity (ODBC)*"

[34.] www.opcfoundation.org, "The OPC Foundation", OPC Standard 1.0