UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET
PODGORICA

Veselin Manojlović

Hardverska i softverska rješenja za automatizaciju nadzora procesa proizvodnje

MAGISTARSKI RAD

Podgorica 2016.

PODACI I INFORMACIJE O MAGISTRANDU

Ime i prezime: Veselin Manojlović

Datum i mjesto rođenja: 22.09.1980.godine, Nikšić

Naziv završenog osnovnog studijskog programa i godina završetka studija: Elektronika, telekomunikacije i računari, 2007.

INFORMACIJE O MAGISTARSKOM RADU

Naziv postdiplomskog studija: Elektronika

Naslov rada:

„HARDVERSKA I SOFTVERSKA RJEŠENJA ZA AUTOMATIZACIJU NADZORA PROCESA PROIZVODNJE“

Fakultet na kojem je rad odbranjen: Elektrotehnički fakultet Podgorica

UDK, OCJENA I ODBRANA MAGISTARSKOG RADA

Datum prijave magistarskog rada: 05.03.2014.

Datum sjednice Vijeća na kojoj je prihvaćena tema: 05.06.2014.

Komisija za ocjenu teme i podobnosti magistranda: prof.dr Veselin Ivanović, prof.dr Zoran Mijanović, doc.dr Milutin Radonjić

Mentor: prof.dr Zoran Mijanović

Komisija za ocjenu rada: prof.dr Veselin Ivanović, prof.dr Zoran Mijanović, doc.dr Milutin Radonjić, doc.dr Milovan Radulović

Komisija za odbranu rada: prof.dr Veselin Ivanović, prof.dr Zoran Mijanović, doc.dr Milutin Radonjić, doc.dr Milovan Radulović

Lektor

Datum odbrane

Datum promocije

Sažetak

Poboljšanje kvaliteta proizvoda i smanjenje udjela gubitaka u industrijskoj proizvodnji zahtijeva efikasno upravljanje proizvodnim procesom, a za to je potrebno prikupljati raznovrsne podatke iz procesa. Pri tome, menadžmentu, čiji je zadatak unapređenje kvaliteta, naročito je korisna informacija o iskorišćenosti mašina i učinku radnika.

U prvom dijelu rada realizovano je kompletno hardversko i softversko rješenje sistema za nadzor proizvodnje u konkretnom industrijskom okruženju. Uslovi za rad uređaja u industriji obično su vrlo nepovoljni (velike promjene temperature, visok nivo elektromagnetskih smetnji od raznih prekidača, zbog preklapanja strujnih krugova, uključenja i isključenja induktivnih potrošača i sl.), što uslovljava smetnje u komunikaciji. Zbog toga je u radu prikupljanje podataka realizovano pomoću distribuiranih kontrolera koji mogu autonomno da prikupljaju informacije sa senzora i da ih šalju na zahtjev centralnog računara. U ovakvim sistemima uobičajeno je da se koristi multipoint mrežna topologija, gdje su svi uređaji povezani na jednu liniju komunikacije, a čije su prednosti jednostavnost i niska cijena izvođenja. Međutim, ova topologija nije dobro rješenje ukoliko je komunikacija učestala, a potrebno je istovremeno komunicirati sa većim brojem uređaja. Da bi se ovaj problem prevazišao, u radu je realizovana point-to-point mrežna topologija korišćenjem koncentratora sa velikim brojem portova, gdje se komunikacija između centralnog računara i koncentratora obavlja mnogo većom brzinom u odnosu na komunikaciju između koncentratora i ostalih uređaja. Ovo rješenje, pored toga što omogućava učestalu komunikaciju sa većim brojem uređaja istovremeno, obezbjeđuje da smetnje na jednoj liniji komunikacije nemaju uticaja na komunikaciju sa ostalim uređajima. Pri tom, da bi se još više smanjio uticaj smetnji, korišćen je diferencijalni prenos podataka. Softver za praćenje proizvodnje realizovan je tako da omogući brzi pregled sistema, odnosno, da na osnovu datih kriterijuma korektnog rada budu prikazane sve mašine na najpregledniji način, tj. sa minimalnim brojem stanja (uključeno, isključeno, zastoj, normalni rad, preopterećenje), kako bi se lako uočile problematične tačke u procesu proizvodnje. Prilikom izrade softvera nastojalo se da se maksimalno koriste open source alati.

U drugom dijelu rada riješen je jedan konkretan problem proširivanja nekog sistema tako da obuhvati djelove sistema na udaljenim lokacijama ali pod uslovom da se ne vrše izmjene na postojećoj aplikaciji sistema. Realizovano je rješenje koje za komunikaciju koristi GSM mrežu, pri čemu je bilo potrebno riješiti niz problema, kao što su slaba pokrivenost signalom mreže, a često i potpuno odsustvo signala u dužem vremenskom periodu, a uz to i blokiran pristup lokalnim mrežama spolja iz razloga sigurnosti (zatvoreni ulazni i većina izlaznih komunikacionih portova). Za prevazilaženje ovih problema iskorišćen je e-mail kao medijum za prenos komandnih skriptova između udaljenih uređaja, čime je postignuto da uređaji koji komuniciraju ne moraju biti istovremeno prisutni na mreži. Osim toga, najčešće su, bez obzira na sigurnosnu politiku, otvoreni izlazni e-mail portovi. čime je izbjegnuta potreba pristupa mreži spolja jer su isključivo uređaji iz lokalne mreže inicijatori komunikacije sa spoljnim serverom.

Abstract

Product quality improvement and waste reduction in industrial production requires efficient management of the production process, where collection of various data from the process is necessary. However, for management, witch is responsible for quality improvement, information about machine utilization and employees performance is particularly useful.

In the first part of this paper, complete hardware and software solution for production supervision in actual industrial environment is implemented. Industrial equipment often operates in harsh conditions (large temperature variation, high levels of electromagnetic interference caused by various switches, circuits overlapping, inductive elements on and off etc.), which causes communication problems. Therefore, in this work, data acquisition is realized using distributed controllers which can autonomously collect data from sensors and send it on central computer demand. In such systems, it is common to use multi-point network topology where all devices are linked to the same communication line, whose advantages are simpliciy and low cost of implementation. However, this topology is not a good solution when the communication is too frequent and simultaneous communication with large number of devices is required. To overcome this problem, in this paper a point-to-point network topology is implemented using a concentrator with large number of ports, where the communication speed between concentrator and central computer is much greater than the speed between concentrator and other devices. This solution, besides allowing frequent communication with greater number of devices simultaneously, provides that interferences on one communication line have no impact on communication with other devices. Additionally, to lower the impact of interferrences even further, differential data transfer is used. Software is implemented so as to allow a quick overview of the system, ie, to present all machines in the the most transparent way, with minimal set of states (on/off, pause/normal operation, overload) so that the problematic points in production are easy to spot. During software development, open source tools are mostly used.

In the second part of the paper, a particular problem of expanding a system to cover parts of the system in remote locations is solved, but on condition not to make changes to existing application running on the system. Implemented solution uses GSM network for communication, where it was necessary to solve a number of problems, such as poor network coverage and, often, complete absence of signals for a longer period, and also, blocked outside access to local networks for security reasons (closed input and most of output communication ports). To overcome these problems, e-mail was used as a medium for the transfer of command scripts to remote devices, which results in the devices that communicate not to have to be present simultaneously on the network. Moreover, regardless of the security policy, most often, the output ports for e-mail are open, thus avoiding the need for access to the network from the outside because only devices from the local network initiate communication with the external server.

Sadržaj

Uvod

Proizvodni proces je izvor velike količine raznovrsnih podataka. Veliki dio tih podataka uglavnom služi samo za eventualno oporavljanje sistema od ispada, čije uzroke nije moguće rekonstruisati bez detaljnog loga. Za rukovodstvo neke kompanije, međutim, većina tih podataka nije od posebne važnosti. Iz perspektive menadžmenta, osnovni cilj nadzora proizvodnje je osiguranje i poboljšanje kvaliteta. Zavisno od okolnosti, kvalitet se može odnositi na različite faktore kao što su: upotrebljivost proizvoda, raspoloživost, brzina isporuke, uspješnost iz prvog pokušaja, minimalni troškovi, usklađenost sa zahtjevima korisnika, efikasan korisnički servis itd.

Od svih ovih značenja riječi “kvalitet”, dva su od posebne važnosti za upravljanje [2]:

- **Projektovani kvalitet** (eng. quality of design) - kvalitet koji se odnosi na *osobine proizvoda* koje zadovoljavaju korisničke zahtjeve. U tom smislu, kvalitet se vezuje za dohodak. Tako, bolji kvalitet ima svrhu da obezbijedi zadovoljnijeg korisnika i, eventualno, poveća prihod. Međutim, dostizanje boljeg kvaliteta obično zahtijeva investicije, pa tako povećava cijenu proizvoda. Dakle, ovako definisan bolji kvalitet više košta.

- **Realizovani kvalitet** (eng. quality of conformance) - kvalitet kao *oslobađanje od gubitaka*, oslobađanje od grešaka koje rezultiraju u potrebi da se neki posao ponovo odradi ili u otkazivanju na terenu, korisničkom nezadovoljstvu, žalbama korisnika i sl. U ovom slučaju značenje kvaliteta je orijentisano na cijenu, tako da ovako definisan bolji kvalitet obično košta manje.

U prošlosti je, uglavnom, bilo dominantno shvatanje kvaliteta vezano za dizajn, tako da su rukovodioci kompanija često zanemarivali potrebu unapređenja kvaliteta, podrazumijevajući da će se time samo povećati troškovi. Međutim, počev od 80-tih godina XX vijeka, mnoge kompanije su uspješno primijenile mjere za smanjenje troškova od lošeg kvaliteta i time ostvarile značajne uštede [3][7]. Tako npr. odjeljenje za rad sa korisnicima kompanije Xerox je za četiri godine (između 1989. i 1993.), primjenjujući programe smanjenja troškova kvaliteta, ostvarilo uštedu od 200 miliona dolara [4]. Kompanija Tenneco je, počev od 1991.godine, za šest godina smanjila troškove kvaliteta za 1,8 milijardi dolara i time povećala dobit za 900 miliona [5]. Slično, kompanija Westinghouse je uspjela da poveća produktivnost za 15%, umanji škart za 58%, smanji vrijeme ciklusa za 66%, smanji povraćaj proizvoda za 69% i unaprijedi efikasnost servisa za 20% [6]. Svi ovi primjeri pokazuju da se, obraćanjem pažnje na realizovani kvalitet, otvara širok prostor za unapređenje poslovanja kompanije.

Unapređenju kvaliteta uveliko doprinosi i saradnja zaposlenih na svim nivoima organizacije, tako da je za uspješno upravljanje kvalitetom, pored praćenja tehnoloških parametara procesa i rada mašina, značajno i prikupljanje podataka o učinku zaposlenih, kako o vremenu provedenom na radnom mjestu, tako i o njihovoj efikasnosti.

U prvom poglavlju ovog rada biće prikazan sistem za nadzor proizvodnje koji sam realizovao za potrebe fabrike biorazgradljivih ambalaža “Garmin” u Danilovgradu. Realizovano je kompletno hardversko i softversko rješenje sa ciljem da obezbijedi sljedeće funkcije:

- praćenje efikasnosti mašina,

- praćenje vremena iskorišćenja mašina,

- uočavanje zastoja u proizvodnji,
- uočavanje uskih grla,
- praćenje poštovanja tehnoloških parametara,
- praćenje učinka radnika radi pravičnog nagrađivanja.

Biće ukratko opisan i istorijski razvoj upravljanja kvalitetom, kako bi se mogla bolje sagledati važnost navedenih funkcija sistema.

U drugom poglavlju opisan je jedan način povezivanja djelova sistema raspoređenih na velikom prostoru. Rješenje koje je ovdje prikazano razvio sam radeći na proširenju sistema za evidenciju radnog vremena u preduzeću A.D.”Plantaže” Podgorica, gdje sam, upotrebom e-maila kao široko dostupne tehnologije, uspio da riješim niz problema kao što su slaba pokrivenost mrežom, nemogućnost istovremenog pristupa mreži svih učesnika komunikacije, zatvorenost lokalnih mreža iz razloga sigurnosti i sl.

1. Nadzor proizvodnje

1.1. Istorija razvoja kontrole kvaliteta

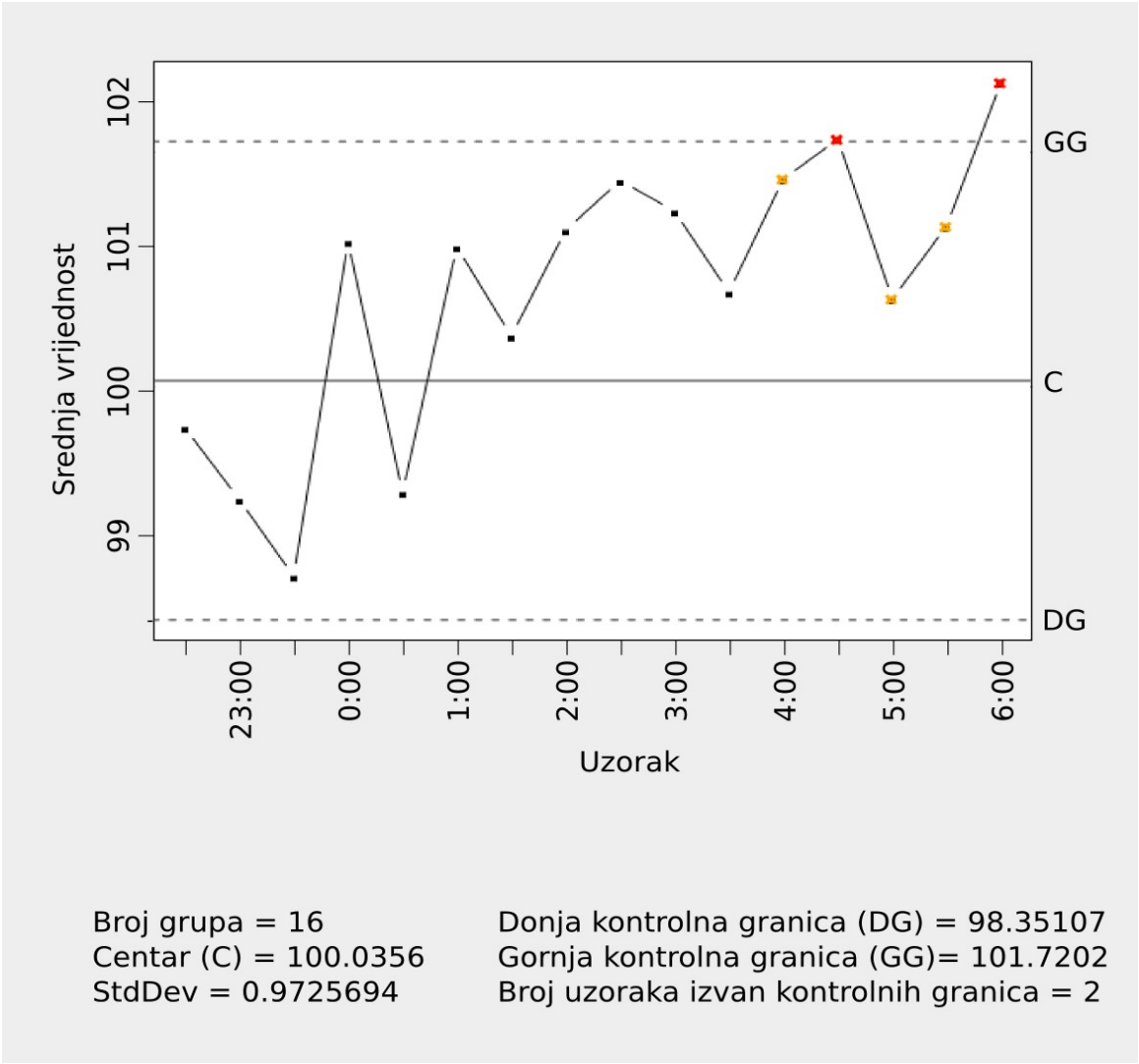
U različitim istorijskim periodima različito se gledalo na značaj kvaliteta, a to je, u velikoj mjeri, bilo uslovljeno obimom i vrstom proizvodnje.

U proizvodnji iz predindustrijskog doba obično je jedan čovjek (majstor, zanatlija) imao kontrolu nad kompletnom izradom nekog proizvoda. On je morao da poznaje tehnologiju kompletnog procesa i sve njegove faze pojedinačno i bio je u prilici da vrlo brzo uoči eventualne propuste nastale u toku izrade proizvoda, a isto tako i da ih odmah otkloni, što je smanjivalo šanse da defektni proizvod dospije do krajnjeg korisnika.

Dolaskom industrijske revolucije, došlo je i do podjele rada na način da je jedan radnik bio zadužen za samo mali dio posla u izradi jednog proizvoda, tako da su se povećale šanse da eventualna greška načinjena u jednoj fazi izrade prođe neprimjećena kroz veći dio procesa, pa i kroz čitav proces proizvodnje. U to vrijeme, jedini način da se spriječi da neispravan proizvod stigne do krajnjeg korisnika bila je inspekcija gotovih proizvoda. Tako je, uporedo sa razvojem industrije i povećanjem obima proizvodnje, rasla i vjerovatnoća proizvodnje defektnih proizvoda, a sa njom i troškovi proizvodnje.

1.1.1. Statistička kontrola procesa

Prve firme koje su prepoznale ovaj problem bile su Bell System Laboratories i Western Electric. Radeći za Western Electric, W.A.Shewhart [8] je u jednom kratkom memorandumu iz 1924.godine postavio temelje za metod statističke kontrole procesa (SPC - statistical process control). Shewhart ukazuje na važnost smanjenja varijacija u proizvodnom procesu i naglašava da kontinualno podešavanje parametara procesa kao reakcija na odstupanja zapravo povećava varijabilnost procesa. Da bi precizirao problem, on varijacije u procesu dijeli na dvije grupe: varijacije kao posljedica očekivanih uzroka (*assignable-causes*) i one koje su posljedica nepredvidivih uzroka (*chance-causes*) i uvodi kontrolni dijagram [9] da bi olakšao razlikovanje ovih dviju grupa. Na kontrolnom dijagramu se može ispratiti da li su parametri proizvedenih primjeraka unutar zadatih okvira.



Slika 1.1: Primjer kontrolnog dijagrama procesa

Iz tekućeg procesa se uzimaju uzorci određene količine i prave se dijagrami varijabilnosti unutar tih uzoraka. Tek kada rezultati počnu da izlaze izvan zadatih okvira, potrebno je analizirati uzroke takvog stanja, tj. utvrditi koji od faktora što djeluju na proces utiču na kvalitet proizvoda. Dok su parametri unutar zadatih granica, proces je u stanju statističke kontrole. Shewhart naglašava da je jedino održavanjem procesa u stanju statističke kontrole, kada do izražaja dolaze samo očekivani uzroci, moguće predvidjeti budući izlaz i ekonomično upravljati procesom.

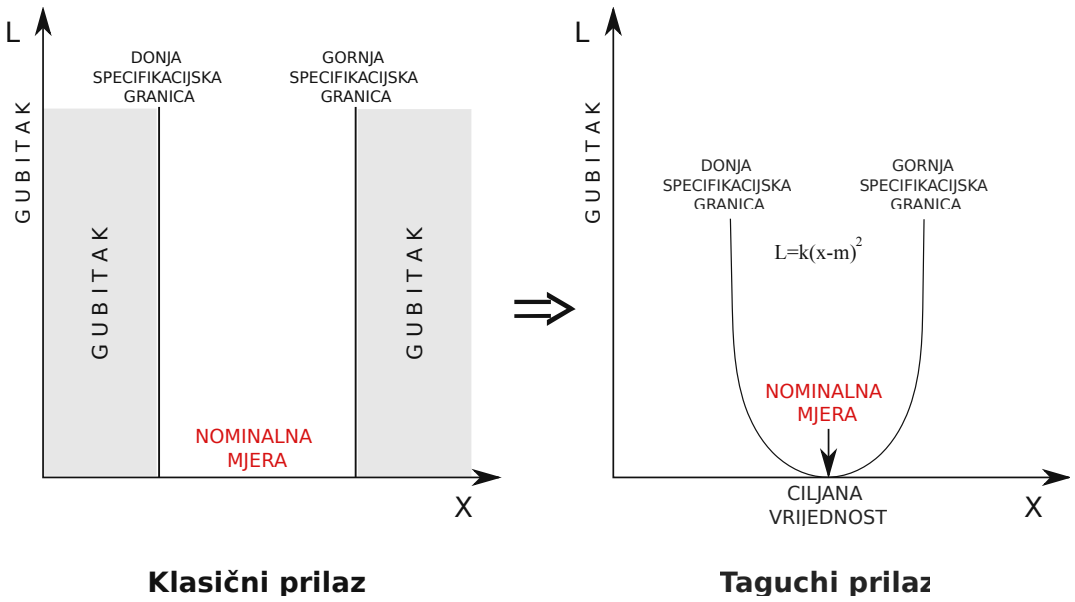
1.1.2. Metod dizajna eksperimenata i Taguči-metoda

Statistička kontrola procesa je bila široko primjenjivana u proizvodnji sve do sredine XX vijeka, naročito u vojnoj industriji tokom II svjetskog rata. Međutim, ovaj metod ne precizira na koji način izlaz zavisi od pojedinačnih parametara procesa. Zato se od 50-tih godina počinje primjenjivati metod dizajna eksperimenata (DOE - design of experiments). Po ovom metodu obavlja se namjerna, planirana manipulacija parametrima kako bi se utvrdio uticaj pojedinih parametara kao i njihova međuzavisnost. I dok se SPC može obavljati direktno u proizvodnom pogonu, za DOE je potrebno u preduzeću formirati posebno odjeljenje za istraživanje i razvoj. Njegov je zadatak isključivo

simulacija procesa i analiza međuzavisnosti njegovih parametara, dok se odjeljenje za proizvodnju bavi samo izvođenjem procesa, ne ulazeći u njegovu analizu.

Veliki doprinos metodu dizajna eksperimenata dao je japanski inženjer i statističar Genići Taguči [10]. Njegova metoda inžinjeringa kvaliteta direktno je uticala na brzi rast konkurentnosti japanske industrije 70-ih i 80-ih godina prošlog vijeka.

Taguči je radio na unapređenju starije statističke metode dizajna eksperimenta koju je 1920-ih godina utemeljio Fišer (R. A. Fisher). Fišer je svoje metode razvijao imajući u vidu poljoprivrednu proizvodnju, tako da su njegovi eksperimenti usmjereni na unapređenje srednjeg prinosa. Taguči, međutim, smatra da za većinu industrijske proizvodnje cilj treba da bude projektovanje tačne vrijednosti ishoda (npr. priključak tačnog prečnika na nekoj mašini ili baterija sa tačnom vrijednošću napona). Jedna od njegovih osnovnih ideja je da je kvalitetan onaj proizvod koji izaziva najmanje troškove za društvo tokom cijelog svog "životnog vijeka". Uvodi funkciju gubitka (Taguchi funkcija) koja predstavlja odnos između gubitka, tj. troškova i tehničkih osobina proizvoda, a kojom je izraženo odstupanje funkcionalne karakteristike proizvoda ili procesa od projektovane - nominalne vrijednosti. Nasuprot klasičnom shvatanju da je proizvod kvalitetan ukoliko se vrijednost njegove posmatrane karakteristike kvaliteta nalazi unutar specifikacijskih granica, Taguči smatra da je optimalni kvalitet postignut za jednu, tj. najpovoljniju vrijednost (nominalna mjera), a da svako odstupanje od te vrijednosti stvara određene gubitke.



Slika 1.2: Tagučijeva funkcija gubitaka

Taguči smatra da inžinjering kvaliteta treba da počne od razumijevanja troškova kvaliteta u različitim okolnostima. Bilo je uobičajeno da se troškovi računaju brojenjem komada proizvedenih van specifikacijskih granica i njihovim množenjem sa cijenom opravke ili škarta. Međutim, Taguči sugeriše da proizvođači prošire svoje horizonte uzimajući u obzir troškove društva. I mada troškovi mogu biti jednostavno posljedica nedovoljne prilagođenosti korisničkim zahtjevima, bilo koji komad proizveden izvan nominalne vrijednosti rezultiraće nekim gubitkom za korisnika ili šire društvo, bilo kroz preranu istrošenost ili zbog neuklapanja sa drugim djelovima koji i sami mogu biti daleko od nominalne specifikacije. Proizvođači su skloni da ignorišu ovakve troškove i da više

obraćaju pažnju na unutrašnje troškove kompanije nego na troškove šireg društva. Međutim, ovi eksterni gubici otežavaju efikasno funkcionisanje tržišta, tako da će svojevremeno i neizbježno pronaći put nazad u kompaniju gdje su nastali, a da će proizvođači radeći na njihovom smanjivanju povećati svoju reputaciju, osvojiti tržište i ostvariti profit.

Slično Shewhart-u, i Taguči smatra da je loš kvalitet uglavnom posljedica velikih varijacija izlaza procesa i da stalne reakcije na promjene faktora sredine samo povećavaju te varijacije. Međutim, Taguči smatra da je najbolja šansa za eliminaciju varijacija unapređenje dizajna proizvoda i dizajna procesa, a ne kontrola njegovog izvođenja [11]. Kvalitet mora biti ugrađen u proizvod i proces proizvodnje, tako da u proizvodnom procesu treba više pažnje posvetiti "off-line" kontroli, kako bi se uklonili problemi koji se javljaju u proizvodnji. S tim u vezi, Taguči je razvio strategiju za upravljanje kvalitetom koja se može koristiti u oba konteksta. Ovaj proces ima tri faze:

- dizajn sistema,
- dizajn parametara i
- dizajn tolerancije.

Dizajn sistema obavlja se na nivou koncepta i uključuje kreativnost i inovacije. Kad se uspostavi koncept, potrebno je odrediti nominalne vrijednosti dimenzija i drugih parametara dizajna. Taguči je zapazio da se, pravilnim izborom vrijednosti parametara koje je moguće kontrolisati (na primer izbor materijala, tehnologija proizvodnje i sl.), može dobiti ***robustan dizajn*** koji ima visoku toleranciju na faktore koje je nemoguće kontrolisati ili su preskupi za kontrolu (varijacije u proizvodnji, uticaji sredine i kumulativna oštećenja). Nakon uspješno obavljenog dizajna parametara pristupa se trećoj fazi, dizajnu tolerancija, gdje se specificiraju uži opsezi tolerancije za faktore dizajna čije promjene imaju negativan uticaj na varijacije izlaza.

1.1.3. TQM, ISO 9000 i "Šest-sigma"

Veliki doprinos razumijevanju potrebe za unapređenjem kvaliteta imala je revolucija kvaliteta u Japanu sredinom XX vijeka. Nakon II svjetskog rata japanski proizvođači su se suočili sa činjenicom da Zapad ne želi da kupuje njihove proizvode, koji su stekli reputaciju lošeg kvaliteta. I dok se, po završetku rata, u Sjedinjenim američkim državama postepeno odustaje od programa kontrole kvaliteta koji su primjenjivani za potrebe ratne vojne industrije, za to vrijeme Japan, u cilju oporavka ekonomije i industrije, prihvata stručnu pomoć stranih eksperata, od kojih su najveći doprinos dali W. Edwards Deming i Joseph M. Juran. Nemogućnost da prodaju svoje proizvode Japanci su shvatili kao alarm i podsticaj da započnu radikalnu reformu kvaliteta. Umjesto da se oslanjaju na inspekciju krajnjeg proizvoda, japanski preduzetnici su pažnju usmjerili na unapređenje organizacije kompletnog procesa, razvijajući tako koncept tzv. "totalnog kvaliteta". Ovo je omogućilo da Japan počne da izvozi proizvode boljeg kvaliteta po nižoj cijeni, što će dovesti do toga da, u roku od jedne decenije, Japan postane druga najmoćnija ekonomija i svjetski lider u kvalitetu [12].

Ovakvom razvoju događaja doprinijela je i činjenica da su se proizvođači sa Zapada dugo držali pretpostavke da je uspjeh Japana povezan sa konkurentnošću u visini cijena. Za to vrijeme, japanske kompanije su nastavile da povećavaju svoje prisustvo na zapadnom tržištu, potiskujući zapadne kompanije koje su bile nemoćne da se uspješno takmiče sa konkurencijom, što je dovelo do krize 70-ih godina XX vijeka.

Međutim, 80-ih godina i na Zapadu dolazi do podizanja opšte svijesti o značaju kvaliteta, što je dovelo do nastanka i razvoja više različitih metoda za njegovo unapređenje. Tako se na osnovama koje su postavili Deming, Juran i rani japanski eksperti za kvalitet, razvija metod totalnog upravljanja kvalitetom (TQM - Total Quality Management). Ovaj pristup ima za cilj ostvarenje dugoročnog uspjeha zadovoljavanjem očekivanja kupaca uz najniže troškove i predviđa da zaposleni na svim nivoima organizacije aktivno učestvuju u unapređenju procesa, proizvoda, usluga i radne kulture u organizaciji. U ovakvom sistemu dat je naglasak na definiciji kvaliteta kao stepena usklađenosti sa potrebama potrošača. TQM predstavlja najopštiji koncept menadžmenta kvaliteta koji uzima u obzir zahtjeve i interese svih zainteresovanih strana organizacije (kupce, zaposlene, akcionare, dobavljače, društvo).

Istovremeno, razvija se ISO 9000 serija standarda za upravljanje kvalitetom sa ciljem da se kompanijama pomogne da dokumentuju elemente koje treba implementirati da bi se održavao efikasan sistem kvaliteta.

Ni TQM ni ISO 9000 ne obezbjeđuju konkretne “alate” za ostvarenje kvaliteta, već predstavljaju samo okvir koji olakšava njegovo dostizanje. Uporedo sa TQM, Motorola 80-tih godina razvija sopstvenu metodologiju za unapređenje kvaliteta, koja je ovoj kompaniji omogućila da izađe iz krize i postane lider u kvalitetu. Ovaj pristup, nazvan “šest-sigma” predstavlja skup mjera koje imaju za cilj da se eliminiše varijabilnost procesa i usluga i rasipanje smanji na 3.4 defekta u milion. Preduzeće se raščlanjuje do nivoa procesa, koji se zatim pojedinačno analiziraju i, tek ukoliko se za neki proces utvrdi da postoji mogućnost unapređenja, na njega se ciljano djeluje sa malom grupom vrhunski obučenih stručnjaka. Za unapređenje postojećih procesa primjenjuje se tzv. DMAIC pristup (Define Measure Analyze Improve Control - definiši, izmjeri, analiziraj, poboljšaj, kontroliši), a za proizvode ili procese koje tek treba razviti primjenjuje se pristup DMADV (Define Measure Analyze Design Verify - definiši, izmjeri, analiziraj, projektuj, provjeri). Šest-sigma koristi razne statističke alate za mjerenje i kontrolu procesa, ali je stvarni naglasak na rukovođenju. Nakon što je 1988. dobila “Malcolm Baldrige” nagradu za kvalitet, Motorola odlučuje da ovu metodologiju objavi pod nazivom “Šest-sigma” i time omogući i drugim kompanijama da upotrebe njeno iskustvo.

1.2. Prikupljanje podataka iz proizvodnog procesa

Industrijska proizvodnja je složen proces koji se obavlja u više faza i na različitim mašinama koje mogu biti raspoređene na širem prostoru. Od mnoštva podataka koji se mogu dobiti iz industrijskog procesa, kako sa senzora, tako i povratnih informacija o stanju aktuatora, nijesu svi podjednako bitni na svim nivoima praćenja. Podaci sa nekih senzora mogu biti bitni samo lokalno, npr. serviseru prilikom neposrednog otklanjanja kvara mašine, dok su neki drugi podaci bitni samo za dugoročnu analizu. Ovo je potrebno uzeti u obzir pri izboru podataka koji će se trajno skladištiti. Raspoloživa trajna memorija obično je ograničena i nije dovoljna da čuva sve raspoložive podatke iz procesa u dužem vremenskom intervalu, pa je potrebno praviti kompromis između dužine snimljenog intervala i opširnosti informacije o procesu. Sa druge strane, snimanje prevelike količine podataka koji pristižu velikom brzinom može predstavljati problem budući da je trajna memorija, obično, usko grlo sistema.

Planirani način prezentacije podataka takođe ima uticaja na projektovanje sistema za prikupljanje podataka. U zavisnosti od širine posmatranog vremenskog intervala, korisnika će zanimati različite vrste podataka. Što je veća količina podataka koja treba da se prikaže, sistem je manje efikasan u pogledu prezentacije u realnom vremenu. Mada se pri odlučivanju čini da je bolje ukoliko su na

raspolaganju raznovrsniji podaci, skup korisnih parametara je obično vrlo uzak.

Zavisno od obima procesa koji se kontroliše, kontrolni sistemi mogu biti različitog stepena složenosti.

PLC (Programmable Logic Controller) uređaji su počeli da se upotrebljavaju kao zamjena za relejnu logiku koja kontroliše uređaje u industrijskim procesima. Obezbjeđuju binarnu logiku, tajmere i analogne i digitalne ulaze i izlaze. Mogu se koristiti za kontrolu sistemskih komponenti u većim sistemima (SCADA i DCS), a mogu biti i primarne komponente u manjim konfiguracijama. Ovakvi sistemi obično ne čuvaju dugoročno podatke iz procesa.

DCS (Distributed Control Systems) su sistemi kod kojih kontrola nije centralizovana, već su kontrolni elementi raspoređeni po podsistemima, tj. svaki podproces je kontrolisan jednim ili više autonomnih kontrolera. Ovi kontroleri su međusobno povezani u mrežu radi komunikacije i nadzora.

SCADA (Supervisory Control And Data Acquisition) sistemi su najprije razvijeni za aplikacije distribucije energije, prirodnog gasa i sl, gdje je potrebno obezbjediti prikupljanje podataka sa potencijalno udaljenih lokacija, sa kojima je ostvarena veza malog protoka i sa velikim kašnjenjima.

1.2.1. Struktura proizvodnog procesa

Proces proizvodnje, o kome se govori u ovom poglavlju, odvija se u fabrici za proizvodnju biorazgradljivih ambalaža “Garmin” u Danilovgradu i obavlja se u četiri faze:

- **dobijanje plastične folije**. Ovo se obavlja pomoću ekstrudera na kojima se folija dobija topljenjem zrna plastične sirovine. Na ovim uređajima se nalazi još i jonizator koji pod visokim naponom (oko 10kV) jonizuje vazduh i tako ubrzanim jonima bombarduje foliju, čime se na površini stvaraju pore, tako da se štampa bolje zadržava. Bitni podaci koji opisuju rad ovih uređaja su dužina proizvedene folije i trenutni napon na jonizatoru.

Slika 1.3: Ekstruder

- **štampanje**. Bitni podaci o radu ovog uređaja su dužina odštampane folije i istezanje folije, koje ima uticaja na kvalitet štampe. Osim slike, na foliji se štampaju i markeri koji određuju na kojim mjestima će se obaviti njeno kidanje.

Slika 1.4: Štampa

- **rezanje folije**. Folija se reže na mjestima označenim markerima odštampanim u prethodnoj fazi. Markeri se detektuju odgovarajućom foto-ćelijom. Za opis rada ovog uređaja bitan podatak je broj odsječenih ambalaža.

Slika 1.5: Sjeckalica (lijevo: Hemingstone, desno: Roll-o-matic)

- **regeneracija**. Škartne ambalaže se u regeneratorima ponovo tope i materijal se ponovo vraća na početak procesa. Folija koja se tako dobija je lošijeg kvaliteta, pa te ambalaže imaju sekundarnu namjenu.

Slika 1.6: Regenerator

1.2.2. Hardver sistema

Hardver sistema za nadzor sastoji se iz nekoliko vrsta uređaja:

- PLC za prikupljanje podataka na mašinama,
- koncentrator i
- centralni PC računar.

1.2.2.1. Umrežavanje komponenata sistema

1.2.2.1.1. Pregled osnovnih mrežnih topologija

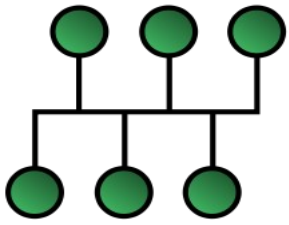
U zavisnosti vrste i uloge uređaja u sistemu, kao i od njihovog prostornog rasporeda, oni se mogu organizovati na različite načine. Organizacije, tj. topologije se mogu podijeliti u dvije kategorije: fizičke i logičke. Fizičke topologije određuju prostorni raspored elemenata mreže, položaj uređaja, instalaciju kablova i sl.. Logičke topologije određuju način na koji podaci prolaze kroz mrežu nezavisno od njene fizičke strukture, tj. dvije mreže sa istom logičkom topologijom mogu imati različitu fizičku topologiju.

Point-to-point topologija

Ovo je najjednostavnija topologija u kojoj je ostvarena direktna veza između dva uređaja. Gledano iz perspektive korisnika, uspostavlja se trajni kanal za komunikaciju, koja nije ometana od strane ostalih kanala. Korišćenjem circuit-switching (vremenski multipleks) i packet-switching tehnologije, point-to-point veza se može dinamički formirati i raskinuti ukoliko više nije potrebna. Prednost on demand (na zahtjev) point-to-point veze postaje izraženija sa povećanjem broja parova između kojih treba uspostaviti vezu.

Topologija magistrale (bus)

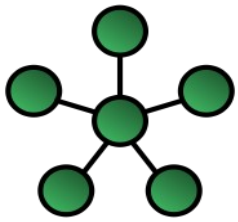
Ovdje su svi uređaji povezani na glavnu magistralu (sabirnicu) u određenim razmacima i odašilju poruke koje se prostiru u oba smjera magistrale, tako da svaki uređaj prima poruke sa svih ostalih i na osnovu adrese prepoznaje koje su poruke njemu namijenjene, a ignoriše ostale. Budući da je potreban samo jedan kabl, implementiranje ove mreže je jeftino. Međutim, održavanje može da bude skupo jer problem u jednoj tački može da onesposobi cijelu mrežu, tj. sistem otkazuje iako otkaže neki manje bitan dio sistema, a otežana je i lokalizacija kvara.



Slika 1.7: Topologija magistrale

Topologija zvijezde

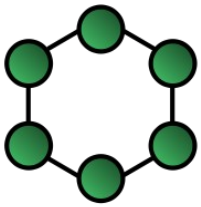
Ovdje su svi čvorovi vezani na centralni čvor point-to-point vezom i sva komunikacija odvija se preko centralnog čvora koji ima ulogu *repeater*-a. Ova topologija je najjednostavnija za implementaciju, a prednost je lako dodavanje novih uređaja. Međutim, osnovni nedostatak je što je cijela mreža osjetljiva na otkazivanje jedne tačke, tj. centralnog čvora.



Slika 1.8: Topologija zvijezde

Lančana (daisy chain) topologija

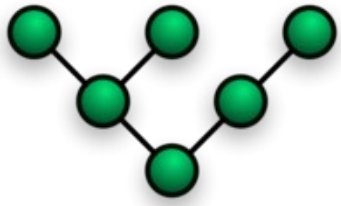
Poslije topologije zvijezde, ovo je najjednostavniji način da se u sistem dodaju novi uređaji i dobija se lančanim nadovezivanjem uređaja jednih na druge. Ova topologija javlja se u dvije forme: linearna i topologija prstena. Linearna topologija je otvorena na krajevima, pa se poruka mora prenositi u oba smjera lanca, tako da svaki uređaj u mreži mora imati po dva prijemnika i dva odašiljača, što povećava cijenu realizacije. Topologija prstena omogućava da se prepolovi broj potrebnih prijemnika i odašiljača u odnosu na linearnu topologiju jer poruka može da putuje samo u jednom smjeru, a da stigne do bilo kojeg uređaja u mreži. Nedostatak ove topologije je što, ako otkaže jedan čvor, cijela mreža gubi funkciju.



Slika 1.9: Topologija prstena

Topologija stabla (hijerarhijska topologija)

Ova topologija se može posmatrati kao skup zvjezdastih topologija organizovanih u hijerarhiju. Periferni čvorovi (listovi) komuniciraju samo sa po jednim čvorom, dok se od ostalih čvorova u mreži očekuje da obavljaju ulogu ripitera ili regeneratora. Slično topologiji zvijezde, proširivanje mreže je jednostavno dodavanjem nove grane u stablo. Zbog hijerarhijske organizacije jednostavnije je izolovati problem na nekom dijelu stabla. Isto tako, i nedostaci su slični onima kod zvjezdaste topologije, tj. otkazivanje jednog čvora osovine odsjeca cijelo podstablo, koje se oslanjalo na taj čvor, od osnovnog stabla.



Slika 1.10: Topologija stabla

Mrežasta (*mesh*) topologija

Ova topologija može biti djelimično ili potpuno povezana.

Potpuno povezana topologija nema potrebe za *switching*-om ili *broadcasting*-om. Međutim, kod ove topologije broj veza raste vrlo brzo sa povećanjem broja čvorova, što je čini nepraktičnom za veće mreže.

Kod djelimično povezane mreže mogu se naći bar dva čvora između kojih postoji dvije ili više putanja, tako da, u slučaju otkazivanja jedne putanje, komunikacija se može obavljati preko druge, rezervne putanje. Ovakva decentralizacija obično služi da bi se kompenzovali nedostaci zvjezdaste ili topologije stabla, gdje cijela mreža zavisi od jednog centralnog čvora.

Slika 1.11: Mrežasta topologija: djelimično povezana (lijevo) i potpuno povezana (desno)

1.2.2.1.2. Komunikacioni protokol

1.2.2.1.2.1. RS-232

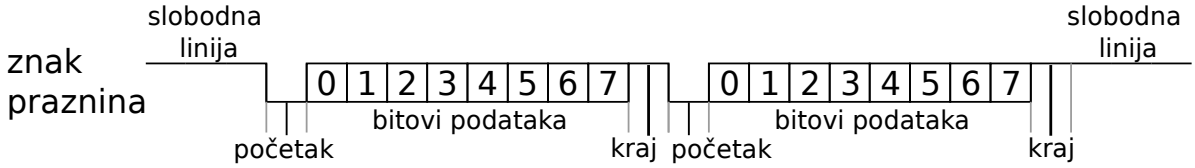
RS-232 je standard za povezivanje DTE (Data Terminal Equipment), kao što je npr. računarski terminal, i DCE (Data Communication Equipment) uređaja, kao što je npr. modem, primjenom asinhrone serijske veze relativno malih brzina. Ovaj standard je bio široko rasprostranjen u serijskim portovima računara, ali ga, zbog relativno malih brzina prenosa podataka, u novije vrijeme postepeno zamjenjuje USB standard.

Standard definiše fizički i logički sloj, tj. električne karakteristike i vremenske promjene signala, značenje signala, kao i veličinu i raspored pinova konektora. Važeća verzija standarda je TIA-232-F, usvojen 1997. godine [14].

Standard definiše point-to-point vezu maksimalne brzine do 20kbps, ali se u novije vrijeme često koriste i brzine veće od 115,2kbps. Maksimalna dužina kabla zavisna je od brzine i, prema standardu, za brzinu od 19,2kbps dozvoljena dužina kabla je do 15m. Podržana je i dupleks veza.

Podaci se prenose naponskim signalima nesimetričnom vezom. Iako standard dozvoljava napone od -25V(logička jedinica) do +25V(logička nula), uobičajeno se koriste ±12V ili niži. Prijemnik mora biti u stanju da signale na ulazu između -25V i -3V primi kao logičku jedinicu, a napone od +3V do +25V kao logičku nulu. Definisana je izlazna otpornost predajnika manja od 50Ω i ulazna otpornost prijemnika između 3kΩ i 7kΩ uz ulaznu kapacitivnost manju od 2,5nF.

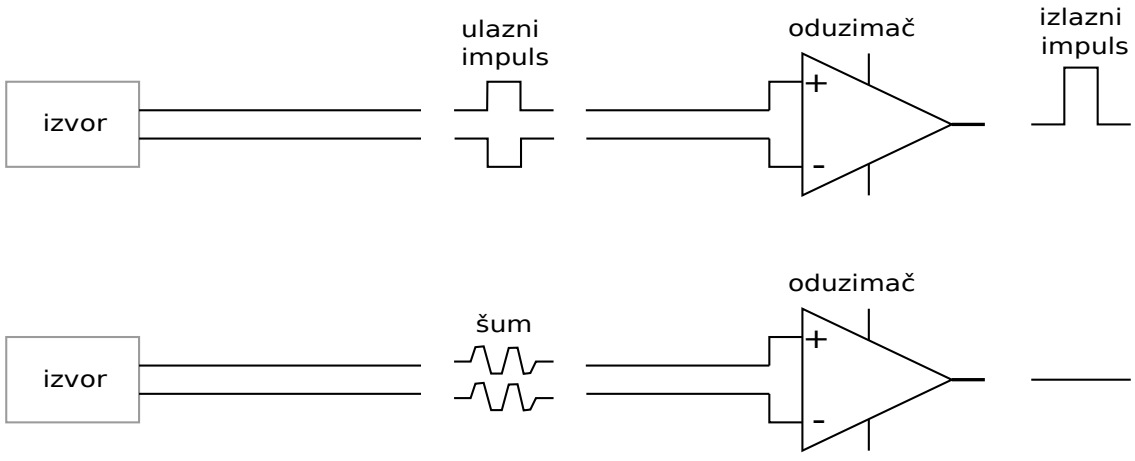
Standard definiše i oblik okvira u koji se pakuje podatak koji se šalje. Okvir se sastoji iz jednog start-bita i jednog stop-bita, između kojih se nalaze 5-8 bitova podataka i, opciono, bit parnosti.



Slika 1.12: RS-232 okvir

1.2.2.1.2.2. RS-422

RS-422 [15] predstavlja proširenje RS-232 standarda i od njega se razlikuje na fizičkom nivou protokola. RS-422 je diferencijalni serijski protokol, gdje se veza veza ostvaruje „balansiranim“ vodovima.



Slika 1.13: Diferencijalni prenos signala

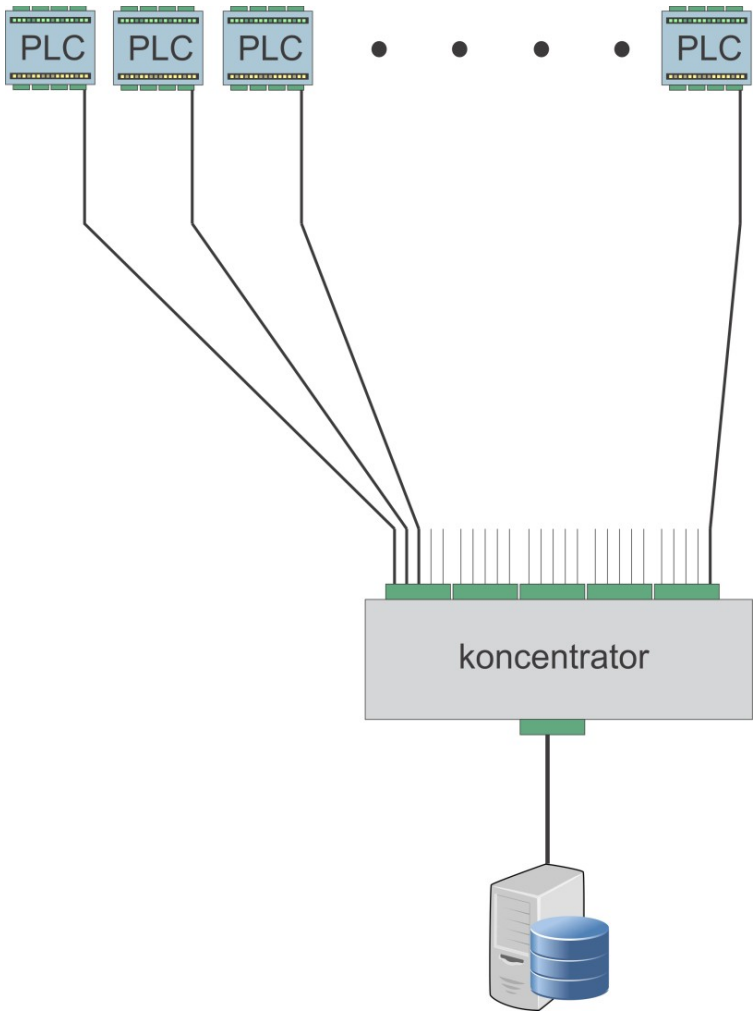
Predajnik na jedan signalni vod predaje korisni signal, a na drugi invertovanu vrijednost tog signala. Prijemnik dobija signal koji predstavlja razliku signala sa dva signalna voda i ima amplitudu duplo

veću od ulaznog signala. Oduzimanjem signala na prijemniku poništava se šum nastao na vodovima, što ovom tipu veze daje prednost u odnosu na nebalansirani RS-232 jer dozvoljava veće dužine vodova i veće brzine komunikacije. Pri brzinama bliskim 100kbps, maksimalna dozvoljena dužina voda je 1200m, a na rastojanjima do 12m moguće je postići brzine komunikacije do 10Mbps. Može se uzeti da je dozvoljena dužina voda u metrima l[m]<=108/baudrate[bps].

Podržan je i tzv. multidrop prenos podataka, sa jednog predajnika na više prijemnika, pri čemu je dozvoljeno maksimalno 10 prijemnika.

1.2.2.1.3. Realizacija mreže u konkretnom sistemu

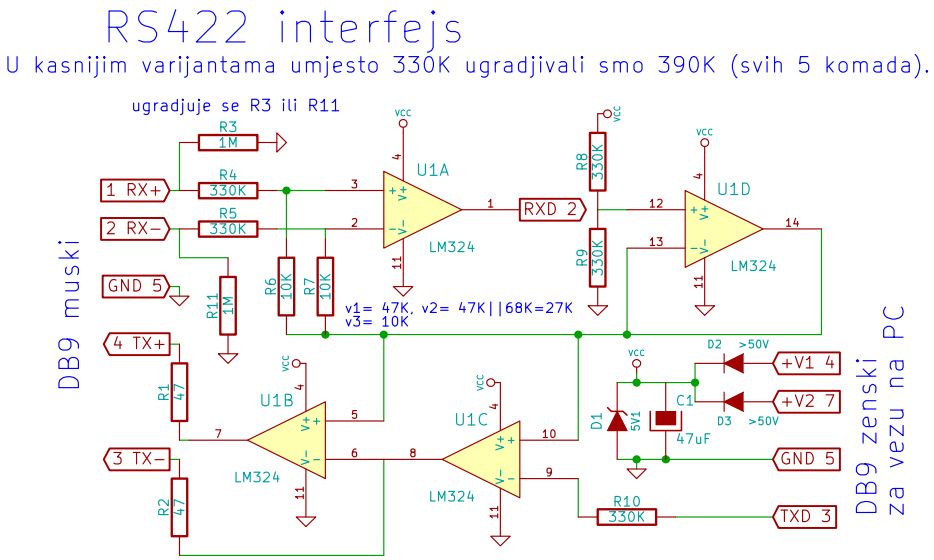
Imajući u vidu da su uslovi za rad uređaja u industriji vrlo nepovoljni zbog velikih varijacija temperature i drugih parametara, kao i raznovrsnih elektromagnetskih smetnji, očekivano je da se u komunikaciji često javljaju smetnje ili prekidi. S obzirom na raspored mašina u fabrici, tj. da su sve mašine približno jednako udaljene od mjesta predviđenog za centralni računar, a uzimajući u obzir već postojeću instalaciju, tj. ožičenje do većine uređaja, odlučio sam se da uređaje povežem u mrežu sa switched point-to-point topologijom, a da se komunikacija obavlja serijskim RS-232 i RS-422 protokolima.



Slika 1.14: Organizacija sistema za nadzor proizvodnje

Pogodnost RS-422 standarda je što ova vrsta veze omogućava da uređaji budu na različitim električnim potencijalima. Osim toga, u odnosu na takođe široko zastupljenu ethernet mrežu, prednosti RS-422 su:

- manja potrošnja struje,
- veće dozvoljeno rastojanje između čvorova mreže, koje je kod etherneta do 100m.
- jednostavnije utvrđivanje i otklanjanje problema budući da se radi o prenosu osnovnog, nemodulisanog signala, pa je, obično, dovoljno samo utvrditi da li je došlo do prekida kabla.



Slika 1.15: Realizacija RS-422 interfejsa

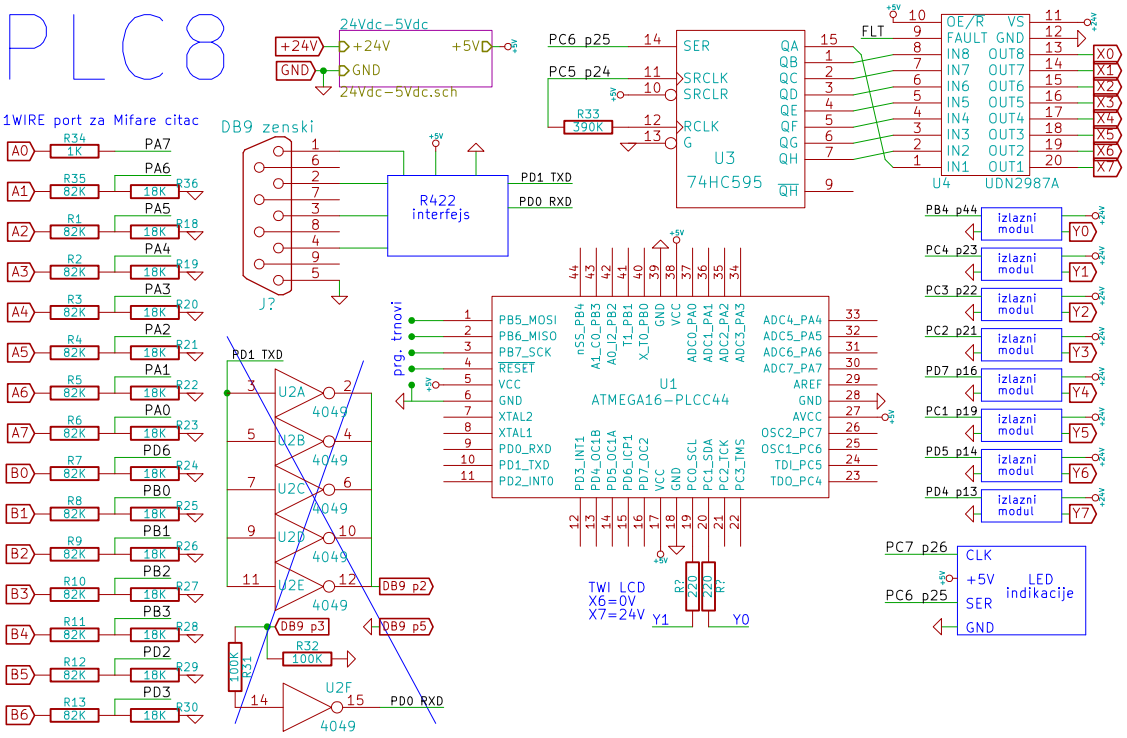
1.2.2.2. Komponente sistema

1.2.2.2.1. PLC uređaji na mašinama

Na svim mašinama koje učestvuju u procesu proizvodnje bilo je potrebno instalirati po jedan PLC. Između više različitih arhitektura koje su mi stajale na raspolaganju (AVR, PIC, 8051, ARM), odlučio sam se za 8-bitnu AVR arhitekturu prvenstveno zbog dužeg prethodnog iskustva u radu sa ovim kontrolerima i relativno dobrog poznavanja njihovih mogućnosti i ograničenja. izabrao sam Atmel-ov AT Mega16 8-bitni AVR mikrokontroler [16]. Ovaj kontroler ima 16kB memorije za smještanje programa, 512B EEPROM memorije u kojoj se čuvaju podešavanja i 1kB RAM-a. Osim toga, kontroler je opremljen sa 32 ulazno/izlazna pina opšte namjene (grupisana u četiri 8-bitna porta). Neki od ovih pinova, pored funkcije digitalnog ulaza/izlaza, mogu imati i druge funkcije, kao što su: analogni ulaz, USART, SPI, PWM i sl..

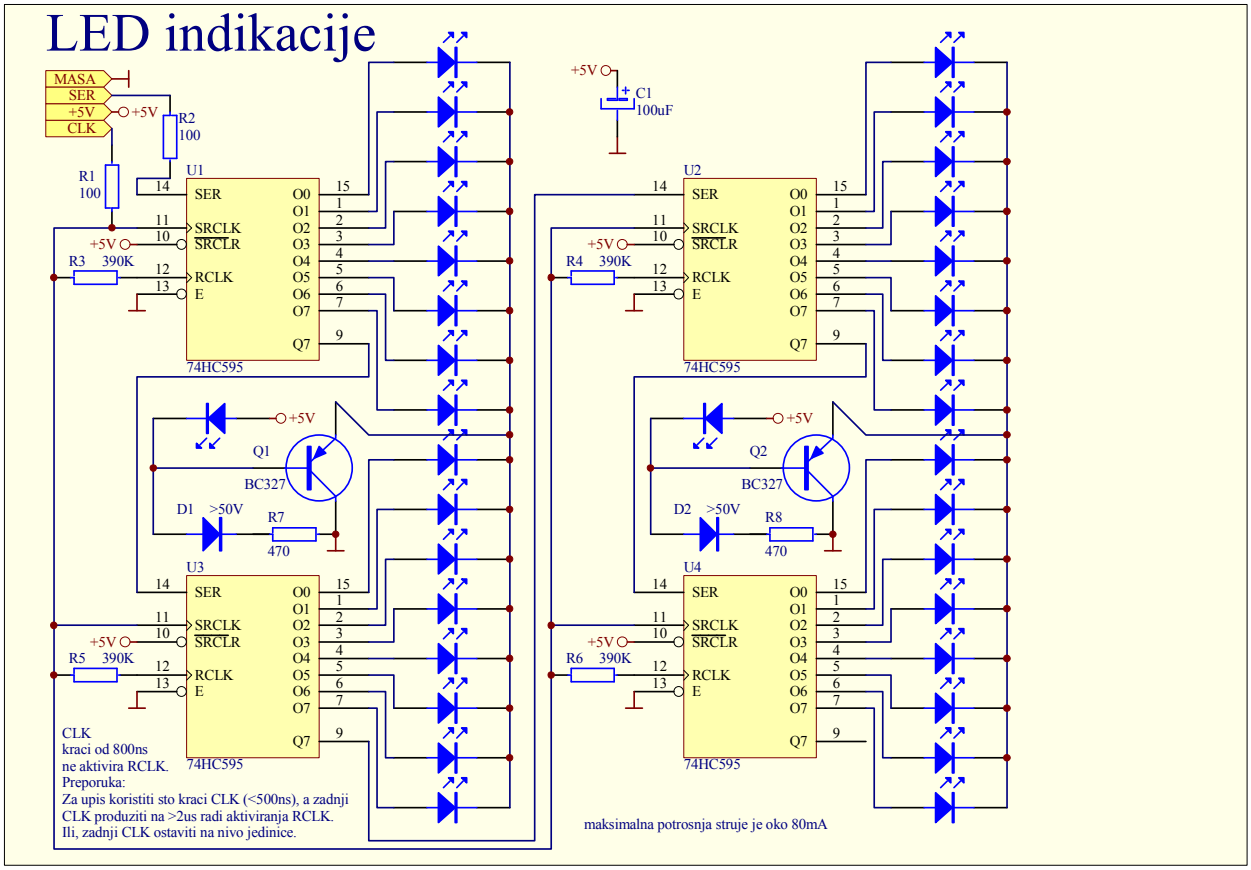
AVR mikrokontroleri imaju Harvard arhitekturu, tj. razdvojene adresne prostore za RAM i za programsku memoriju. Mogu direktno da adresiraju sav raspoloživi RAM, za razliku od nrp. PIC kontrolera koji u jednom momentu mogu adresirati samo 256 bajtova, a za ostatak RAM-a moraju

koristiti tzv. bank-switching mehanizam. Mada C kompajler skriva ovaj mehanizam od programera, on ipak uzrokuje smanjenje brzine. Takođe, za razliku od PIC ili 8051 kojima je za izvršenje jedne instrukcije potrebno više taktova, AVR većinu instrukcija izvršava u toku jednog takta.



Slika 1.16: Električna šema PLC uređaja na mašinama

PLC uređaji treba da sakupljaju različite vrste podataka sa mašina, kao što su: napon (analogni ulaz), dužina proizvedene ili odštampane folije, broj izrezanih komada i sl.. Predvidio sam da za svaki PLC može biti vezan i po jedan čitač kartica kojim bi se evidentiralo prisustvo radnika. Prikupljeni podaci sa svih PLC-ova treba da se serijskom vezom šalju na koncentrator, koji će ih dalje proslijeđivati PC računaru na kome treba da se vrši njihova dalja obrada, čuvanje i prikaz. Kumulativne podatke, kao što su dužina i broj komada, PLC treba da čuva u lokalnim brojačima, čime se obezbjeđuje da podaci ne budu izgubljeni čak i ukoliko na neko vrijeme dođe do prekida komunikacije sa glavnim računarom.



Slika 1.17: LED indikacija na PLC uređajima

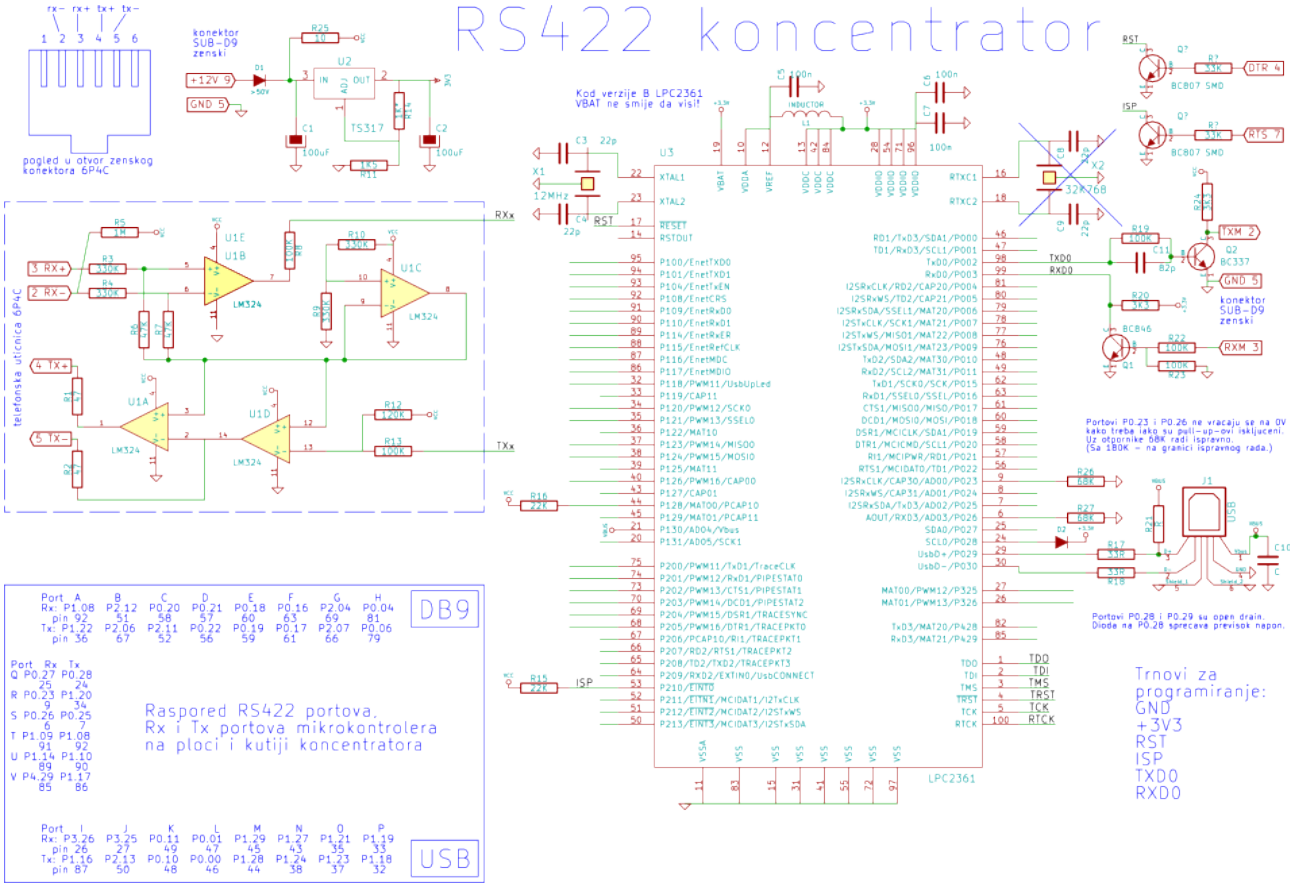
1.2.2.2.2. Koncentrator

Budući da je u konkretnom industrijskom okruženju vjerovatnoća da dođe do elektromagnetskih smetnji na vezama veoma velika, povezivanje svih uređaja na zajedničku magistralu moglo bi da predstavlja veliki problem zato što bi smetnje na jednoj vezi ometale komunikaciju na svim ostalim vezama. Da bih riješio ovaj problem, realizovao sam koncentrator čija je uloga da izoluje kanale komunikacije prema pojedinačnim PLC-ima, tako da smetnja na jednom kanalu nema uticaja na komunikaciju u drugim kanalima. Koncentrator na strani prema centralnom računaru sabira komunikaciju sa svih kanala, ali, pošto je ova veza izmještena iz postrojenja i mnogo kraća od kanala prema PLC-ima, mnogo je manja mogućnost da dođe do smetnji.

Koncentrator je zasnovan na Philips-ovom LPC2366 ARM7 mikrokontroleru [17], koji sadrži veći broj portova opšte namjene, koji mogu imati različite funkcije (digitalni ulazi/izlazi, ADC i DAC, PWM, SPI i sl. U konkretnom slučaju ovi portovi su upotrebljeni kao softverski USART za serijsku komunikaciju sa PLC uređajima, kao i serijski port preko koga komunicira sa glavnim računarom. ARM procesor ima von Neumann-ovu arhitekturu, tj. programska i RAM memorija dijele isti adresni prostor. Počev od ARM7TDMI serije, ARM procesori, pored 32-bitnog seta instrukcija, mogu raditi i sa setom 16-bitnih, tzv. Thumb instrukcija ('T' u TDMI označava Thumb). Za korišćenje ovih instrukcija procesor se mora dovesti u poseban ražim sa posebnim stanjem i skupom registara. Prednost korišćenja ovog seta instrukcija je veća gustina koda, tj. bolja iskorišćenost programske memorije. Ova ušteda dolazi od toga što neki od parametara ovih instrukcija mogu

postati implicitni, za razliku od parametara odgovarajuće 32-bitne instrukcije. Time se, doduše, ograničava broj mogućih funkcija, zbog čega se u nekim slučajevima gubi na brzini budući da za istu funkcionalnost procesor mora da izvrši više instrukcija u odnosu na 32-bitni set. Međutim, u uslovima kad je funkcija ograničena na 16-bitnu adresnu magistralu, Thumb instrukcije su efikasnije jer procesor učitava manje programskog koda.

Brzina kojom koncentrator komunicira sa glavnim računarom treba da bude mnogo veća od brzine komunikacije sa PLC-uređajima. U konkretnom slučaju, za komunikaciju sa centralnim računarom izabrana je brzina od 115200 bps, a za komunikaciju sa PLC uređajima 4800 bps.



Slika 1.18: Električna šema koncentratora

1.2.2.2.3. Centralni računar

Centralni računar je PC sa linux operativnim sistemom. Ovaj računar pokreće aplikaciju za prikupljanje podataka iz procesa proizvodnje, kao i mysql server sa bazom podataka u koju treba smjestiti prikupljene podatke. Novi podaci treba da se upisuju u bazu sa učestalošću od jednog minuta. Zbog toga je, zavisno od broja uređaja u sistemu, potrebno da računar ima hard-disk dovoljno velikog kapaciteta. Na računaru je pokrenut i apache web-server, na kome se nalazi aplikacija za pregled prisustva radnika.

1.2.3. Baza podataka

1.2.3.1. MySQL

Za skladištenje podataka iskoristio sam MySQL bazu. MySQL [18] je najpopularniji open-source sistem za upravljanje relacionom bazom podataka (RDBMS - Relational DataBase Management System). Podržava mnoge napredne mogućnosti i, za razliku od konkurentskih sistema, daje korisniku mogućnost izbora između više različitih mehanizama za smještanje podataka (storage engines). Koriste ga mnogi open-source projekti i veliki web projekti kao što su Wikipedia, Google, Facebook i dr.

MySQL omogućava korišćenje više potpuno različitih mehanizama za smještanje podataka u okviru iste baze. Svaki od tih mehanizama (storage engines) ima svoje prednosti i mane, što određuje oblast njegove upotrebe.

1.2.3.2. MyISAM

MyISAM mehanizam za smještanje podataka je unaprijeđena verzija ISAM mehanizma (ISAM je skraćenica od *indexed sequential access method*), a omogućava brz pristup podacima zahvaljujući ne-klasterovanoj organizaciji indeksa i podataka.

Kod ***ne-klasterovane organizacije indeksa*** tabela podataka nije sortirana, već se novi zapisi dodaju na kraj tabele, dok indeksna tabela sadrži sortirane vrijednosti indeksiranog polja koje povezuje sa identifikacionim brojem zapisa iz tabele podataka (koji predstavlja offset datog zapisa u toj tabeli). Prilikom pretraživanja, da bi se pročitao traženi zapis potrebno je da se izvrši dodatna lookup operacija, tj. pronalaženje zapisa na osnovu njegovog identifikacionog broja dobijenog iz indeksne tabele.

Sa druge strane, kod ***klasterovane organizacije***, tabela podataka je u memoriji fizički sortirana po vrijednostima jedne ili grupe kolona koje sačinjavaju primarni ključ, tzv. klasterovani (grupisani) ključ. Pretraživanje po ovom ključu je veoma brzo jer nije potrebno izvršavati lookup pri prelazu sa indeksnog zapisa na zapis u primarnoj tabeli. Ako indeksni zapis sadrži sve informacije koje se traže upitom, takav indeks se naziva obuhvatajući (covering) indeks. Svi ostali indeksi (sekundarni) se oslanjaju na klasterovani ključ. U MySQL-u ovu organizaciju koristi innoDB engine.

Organizacija fajlova i direktorijuma

MySQL u data_dir direktorijumu sadrži po jedan poddirektorijum za svaku od baza koje se nalaze na serveru. MyISAM za svaku tabelu formira po tri fajla:

- ime_tabele.frm sadrži meta-podatke o definiciji tabele,
- ime_tabele.MYD sadrži zapise sa podacima,
- ime_tabele.MYI sadrži indeksne zapise.

Zahvaljujući ovakvoj organizaciji moguće je premještati tabele sa jednog servera na drugi jednostavnim premještanjem ovih fajlova, što nije moguće kod innoDB tabela.

MyISAM vodi računa o organizaciji indeksnih zapisa, ali ne i zapisa sa podacima, već svaki novi zapis dopisuje na kraj .MYD fajla. Pošto MyISAM ne koristi stranice za smještanje zapisa na disku, ne postoji prazan prostor između zapisa u .MYD fajlu, čime se postiže smanjenje veličine fajla. Za

smještanje indeksnih zapisa koriste se stranice veličine 1kB.

Osim strukture indeksa B-stabla, MyISAM podržava i strukturu R-stabla, kao i FULLTEXT indekse.

Format zapisa

U zavisnosti od tipova podataka koji se koriste, zapisi u osnovnoj tabeli mogu biti fiksne ili promjenljive veličine. Za ove dvije vrste formati .MYD fajla su različiti. Kad su zapisi fiksne dužine, server može brže pronaći traženi zapis u fajlu, oslanjajući se na dužinu zapisa prilikom računanja njegovog offset-a. Druga razlika je što, kod zapisa promjenljive dužine, u slučaju promjene zapisa koji se ne nalazi na kraju fajla, može doći do promjene dužine zapisa, tako da cio zapis više ne može da stane u predviđeni prostor (iako server unaprijed rezerviše nešto slobodnog prostora za ovakve slučajeve). Tada se u zapis dodaje pokazivač na ostatak tog zapisa koji se dopisuje na kraj fajla.

Pored ove dvije vrste formata, postoji još jedan - kompresovani format zapisa.

Zaključavanje tabela

Da bi se očuvao integritet podataka kad više procesa istovremeno pristupa tabeli, MyISAM omogućava zaključavanje (locking) tabele, što znači da, kad jedan proces obavlja neku operaciju nad tabelom, ostali procesi moraju da čekaju dok se ta operacija ne završi. Podržano je samo zaključavanje na nivou cijele tabele, čija je prednost to što zauzima manje resursa, jer se ne mora voditi računa o pojedinačnim zapisima ili poljima. Međutim, nedostatak ovog tipa zaključavanja je što, u slučaju da veliki broj procesa istovremeno pristupa tabeli, svi će oni biti blokirani, bez obzira na to što pristupaju različitim djelovima te tabele. Zavisno od tipa upita nad tabelom, kod ovog tipa zaključavanja razlikujemo tri slučaja:

- READ LOCAL - kad je zadat SELECT upit nad kopijom tabele u memoriji. U ovom slučaju neće biti blokirane INSERT operacije ako se zapisi dodaju na kraj fajla. Zbog toga su MyISAM tabele pogodne za logovanje podataka.
- READ - kad se obavljaju operacije nad .MYD tabelom na disku. U ovom slučaju biće blokirani svi INSERT, UPDATE i DELETE upiti nad tabelom.
- WRITE - kad je zadat UPDATE ili DELETE upit, ili kad je zadat INSERT upit koji popunjava prostor koji je ostao nakon brisanja zapisa.

Nedostaci MyISAM engine-a

Glavni nedostaci ovog mehanizma su nepostojanje podrške za spoljne ključeve i za transakcije koje obuhvataju više komandi.

1.2.3.3. innoDB

InnoDB engine je podržan od strane MySQL-a počev od verzije 3.23, a od verzije 5.5 je podrazumijevani mehanizam. InnoDB rješava neke od nedostataka MyISAM mehanizma, ali po cijenu brzine i povećanog korišćenja resursa sistema.

Podrška za spoljne ključeve

InnoDB obezbjeđuje referencijalni integritet relacija spoljnih ključeva na nivou baze. Kad se u

CREATE TABLE komandu uključi opcija FOREIGN KEY...REFERENCES, prilikom upisa reda u tabelu provjerava se postojanje zadate vrijednosti ključa u referenciranoj tabeli. Referencirana tabela mora imati jedinstveni ključ koji sadrži referenciranu kolonu i tip podataka mora biti isti sa obje strane relacije. Mogu se koristiti i opcije ON UPDATE CASCADE i ON UPDATE DELETE da bi se automatski obradili slučajevi izmjene ili brisanja vrijednosti referenciranog polja.

Zaključavanje tabela

Osim zaključavanja na nivou tabele, innoDB podržava i zaključavanje na nivou reda. Ovaj tip je podrazumijevan, ali se može forsirati zaključavanje na nivou tabele korišćenjem LOCK TABLES komande. Za razliku od zaključavanja na nivou tabele, gdje server vodi računa samo o jednom dijeljenom resursu, kod zaključavanja na nivou reda server mora da održava tabelu sa informacijama o svakom zaključanom redu. Ova tabela je izgrađena nad primarnim ključem osnovne tabele, zbog čega svaka innoDB tabela mora da ima primarni ključ.

U sistemima sa velikim brojem konkurentnih upisa, zaključavanje cijele tabele može da predstavlja problem jer blokira veliki broj procesa, dok zaključavanje na nivou reda omogućava da se blokiraju samo oni procesi čitanja ili upisa koji se odnose na taj red.

Kontrola transakcija

Transakcija predstavlja jedinicu posla koji se obavlja nad bazom, a koji mora da bude:

- atomski, tj. da se ili izvrši u potpunosti ili da se nikako ne izvrši,
- konzistentan - da njegovi rezultati budu u skladu sa ograničenjima postavljenim u samoj bazi,
- izolovan - da ne zavisi od ostalih transakcija i
- da zadovoljava zahtjev trajnosti, tj. da njegovi rezultati budu upisani u trajnu memoriju.

Ovi kriterijumi predstavljaju tzv. ACID(atomicity, consistency, isolation, durability) test koji sistem za upravljanje bazom mora da zadovolji da bi se smatrao bezbijednim za transakcije. InnoDB postiže kontrolu transakcija pomoću MVCC(multiversion concurency control). Podrazumijevani nivo izolacije je REPEATABLE READ. Ukoliko je potreban viši nivo izolacije, innoDB nudi SERIALIZABLE izolacioni nivo koji se postavlja komandom SET TRANSACTION ISOLATION LEVEL prije izvršenja bilo koje komande u tekućoj konekciji.

Struktura fajlova i direktorijuma

Iako MySQL server održava .frm fajl za svaku innoDB tabelu (slično kao kod MyISAM), innoDB održava i svoje posebne meta-informacije o innoDB tabelama. Zbog toga nije moguće (za razliku od MyISAM) jednostavno prenijeti bazu na drugi server jednostavnim kopiranjem fajlova odgovarajućih tabela.

InnoDB održava tabele u tzv. tablespace-u. Tablespace sačinjava više fajlova koji mogu da narastu do maksimalne veličine ograničene operativnim sistemom. Njihov naziv podrazumijevano počinje sa “ibdata”, a zatim slijedi broj. Ovi fajlovi se nadovezuju jedan na drugi i sadrže i zapise podataka i indeksne zapise svih tabela. Autoextend funkcionalnost omogućava da veličina fajlova raste sa porastom veličine baze. Na ovaj način se izbjegava ograničenje operativnog sistema za maksimalnu veličinu fajla, jer tablespace može da sadrži veći broj fajlova (za razliku od pojedinačnih .MYD

fajlova kod MyISAM).

Unutar tablespace-a, dva interna fajla (zvana segmenti) održavaju sve innoDB tabele. Jedan od segmenata se koristi za stranice podataka klasterovanog indeksa, a drugi za ostale (sekundarne) indekse izgrađene nad klasterovanim ključem. Ovakva struktura služi da bi se mogli dodavati veliki blokovi sekvencijalnih zapisa, kako u tabelu podataka, tako i u indeksnu tabelu.

Da bi implementirao sistem za obradu transakcija, innoDB kreira i log fajlove.

Od verzije 4.1.1 postoji opcija da innoDB organizuje fajlove u formatu jedan fajl po tabeli. Međutim, ovo još uvijek ne omogućava premještanje tabela sa servera na server prostim kopiranjem fajlova. Takođe, nije moguće ručno podesiti da se tabele dodijele različitim *ibdata* fajlovima, pa se zbog toga innoDB ne može natjerati da različite tabele smješta na različite diskove.

Organizacija innoDB stranice podataka

InnoDB zapise podataka i indeksne zapise i u memoriji i na disku smješta u 16kB stranice. Ove stranice su organizovane kroz *ibdata* fajlove kao opseg od po 64 uzastopne stranice. Razlog za ovo je da se odjednom alocira veliki prostor u memoriji i na disku, čime se postiže da zapisi na disku budu sekvencijalni i da se sistem održi defragmentiran koliko je to moguće.

Skoro svaki od ovih opsega sadrži podatke vezane za jedan indeks. Jedini izuzetak je jedan opseg koji sadrži katalog stranica, koji sadrži povezano stablo pokazivača na ostale opsege u *tablespace*-u.

Pošto innoDB koristi klasterovanu indeks organizaciju, stranice koje predstavljaju *leaf*-čvorove indeksa sadrže zapise sa podacima. B-stabla sekundarnih indeksa oslanjaju se na stranice podataka klasterovanog indeksa.

Format zapisa iz tabele podataka se znatno razlikuje od formata kod MyISAM mehanizma. Između ostalih polja, treba izdvojiti dva polja bitna za organizaciju zapisa: *heap*-broj i pokazivač na zapis koji sadrži sljedeću vrijednost ključa. Iako se primjenjuje klasterovani indeks, zapisi u memoriji nijesu fizički poređani po vrijednostima tog indeksa, jer bi održavanje toga poretka, prilikom *heap* unosa novog zapisa, zahtijevalo reorganizaciju velikih blokova memorije. Zato se zapis dodaje na prvo slobodno mjesto i pridružuje mu se pokazivač na zapis sa sljedećom vrijednošću klasterovanog ključa.

Interni innoDB baferi

InnoDB kešira informacije u dva veća interna bafera:

- *buffer pool* u kome se keširaju indeksni podaci i
- *log buffer* koji sadrži keširane log-zapise.

Doublewrite bafer i log format

Da bi se osigurala ACID svojstva sistema za kontrolu transakcija, innoDB koristi sistem za logovanje zvani *doublewrite buffer* sistem. To je proces dvojnog upisa koji se dešava kad se zapisi mijenjaju u toku transakcije. InnoDB mora da osigura da će svaka operacija koja mijenja podatke u memoriji biti snimljena na disk (u log) prije nego što se izvrši COMMIT cijele transakcije. Ovim se osigurava da će, u slučaju pada sistema ili otkazivanja, biti moguće rekonstruisati transakciju na osnovu log zapisa. Međutim, ukoliko je odrađen *roll back* prije nego što je stigla komanda COMMIT, tada nema potrebe da se rekonstruiše ova transakcija tokom procesa oporavka sistema.

Prema tome, innoDB ubacuje komande transakcije u log bafer, dok ih istovremeno izvršava nad kopijom zapisa u memoriji, koji su na raspolaganju u pool baferu. Otuda termin doublewrite buffer.

Kad primi COMMIT, innoDB odrađuje flush na disk zapisa iz log bafera.

Pošto je broj zapisa u log fajlu fiksan, a zapisi će se i dalje dodavati, novi zapisi moraju se prepisivati preko starih. InnoDB flushing sistem je cirkularan, tj. kad se dođe do kraja log fajla, novi zapisi se upisuju na početak fajla.

Format log zapisa sadrži log serijski broj (LSN), koji predstavlja 8-bajtni fajl-offset i bajt-offset za određeni log zapis. Osim serijskog broja, log zapis sadrži kompresovanu formu modifikacije podataka koju je napravila odgovarajuća komanda.

U slučaju da se prekorači kapacitet buffer pool-a, sistem je prinuđen da flush-uje stranice podataka na disk da bi se oslobodio stranica podataka koje nijesu skoro korištene. Međutim, prije nego što to uradi, sistem provjerava LSN iz zaglavlja stranice podataka i poredi da li je manji od LSN posljednjeg zapisa u log fajlu. Ako nije, innoDB upisuje log zapise u fajl prije nego što izvrši flush podataka.

Proces oporavka sistema

Sistem za upravljanje transakcijama izvršava checkpointing proces da bi označio u log fajlu da su flush-ovane na disk sve stranice u kojima su vršene izmjene zapisa. Ova oznaka sadrži listu brojeva otvorenih transakcija u momentu kada je urađen checkpoint. Kod innoDB mehanizma, checkpoint sadrži povezanu listu stranica podataka koja još uvijek može biti u neredu zbog transakcija koje čekaju. U određenim vremenskim intervalima pokreće se u pozadini nit koja flush-uje izmijenjene stranice iz buffer pool-a na disk.

1.2.3.4. Kriterijumi za izbor mehanizma za skladištenje podataka

Jedan kriterijum za izbor mogu biti tipovi podataka koji se koriste u tabelama. U zavisnosti od izbora engine-a, određujemo način indeksiranja koji može biti manje ili više pogodan za odabrane tipove podataka. Na primjer, ako je potrebno koristiti FULLTEXT indekse, onda je jedini izbor MyISAM.

MyISAM pruža mogućnost konkurentnog, tj. paralelnog izvršavanja upita čitanja i unosa podataka, pa je idealan za log tabele, gdje se zapisi dodaju na kraj tabele, a rijetko se vrše izmjene i brisanja zapisa.

Takođe, MyISAM je pogodan ako se često koriste SELECT COUNT(*) upiti, jer zahvaljujući svojoj statistici indeksa, daje odgovor skoro trenutno.

Ako je potrebno koristiti transakcije ili spoljne ključeve, tada je innoDB jedini izbor, jer ih MyISAM ne podržava.

InnoDB je takođe dobar izbor za čuvanje sesija web aplikacija jer su ovdje česte UPDATE komande, a zaključavanje na nivou reda obezbjeđuje brz konkurentni pristup za operacije čitanja i upisa.

Nije dobro pretpostaviti da će izabrani engine i u budućnosti biti odgovarajući. Tokom vremena mogu se promijeniti ne samo zahtjevi za prostorom za skladištenje podataka, već i selektivnost, pa i tipovi podataka u tabelama, tako da treba razmotriti i mogućnost naknadne promjene izabranog

mehanizma.

1.2.3.5. Struktura baze sistema za nadzor proizvodnje

U bazu treba smjestiti različite podatke koji se, prema učestanosti transakcija, tj. promjena u bazi, mogu podijeliti u dvije vrste: struktura sistema koja se rijetko mijenja i istorija vrijednosti signala koja se upisuje sa učestalošću od jednog minuta. Na osnovu prethodne analize proizilazi da je za opis strukture sistema, kao i čuvanje aktuelnih vrijednosti signala, najpogodnije koristiti innoDB tabele, dok je za čuvanje istorije vrijednosti signala pogodnija MyISAM tabela.

Osnovni elemenat strukture je signal, koji, uglavnom, odgovara podatku dobijenom sa pojedinačnog porta PLC-a. Tabela signali definisana je na slijedeći način:

```
CREATE TABLE IF NOT EXISTS `signali` (
  `id` int(10) unsigned NOT NULL,
  `naziv` varchar(50) DEFAULT NULL,
  `tip` char(1) NOT NULL,
  `podtip` char(1) NOT NULL,
  `velicina` char(1) NOT NULL,
  `snimanje` char(1) NOT NULL,
  `jedinica` varchar(20) NOT NULL,
  `uredjaj` int(11) DEFAULT NULL,
  `pozicija` int(11) NOT NULL,
  `traka` tinyint(4) NOT NULL,
  `y_min` int(11) DEFAULT NULL,
  `y_max` int(11) DEFAULT NULL,
  `p0` int(11) NOT NULL,
  `p1` int(11) NOT NULL,
  `k0` float(10,4) DEFAULT NULL,
  `k1` float(10,4) DEFAULT NULL,
  `opis` varchar(100) DEFAULT NULL,
  `vrijednost` int(11) DEFAULT NULL,
  `vrijeme_upisa` datetime NOT NULL,

...

  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC;
```

Predviđeno je da *Tip* signala može biti:

- **analogni** - napon, struja i sl, uzima se trenutna vrijednost,
- **digitalni** - razni digitalni ulazi i izlazi, alarmi i sl,
- **minimum, maksimum, prosjek** - u bazu se upisuje minimum, maksimum, odnosno prosjek vrijednosti signala za posmatrani vremenski interval,
- **kumulativni** - brojači komada, dužine folije i sl,
- **relativni** - istezanje folije na štampi, ne dobija se direktno od PLC-a, već se računa na osnovu drugih signala (izmjerena dužina na izlazu i dužina na ulazu u štampu),
- **radnik** - broj kartice radnika koji šalje MIFARE čitač povezan na PLC uređaja. Ovdje se, u

stvari, radi o dva moguća tipa signala. U jednom slučaju se prisustvo radnika mjeri samo za vrijeme prisustva kartice u čitaču, dok u drugom slučaju čitač služi za prijavu ili odjavu radnika. U slučaju prijave/odjave radnika polje podtip određuje da li se radi o čitaču na ulazu ili izlazu.

Polje *veličina* određuje vrstu veličine koja se mjeri (dužina, napon, broj komada i sl.). *Jedinica* određuje mjernu jedinicu za odgovarajuću fizičku veličinu, koja će se koristiti prilikom prikaza.

Polje *snimanje* određuje da li će se u bazu snimati istorija vrijednosti ovog signala. Npr. kod istezanja, u tabeli se specificiraju tri signala: dužina folije na ulazu, dužina na izlazu i istezanje. Pošto se prva dva signala koriste samo za računanje trećeg, tj. istezanja, onda nema potrebe za njihovim snimanjem, već se snima samo izvedeni (izračunati) signal istezanja.

Polje *uređaj* određuje kojem uređaju pripada signal. Ako uređaj ima više traka, polje *traka* određuje za koju je traku signal vezan. Polje *pozicija* određuje značaj signala, tj. da li signal odražava aktivnost uređaja i da li će vrijednost signala biti prikazana u skraćenom pregledu rada uređaja. Ukoliko signal odražava aktivnost uređaja, polja *p0* i *p1* određuju donju i gornju granicu opsega vrijednosti signala za koje se smatra da je uređaj aktivan i da ispravno radi. Ukoliko je vrijednost signala ispod donje granice, smatra se da je uređaj neaktivan, a ako je iznad gornje granice, znači da je došlo do preopterećenja i treba poslati alarm.

Budući da se signali sa PLC-a dobijaju u neobrađenom obliku (npr. u jedinicama A/D konvertora), polja *k0* i *k1* služe za kalibraciju signala u standardne mjerne jedinice.

Istorija vrijednosti signala se u bazu upisuje sa periodom od jednog minuta, ali je, u slučajevima prekida komunikacije sa PLC-uređajima ili prekida rada aplikacije, npr. zbog nestanka električnog napajanja, potrebno u bazu upisivati i trenutne vrijednosti signala, kao i vrijeme tog upisa, za šta služe polja *vrijednost* i *vrijeme_upisa*.

Istorija vrijednosti signala treba da se snima u tabelu *vrijednosti_signala*. Tabela je definisana na slijedeći način:

```
CREATE TABLE `vrijednosti_signala` (
  `id` bigint(20) NOT NULL DEFAULT '0',
  `vrijeme` datetime NOT NULL,

...

  `E1_mfr` int(11) DEFAULT NULL,
  `E3_mfr` int(11) DEFAULT NULL,
  `E4_mfr` int(11) DEFAULT NULL,
  `E5_mfr` int(11) DEFAULT NULL,

...

  `E1_napon` int(11) DEFAULT NULL,
  `E1_suma` int(11) DEFAULT NULL,
  `E1_param3` int(11) DEFAULT NULL,
  `E3_napon` int(11) DEFAULT NULL,
  `E3_suma` int(11) DEFAULT NULL,
  `E3_param3` int(11) DEFAULT NULL,

...
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

Obavezna polja su *id* i *vrijeme*, a ostala polja imaju naziv koji odgovara nazivu signala iz tabele signali. Iskoristio sam MyISAM mehanizam zbog toga što će se nad ovom tabelom zadavati pretežno INSERT i SELECT upiti.

Informacija koja se od PLC-a dobija sa čitača kartica predstavlja broj kartice koji se sastoji od osam heksadecimalnih cifara i dvije kontrolne cifre. Međutim, u tabelu *vrijednosti_signala* se kao vrijednost signala upisuje id radnika. Veza između kartice i radnika je definisana u tabeli *kartice*:

```
CREATE TABLE `kartice` (
  `id` varchar(20) NOT NULL,
  `radnik` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Podaci o radnicima smještaju se u tabelu *radnici*:

```
CREATE TABLE `radnici` (
  `id` int(10) unsigned NOT NULL,
  `ime` varchar(40) COLLATE utf8_slovenian_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_slovenian_ci
ROW_FORMAT=DYNAMIC;
```

Sve mašine u fabrici koje su dio sistema za nadzor vezane su za po jedan PLC. Svakom PLC uređaju odgovara po jedan red u tabeli *uredjaji*.

```
CREATE TABLE `uredjaji` (
  `id` int(10) unsigned NOT NULL,
  `naziv` varchar(40) DEFAULT NULL,
  `funkcija` char(1) NOT NULL,
  `plc_adresa` char(1) NOT NULL,
  `koncentrator_port` char(1) NOT NULL,
  `opis` text,
  `sig_citac_kartica` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC;
```

Polje *funkcija* određuje vrstu mašine za koju je PLC vezan. U konkretnom sistemu to mogu biti: ekstruder, štampa, sjeckalice i regenerator. Svaka od ovih mašina centralnom računaru može da šalje različit broj ili vrstu podataka, pa je različit i format primljenih poruka, koji centralni računar tumači na osnovu polja *funkcija*.

Komunikacija centralnog računara sa PLC-ima i sa koncentratorom treba da se obavlja različitim protokolima, tako da adresa PLC-a i kanal koncentratora na koji je PLC priključen predstavljaju različite podatke. I mada je predviđeno da jednom kanalu odgovara jedan PLC, ostavljena je mogućnost da više njih bude priključeno na jedan kanal. Budući da prilikom prijema centralni računar, prije čitanja poruke sa PLC-a, apstrahuje protokol koncentratora, to je adresa kanala bitna samo prilikom slanja poruke sa centralnog računara.

Za PLC jedne mašine može biti fizički vezan najviše jedan čitač. Međutim, može se desiti da, pored

glavnog radnika, na nekoj mašini radi još i jedan ili više pomoćnih radnika. Zbog toga se za svaku mašinu određuje primarni čitač, što je definisano poljem *sig_citac_kartica*. Primarni čitač neke mašine ne mora biti fizički vezan za PLC te mašine, već može biti vezan i za PLC neke druge mašine, a jedan čitač može biti primarni i za više mašina. U konkretnom slučaju, u fabrici je za PLC na jednom ekstruderu vezan primarni čitač za taj, ali i za sve ostale ekstrudere, što znači da će radniku prijavljenom na taj čitač, u izvještajima biti dodijeljena proizvodnja sa svih ekstrudera, koja je ostvarena za vrijeme dok je bio prijavljen. Čitači vezani za PLC ostalih ekstrudera, ukoliko nijesu registrovani kao primarni za neku drugu mašinu, služe za registrovanje pomoćnih radnika kojima će se računati samo vrijeme prisustva.

1.2.4. Softver sistema za nadzor

Softver sistema obuhvata aplikacije za komunikaciju i prikupljanje podataka na PLC uređajima i na centralnom računaru, program za komunikaciju na koncentratoru i aplikaciju za pregled podataka.

1.2.4.1. Softver za prikupljanje podataka na PLC uređaju

Program koji se izvršava na PLC-u uzima podatke sa analognih i digitalnih ulaza u zavisnosti od mašine za koju je vezan. Osim operativnih podataka mašine, PLC uzima i podatke o prisustvu radnika sa čitača kartica. Podaci se prikupljaju u RAM memoriju, a zatim se, na zahtjev, šalju centralnom računaru. Program je pisan u C jeziku za gcc (GNU C) kompajler, a za funkcije specifične za AVR korištena je *avr-libc* biblioteka.

Za komunikaciju sa PLC uređajima koristi se tekstualni protokol, tako da se komande mogu zadavati i ručno, što omogućava jednostavno uočavanje i otklanjanje problema u sistemu. Komunikaciju uvijek inicira centralni računar, a poruke imaju različit format u zavisnosti od smjera komunikacije.

Paketi koje PLC očekuje sa centralnog računara imaju sljedeći format:

```
[adresa_PLCa komanda podaci] kontrolna_suma
```

gdje je:

"[" – početak paketa,

adresa_PLCa – jedinstvena adresa dužine 1 karakter (adresa '*' je univerzalna adresa i nju prihvata svaki PLC),

komanda – 1 znak(A-Z a-z),

podaci – zavise od formata komande,

(blokovi (komanda podaci) se mogu ponavljati)

"]" – kraj paketa,

kontrolna_suma – suma bajtova paketa unutar zagrada izrazena sa 2 tekstualna heksadecimalna znaka.

Primjer ispravnog paketa:

```
[1U]86
```

gdje je 1=adresa PLC-a, U=komanda, a kontrolna suma ova dva znaka je 0x86

Slijedi dio programa koji obavlja prijem paketa:

```
ISR(USART_RXC_vect){    // prekid, primljen karakter
        rxbuf[wrx++]=UDR;
        if( wrx >= RX_BUFFER_SIZE ) wrx = 0;
}

int main(void)
{
    //  ....
    while (1)
    {
        //  ....
        if(wrx != rdx) { // nesto je stiglo od PC-a preko RS422
            comLED=50;
            uint8_t ch = rxbuf[rdx];
            if( ++rdx >= RX_BUFFER_SIZE ) rdx = 0;
            if(ch=='[') rxStanje=1;
            else switch(rxStanje){
                case 1: if((ch==moja_adresa) || (ch=='*')) {
                            ++rxStanje; rxSuma=ch; pstart=rdx;
                        } // * = svima (broadcast)
                        else rxStanje=0;
                            break;
                case 2: if(ch==']') ++rxStanje;
                        else rxSuma += ch;
                            break;
                case 3: if(ch>='a') ch -= 'a'-10;
                        else if(ch>='A') ch -= 'A'-10;
                        else ch -= '0';
                        rxSuma -= ch*16; ++rxStanje;
                        break;
                case 4: if(ch>='a') ch -= 'a'-10;
                        else if(ch>='A') ch -= 'A'-10;
                        else ch -= '0';
                        rxSuma -= ch; rxStanje=0;
                        if(rxSuma==0) {
                            uradiPaket(pstart); comLED = 250;
                        }
                }
            }
        }
        //  ....
    }
}
```

Zavisno od mašine na kojoj se nalazi, PLC treba da prikuplja različite vrste podataka. Program učitan u svaki PLC je identičan, a funkcija konkretnog PLC-a se određuje upisom odgovarajuće vrijednosti u EEPROM i može se zadati daljinski odgovarajućom komandom. Slijedi dio programa koji obavlja prikupljanje podataka u zavisnosti od vrste mašine:

```
int main(void)
{
```

```
//  ....
moja_funkcija =  eeprom3read(EE_MOJA_FUNKCIJA);
if(moja_funkcija ==0xFF) moja_funkcija='?';
//  ....
while (1)
{
    //  ....
    uint8_t duz, kom;
    switch(moja_funkcija){
    case F_EKSTRUDER:
        mjeriNapon();
        duz = brojacC;  // impulsi metraze dolaze na ulaz A4
        duzinaA += (uint8_t)(duz - brojacPreC); brojacPreC=duz;
        break;
    case F_STAMPA_MATILA:
    case F_STAMPA_YOLANDA:
        // induktivni davac sa ulaza A2
        duz = brojacA; duzinaUl += (uint8_t)(duz-brojacPreA);
        brojacPreA=duz;
        // induktivni davac sa ulaza A4
        duz = brojacC; duzinaA += (uint8_t)(duz-brojacPreC);
        brojacPreC=duz;
        // induktivni davac sa ulaza A5
        duz = brojacD; duzinaB += (uint8_t)(duz-brojacPreD);
        brojacPreD=duz;
        break;
    case F_SJ_MATILA_STARA:
    case F_SJ_TURKINJA:
    case F_SJ_ROLOMATIC:
        kom = brojacA; komadiA += (uint8_t)(kom-brojacPreA);
        brojacPreA=kom; break;
    case F_SJ_PAPIRNA:
        // kapacitivni senzor noza na ulazu A2
        kom = brojacA; komadiA += (uint8_t)(kom-brojacPreA);
        brojacPreA=kom;
        // inkrementalni impulsi sa ulaza A4
        duz = brojacC; duzinaA += (uint8_t)(duz-brojacPreC);
        brojacPreC=duz;
        // fotoceliju za sada ne gledamo
        break;
    case F_SJ_MATILA_NOVA:
        kom=brojacA;
        // ako je ukljucena traka A - ulaz A4
        if(dUlazi & 16) komadiA += (uint8_t)(kom-brojacPreA);
        // ako je ukljucena traka B - ulaz A5
        if(dUlazi & 32) komadiB += (uint8_t)(kom-brojacPreA);
        brojacPreA=kom;
        break;
    case F_SJ_HEMINGSTONE:
        // brojanje kesa
        kom = brojacA;
        if(kom-brojacPreA) { // Za svaku kesu
            uint16_t duzA, duzB;
            TCCR0=0; // zaustavljamo brojac
            if(TIFR & _BV(TOV0)) { ++tcnt0h; TIFR = _BV(TOV0);}
            duzA = 256*tcnt0h + TCNT0;
```

```
                    // resetujemo i ponovo ukljucujemo brojac
                    TCNT0=0; TCCR0=7; tcnt0h=0;
                    duzB=TCNT1; TCNT1=0;
                    duzinaA += duzA;
                    duzinaB += duzB;
#define MIN_DUZ 300 // 1 impuls = 1mm
                    if(duzA > MIN_DUZ)
                        komadiA += (uint8_t)(kom-brojacPreA);
                    if(duzB > MIN_DUZ)
                        komadiB += (uint8_t)(kom-brojacPreA);
                }
                brojacPreA = kom;
                break;
            case F_REGENERATOR: // regenerator
            default:
                break;
            }
            //  ....
        }
        //  ....
}
```

Odgovor koji PLC vraća ima oblik:

```
{ adresa_PLCa funkcija_PLCa status PLCa kod_komande podaci }
kontrolna_suma
```

gdje je:

"{" – početak paketa,

adresa_PLCa – 1 karakter,

funkcija_PLCa – 1 karakter

status PLCa – 1 karakter

kod_komande – 1 znak(A-Z a-z) , isti znak kao komanda PC-ja,

podaci – podaci prema formatu odgovora,

(blokovi (kod_komande podaci) se mogu ponavljati)

"}" – kraj paketa,

kontrolna suma – suma bajtova unutar zagrada izražena sa 2 tekstualna heksadecimalna znaka.

Najčešće korišćena komanda je 'U', koja vraća vrijednost mjerenih veličina na mašinama. Broj mjerenih veličina i format vrijednosti zavise od tipa mašine, odnosno od funkcije PLC-a.

Pored čitanja i obrade podataka sa ulaza, moguće je i komandovanje izlazima PLC-a, što se obavlja slanjem komande 'I'. Ova mogućnost je ostavljena sa namjerom da se automatski upravlja radom ekstrudera, odnosno jonizatora na ekstruderima. Naime, ekstruder se smije uključiti tek nakon nekoliko minuta od uključenja jonizatora, a ukoliko dođe do prestanka rada jonizatora, potrebno je isključiti i ekstruder. Međutim, ova funkcionalnost još nije implementirana na ostalim nivoima

sistema.

Slanjem 'K' komande PLC-u moguće je zadati stanje LED na čitaču kako bi se signaliziralo da li je sistem prepoznao prijavljenog radnika, a komanda kao rezultat vraća kod pročitane kartice (8 karaktera) i kontrolnu sumu (2 karaktera). Ako kartica nije prisutna, odgovor komande je “00000000 01”, dok, ukoliko čitač nije priključen ili je došlo do problema u komunikaciji sa čitačem, rezultat komande će biti “00000000 00”. Komunikacija između PLC-a i čitača kartica obavlja se 1-Wire protokolom.

U odgovoru, pored rezultata izvršavanja zadate komande, PLC vraća i trenutno stanje izvršavanja programa, tj. da li je došlo do reseta i koji je bio razlog. Razlog reseta može se pročitati iz MCUCSR registra. Nakon reseta, centralni računar treba sljedećom komandom da obriše sadržaj ovog registra, tj. da stanje vrati na normalno.

Slijedi funkcija za obradu primljenog paketa i slanje odgovora:

```
void uradiPaket(char pok){
    SaljiChar('{'); txSuma=0; SaljiChar(moja_adresa);
    SaljiChar(moja_funkcija); SaljiChar(status);
    uint8_t cmd=0;
    char txt[20]; uint8_t u18;
    while(1){// Analiziramo jednu po jednu komandu i vracamo odgovor
        uint8_t ch=rxbuf[pok]; if( ++pok >= RX_BUFFER_SIZE ) pok=0;
        if(ch==']') break; // kraj paketa sa komandama
        switch(cmd){
        case 0:
            SaljiChar(' '); // razmak izmedju 2 komande
            switch(ch){
            case 'U':// čitanje izmjerenih veličina sa mašina,
                // format odgovora zavisi od vrste mašine
                // cmd='U'; Stavicemo ovo ako bismo
                // ocekivali dodatne podatke uz komandu.
                u18 = dUlazi / 8;
                switch(moja_funkcija){
                case F_EKSTRUDER:
                    sprintf(txt, "U %03X %04X",
                        (uint16_t) fnapon, duzinaA);
                    break;
                case F_STAMPA_MATILA:
                case F_STAMPA_YOLANDA:
                    sprintf(txt, "U %04X %04X %04X",
                         duzinaU1, duzinaA, duzinaB);
                    break;
                case F_SJ_HEMINGSTONE:
                    sprintf(txt, "U %04X %04X %04X %04X",
                        komadiA, komadiB, duzinaA, duzinaB);
                    break;
                case F_SJ_PAPIRNA:
                    sprintf(txt, "U %04X %04X",
                        komadiA, duzinaA);
                    break;
                case F_SJ_TURKINJA:
                case F_SJ_MATILA_STARA:
                case F_SJ_ROLOMATIC:
```

```
                sprintf(txt, "U %04X", komadiA); break;
            case F_SJ_MATILA_NOVA:
                sprintf(txt, "U %04X %04X",
                    komadiA, komadiB);
                break;
            case F_REGENERATOR: // regenerator
            default: sprintf(txt, "U ? ");
            }
            SaljiTekst(txt);
            break;
        case 'K':
            cmd='K'; // Ocekujemo dodatne podatke uz komandu
            iwireKomanda(IWIRE_POSALJI_MIFARE_ID);
            sn[0]=iwirePrimi8();    sn[1]=iwirePrimi8();
            sn[2]=iwirePrimi8();    sn[3]=iwirePrimi8();
            sn[4]=iwirePrimi8();
            sprintf(txt, "K%02X%02X%02X%02X %02X",
                sn[0], sn[1], sn[2], sn[3], sn[4]);
            SaljiTekst(txt); break;
        case '@':
            cmd='@'; // ocekujemo dodatne podatke uz komandu
            SaljiChar(ch); break;
        case 'I': cmd='I';
            SaljiChar(ch); break;
        case 'S': cmd='S';
            SaljiChar(ch); break;
        case ' ':
            break; // Ignorisemo razmak izmedju komandi
        default: // nepoznata komanda
            SaljiChar(ch); SaljiChar('?');
        }
        break;
    case 'K':
        cmd=0; // ne ocekujemo vise paramatara
        switch(ch){
        case '0': iwireKomanda(IWIRE_UGASI_OBJE_LED); break;
        case '1': iwireKomanda(IWIRE_UPALI_CRVENU); break;
        case '2': iwireKomanda(IWIRE_UPALI_ZELENU); break;
        }
        break;
    case '@':
        cmd='A'; // ocekujemo jos jedan paramatar
                 // uz ovu komadu
        // upisujemo novu adresu
        // i odmah zatim uzimamo ju iz eeproma
        eeprom_write_byte( (uint8_t*)EE_MOJA_ADRESA, ch);
        eeprom_write_byte( (uint8_t*)EE_MOJA_ADRESA+1, ch);
        eeprom_write_byte( (uint8_t*)EE_MOJA_ADRESA+2, ch);
        moja_adresa = eeprom3read(EE_MOJA_ADRESA);
        if(moja_adresa ==0xFF) moja_adresa='?';
        SaljiChar(moja_adresa); break;
    case 'A': cmd=0; // ne ocekujemo vise paramatara
        // upisujemo novu funkciju
        // i odmah zatim uzimamo ju iz eeproma
        eeprom_write_byte((uint8_t*)EE_MOJA_FUNKCIJA,ch);
        eeprom_write_byte((uint8_t*)EE_MOJA_FUNKCIJA+1, ch);
```

```
                eeprom_write_byte((uint8_t*)EE_MOJA_FUNKCIJA+2, ch);
                moja_funkcija = eeprom3read(EE_MOJA_FUNKCIJA);
                if(moja_funkcija ==0xFF) moja_funkcija='?';
                SaljiChar(moja_funkcija); break;
            case 'S': cmd=0; // ne ocekujemo vise paramatara
                status=ch; SaljiChar(status);
                eeprom_write_byte( (uint8_t*)EE_OSCCAL, OSCCAL);
                eeprom_write_byte( (uint8_t*)EE_OSCCAL+1, OSCCAL);
                eeprom_write_byte( (uint8_t*)EE_OSCCAL+2, OSCCAL);
                break;
            case 'I': cmd=0;
                SaljiChar(ch);
                switch(ch){
                case '0': clr(Y6); clr(Y7); break;
                case '1': set(Y6); clr(Y7); break;
                case '2': clr(Y6); set(Y7); break;
                case '3': set(Y6); set(Y7); break;
                }
                break;
            }
        }
        sprintf(txt, "}%02X", txSuma);
        SaljiTekst(txt);
    }
```

Da bi se eliminisao uticaj smetnji na ulazima, primjenjuje se filtriranje. Slijedi dio koda za čitanje ulaza:

```
    void procitajUlaze(void){
            //PORTA = 0; PORTB &=~0x07; PORTD &=~0x4f;
            //DDRA = 0; DDRB &=~0x07; DDRD &=~0x4f;
            static uint8_t ulaziA[2], ulaziB[2], ulaziD[2];
            static uint8_t prolaz = 0;
            static uint8_t pA, pB, pD;
            if(++prolaz >1) prolaz = 0;
            ulaziA[prolaz] = PINA;
            ulaziB[prolaz] = PINB;
            ulaziD[prolaz] = PIND;
            pA |= ulaziA[0] & ulaziA[1]; // & ulaziA[2] & ...
                                         // za jace filtriranje
            pA &= ulaziA[0] | ulaziA[1]; // | ulaziA[2] | ...
            pB |= ulaziB[0] & ulaziB[1];
            pB &= ulaziB[0] | ulaziB[1];
            pD |= ulaziD[0] & ulaziD[1];
            pD &= ulaziD[0] | ulaziD[1];
            dUlazi=0;
            // ulaz A0 rezervisan za vezu sa 1wire Mifare citacem
            if(pA & _BV(6)) dUlazi |= 2; // ulaz A1
            if(pA & _BV(5)) dUlazi |= 4; // ulaz A2
            if(pA & _BV(4)) dUlazi |= 8; // ulaz A3
            //  ...
            if(pD & _BV(6)) dUlazi |= 0x100; // ulaz B0
            if(pB & _BV(0)) dUlazi |= 0x200; // B1
            if(pB & _BV(1)) dUlazi |= 0x400; // ulaz B2
            //  ...
```

```
}
#define HISTEREZIS 4
#define LIMIT 4
ISR(TIMER2_OVF_vect, ISR_NOBLOCK){
    static signed char filterA2, filterA3, filterA4, filterA5;
    TCNT2|=128;//Perioda prekida=128us. Prescaler=1/8. Fclk=8MHz
    if(ulazA2){
        if(filterA2 < LIMIT) if(++filterA2 == 0) {
            filterA2 += HISTEREZIS; ++brojacA;
        }
    } else if(filterA2 > -LIMIT) if(--filterA2 == 0)
        filterA2 -= HISTEREZIS;
    if(ulazA3){
        if(filterA3 < LIMIT) if(++filterA3 == 0) {
            filterA3 += HISTEREZIS; ++brojacB;
        }
    } else if(filterA3 > -LIMIT) if(--filterA3 == 0)
        filterA3 -= HISTEREZIS;
    //      ...
}

void mjeriNapon(void)
{
    unsigned int rez;
    // Koristimo internu referencu 2,56V. Ulaz je PA6 (A1 na PLC-u)
    ADMUX = _BV(REFS1) + _BV(REFS0) + 6;
    // start ADC, brisemo ADIF, preskaler=64
    ADCSRA = _BV(ADEN)+_BV(ADSC)+_BV(ADIF)+_BV(ADPS2)+_BV(ADPS1);
    while(!(ADCSRA & _BV(ADIF))); // Cekamo da se konverzija zavrsi
    // Uvijek se mora prvo citati nizi dio ADC data registra
    rez = ADCL + 256*ADCH;
    fnapon=0.9*fnapon + 0.1*rez; // filter za usrednjavanje
}
```

Budući da, zbog čestih elektromagnetskih smetnji, može doći do oštećenja sadržaja EEPROM-a, svaki podatak se upisuje na po tri uzastopne lokacije, tako da se kasnije može rekonstruisati ukoliko bar dvije lokacije imaju isti sadržaj. Slijedi funkcija za čitanje podataka iz EEPROM-a:

```
uint8_t eeprom3read(int adr){
    uint8_t data, data1, data2, data3;
    data1= eeprom_read_byte( (uint8_t*) adr);
    data2= eeprom_read_byte( (uint8_t*) adr+1);
    data3= eeprom_read_byte( (uint8_t*) adr+2);
    // 2 od 3 odlucuju
    if(data2==data3) data=data2;
    else data=data1;
    // popravke po potrebi
    if(data1 != data) eeprom_write_byte( (uint8_t*) adr, data);
    if(data2 != data) eeprom_write_byte( (uint8_t*) adr+1, data);
    if(data3 != data) eeprom_write_byte( (uint8_t*) adr+2, data);
    return(data);
}
```

Kao izvor takta procesora PLC koristi interni RC oscilator čija frekvencija može da varira, tako da je za ostvarivanje stabilne komunikacije potrebno vršiti kalibraciju oscilatora na osnovu trajanja

bitova u primljenoj poruci. Podešavanje frekvencije internog oscilatora vrši se promjenom vrijednosti OSCCAL registra. Slijedi dio koda koji obavlja kalibraciju:

```
#define RXD    (PIND & _BV(0)) // PD0, USART RX pin
ISR(TIMER2_OVF_vect, ISR_NOBLOCK){ //  TIMER2 OVERFLOW prekid
    static signed char filterA2, filterA3, filterA4, filterA5;
    TCNT2 |= 128; // Perioda prekida=128us. Prescaler=1/8. Fclk=8MHz
    cPrekid = 1; ++cth;
    //      ...
}

ISR(USART_RXC_vect){    // prekid, primljen karakter
    //      ...
    cPrekid = 1;
}

void kalibracija_baud(void){
    unsigned char ctl;
    cth=0; // cth se svakih 128us uvecava u prekidnom programu
    while(1){
        if(cth > 3) return; // ogranicavamo vrijeme do 2-3 *128us
        // Ako se tokom prekida pojavila NULA, napustamo.
        if(cPrekid & 1) { if(!RXD) return; else cPrekid=0;}
        // if(cPrekid & 1) se izvrsava jednom
        // masinskom instrukcijom BRS, a if(cPrekid) sa nekoliko
        if(!RXD) break;
    }
    // Mjerimo priblizno trajanje NULE, pa nam prekidi ne smetaju.
    ctl=TCNT2; cth=0; if(cPrekid & 1) return;
    while(1){
        if(cth >2) return; // ako je nula preduga
        if(RXD) break;
    }
    if(cth==0) return; // ako je nula bila prekratka
    // mjerimo trajanje JEDINICE - hvatamo pocetak nule
    while(1){
        TCNT2; if(cth > 4) return; // ako je jedinica
        if(!RXD) break;
        // Ako se tokom prekida pojavila NULA, napustamo.
        if(cPrekid & 1) { if(!RXD) return; else cPrekid=0;}
    }
    unsigned char c0, c1, c0new;
    c0=TCNT2; c1=cth; c0new=TCNT2;
    if(cPrekid & 1) return;
    if(c0new<c0) { c1=cth; c0=c0new; } // Ako je u trenutku citanja
                    // tajmer prevalio, uzimamo novije ocitavanje.
    uint16_t trajanje; trajanje=c1*128 + c0 - ctl;
    if(trajanje < 416 - 80) return; // ako je greska veca od 20%
    if(trajanje > 416 + 80) return;
    if(trajanje < 416 - 4) ++OSCCAL; // ako je greska > 1%,
                                     // popravljamo OSCCAL
    if(trajanje > 416 + 4) --OSCCAL;
}
```

1.2.4.2. Softver na koncentratoru

Koncentrator treba da obezbijedi dvosmjernu komunikaciju između centralnog PC računara sa jedne i većeg broja PLC uređaja sa druge strane. Pri tome je potrebno da komunikacioni kanal prema bilo kom PLC-u bude izolovan od komunikacionih kanala prema ostalim PLC-uređajima.

Veza sa centralnim računarom ostvarena je korišćenjem namjenskih UART pinova kontrolera, dok je veza sa PLC uređajima ostvarena korišćenjem pinova opšte namjene kojima je softverski obezbijeđena UART funkcija. Brzina komunikacije sa centralnim računarom (brzi kanal) je 115200bps, dok je brzina komunikacije sa PLC uređajima (spori kanal) 4800bps. Da bi se realizovala komunikacija sa ovim brzinama, potrebno je, korišćenjem PLL (Phase-locked loop) množača/djelitelja, frekvenciju takta, sa 12MHz (frekvencija internog oscilatora) podići na 51MHz. Slijedi dio koda kojim se obavlja inicijalizacija kontrolera (softver je pisan u C programskom jeziku za gcc kompajler):

```
static void lowInit(void) // setup clocks and processor port pins
{
    PCONP = (1<<1) + (1<<3); // Ukljucujemo tajmer0 i uart0
    // 0x20=enable main oscllator,
    // 1=configure pin I/O's for Fast GPIO
    SCS = OSC_ENABLE + USE_FIO;
    // cekamo da se oscilator zaleti
    while(!(SCS & OSC_STAT)); // (1<<6)
    CCLKCFG= 31;     // OVO JE VEOMA VAZNO jer nasledjujemo
                     // premali djelitelj od boot loadera
    //  Biramo glavni oscilator kao izvor takta za PLL
    CLKSRCSEL = CLKSRC_MAIN_OSC; // 01;
    // postavljamo PLL množač & djelitelj.
    //  vrijednosti izračunate na osnovu config.h
    // PLLCFG = PLLCFG_MSEL | PLLCFG_PSEL;
    PLLCFG = PLL_MUL; // 16;  N=0+1, M=16+1
                      // umnozavamo Fin=12MHz
                      // da dobijemo Fcco=2*Fin*M/N=408MHz
    PLLCON = PLL_ENABLE; // PLLCON_PLLE enable PLL
    PLLFEED = PLL_FEED_BYTE1; // 0xAA; Ove dvije komande
    PLLFEED = PLL_FEED_BYTE2; // 0x55; moraju se izvršiti za redom.
    //  čekamo PLL lock
    //      while (!(PLLSTAT & PLLSTAT_LOCK))  continue;
    for(int i=5000; --i; ) asm("nop");
    // enable & connect PLL
    PLLCON = PLL_CONNECT; // PLLCON_PLLE | PLLCON_PLLC;
    PLLFEED = PLL_FEED_BYTE1;
    PLLFEED = PLL_FEED_BYTE2;
    // CPU clock najmjestamo na 408/8= 51MHz.
    // Moze samo 0 i neparni brojevi
    CCLKCFG= CPU_CLK_DIV; // 7;
    PCLKSEL0 =0x88; // Periferal CLK za UART0 i TC0 = cpuclk/2.
                    // To je 2 puta brze od pocetne vrijednosti.
    // setup & enable the MAM
    MAMTIM = MAMTIM_CYCLES;
    MAMCR = MAMCR_FULL;
```

```
}
static void sysInit(void)
{
    // set the interrupt controller defaults
    //      ...

    uart0Init(UART_BAUD(115200), UART_8N1, UART_FIFO_OFF);
    // popravljamo baudrate da bude tacno 115200
    //  U0FDR = (int8_t)(11*16 + 8);
    T0PR = 1771-1; // set the prescale divider - za periodu 69,44us
    T0TCR = TCR_ENABLE; // enable timer 0
    // postavljamo TX portove kao izlazne
    uint32_t        bits[5];
    for(int i=0; i<5; ++i) bits[i]=0;
    for(int kanal=0; kanal<MAX_KANAL; ++kanal)
        bits[TXPORT[kanal]/0x20] |= TXBIT[kanal];
    for(int i=0; i<5; ++i)
        *(volatile unsigned int *)(FIO_BASE_ADDR + i*0x20)
            |= bits[i]; // FIOiDIR |= txbits[i];
    // Iskljucujemo pull-ups na RX pinovima
    //!! PAZNJA SVAKA IZMJENA PORTA MORA SE PRENIJETI
    // TAKODJE U funkciju PrijemSaSvihKanala()
    #define PML(i) (2<<(2*i))
    #define PMH(i) (2<<(2*(i-16)))
    PINMODE0 = PML(11) + PML( 1) + PML( 4); // za bitove P0.0-P0.15
    PINMODE1 = PMH(20) + PMH(21) + PMH(18) + PMH(16) + PMH(27) +
               PMH(23) + PMH(26); // P0.16-P0.31
    PINMODE2 = PML( 9) + PML(14); // P1.0-P1.15
    PINMODE3 = PMH(29) + PMH(27) + PMH(21) + PMH(19); // P1.16-P1.31
    PINMODE4 = PML( 8) + PML(12) + PML( 4); // P2.0-P2.15
//      PINMODE5 = PMH(  ); // P2.16-P2.31
//      PINMODE6 = PML(  ); // P3.0-P3.15
    PINMODE7 = PMH(26) + PMH(25); // P3.16-P3.31
//      PINMODE8 = PML(  ); // P4.0-P4.15
    PINMODE9 = PMH(29); // P4.16-P4.31
    // initialize the watchdog timer
    WDTC = 4000; // Interni RC oscilator radi na 4MHz.
                 // Ovo daje WD periodu od 1ms
    // !! Ako se WD perioda napravi prekratka,
    // Flas magic nece moci da preuzme kontrolu nad mikrokontrolerom
    WDFEED=0xAA; WDFEED=0x55;
    WDMOD = 3; // WDRESET+WDENABLE;
               // WD podesen da resetuje mikrkokontroler
    WDFEED=0xAA; WDFEED=0x55;
}
```

Svaki spori kanal ima bafer od 8 okteta za slanje podataka ka glavnom (brzom) kanalu (u kodu označen kao *kaGlav*) i bafer od 64 okteta za prijem podataka od glavnog kanala (*odGlav*). Brzi kanal nema svoje bafere, već koristi bafere sporih kanala. Slijedi dio koda sa definicijama bafera:

```
/*** SPORI KANALI 4800 bps ***/
#define MAX_KANAL  22
#define KA_GLAV_SIZE 16
// Stanje kanala u toku prijema (u kojoj je fazi, sta radi)
unsigned char cSt[MAX_KANAL];
```

```
// Prijemni znak. Popunjava se bit po bit u toku prijema.
unsigned char cPrijem[MAX_KANAL];
// Bafer za prijem sa kanala.
// Podaci se prebacuju u odlazni bafer 115200.
unsigned char kaGlav[MAX_KANAL][KA_GLAV_SIZE];
unsigned char kaGLav_wrx[MAX_KANAL]; // index za upisivanje u bafer
unsigned char kaGLav_rdx[MAX_KANAL]; // index za citanje iz bafera
unsigned char kaGlavSaljeKanal; // Koji je tekuci kaGlav bafer
                                // izabran za slanje prema PC-u
unsigned char kaGlavGledaKanal; // Koji bafer gledamo
                                // da li ima nesto za slanje
unsigned char promjenaKanala; // ukazuje da treba poslati escape
                              // char za izbor kanala ~0..~7
#define OD_GLAV_SIZE 64
// Predajni znak. U toku predaje se pomjera bit po bit.
unsigned char cPredaja[MAX_KANAL];
// Koji bit treba poslati 0-9. 0=START, 1-8=DATA, 9=STOP
unsigned char cTxBit[MAX_KANAL];
// Bafer za predaju sa kanala. Podaci stizu sa mux kanala 115200.
unsigned char odGlav[MAX_KANAL][OD_GLAV_SIZE];
unsigned char odGLav_wrx[MAX_KANAL]; // index za upisivanje u bafer
unsigned char odGLav_rdx[MAX_KANAL]; // index za citanje iz bafera
// koji kanal je izabran i
// u koji odGlav bafer treba puniti podatke sa PC-a
unsigned char odGlavKanal;
uint16_t timeout;
#define ESCP  '~'
#define BV(i) (1UL << i)
#define _P(i) (i*0x20)
#define FIO_BASE_SET 0x3FFFC018
#define FIO_BASE_CLR 0x3FFFC01C
#define setTX(kanal) (*(volatile unsigned int *)(FIO_BASE_SET +
TXPORT[kanal]) = TXBIT[kanal])
#define clrTX(kanal) (*(volatile unsigned int *)(FIO_BASE_CLR +
TXPORT[kanal]) = TXBIT[kanal])
const uint8_t TXPORT[MAX_KANAL]={ _P(1),  _P(2),  _P(2),  _P(0),
                                  _P(0),  _P(0),  _P(2),  _P(0),
                                  _P(1),  _P(2),  _P(0),  _P(0),
                                  _P(1),  _P(1),  _P(1),  _P(1),
                                  _P(0),  _P(1),  _P(0),  _P(1),
                                  _P(1),  _P(1) };
const uint32_t TXBIT[MAX_KANAL]={BV(22), BV( 6), BV(11), BV(22),
                                 BV(19), BV(17), BV( 7), BV( 6),
                                 BV(16), BV(13), BV(10), BV( 0),
                                 BV(28), BV(24), BV(23), BV(18),
                                 BV(28), BV(20), BV(25), BV( 8),
                                 BV(10), BV(17) };
```

Ono što primi sa glavnog kanala, koncentrator odmah raspoređuje u *odGlav* bafere pojedinih kanala. Takođe, kad treba da šalje na glavni kanal, to radi direktno iz *kaGlav* bafera pojedinih kanala. Da bi se naznačilo na koji od sporih kanala treba proslijediti poruku sa glavnog, prilikom slanja podataka koncentratoru poruku treba započeti sa “~k”, gdje je k oznaka sporog kanala. Slično, kad koncentrator prosljeđuje podatke sa sporih na glavni kanal, na početku svakog paketa šalje sekvencu “~k”. Slijedi dio koda koji vrši raspoređivanje podataka po odgovarajućim kanalima:

```
int main(void)
{
    lowInit();
    sysInit();
    while(1) {
        WDFEED=0xAA; WDFEED=0x55; // wd reset
        // Jedan prolaz kroz ovu petlju traje 208 us,
        // a sihronizovan je pomocu 3 poziva
        // funkcije PrijemSaSvihKanala na 3*69.44us.
        if(timeout) --timeout;
        int i;
        // 1. PROLAZ
        PrijemSaSvihKanala(); // traje oko 20us
        for(i=0; i<7; ++i) PredajaNaKanal(i);
            // predaja na prvih 7 kanala. Traje oko 8us
            KomunikacijaSaPC(); // traje oko 4us
        // 2. PROLAZ
        PrijemSaSvihKanala();
        for(i=7; i<14; ++i)
            PredajaNaKanal(i); // predaja na drugih 7 kanala
        KomunikacijaSaPC();
        // 3. PROLAZ
        PrijemSaSvihKanala();
        for(i=14; i<22; ++i)
            PredajaNaKanal(i); // predaja na zadnjih 8 kanala
        KomunikacijaSaPC();
    }
    return 0;
}

void PrimajSaKanala(uint8_t kanal, uint32_t rxbit){//NIJE TTL RS232.
    //Kao +-12V RS232 ali sa nivoima 0V i +5V. START bit = +5V.
    char cs, cp;
    cs=cSt[kanal];
    if(cs==0) { if(rxbit) cs=4; } // Ako je start bit
    else if((++cs & 4) == 0){ // Prvi bit uzimamo u 4. prolazu
       if(cs>=72) { //STOP bit
            cs=0;
            if(!rxbit){ // ako je STOP bit ispravan
                kaGlav[kanal][kaGLav_wrx[kanal]++]=cPrijem[kanal];
                kaGLav_wrx[kanal] %= KA_GLAV_SIZE;
            }
        } else {
            cs+=5; // Ostale bitove uzimamo u svakom 3. prolazu
            cp=cPrijem[kanal];
            cp>>=1; if( !rxbit ) cp |=0x80;
            cPrijem[kanal]=cp;
        }
    }
    cSt[kanal]=cs;
}

uint32_t lastTC;
void PrijemSaSvihKanala(void)
{
    uint32_t now;
```

```
    // Sinhronizacija na 69.44us
    do  now = T0TC; while( now==lastTC); lastTC=now;
    //!! PAZNJA SVAKA IZMJENA PORTA MORA SE PRENIJETI
    //   TAKODJE U INICIJALIZACIJU RX portova u sysinit()
    uint32_t rxbit;
    rxbit=FIO2PIN & BV( 8); PrimajSaKanala(0, rxbit); // A port
    rxbit=FIO2PIN & BV(12); PrimajSaKanala(1, rxbit); // B
    rxbit=FIO0PIN & BV(20); PrimajSaKanala(2, rxbit); // C
    //      ...
    rxbit=FIO1PIN & BV(14); PrimajSaKanala(20, rxbit); // U
    rxbit=FIO4PIN & BV(29); PrimajSaKanala(21, rxbit); // V
}

void PredajaNaKanal(int kanal){// NIJE TTL RS232.
    //Kao +-12V RS232 ali sa nivoima 0V i +5V. START bit = +5V.
    if(cTxBit[kanal] == 0) {
        if(odGLav_wrx[kanal] != odGLav_rdx[kanal]) {
            // Ako nesto ima u baferu, poslacemo na kanal
            cPredaja[kanal]=odGlav[kanal][odGLav_rdx[kanal]++];
            odGLav_rdx[kanal] %= OD_GLAV_SIZE;
            setTX(kanal); // Postavljamo START bit
            cTxBit[kanal]=1;
        }
    } else if(++cTxBit[kanal]>=10) {
        // STOP bit i ponovo smo spremni
        clrTX(kanal); cTxBit[kanal]=0;
    } else {
        // Nadalje postavljamo jedan bit
        if(cPredaja[kanal] & 1) clrTX(kanal); else setTX(kanal);
        cPredaja[kanal]>>=1;
    }
}

void KomunikacijaSaPC(void)
{
    // Saljemo PC-u iz kaGlav bafera
    /*
        Varijanta 1: Salje ~k (oznaka kanala) samo kada ima podatke
                     i kanal se promijenio
        Varijanta 2: Kao varijanta 1, ali povremeno (na 4 sec)
                     ubacuje ~k cak i kada se kanal nije promijenio
        Varijanta 3: Salje ~k svaki put kada iznova pocne da prazni
                     bafer (da salje podatke PC-u).
                     Tada iza ~k ide najcesce 1 do 2 znaka, jer se
                     bafer samo toliko napuni izmedju 2 obilaska
                     svih 8 bafera.
    */
#define VARIJANTA 2
    if(uart0TxEmpty()) { // Ako je UART spreman za slanje ka Glavi
        static char umetanje=0;
        if(umetanje) {
            if(umetanje==1){ // umetanje broja kanala
                kaGlavSaljeKanal = kaGlavGledaKanal;
                uart0Putch('A' + kaGlavSaljeKanal);
            } else // umetanje==2, umetanje escape karaktera
                uart0Putch(ESCP);
```

```
                umetanje=0;
            } else if(kaGLav_rdx[kaGlavGledaKanal]
                    != kaGLav_wrx[kaGlavGledaKanal]) {
                if(kaGlavGledaKanal != kaGlavSaljeKanal) {
                    umetanje=1; uart0Putch(ESCP);
                } else { // Ako imamo jos podataka u baferu, saljemo
                    char ch;
                    ch=kaGlav[kaGlavGledaKanal]
                       [kaGLav_rdx[kaGlavGledaKanal]++];
                    uart0Putch(ch);
                    kaGLav_rdx[kaGlavGledaKanal] %= KA_GLAV_SIZE;
                    if(ch == ESCP) umetanje=2;
                }
            } else { // Potrosili smo bafer. Idemo na sledeci kanal
#if VARIJANTA==2
                if(!timeout){
                    timeout= 20000; kaGlavSaljeKanal = MAX_KANAL;
                } //20000*0.2ms=4s
#elif VARIJANTA==3
                kaGlavSaljeKanal = MAX_KANAL;
#endif
                if(++kaGlavGledaKanal == MAX_KANAL) kaGlavGledaKanal=0;
            }
        }
        // Primamo od PC-a i rasporedjujemo u kanale
        int znak=uart0Getch();
        if( znak >= 0) { // Ako je nesto stiglo preko UARTa
            // rasporedjujemo u odGlav bafere prema kanalima (citacima)
            static char escap=0;
            if( !escap) {
                if(znak == ESCP) escap=1;
                else {
                    odGlav[odGlavKanal][odGLav_wrx[odGlavKanal]++]
                         = znak;
                    odGLav_wrx[odGlavKanal] %= OD_GLAV_SIZE;
                }
            } else {
                escap=0;
                if((znak >= 'A') && (znak < 'A'+MAX_KANAL))
                    odGlavKanal=znak-'A'; // znak nije poruka,
                                          // nego broj kanala
                else { // znak '~' koji je morao da se posalje
                       // kao dva '~', jer je prvi '~' shvacen kao
                       // escape char.
                    odGlavKanal %= MAX_KANAL;
                    odGlav[odGlavKanal][odGLav_wrx[odGlavKanal]++]
                        = znak;
                    odGLav_wrx[odGlavKanal] %= OD_GLAV_SIZE;
                }
            }
        }
    }
```

1.2.4.3. Softver na centralnom PC računaru

1.2.4.3.1. Qt framework

Obje aplikacije na centralnom računaru, za prikupljanje i za pregled podataka, realizovane su na C++ programskom jeziku koristeći Qt framework [19]. Qt je softverski okvir koji omogućava razvoj, kako grafičkih, tako i konzolnih aplikacija, nezavisno od platforme, tj. operativnog sistema na kome će se aplikacija pokretati. Podržava veliki broj platformi kao što su:

- X11 - X Windows sistem (GNU/Linux, FreeBSD...)
- Android
- Windows (XP, Vista,7,8)
- Windows CE
- iOS
- OS X
- QNX/BlackBerry 10
- OpenSolaris
- OS/2
- Amazon Kindle DX
- AmigaOS i dr.

Ovo praktično znači da aplikacija razvijena za jednu platformu u najvećem broju slučajeva može raditi na većini ostalih podržanih platformi sa minimalnim izmjenama ili čak i bez izmjena.

Qt je od 2011. vlasništvo firme Digia. Raspoloživ je uz komercijalnu, GPL v3 i GPL v2 licencu. Svaka od edicija podržava više različitih kompajlera, uključujući GCC C++ i Visual Studio kompajler.

Qt se oslanja na C++, ali koristi i posebni generator koda zvani Meta Object Compiler, tj. moc, zajedno sa još nekim makroima kako bi obogatio jezik. Osim u C++, Qt se može koristiti i iz drugih programskih jezika (Phyton, Java, Ruby, Perl...) kroz tzv. language bindings.

Ključni elementi Qt okruženja su:

- **potpuna apstrakcija grafičkog interfejsa** - Ovim je olakšana izrada aplikacija sa Model-View-Controller (MVC) arhitekturom koja omogućava autonomni razvoj tri osnovna dijela aplikacije: model - dio za komunikaciju sa bazom podataka, view - grafički interfejs i controller - logika aplikacije. Ova arhitektura predstavlja osnovu i velikog broja web-aplikacija, pa ovdje treba pomenuti i još jedan C++ framework, Wt - WebToolkit, koji je rađen po uzoru na Qt, a namijenjen je za izradu web-aplikacija.
- **signali i slotovi** - omogućavaju jednostavnu obradu događaja (events) obezbjeđujući

razmjenu poruka među objektima, bilo da su ti objekti unutar iste niti ili u različitim nitima.

- **metaobject kompajler** - omogućava prevođenje Qt-specifičnih preprocesorskih direktiva u standardne C++ konstrukcije.

Pored ovoga, Qt ima i razvijenu podršku za internacionalizaciju, a od ne-grafičkih mogućnosti obezbjeđuje SQL pristup bazama podataka, XML obradu, upravljanje nitima, pristup mreži i unificirani interfejs za rad sa fajlovima nezavisno od platforme.

1.2.4.3.2. Aplikacija za prikupljanje podataka na centralnom računaru

Ova aplikacija proziva PLC na mašinama i podatke iz odgovora obrađuje i upisuje u mysql relacionu bazu podataka. Ovdje treba napomenuti da, pored relacionih (strukturiranih) baza podataka kod kojih se podaci upisuju u unaprijed pripremljenu i fiksiranu strukturu, za čuvanje velike količine podataka, za koje se ne može unaprijed predvidjeti hijerarhijska struktura, koriste se nestrukturirane, tzv. NoSQL baze podataka. NoSQL baze se mogu posmatrati i kao unaprijeđena verzija logovanja događaja. Jedan primjer nestrukturiranog čuvanja podataka je, već široko primijenjivani, JSON (JavaScript Object Notation) format.

Logovanje događaja u ovoj aplikaciji je, radi uštede prostora na disku zbog velike količine podataka, izvedeno tako da se događaji u toku dana snimaju u tekstualni fajl koji u nazivu sadrži redni broj tog dana u mjesecu, npr. za 24.april naziv fajla će biti log24.txt. Na ovaj način se podaci čuvaju mjesec dana, a zatim se preko fajla prepisuje novi fajl iz sljedećeg mjeseca čiji je redni broj isti kao kod starog fajla. Za detaljnije praćenje komunikacije, gdje se loguje mnogo više podataka, primijenjen je sličan način rasterećenja diska, s tim što je u nazivu fajla naveden sat, tako da se ti fajlovi čuvaju samo dvadeset i četiri sata.

Strukturu sistema aplikacija čita iz *mysql* baze i formira liste objekata *QList<Uredjaj*> i QList<IOSignal*>* za smještanje nizova zapisa iz odgovarajućih tabela `*uredjaji*`, odnosno `*signali*`. Za svako od polja u ovim tabelama postoji istoimeni atribut u odgovarajućoj klasi.

```
QSqlQuery upit;
upit.exec("SELECT id, naziv, funkcija, plc_adresa,"
    " koncentrator_port, opis FROM uredjaji ORDER BY id ASC");
while(upit.next())
{
    uredjaj.append(new Uredjaj());
    uredjaj.last()->id=upit.value(0).toUInt();
    uredjajById.insert(uredjaj.last()->id,uredjaj.last());
    uredjaj.last()->funkcija
        = upit.value(2).toString().at(0).toAscii();
    uredjaj.last()->plc_adresa
        = upit.value(3).toString().at(0).toAscii();
    uredjaj.last()->koncentrator_port
        = upit.value(4).toString().at(0).toAscii();
    //      ...
}
```

Na osnovu polja `*traka*`, `*pozicija*`, `*tip*`, `*podtip*` i `*velicina*` iz tabele `*signali*` formira se hijerarhija signala vezanih za objekte iz liste *"uredjaj"*. Pri tom se za svaki uređaj, kao i za svaku traku, na osnovu vrijednosti polja `*pozicija*` u tabeli `*signali*`, definiše po jedan glavni signal koji će biti prikazan u brzom pregledu stanja mašina. Slijedi dio koda za formiranje hijerarhije signala po

uređajima:

```
for(int i=0; i<uredjaj.count(); i++)
{
    QString strUpit="SELECT id, naziv, tip, podtip, velicina,"
        " snimanje, jedinica, uredjaj, pozicija, traka, y_min,"
        " y_max, p0, p1, k0, k1 FROM signali WHERE uredjaj="
        +QString::number(uredjaj[i]->id)
        +" ORDER BY traka ASC, tip ASC, velicina ASC,"
        +" pozicija ASC,id ASC";
    upit.exec(strUpit);
    while(upit.next())
    {
        IOSignal *novi_signal=new IOSignal(this);
        uredjaj[i]->io_signal.append(novi_signal);
        novi_signal->id=upit.value(0).toInt();
        novi_signal->tip = upit.value(2).toString().at(0).toAscii();
        if((upit.value(3).isNull()) ){
            novi_signal->podtip=' ';
        }else if(upit.value(3).toString().size()<1){
            novi_signal->podtip=' ';
        }else{
            novi_signal->podtip
                = upit.value(3).toString().at(0).toAscii();
        }
        novi_signal->velicina
            = upit.value(4).toString().at(0).toAscii();
        novi_signal->snimanje
            = (upit.value(5).toString().at(0).toAscii()=='y')?
              true:false;
        novi_signal->uredjaj=upit.value(7).toInt();
        novi_signal->pozicija=upit.value(8).toInt();
        novi_signal->traka=upit.value(9).toInt();
        //  ...
        while(uredjaj[i]->traka.count()<(novi_signal->traka+1)){
            uredjaj[i]->traka.append(Traka());
        }
        io_signal.append(novi_signal);
        switch(novi_signal->tip){
        case IOSignal::SIGTIP_RELATIVNI:
            uredjaj[i]->traka[novi_signal->traka].relativni
                .append(novi_signal);
            switch(novi_signal->velicina){
            case IOSignal::SIGVR_ISTEZANJE:
                uredjaj[i]->traka[novi_signal->traka].istezanje
                    .append(novi_signal);
                break;
            }
            break;
        case IOSignal::SIGTIP_ANALOGNI:
            uredjaj[i]->traka[novi_signal->traka].analog
                .append(novi_signal);
            switch(novi_signal->velicina){
            case IOSignal::SIGVR_NAPON:
                uredjaj[i]->traka[novi_signal->traka].napon
                    .append(novi_signal);
```

```
                    break;
                }
                break;
            case IOSignal::SIGTIP_KUMULATIVNI:
                uredjaj[i]->traka[novi_signal->traka].kumulativni
                    .append(novi_signal);
                switch(novi_signal->velicina){
                case IOSignal::SIGVR_DUZINA:
                    uredjaj[i]->traka[novi_signal->traka].duzina
                        .append(novi_signal);
                    break;
                case IOSignal::SIGVR_KOMADI:
                    uredjaj[i]->traka[novi_signal->traka].komadi
                        .append(novi_signal);
                    break;
                }
                break;
            //  ...
            }
            if(novi_signal->pozicija==0){
                novi_signal->glavni=true;
                uredjaj[i]->traka[novi_signal->traka].glavni
                    .append(novi_signal);
            }
        }
    }
```

U slučajevima gdje je potrebno vršiti brzu pretragu objekata na osnovu indeksa, na primjer pronalaženje radnika na osnovu koda kartice, korišćene su *heš*-tabele (*QHash*) koje u C++ Qt okruženju obezbjeđuju funkcionalnost koju u nekim drugim jezicima imaju asocijativni nizovi. Slijedi primjer korišćenja *QHash* klase:

```
QHash<QString, unsigned int>radnikByKartica;
// ...
radnikByKartica.clear();
QSqlQuery upit;
upit.exec("SELECT id, radnik FROM kartice");
while(upit.next()){
    radnikByKartica.insert(upit.value(0).toString()
                           ,upit.value(1).toUInt());
}
// ...
if(radnikByKartica.contains(id_kartice))
     prijaviUlazRadnika(radnikByKartica.value(id_kartice));
```

Svi PLC uređaji se prozivaju svake sekunde, dok se upis u bazu vrši u rezoluciji od jednog minuta. U slučaju prekida komunikacije prije isteka datog minuta, u bazu će se snimiti podaci na osnovu posljednje ispravne komunikacije u toku tog minuta, a ukoliko cio minut nije bilo komunikacije, u bazu će biti upisana informacija da nije bilo komunikacije. Prilikom prijema odgovora sa PLC uređaja, aplikacija mora voditi računa i o sloju protokola koji unosi koncentrator. Da bi se obezbijedila ravnopravna komunikacija sa svim uređajima, koncentrator prijem sa kanala razbija u okvire koji počinju znakom '~', a zatim slijedi adresa kanala i primljeni podaci. Zbog toga se često dešava da primljeni paket sa PLC-a bude podijeljen na više okvira, tako da je zadatak aplikacije na centralnom računaru da prvo objedini podatke iz okvira dobijenih od koncentratora, a zatim da tumači pakete odgovora. Slijedi dio koda iz klase *Komunikacija* koji apstrahuje protokol

koncentratora i raspoređuje primljene podatke u prijemne bafere vezane za objekte tipa *Uredjaj*, a na osnovu informacije dobijene od koncentratora o tome sa kog kanala su stigli:

```
class Komunikacija:public QObject
{Q_OBJECT
//     ...
    QList<Uredjaj*>uredjaj;
    QHash<char,int>uredjajIndexFromKoncPort;
    unsigned char koncentrator_port_i, koncentrator_stanje;
    unsigned char paket_rcv[MAX_VELICINA_PAKETA];
//     ...
};

bool Komunikacija::procitajKoncentrator(unsigned char *buffer,
                                        int procitano)
{
    for(int i=0;i<procitano;i++){
        if(koncentrator_stanje==KONC_ST_CEKA_PORT){
            if(buffer[i]=='~'){
                uredjaj.value(
                          uredjajIndexFromKoncPort.value(
                                        koncentrator_port_i
                 ))->strRcvBuffer += QString::fromAscii(
                          (const char*)(buffer+i),1);
            }else if(uredjajIndexFromKoncPort.contains(buffer[i])){
                koncentrator_port_i=buffer[i];
            }
            koncentrator_stanje=KONC_ST_PRIMA_PKT;
        }else if(koncentrator_stanje==KONC_ST_PRIMA_PKT){
            if(buffer[i]=='~'){
                koncentrator_stanje=KONC_ST_CEKA_PORT;
            }else if(uredjajIndexFromKoncPort
                          .contains(koncentrator_port_i)){
                uredjaj[uredjajIndexFromKoncPort.value(koncentrator_port_i)
                    ]->strRcvBuffer
                    +=QString::fromAscii((const char*)(buffer+i),1);
            }
        }
        if(uredjajIndexFromKoncPort.contains(koncentrator_port_i)){
             uredjaj[uredjajIndexFromKoncPort.value(koncentrator_port_i)]
             ->ceka_prijem=true;
        }
    }
    return true;
}
```

Sljedeći nivo komunikacije vrši obradu sadržaja prijemnog bafera pojedinih uređaja i rekonstrukciju paketa koje su poslali PLC uređaji. Na ovom nivou još se ne uzima u obzir adresa i funkcija PLC-a, što znači da će PLC, ukoliko se prebaci na neki drugi kanal koncentratora, nastaviti da funkcioniše bez potrebe za rekonfiguracijom sistema, pod uslovom da se prilikom prozivanja uređaja umjesto adrese koristi '*'.

```
bool Komunikacija::primi()
{
    int procitano=0;
    do{
        procitano=procitajPort(paket_rcv);
        procitajKoncentrator(paket_rcv,procitano);
        for(int i=0;i<uredjaj.count();i++){
            if(uredjaj[i]->ceka_prijem){
```

```
                procitajUredjaj(i);
                uredjaj[i]->ceka_prijem=false;
            }
        }
    }while(procitano>0);
    return true;
}

void Komunikacija::procitajUredjaj(unsigned char id)
{
    for(int i=0;i<uredjaj[id]->strRcvBuffer.length();i++){
        QChar qCh=uredjaj[id]->strRcvBuffer.at(i);
        char ch=qCh.toAscii();
        if(ch<0x20){
            uredjaj[id]->paket_stanje=ST_RCV_CEKA_START;
            uredjaj[id]->paket="";
            continue;
        }
        if(ch=='{'){
            uredjaj[id]->paket_stanje=ST_RCV_DATA;
            uredjaj[id]->paket="";
            uredjaj[id]->suma_rcv=0;
            continue;
        }
        if(uredjaj[id]->paket_stanje==ST_RCV_DATA){
            if(ch=='}'){
                uredjaj[id]->paket_stanje=ST_RCV_SUMA0;
            }else{
                uredjaj[id]->paket+=QChar::fromAscii(ch);
            }
            continue;
        }
        if(uredjaj[id]->paket_stanje==ST_RCV_SUMA0){
            uredjaj[id]->paket_stanje=ST_RCV_SUMA1;
            if((ch>='0')&&(ch<='9')){
                uredjaj[id]->suma_rcv=0x10*(ch-'0');
            }else if((ch>='A')&&(ch<='F')){
                uredjaj[id]->suma_rcv=0x10*(10+ch-'A');
            }else if((ch>='a')&&(ch<='f')){
                uredjaj[id]->suma_rcv=0x10*(10+ch-'a');
            }else{
                uredjaj[id]->paket_stanje=ST_RCV_CEKA_START;
                uredjaj[id]->paket="";
            }
            continue;
        }
        if(uredjaj[id]->paket_stanje==ST_RCV_SUMA1){
            if((ch>='0')&&(ch<='9')){
                uredjaj[id]->suma_rcv+=(ch-'0');
            }else if((ch>='A')&&(ch<='F')){
                uredjaj[id]->suma_rcv+=(10+ch-'A');
            }else if((ch>='a')&&(ch<='f')){
                uredjaj[id]->suma_rcv+=(10+ch-'a');
            }else{
                uredjaj[id]->paket_stanje=ST_RCV_CEKA_START;
                uredjaj[id]->paket="";
                continue;
            }
            procitajPaket(uredjaj[id]->paket,uredjaj[id]->suma_rcv);
            uredjaj[id]->suma_rcv=0;
            uredjaj[id]->paket_stanje=ST_RCV_CEKA_START;
```

```
                continue;
            }
        }
        uredjaj[id]->strRcvBuffer="";
    }
```

Na kraju ove faze u obradi, prijem je dovoljno isfiltriran da se može vršiti logovanje. To nije bilo pogodno u ranijim fazama jer najveći dio prijema sa kanala koncentratora potiče od smetnji, tako da bi njihovo logovanje bespotrebno zauzelo veliku količinu memorijskih resursa.

U sljedećoj fazi obrade vrši se provjera ispravnosti paketa na osnovu ček-sume, a zatim se analizira njegov sadržaj. Na osnovu komandi sadržanih u paketu vrši se osvježavanje stanja objekta iz liste uređaja određenog na osnovu adrese i funkcije PLC-a, a koje su takođe sadržane u paketu:

```
    //      ...
unsigned char chCmd=paket[0].toAscii();
unsigned char komanda=getCmdId.value(chCmd);
paket=paket.mid(1);
QList<QString>strParametri;
QList<int>intParametri;
bool ispravan=false;
switch(komanda){
    case CMD_POSALJI_UL:
        switch(funkcija){
        case 'e':   // ekstruder
            ispravan = procitajHexParametar(paket,intParametri);
            ispravan &= procitajHexParametar(paket,intParametri);
            if(ispravan){
                uredjaj.value(adresa)->primljenoExtruder(status,
                                        intParametri.value(0),
                                        intParametri.value(1)
                                        );
            }
            break;
        case 's':   // stampa (Matila)
            ispravan = procitajHexParametar(paket,intParametri);
            ispravan &= procitajHexParametar(paket,intParametri);
            ispravan &= procitajHexParametar(paket,intParametri);
            if(ispravan)
            uredjaj.value(adresa)->primljenoStampaMatila(status,
                                    intParametri.value(0),
                                    intParametri.value(1),
                                    intParametri.value(2));
            break;
      //    ...
}
//    ...

void Uredjaj::primljenoStampaMatila(char status, int duzinaU,
                                    int duzinaA, int duzinaB)
{
    //      ...
    this->traka[1]./* QList<IOSignal*> */duzina[0]
                        ->setVrijednost(init,duzinaA);
    this->traka[2].duzina[0]->setVrijednost(init, duzinaB);
    this->traka[0].duzina[0]->setVrijednost(init,duzinaU);
```

```
        this->traka[1].istezanje[0]->setVrijednost(init,
                            traka[0].duzina[0]->vrijednost,
                            traka[1].duzina[0]->vrijednost);
        this->traka[2].istezanje[0]->setVrijednost(init,
                            traka[0].duzina[0]->vrijednost,
                            traka[2].duzina[0]->vrijednost);
    }
```

Metod *setVrijednost()* klase *IOSignal* obavlja osvježavanje trenutne vrijednosti signala u tabeli `signali`. Svakom signalu u sistemu odgovara po jedan zapis u tabeli `signali`. Sa izuzetkom polja `vrijednost` i `ostatak_mili` (decimalni dio vrijednosti), polja zapisa, koja su mapirana na odgovarajuće atribute klase *IOSignalI*, određuju način na koji će odgovarajući signal biti obrađen.

```
    void IOSignal::setVrijednost(bool init, int v_raw, int v1_raw)
    {
        aktivan=1;  //  prilikom snimanja vrijednosti
                    // u tabelu `vrijednosti_signala` na kraju minuta,
                    // označava da je vrijednost ovog signala aktuelna.
                    // resetuje se prilikom snimanja na kraju minuta.
        value_raw=v_raw;
        if(init){   // ako se uredjaj resetovao
            // proglasimo primljene vrijednosti za pocetne
            value0_raw=v_raw;
            QString sUpit_Aktuelni="UPDATE signali SET vrijeme_upisa=\'"
                +QDateTime::currentDateTime()
                      .toString("yyyy-MM-dd hh:mm:ss")
                +"\', "
                "vrijednost="+QString::number(value_raw)
                +",ostatak_mili="+QString::number(ostatak_mili)
                +" WHERE id="+QString::number(id);
            QSqlQuery upit_aktuelni;
            upit_aktuelni.exec(sUpit_Aktuelni); // upisujemo u bazu
                                        // aktuelne vrijednosti signala
            reset();    // gube se prethodni podaci za tekuci minut
        }
        if(tip==SIGTIP_KUMULATIVNI){
            int vrijednost_temp;
            if(value_raw<value0_raw){
                // ako je nakon prethodnog očitavanja
                // brojac prošao kroz nulu
                vrijednost_temp=0xffff-value0_raw+value_raw;
            }else{
                vrijednost_temp=value_raw-value0_raw;
            }
            fVrijednost+=k0 + k1*vrijednost_temp+ostatak_mili;
            vrijednost=(int)fVrijednost;
            ostatak_mili=fVrijednost-vrijednost;
            fVrijednost=vrijednost;
        }else if(tip==SIGTIP_RELATIVNI){
            if(velicina==SIGVR_ISTEZANJE){
                if((v_raw!=0)&&((v1_raw-v_raw)>0)){
                    fVrijednost=1000.0*(v1_raw-v_raw)/(float)v_raw;
                }else{
                    fVrijednost=0;
                }
```

```
                vrijednost=(int)fVrijednost;
            }
        }else{
            fVrijednost=k0 + k1 * value_raw;
            vrijednost=(int)fVrijednost;
        }
        value0_raw=value_raw;
    }
```

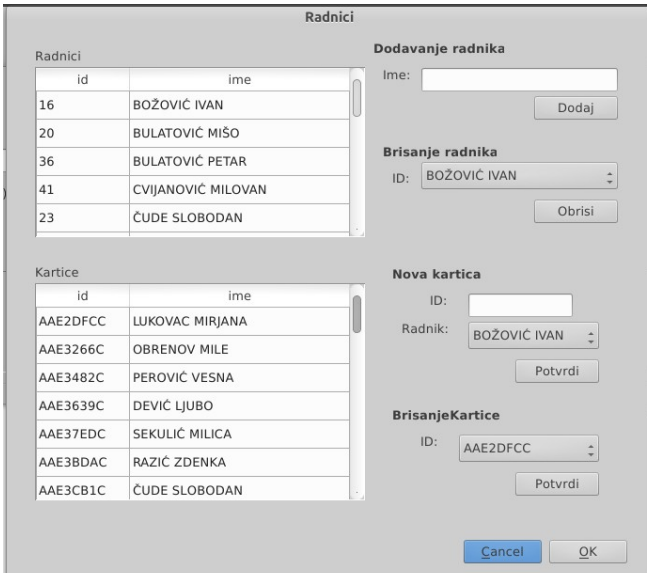
Slika 1.19: Aplikacija za prikupljanje podataka

Na slici 1.19 prikazan je glavni prozor aplikacije. U tabeli 'uređaji' nalazi se spisak i adrese uređaja, kao i njihovi osnovni parametri. Tabela 'signali' sadrži spisak i osobine svih signala u sistemu, kao i uređaj za koji je taj signal vezan. U tabeli 'Čitači kartica' prikazano je trenutno stanje čitača. Ukoliko je u čitaču prisutna kartica, biće prikazan njen kod i ime radnika kome je pridružena.

U sistemu se prisustvo radnika bilježi na dva načina: za radnike koji rade na mašinama prisustvo se računa kao vrijeme dok se kartica nalazi u čitaču, a za ostale radnike prisustvo se računa na osnovu prijave i odjave na dva odvojena čitača koji se nalaze na ulazu i izlazu iz fabike. Na jednoj mašini može da radi više radnika istovremeno, tako da joj je moguće pridružiti više čitača. Isto tako, jedan radnik, odnosno čitač kartica, može da pokriva više mašina. Za čitače kartica na mašinama koje mjere vrijeme prisustva kartice u uređaju, prilikom upisa u bazu podataka na kraju minuta, informacija da radnik nije bio prisutan upisivaće se samo ako nijedna kartica nije bila prisutna u toku cijelog tog minuta, dok će u suprotnom biti zabilježena posljednja prisutna kartica. Zavisno od toga da li je radnik prepoznat u sistemu, prilikom prozivanja uređaja program šalje PLC-u komandu

za postavljanje odgovarajuće LED indikacije na čitaču. Ukoliko je prijavljena nova kartica, automatski se otvara dijalog u kom je omogućeno pridruživanje radnika toj kartici, čime je pojednostavljen unos novih kartica u sistem.

Upis novog i deaktivacija postojećeg radnika, kao i izmjena podataka o radnicima, može se obaviti u dijalogu koji se otvara izborom opcije *Radnici* iz menija *Podešavanja*.

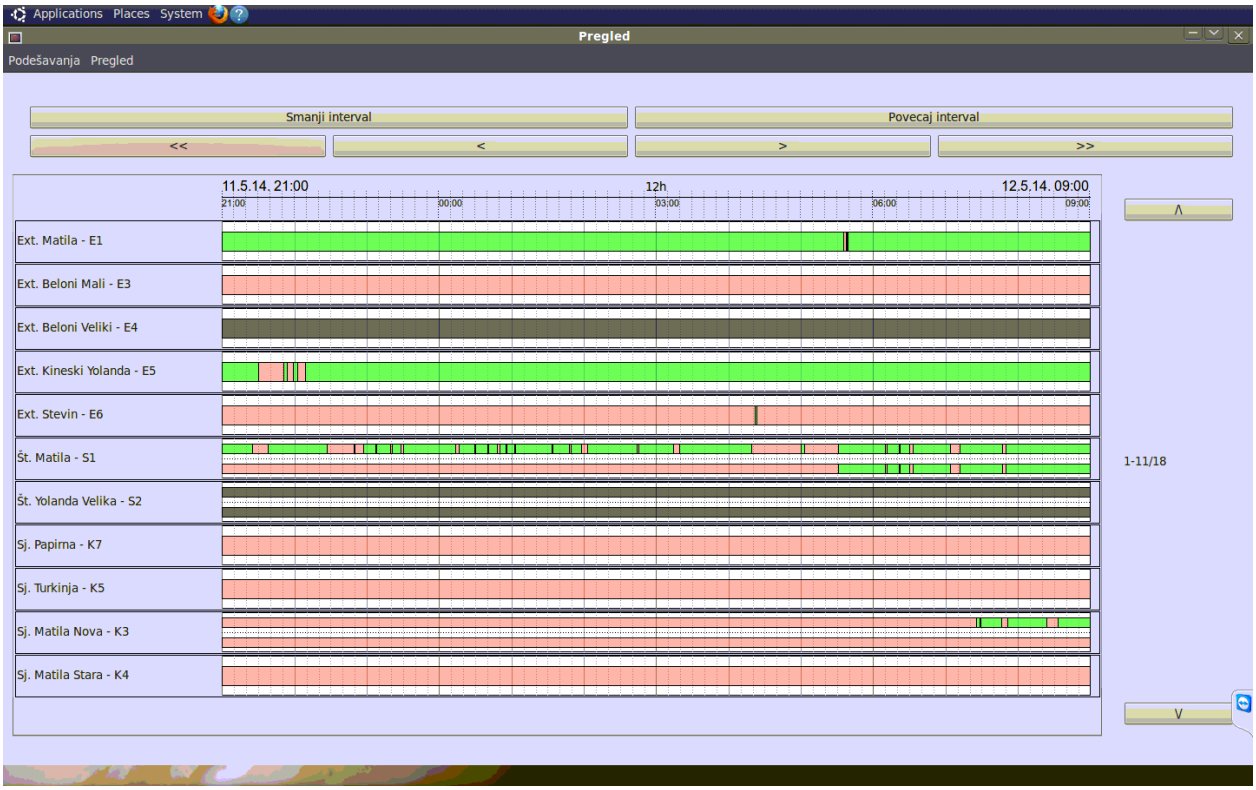


Slika 1.20: Uređivanje radnika i kartica

1.2.4.3.3. Aplikacija za pregled podataka

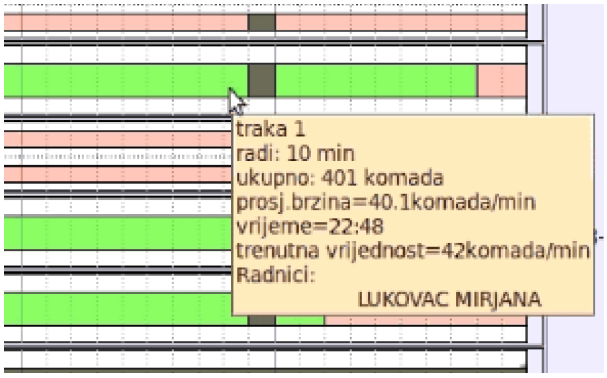
Ova aplikacija omogućava da se korisniku pregledno prikažu najbitniji podaci iz procesa proizvodnje u nekom vremenskom intervalu. Iz aplikacije je moguće dobiti i izvještaje o proizvodnji po pojedinim mašinama, kao i izvještaje o učinku pojedinih radnika.

Glavna forma (Slika 1.21) sadrži prikaz rada svih mašina u procesu na osnovu jednog parametra koji je definisan kao glavni. To može biti npr. dužina odštampane folije, broj komada kod sjeckalica i sl. Dio trake koji je zelene boje označava da je mašina u tom periodu bila aktivna i da se vrijednost glavnog parametra kretala u očekivanim okvirima. Crveni dio trake predstavlja intervale vremena kada je mašina bila neaktivna, dok djelovi označeni sivom bojom predstavljaju periode kad mašina nije imala komunikaciju sa centralnim računarom. Interval u kome se posmatra proces može biti 1h, 12h i 24h, a rezolucija u svakom od ovih slučajeva je jedan minut.

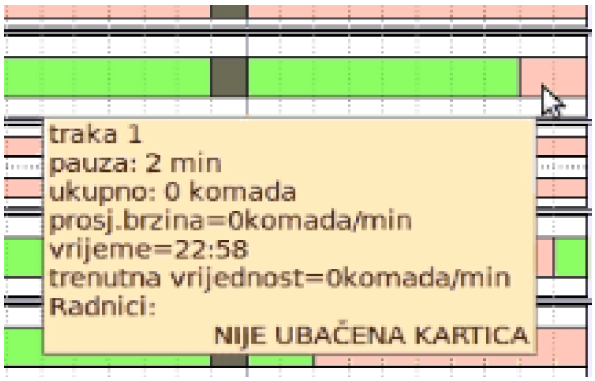


Slika 1.21: Aplikacija za pregled podataka

Klikom miša na bilo koji od ovih djelova otvara se tooltip sa informacijom o tome koliko ukupno traje posmatrani period, kolika je bila ukupna proizvodnja i prosječna brzina proizvodnje u posmatranom periodu, trenutna vrijednost brzine proizvodnje u izabranom momentu (najmanja vremenska jedinica je minut) i koji je radnik radio na mašini u tom periodu. Ako je mašina bila neaktivna, biće prikazana dužina perioda neaktivnosti. Neaktivnost ne podrazumijeva samo periode potpunog zastoja u radu mašine, već i periode u kojima je mašina radila ali nije dostignuta zadata norma.



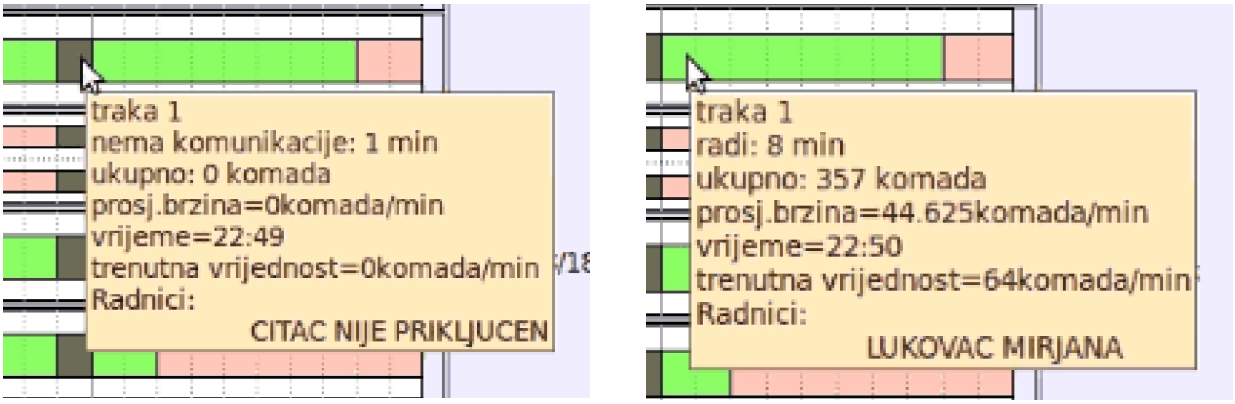
Slika 1.22: Aktivan rad mašine



Slika 1.23: Vrijeme neaktivnosti mašine

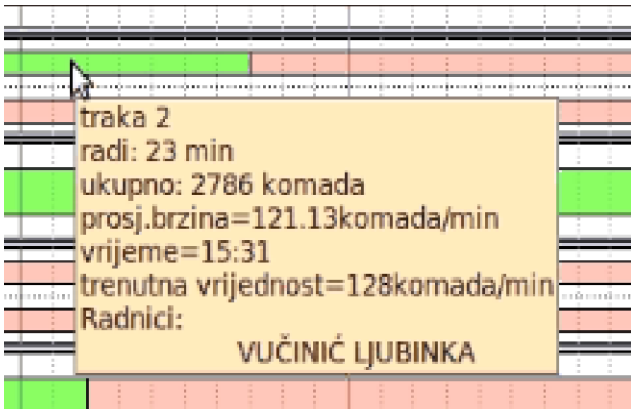
Ukoliko dođe do prekida komunikacije, PLC nastavlja da prikuplja i čuva kumulativne podatke, tako da se, nakon ponovnog uspostavljanja komunikacije, ponovo mogu dobiti tačni podaci o

ukupnoj proizvodnji. Međutim, pošto se u bazu upisuju vrijednosti kumulativnih podataka na kraju svakog minuta, a program računa trenutnu brzinu proizvodnje na osnovu razlike dva uzastopna upisa u bazi, za prvi minut nakon uspostavljanja komunikacije, za trenutnu brzinu treba očekivati mnogo veće vrijednosti od stvarnih.



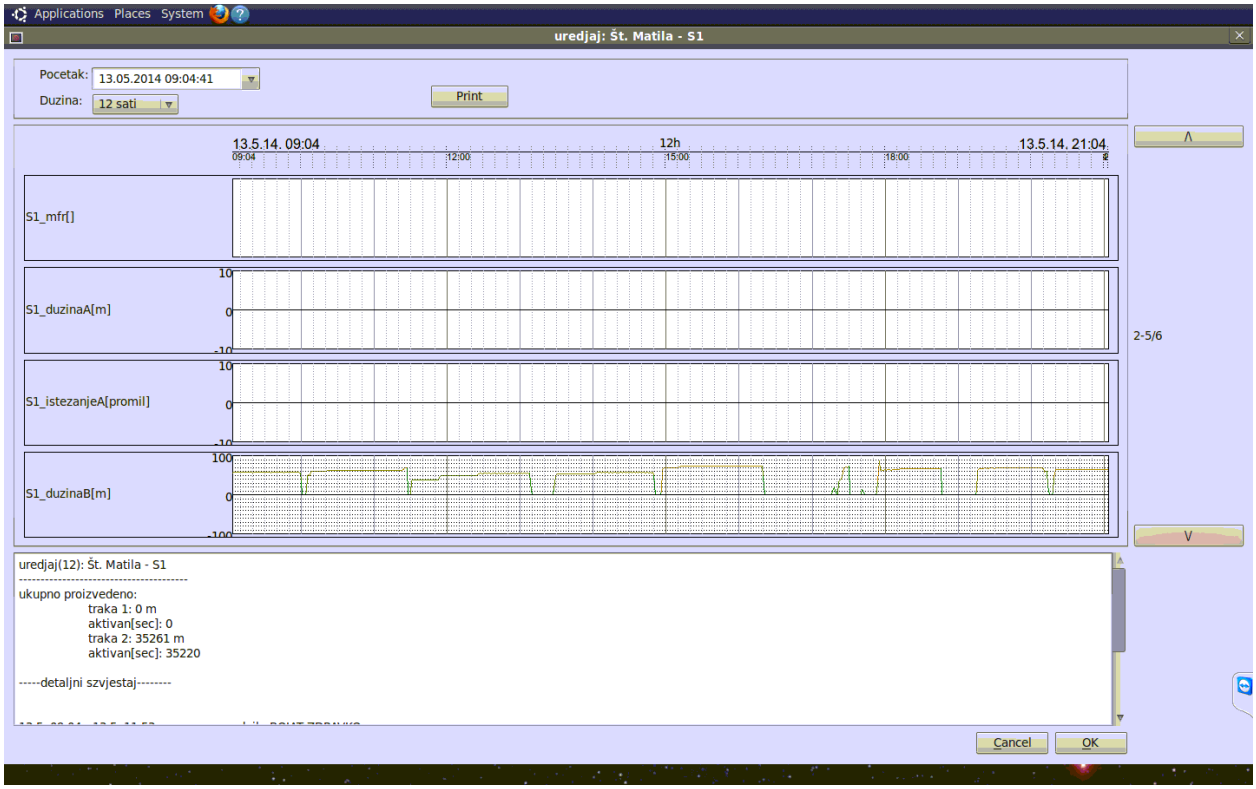
Slika 1.24: Prikaz prekida komunikacije (lijevo) i stanja nakon prekida (desno)

Za mašine sa više traka, prikaz je takođe podijeljen na odgovarajući broj traka, a klikom na jednu od njih dobijaju se informacije samo za tu traku.



Slika 1.25: Pregled pojedinačnih traka

Duplim klikom na prikaz neke mašine otvara se novi prozor u kome su prikazani i ostali ulazi te mašine.



Slika 1.26: Izvještaj za pojedinačnu mašinu

U ovom prozoru može se dobiti izvještaj o ukupnoj proizvodnji mašine u izabranom vremenskom intervalu, kao i o učinku pojedinih radnika na toj mašini u tom periodu.

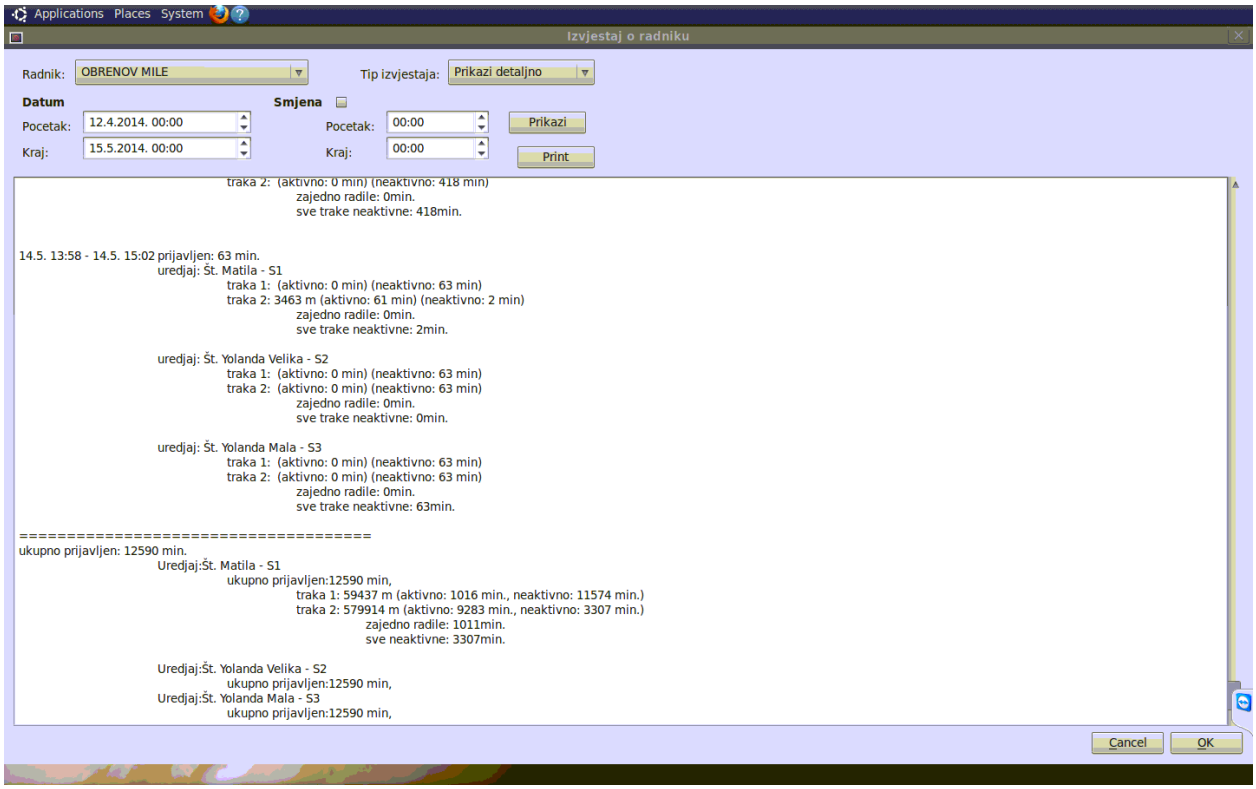
Prikazan je primjer izvještaja za vremenski period od dvanaest sati.

```
uredjaj(12): Št. Matila - S1
   ----------------------------------------
   ukupno proizvedeno:
        traka 1: 0 m
        aktivan[sec]: 0
        traka 2: 35261 m
        aktivan[sec]: 35220

   -----detaljni szvjestaj--------


   13.5. 09:04 - 13.5. 11:53  -    radnik: BOJAT ZDRAVKO
                   traka 2    proizvedeno: 9373m

   13.5. 11:53 - 13.5. 14:56  -    radnik: PRINCIP SVETOZAR
                   traka 2    proizvedeno: 8643m

   13.5. 14:56 - 13.5. 21:03  -    radnik: OBRENOV MILE
                   traka 2    proizvedeno: 17179m
```

Pored izvještaja o proizvodnji, u ovom dijalogu se mogu pratiti vremenske promjene svih

parametara uređaja koji se prate, tako da se lako mogu uočiti npr. mogući periodi opterećenja i drugi faktori bitni za ispravan rad uređaja.

Izborom opcije Radnik iz menija Izvještaj otvara se prozor iz koga se može dobiti izvještaj o pojedinačnom radniku, tj. na kojim mašinama je radio u izabranom periodu i koliki je bio njegov učinak. Pored količina koje je radnik proizveo, iz izvještaja se može pratiti i odnos vremena aktivnog i neaktivnog rada mašine, a za mašine sa više traka može se pratiti i vrijeme istovremenog rada traka. Prilikom zadavanja granica posmatranog vremenskog intervala moguće je izvještaj ograničiti samo na pojedinačnu smjenu.



Slika 1.27: Izvještaj o učinku radnika

1.2.4.3.4. Mogućnost daljeg unapređenja sistema

U novije vrijeme postoji tendencija da se, ako je to ikako moguće, desktop aplikacije modifikuju kako bi funkcionisale preko web-interfejsa. Ovo je naročito pogodno zbog toga što tada nije potrebno instalirati dodatni softver za podršku aplikacije, već se sa aplikacijom može raditi direktno iz standardnog web-pregledača.

Tradicionalno, web-aplikacije su koristile HTTP protokol i bile uređene na način da se, po prikupljanju podataka na formi klijentskog browser-a, na zahtjev korisnika ti podaci proslijeđuju serveru putem GET ili PUT HTTP zahtjeva. HTTP server isključivo odgovara na HTTP zahtjeve, tj. nikad ne šalje podatke samoinicijativno. U početku je server kao odgovor morao da šalje kompletnu HTML stranicu, što je bilo zadovoljavajuće za pregledanje statičkog sadržaja (tekstualnih dokumenata, slika i sl.), a kasnije je razvijen AJAX protokol koji je omogućio da se ažuriraju samo pojedini lokalni djelovi stranice, što je omogućilo prikazivanje i dinamičkih sadržaja, istovremeno rasterećujući internet saobraćaj i klijenta jer više nije bilo potrebno za bilo kakvu, pa i najmanju, promjenu učitavati kompletnu web-stranicu.

Međutim, još uvijek je bilo potrebno da klijent izričito traži ažurirane podatke od servera. Jedno od rješenja za ovaj problem je bilo korišćenje tajmera u klijentskoj aplikaciji koji bi periodično tražio od servera ažurirane podatke. Drugo rješenje je bilo otvaranje zasebnih socket-a za TCP ili UDP komunikaciju, gdje je druga strana mogla biti i inicijator komunikacije. Ovo je donedavno obezbjeđivano uglavnom nestandardnim rješenjima kao što su npr. Java applet-i ili adobe flash, gdje je bilo potrebno instalirati dodatn plug-in kako bi browser mogao podržati ovu vrstu komunikacije.

Da bi se ovo prevazišlo, razvijen je WebSocket standard. Nakon inicijalne handshaking faze, http prelazi u full-duplex TCP protokol gdje obje strane mogu da iniciraju komunikaciju koja se odvija sa minimalnim kašnjenjima. Standardizacija obezbjeđuje da ovaj način komunikacije bude podržan u svim modernijim browser-ima.

Wt (Web Toolkit) je web framework razvijen po ugledu na Qt, koji je uglavnom orijentisan na desktop okruženje. Baziran je na C++ jeziku ali je raspoloživ i za neke druge jezike. Hijerarhija raspoloživih klasa se u velikoj mjeri poklapa sa onom kod Qt okruženja, a za većinu klasa iz Qt okruženja postoji odgovarajuća klasa u Wt okruženju čiji se naziv razlikuje samo u početnom slovu (naziv Qt klasa počinje velikim slovom Q, a naziv Wt klase velikim slovom W). Ovim se obezbjeđuje da se veliki dio Qt koda može jednostavno prevesti u Wt, tj. pojednostavljena je izrada web aplikacija od već postojećih desktop aplikacija.

Kod većine raspoloživih web tehnologija potrebno je različite djelove aplikacije razvijati u različitim okruženjima. Tako se za razvoj klijentske strane aplikacije skoro isključivo koristi kombinacija HTML i JavaScript jezika, dok se sa serverske strane koriste PHP, Ruby, Java, C# itd. Wt pojednostavljuje razvoj aplikacija tako što ne razdvaja klijentsku i serversku stranu, već programer cijelu aplikaciju posmatra integralno i razvija je u jednom okruženju i jeziku (uglavnom C++). Osim toga, različiti browser-i na različite načine i u različitom stepenu podržavaju standardne web tehnologije, što je od programera zahtijevalo da vodi računa o ponašanju aplikacije posebno za svaki browser. Wt rješava ovaj problem tako što automatski degradira prikaz aplikacije u zavisnosti od podrške odgovarajućih web standarda od strane browser-a. Na primjer, Wt će, ukoliko to browser podržava, pokušati da iskoristi WebSocket komunikaciju. Međutim, ako taj standard nije podržan, Wt će pokušati da iskoristi periodične Ajax upite, vodeći računa da korisnik ne osjeti razliku u funkcionalnosti aplikacije.

2. Sinhronizacija uređaja na udaljenim lokacijama

2.1. Uvod

Porastom popularnosti interneta u posljednje dvije decenije javila se tendencija da se ta mreža proširi na svaki dio prostora gdje se obavlja bilo kakva ljudska djelatnost. Ovo je uslovilo da se najrazličitije vrste uređaja, ukoliko za to postoje minimalni uslovi, povezuju na globalnu mrežu. U skladu s tim razvijan je hardver i softver koji se u velikoj mjeri oslanjao na stalnu dostupnost i funkcionisanje interneta, a razvijala se i autonomija uređaja koji počinju međusobno da komuniciraju bez potrebe za stalnim ljudskim prisustvom, što je dovelo do pojave koncepta tzv. Internet Of Things (IoT) [20][21]. Karakteristika IoT je da sada autonomni uređaji samostalno uspostavljaju i održavaju komunikaciju i usklađuju svoj rad bez stalnog ljudskog nadzora, za razliku od uobičajene komunikacije između PC uređaja koji su obično imali ulogu terminala, tj. posrednika u komunikaciji između ljudi.

Međutim, u novije vrijeme mnogi proizvođači, kako hardvera, tako i softvera, počinju da se mire sa činjenicom da takvu opštu pristupačnost internetu, sa bilo kog mjesta i u bilo kom momentu, često nije moguće ostvariti. Osim toga, korisnici često ne žele da koriste internet iako za to postoje tehnički uslovi, i to, uglavnom, zbog relativno visokih cijena mobilnog prenosa podataka u odnosu na prenos u WiFi mreži vezanoj na internet npr. DSL vezom. Zato se počinju razvijati pouzdani sistemi koji ostaju stabilni i funkcionalni i u uslovima trajne nepokrivenosti internetom, kao i u slučaju kratkotrajnih otkazivanja mreže.

Da bi se riješio problem stalne ili povremene nemogućnosti pristupa internetu sa određene lokacije, kao jedno od rješenja razvijen je tzv. *meshnet* [22]. Ovaj način povezivanja obezbjeđuje da čvorovi u mreži ne budu samo krajnji korisnici, već da svi korisnici mogu biti i mostovi, tj. međučvorovi na koje mogu da se nadovezuju drugi djelovi mreže. Ovim se povećava vjerovatnoća uspješnog pristupa globalnoj mreži putem lančanog nadovezivanja više lokalnih mreža i nalaženja bilo koje moguće putanje prema globalnoj mreži.

Takođe, i razvoj aplikacija, koje su u jednom periodu bile postale zavisne od on-line podataka, sad se odvija u smjeru ponovnog povratka na korišćenje lokalnog skladištenja podataka kao osnove za funkcionisanje aplikacije. Tako, aplikacije za GPS navođenje, kao što je npr. Google Maps, ostavljaju mogućnost snimanja geografskih mapa u lokalnu memoriju mobilnog uređaja, kako bi aplikacija mogla da funkcioniše i u odsustvu interneta.

Osim objektivne nemogućnosti uspostavljanja fizičke veze sa internetom, prilikom projektovanja sistema javlja se i problem što se velike kompanije često odlučuju da zbog sigurnosti ograniče komunikaciju između svoje lokalne mreže i interneta koristeći firewall kako bi potpuno blokirale portove ili ih ograničili na upotrebu samo određenih protokola. Upravo na ovaj slučaj sam naišao tokom rada na proširenju sistema za evidenciju radnog vremena u preduzeću “Plantaže” - Podgorica. Sistem, koji je do tada funkcionisao samo u upravnoj zgradi, trebalo je proširiti i na neke udaljene objekte, a jedini način da se to postigne bio je putem GSM mreže. Pri tom je bilo poželjno izbjeći intervencije na postojećoj aplikaciji koja je već duže vrijeme stabilno funkcionisala. Pored problema nemogućnosti pristupa lokalnoj mreži u upravnoj zgradi, postojao je i problem slabe pokrivenosti udaljenih lokacija GSM mrežom, tako da su se uređaji samo povremeno povezivali na mrežu.

Rješenjem, koje sam tamo realizovao i koje će biti opisano u ovom poglavlju, omogućena je komunikacija između uređaja koji ne moraju istovremeno biti prisutni na mreži i koji mreži samo povremeno pristupaju. Takođe, izborom e-mail-a kao sredstva komunikacije, riješio sam i problem blokirane komunikacije od strane *firewall*-a, budući da se, ipak, portovi za HTTP i e-mail komunikaciju obično ostavljaju otvoreni.

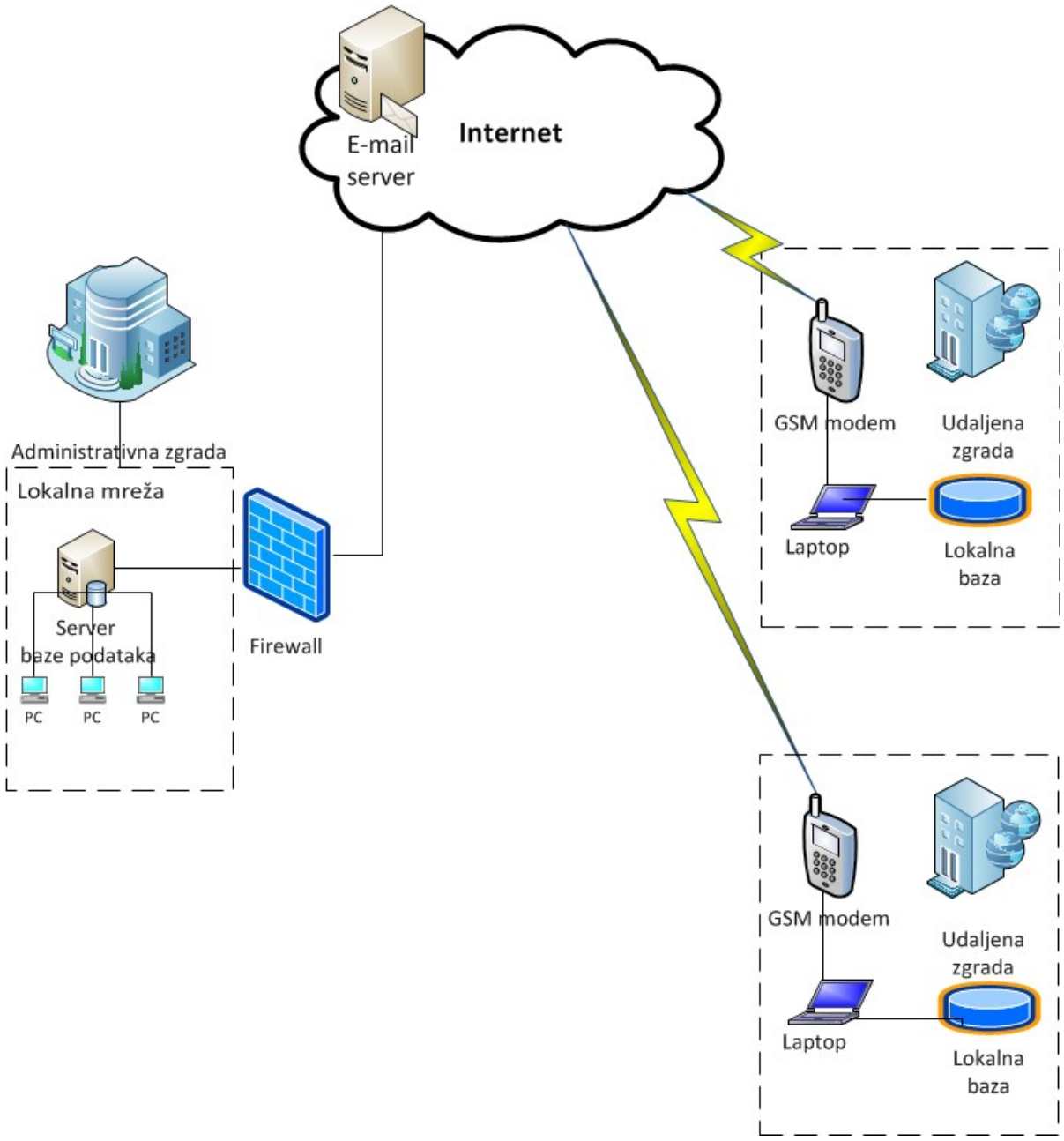
2.2. Sistem za evidenciju radnog vremena

2.2.1. Organizacija sistema

Originalna aplikacija se izvršava na PC računarima sa Linux operativnim sistemom. Podaci sa čitača kartica se prikupljaju pomoću PLC uređaja baziranog na Atmel ATmega16 AVR mikrokontroleru, a zatim se šalju PC računaru koji se nalazi u centralnoj zgradi, gdje se smještaju u MySQL bazu podataka. Sistem je trebalo proširiti na još dva udaljena objekta, tako da se tabela sa spiskom radnika iz udaljene baze podataka sinhronizuje sa tabelom radnika centralne baze, a takođe i da se podaci o vremenu prijave i odjave radnika šalju sa udaljenih objekata u centralnu bazu podataka. Prilikom proširenja sistema trebalo je, takođe, voditi računa i o tome da neki vremenski osjetljivi procesi centralne aplikacije ostanu funkcionalni bez potrebe za izmjenom programa. Na primjer, ako klijentski proces šalje upit serveru baze podataka, on čeka na odgovor određeno vrijeme i, ukoliko odgovor za to vrijeme ne stigne, klijent raskida vezu sa bazom. Ovo može biti problem ukoliko su klijent i server na različitim računarima, a server samo povremeno ima pristup mreži. Jedno od mogućih rješenja je da se ovi vremenski osjetljivi procesi izmjeste na udaljeni računar, a da se na centralnom računaru instalira jedna pomoćna aplikacija. Ova aplikacija udaljenom računaru šalje skript koji poziva proces na tom računaru kao lokalni, snima rezultate u fajl, a zatim poziva drugu pomoćnu aplikaciju koja će taj fajl putem e-maila poslati nazad centralnom računaru.

Uzimajući u obzir veliko rastojanje između objekata, jedina praktično izvodljiva veza između njih bio je internet. Računar u administrativnoj zgradi, na kome je instalirana glavna aplikacija i baza podataka, nalazi se iza rutera koji obezbjeđuje firewall i NAT. Udaljeni objekti su na internet povezani preko GSM modema, koji radi u jednom od režima paketnog prenosa podataka (GPRS, EDGE, HSDPA). Bilo je poželjno da udaljeni računari ne budu stalno povezani na internet, već da samo povremeno ostvaruju kratke konekcije. Ovo stoga što se udaljeni objekti nalaze u ruralnom području sa dosta lošom pokrivenošću mobilnim signalom i čestim prekidima veze. Budući da su u tim oblastima česti prekidi električne energije, trebalo je voditi računa i o štednji i efikasnoj upotrebi energije, tj. vrijeme trajanja veze sa internetom bi trebalo da bude što kraće, kako bi uređaj što duže radio sa autonomnim napajanjem. Osim toga, mobilni operateri, u zavisnosti od tarife, mogu dodatno da naplate prekoračenje neke određene količine podataka ili da ograniče brzinu prenosa.

Pošto nijedan od računara koji učestvuju u komunikaciji ne prihvata dolazne konekcije, bilo je potrebno da između njih postoji posrednik kome će se obraćati po potrebi. Bilo je najpogodnije da to bude e-mail server jer su portovi za e-mail komunikaciju skoro uvijek otvoreni. Postoje i druga rješenja koja omogućavaju tunneling protokola, kao što je VPN, ali prednost e-mail servera je što obezbjeđuje privremeno čuvanje poruka, tako da uređaji koji učestvuju u komunikaciji ne moraju biti istovremeno prisutni na mreži.



Slika 2.1: Organizacija sistema

E-mail porukom se šalje shell-skript [24] koji će se izvršiti na udaljenom računaru. Skript sadrži komande operativnom sistemu, tako da može da pokrene standardne programe tog operativnog sistema, a, takođe, i pomoćne procese vezane za korisničku aplikaciju. Koristeći redirekciju izlaza procesa u fajl, kao i mogućnost proslijeđivanja fajla kao ulaza u sljedeći proces, skript može rezultate izvršenog procesa da proslijedi kao fajl standardnom programu za slanje e-pošte, tako da se taj fajl e-mail porukom šalje centralnom računaru. Format tog odgovora zavisi od izvršenog procesa, ali se, isto tako, može kontrolisati od strane inicijatora komunikacije, tj. centralnog računara. Ovo rješenje je testirano na Linux operativnom sistemu, ali se, isto tako, može primijeniti i na druge operativne sisteme koji omogućavaju redirekciju ulaza i izlaza na fajl (npr. UNIX, MS Windows i sl.).

2.2.2. E-mail

E-mail je metod razmjene digitalnih poruka između jednog pošiljaoca i jednog ili više primalaca. Neki najraniji e-mail sistemi su zahtijevali da pošiljalac i primalac budu istovremeno na mreži (slično kao kod instant messaging). Međutim, svi savremeni e-mail sistemi se zasnivanju na store and forward modelu, gdje e-mail server prima, proslijeđuje, čuva i isporučuje poruke, tako da ni korisnici ni njihovi računari ne moraju biti istovremeno na mreži, već je dovoljno da se na server povežu samo kratko vrijeme, dovoljno da pošalju ili prime poruku.

E-mail poruka je definisana u RFC5322 dokumentu, dok su prilozi sa multimedijalnim sadržajem (obično nazivani Multipurpose Internet Mail Extensions, odnosno MIME) definisani u dokumentima RFC 2045 do RFC 2049. Poruka se sastoji od zaglavlja (header) i tijela poruke (body). Zaglavlje i tijelo poruke odvojeni su jednom praznom linijom.

Svaka poruka sadrži tačno jedno zaglavlje koje je organizovano kao niz polja, koja se sastoje od naziva i vrijednosti. Jedno polje može da obuhvati i više linija. Svaka linija zaglavlja koja počinje nekim vidljivim znakom predstavlja početak polja. Naziv polja počinje prvim znakom u toj liniji, a završava neposredno prije separatora (znak dvotačka). Nakon separatora slijedi vrijednost polja, koja se može nastaviti i na naredne linije ukoliko počinju sa TAB ili razmakom. Ranije su imena i vrijednosti polja bili ograničeni na 7-bitne ASCII karaktere, međutim, u novije vrijeme sve veći broj sistema ima podršku za UTF-8 Unicode, što, pored ostalog, omogućava da se u nazivu email adresa koriste znakovi specifični za druge jezike osim engleskog.

Zaglavlje poruke mora sadržati bar slijedeća polja:

- **From**: email adresa i, neobavezno, ime autora i
- **Date**: lokalno vrijeme i datum slanja poruke.

Kod mnogih email klijenata korisnik ne može direktno da mijenja vrijednost ovih polja (osim kroz podešavanja sistema), već se ona automatski popunjavaju.

Zaglavlje bi trebalo da sadrži i polje Message-ID, čija je vrijednost automatski generisana i služi da bi se spriječio višestruki prijem iste poruke i kao referenca prilikom odgovora na poruku. Poruke odgovora trebalo bi da sadrže i polje In-Reply-To, čija je vrijednost Message-ID poruke na koju se odgovara.

Ostala uobičajena polja zaglavlja su:

- **To**: jedna ili više adresa i, opciono, imena primalaca. Ove adrese ne moraju imati veze sa stvarnim odredištima na koja se poruka šalje, a čija lista se isporučuje odvojeno u okviru SMTP protokola i može, ali ne mora, biti formirana na osnovu vrijednosti To polja.
- **Subject**: naslov poruke.
- **Cc**: Carbon copy. Vrijednost slično kao kod To polja, ali ih mnogi email klijenti različito označavaju u inbox-u.
- **Bcc**: Blind carbon copy. Adrese primalaca koje su nevidljive za ostale primaoce.

- **Content-Type**: Informacija o tome kako će sadržaj biti prikazan, obično MIME,
- **Precedence**: vrijednosti su obično “bulk”, “junk” ili “list”. Označava da se, kao odgovor, ne šalju automatske poruke o odsustvu primaoca,
- **References**: niz identifikatora roditeljskih poruka u niti kojoj pripada poruka,
- **Reply-To**: adresa koju bi trebalo koristiti prilikom slanja odgovora na datu poruku,
- **Sender**: adresa pošiljaoca. Najčešće je ista kao u polju From ali može i da se razlikuje, tj. da predstavlja adresu agenta koji šalje poruku u ime autora, gdje je adresa autora specificirana u polju From,
- **Archived-At**: direktna veza ka arhiviranoj formi poruke.

Tijelo poruke može se sastojati iz više djelova, zavisno od toga da li se uz tekst poruke šalju i datoteke. Ukoliko se šalju datoteke, to se u zaglavlju poruke može naznačiti na slijedeći način:

```
Content-Type: multipart/alternative;
      boundary="oznaka"
```

Tekst označen sa boundary služi za razdvajanje teksta i fajlova u tijelu poruke i ne smije se pojavljivati nigdje u tekstu poruke.

Datoteke su u poruci kodirane tako što se binarni sadržaj prevodi u tekst koristeći Base64 algoritam. Ovim algoritmom se tri bajta binarnih podataka prevode u četiri bajta teksta. Na taj način se veličina datoteke povećava za jednu trećinu, tako da je za prenos većih datoteka bolje koristiti neke druge protokole koji su posebno predviđeni za ovu namjenu i podržavaju kompresiju.

2.2.2.1. SMTP

SMTP (Simple Mail Transfer Protocol) je internet standard za prenos elektronske pošte. Prvi put je definisan 1982. godine u dokumentu RFC 821, a najnovija verzija, koja se sada najčešće koristi, je definisana 2008. u RFC 5321 [25].

Dok mail serveri i drugi mail agenti koriste SMTP i za slanje i za prijem poruka, klijentske aplikacije koriste SMTP samo za slanje poruka, dok za prijem koriste POP3 (Post Office Protocol) ili IMAP (Internet Message Access Protocol).

I mada webmail sistemi, kao što su GMail, Yahoo! mail ili Hotmail, za pristup mailbox-u na svojim serverima koriste svoje nestandardne protokole, prema spoljnim serverima, za prijem ili slanje poruka, obično koriste SMTP.

SMTP podrazumijevano koristi TCP port 25. Protokol za slanje poruka je isti, osim što koristi port 587, dok SMTP veza osigurana pomoću SSL (SMTPS) koristi port 465.

SMTP je tekstualni protokol, orijentisan na sesiju, gdje se komunikacija obavlja slanjem komandnih stringova i potrebnih podataka preko TCP kanala. SMTP sesija sadrži komande SMTP klijenta i odgovore SMTP servera kojima se otvara sesija i razmjenjuju podaci. Jedna sesija može sadržati nijednu ili više SMTP transakcija. Transakcija se sastoji od tri komande i odgovarajućih odgovora. Komande su slijedeće:

- **MAIL** komandom se uspostavlja povratna putanja, tj. zadaje se adresa na koju se isporučuju poruke sa različitim izvještajima o statusu isporučenog maila.
- **RCPT** komandom se zadaje adresa promaoca poruke. Ova komanda se može zadavati više puta, po jedna za svakog novog primaoca. Te adrese će biti uključene u ovojnicu poruke. SMTP protokol definiše način transporta poruke, a ne njen sadržaj, tako da se definiše ovojnica poruke (koja pored ostalog sadrži i adresu pošiljaoca), ali ne i zaglavlje poruke koje je definisano drugim standardom.
- **DATA** označava početak teksta poruke koji se sastoji od zaglavlja i tijela poruke odvojenih praznim redom. Kraj unosa teksta poruke označava se linijom koja sadrži samo tačku. DATA je, u stvari, grupa komandi i server na nju odgovara dva puta, prvi put da javi da je spreman za prihvatanje teksta poruke, a drugi put nakon završne sekvence da označi da li je poruka prihvaćena ili odbijena.

Pored odgovora na pojedinačne DATA komande, odgovor servera na kompletnu poruku može biti pozitivan (2xx kodovi) ili negativan. Negativni odgovor može biti trajni (5xx) ili prelazni (4xx). Reject je trajni tip negativnog odgovora koji označava trajnu nemogućnost SMTP servera da prihvati poruku i u tom slučaju se šalju povratne (bounce) poruke. Drop je tip pozitivnog odgovora koji je praćen odbacivanjem poruke.

Inicijalni pošiljalac poruke (SMTP klijent) može biti ili korisnički email klijent (mail user agent - MUA) ili SMTP server koji, da bi poslao poruku, obavlja funkciju klijenta (MTA - mail transfer agent). MUA informacije o SMTP serveru čuva u konfiguraciji, dok MTA adresu odlaznog SMTP servera određuje na osnovu naziva domena svakog pojedinačnog krajnjeg primaoca poruke. Kad SMTP server obavlja ulogu klijenta, on započinje TCP komunikaciju sa SMTP serverom na podrazumijevanom portu 25, dok mail user agenti koriste port 587.

2.2.2.2. IMAP

IMAP (Internet Message Access Protocol) je protokol aplikativnog nivoa koji omogućava preuzimanje elektronske pošte sa e-mail servera. Razvijen je 1986.god. kao alternativa za POP (Post Office Protocol), a najnovija verzija protokola (IMAP4rev1) definisana je u RFC3501 [26].

IMAP server koristi TCP port 143, dok IMAP preko SSL (IMAPS) server koristi port 993. Nakon uspostavljanja veze i *handshaking* faze, komunikacija se dalje obavlja na način što klijent serveru šalje tekstualne komande i podatke, a server izvršava komande i vraća odgovor. Na početku linije svake nove komande koju šalje klijent nalazi se jedinstvena oznaka za identifikaciju komande. Jedna komanda može biti raspoređena na više linija. Prilikom obrade komande, ako server konstatuje grešku, odgovoriće sa “BAD” i identifikatorom linije u kojoj je našao grešku. Ako je komanda uspješno izvršena, server šalje odgovor, a na kraju odgovora šalje se linija koja počinje identifikatorom linije komande i porukom “OK”. Ako komanda iz nekog razloga ne može biti izvršena, server vraća poruku “NO” uz identifikator komande koju nije izvršio.

IMAP korisniku omogućava sljedeće operacije:

- provjeravanje novih poruka,

- trajno brisanje poruka sa servera,
- kreiranje i brisanje poštanskih sandučića,
- podešavanje stanja poruke (nepročitana, pročitana, hitna, odgovorena, nova, obrisana i status formiranja poruke),
- čitanje atributa poruka (veličina, struktura, zaglavlje i tijelo poruke).

IMAP, za razliku od POP3 protokola, dozvoljava da više korisnika istovremeno pristupaju istom sandučetu, tj. dozvoljava istovremeni pristup sandučetu sa više lokacija.

Kod POP3 protokola klijent odmah prima poruke u cjelosti (zaglavlje, tijelo i sve pridružene datoteke), tako da nema mogućnost da odustane od prijema poruke u toku samog prijema, niti može da spriječi njen prijem. Pri prijemu korisnik sam odlučuje da li će neku poruku smatrati primljenom, što kod nekih aplikacija za prijem poruka može izazvati grešku da klijent iznova prima poruke koje je već pročitao. Osim toga, poruke na serveru se poslije nekog vremena automatski brišu, tako da klijent raspolaže samo lokalnom kopijom poruke.

Sa druge strane, IMAP podrazumijeva čuvanje i rad sa porukama na serveru, mada mogu da se čuvaju i lokalne kopije. Osim toga, prilikom prijema, klijent prvo prima samo zaglavlja poruka, a istovremeno je omogućena i pretraga po ključnim riječima cijelog sadržaja poruka na serveru, tako da klijent može da odluči da li želi da obriše poruku prije nego što je primi, što je naročito pogodno u slučaju nepoželjnih poruka.

IMAP komande

Komande dostupne u bilo kom stanju:

- CAPABILITY — klijent traži informaciju o mogućnostima servera, kao npr. dozvoljeni mehanizmi identifikacije,
- NOOP — prazna komanda, koristi se da se ne prekine veza kada istekne vrijeme (timeout) ili za periodično podsticanje primanja novih poruka ili aktualizaciju statusa postojećih poruka,
- LOGOUT — odjavljivanje klijenta.

Komande dostupne u neautorizovanom stanju:

- STARTTLS — proširenje običnog protokola komunikacije koje omogućava šifrovanu TCL vezu umjesto posebnih portova za šifrovanje komunikacija,
- LOGIN — prijavljivanje klijenta na sever poslije uspostavljanja veze. Argument sadrži ime naloga i šifru u svom osnovnom obliku, bez šifrovanja,
- AUTHENTICATE prijavljivanje uz pomoć određenog mehanizma šifrovanja iza kojeg slijede korisničko ime naloga i šifra klijenta.

Komande dostupne u autorizovanom stanju, kada je korisnik već prijavljen na server:

- SELECT — uspostavlja aktivnu fasciklu (direktorijum), odnosno sanduče sa kojim će

klijent dalje raditi (npr. INBOX, OUTBOX itd.),

- EXAMINE — komanda slična komandi SELECT, ali je sanduče namijenjeno samo za čitanje,
- CREATE — napraviti novo sanduče,
- DELETE — brisanje sandučeta,
- RENAME — promjena imena sandučeta,
- SUBSCRIBE — dodaje određeno sanduče u listu aktivnih računa klijenata na serveru. Komanda sadrži samo jedan argument: ime sandučeta korisnika, pri čemu ono ne mora da postoji prije dodavanja na listu,
- UNSUBSCRIBE — briše određeno sanduče sa liste aktivnih računa klijenata na serveru,
- LIST — zahtjev za sadržaj određenog sandučeta i njegovih fascikli (poddirektorijuma). Ukoliko je u pitanju trenutno aktivno sanduče, šalje se prazan niz "", kao argument komande,
- LSUB — vraća podskup iz seta poštanskih sandučića koje je korisnik označio kao aktivne na serveru,
- STATUS — upit o trenutnom stanju poruka u sandučetu, uključujući i broj nepročitanih poruka, pri čemu mogu da se zahtijevaju sljedeći kriterijumi:
    1. MESSAGES — ukupan broj poruka u sandučetu,
    2. RECENT — broj poruka sa zastavicom „nedavna“,
    3. UIDNEXT — jedinstveni identifikator za sljedeću novu poruku,
    4. UIDVALIDITY — jedinstveni identifikator sandučeta,
    5. UNSEEN — broj poruka koji ne nose zastavicu „pregledano“.
- APPEND — dodaje poruku u određeno elektronsko sanduče.

Komande dostupne u stanju SELECTED:

- CHECK — traži kontrolnu tačku za trenutno odabrano sanduče,
- CLOSE — zatvara poštansko sanduče,
- EXPUNGE — uklanja iz sandučeta sve poruke obilježene kao obrisane, pri čemu sanduče još uvek nije zatvoreno,
- SEARCH — pretraga poruka sa određenim filterom,
- FETCH — nakon dobijanja liste glavne fascikle (direktorijuma) sa poddirektorijumima (komandom LIST), klijent može zatražiti preuzimanje poruka koje se nalaze u njoj,

- STORE — pošto klijent pročita poruku, može da je obilježi određenom zastavicom za smiještanje ili za brisanje,
- COPY — kopiranje poruke iz određene fascikle,
- UID — koristi se u kombinaciji sa komandama SEARCH, FETCH ili STORE i vraća jedinstvene identifikacione brojeve poruka koje odgovaraju rezultatima tih komandi.

2.2.3. Realizacija softvera

Konkretno rješenje koje je ovdje realizovano sastoji se od dvije pomoćne aplikacije, od kojih se jedna nalazi na centralnom, a druga na udaljenom računaru. Obje su realizovane u programskom jeziku C++, ali to može biti bilo koji jezik koji omogućava pozivanje komandi operativnog sistema.

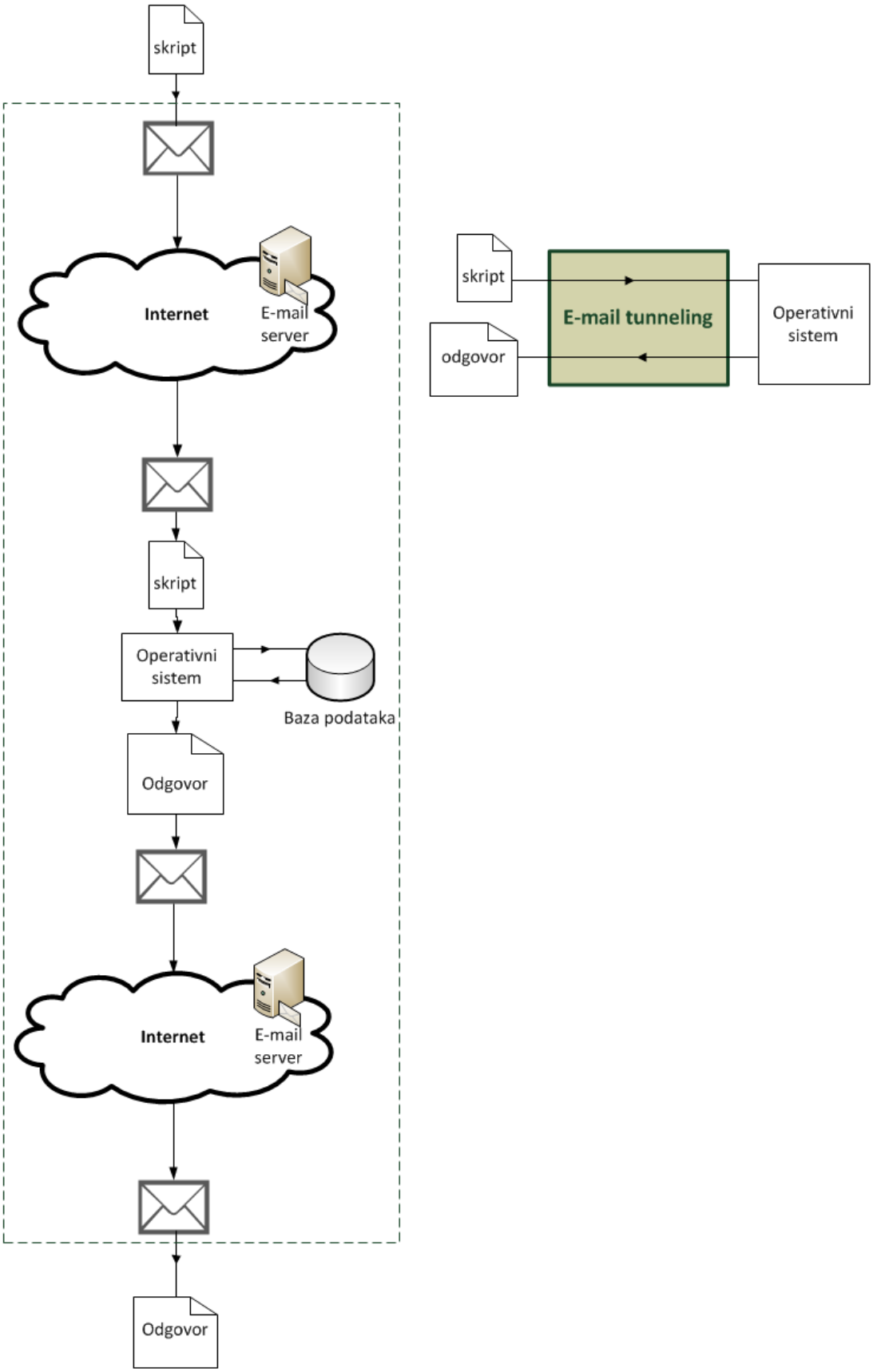
2.2.3.1. Aplikacija na centralnom računaru

Ova aplikacija se periodično pokreće i ima dva zadatka. Prvi zadatak je da kreira i šalje e-mail poruke prema udaljenim računarima, a drugi je da prima i interpretira poruke sa odgovorima udaljenih računara.

Slanje poruka je realizovano korišćenjem SMTP (Simple Mail Transfer Protocol). Naslov (subject) svih poruka sastoji se od niza polja odvojenih razmacima. Polja su slijedeća:

- id_izvora,
- id_izvorne_aplikacije,
- id_odredišta,
- id_odredišne_aplikacije,
- redni_broj.

Izvorne i odredišne e-mail adrese su iste za sve poruke, tj. sve poruke završavaju u istom sandučetu (mailbox). Računar određuje koja je poruka njemu namijenjena na osnovu vrijednosti polja id_odredišta. Format tijela poruke se razlikuje u zavisnosti od toga da li je poruka poslata sa centralnog ili udaljenog računara.



Slika 2.2: e-mail script-tunneling

Poruka poslata sa centralnog računara sadrži skript koji će se izvršiti na udaljenom računaru. Skript je oivičen <SCRIPT></SCRIPT> oznakama, što omogućava da ostatak tijela poruke može kasnije biti upotrebljen u druge svrhe.

U ovom konkretnom slučaju, skript sukcesivno izvršava sljedeće operacije:

1. Koristeći echo komande i redirekciju izlaza, kreira tekst fajl koji sadrži SQL upit.
2. Poziva mysql klijentsku aplikaciju koja kao ulaz uzima ranije formirani tekst fajl, a izlaz, tj. rezultat upita, preusmjerava u novi fajl.
3. Poziva aplikaciju za slanje e-maila, koja prethodno kreirani fajl sa rezultatima upita uzima kao ulaz. Polja u naslovu poruke popunjavaju se slijedećim vrijednostima: id_odredišta – tako da se poruka šalje centralnom računaru, id_*_aplikacije – tako da centralni računar po prijemu zna kako da interpretira odgovor.

Za prijem poruka korišćen je IMAP protokol koji, za razliku od POP protokola, omogućava da više klijenata bude istovremeno prijavljeno na isto sanduče (mailbox). Zahvaljujući tome moguće je da svi učesnici u komunikaciji dijele istu e-mail adresu, odnosno da pojedinačni računar ne mora da provjerava da li je još neko prijavljen na mailbox.

Prilikom prijema, aplikacija izvršava slijedeće korake:

1. Prijavljuje se na e-mail server.
2. Zahtijeva listu svih primljenih poruka, što uključuje: redni broj poruke u inbox-u, e-mail adresa izvora, naslov poruke i informaciju o tome da li je poruka već pročitana.
3. U zavisnosti od polja id_odredišta u naslovu poruke, filtrira poruke koje se odnose na taj računar i pravi listu njihovih rednih brojeva.
4. Na osnovu formirane liste čita jednu po jednu poruku i interpretira ih u zavisnosti od vrijednosti polja id_*_aplikacije.
5. Briše sve pročitane poruke, oslobađajući tako prostor na serveru za prijem novih poruka.
6. Odjavljuje se sa IMAP servera.

Najkraći period između dva uzastopna pokretanja aplikacije treba da bude dovoljno dug da omogući aplikaciji da pročita sve poruke upućene tom računaru, kao i listu sa osnovnim informacijama o svim porukama koje se trenutno čuvaju na serveru, tj. na zajedničkoj e-mail adresi. Zbog toga je neophodno da se poruke redovno brišu, ne samo one koje aplikacije međusobno razmjenjuju, već i poruke slučajno prispjele na ovu adresu. Međutim, ponekad može biti korisno da se poruke, prije brisanja, proslijeđuju na neku drugu e-mail adresu radi logovanja procesa.

Dodatna pouzdanost komunikacije realizovana je korišćenjem polja redni_broj u naslovu poruke. Aplikacija poredi vrijednosti ovog polja za poslate i primljene poruke, tako da može ispravno da rasporedi primljene odgovore. Takođe, može da registruje da neki odgovor nedostaje i da ponovo pošalje odgovarajući upit.

E-mail serveri često dodatno modifikuju tekst tijela poruke (npr. zamjenom tab i drugih format karaktera razmacima). Ovo može da predstavlja problem prilikom interpretacije odgovora

sadržanog u tekstu tijela poruke. Na primjer, ako neki proces kao odgovor vraća tabelu u kojoj su string polja odvojena TAB znakovima, nakon prijema poruke više neće biti moguće razlikovati razmake u string polju od graničnika između kolona. U konkretnom slučaju problem može da se riješi tako što se zahtijeva od procesa da dodatno oiviči string polja (npr. znakovima navoda). Međutim, opštije rješenje bi bilo da se komandni skript i odgovor, umjesto kao tekst poruke, šalju kao fajlovi u attachment-u, jer ih server tada neće modifikovati.

2.2.3.2. Aplikacija na udaljenom računaru

Zadatak ove aplikacije je da se periodično povezuje na e-mail server i da čita poruke namijenjene tom računaru. To obavlja kroz slijedeće korake:

1. Prijavljuje se na IMAP server.
2. Zahtijeva listu svih primljenih poruka (slično kao aplikacija sa centralnog računara).
3. U zavisnosti od polja id_odredišta iz naslova poruke, filtrira poruke namijenjene tom računaru i pravi listu njihovih rednih brojeva u sandučetu.
4. Čita jednu po jednu poruku iz prethodno kreirane liste i izvršava skript sadržan unutar <SCRIPT></SCRIPT> oznaka. Pošto su prijem (IMAP) i slanje poruka (SMTP) nezavisni, skript može poslati poruku sa odgovorom dok je još u toku čitanje poruke koja sadrži taj skript.
5. Briše sve pročitane poruke.
6. Odjavljuje se sa IMAP servera.

Slično kao kod aplikacije na centralnom računaru, moguće je, prije brisanja, poruke slati na neku drugu e-mail adresu radi logovanja.

U konkretnom slučaju aplikacija se povezuje na server svakih šest minuta. Vjerovatnoća uspješnog povezivanja na e-mail server je veoma velika (samo jedna ili dvije neuspješne konekcije dnevno). U normalnim uslovima, kada komandni skript sadrži samo jedan SELECT upit, konekcija i prijem poruka traje oko deset sekundi. Međutim, ako skript sadrži više INSERT ili UPDATE upita (ograničeno na maksimalno 500 upita po poruci), prijem može da potraje i do minut. Vrijeme potrebno za slanje poruka zavisi od broja redova odgovora na upit (takođe ograničeno na 500 redova po poruci) i takođe se kreće od deset sekundi do jedan minut.

Pošto ova aplikacija samo čita skript i komande iz njega proslijeđuje operativnom sistemu na izvršenje, nije potrebno da se modifikuje ukoliko se promijeni njena namjena, odnosno funkcija osnovne aplikacije, (što je u ovom slučaju registracija radnog vremena), tj. ukoliko se upotrebi u druge svrhe umjesto obraćanja mysql bazi.

2.3. Mogućnosti daljeg poboljšanja sistema

Pošto se koristi posrednik, tj. e-mail server, računari, koji, u svakom slučaju, moraju inicirati

komunikaciju, mogu biti sakriveni iza firewall-a ili NAT rutera. Iako postoje i druga rješenja koja zaobilaze NAT i firewall koristeći tunneling protokola, kao što je npr. VPN, prikazano rješenje ima prednost što se ne moraju otvarati dodatni portovi, jer se za IMAP i SMTP portove, u najvećem broju slučajeva, može računati da su otvoreni. Budući da računari nikad ne primaju dolazne konekcije, već su uvijek inicijatori, oni mogu da imaju i dinamičku adresu, tj. ne postoji potreba za servisima kao što su npr. dyndns. Ovo, takođe, otvara mogućnost korišćenja mobilne komunikacije, što je iskorišćeno i u konkretnom slučaju, gdje je korišćena GSM mreža zbog nedostatka kablovske veze.

Prikazano rješenje omogućava da originalna aplikacija ostane funkcionalna u novim okolnostima gdje računari samo povremeno pristupaju internetu. Osim toga, obezbjeđuje fleksibilnost i mogućnost jednostavne modifikacije sistema, jer je omogućena puna kontrola nad udaljenim računarom. Na primjer, moguće je jednostavno nadograditi sistem samo slanjem odgovarajućeg skripta. Sličnu mogućnost pružaju i neki drugi servisi kao što je TeamViewer, ali prednosti ovog rješenja su manji protok podataka, mogućnost automatizacije procesa i, kao što je ranije spomenuto, već obezbijeđeni slobodni portovi za komunikaciju.

Pošto opisano rješenje omogućava potpunu kontrolu udaljenog sistema, mora se uzeti u obzir i njegova bezbijednost. U ovom konkretnom slučaju oslanjalo se na osnovni nivo bezbijednosti koji obezbjeđuje sam e-mail server. Međutim, u nekim drugim okolnostima može biti neophodno uvesti i dodatne mehanizme zaštite. Obično se koriste dva tipa zaštite: provjera autentičnosti izvora i tajnost komunikacije. Za provjeru autentičnosti izvora obično se koristi digitalni sertifikat. Ukoliko je potrebno uvesti i tajnost komunikacije, najprije je pogodno skript ili odgovor iz tekst tijela poruke izmjestiti u fajl u attachment-u. Ovaj fajl tada može biti zaštićen nekom vrstom enkripcije.

Najzad, kako je ranije spomenuto, iako se sve pročitane poruke brišu sa servera, moguće je njihovo slanje na neku drugu adresu, čime je omogućeno otklanjanje grešaka u originalnoj aplikaciji. Redni broj u naslovu poruke obezbjeđuje ispravan redoslijed poruka, a poređenje rednih brojeva poslatih i primljenih poruka omogućava da se uoči nedostatak određene poruke, što dodatno poboljšava pouzdanost komunikacije.

Zaključak

Efikasan sistem nadzora proizvodnje omogućava proizvodnim kompanijama da izvuku maksimum iz svojih kapaciteta. U uslovima savremene industrijske proizvodnje, gdje je proizvodni proces najčešće složen i raspoređen na više organizacionih jedinica, a sama proizvodnja veoma intenzivna, sistem nadzora proizvodnje je, ne samo poželjan, već je često i jedini način da kompanija opstane na tržištu gdje vlada žestoka konkurencija.

Polazeći od najnižeg, fizičkog nivoa strukture, preko komunikacionih protokola, pa sve do aplikativnog nivoa, u ovom radu je detaljno opisana realizacija kompletnog sistema za nadzor proizvodnje koji sam razvio u konkretnim industrijskim uslovima i koji se u praksi već neko vrijeme uspješno primjenjuje.

Organizacija hardverskih komponenata realizovanog sistema, korišćenjem switched point-to-point topologije, obezbjeđuje da bilo koji dio sistema bude imun na smetnje nastale u nekom od ostalih djelova, što je naročito značajno u industrijskim uslovima, gdje su smetnje veoma učestale i uzrokovane brojnim faktorima, tako da ih je jako teško u potpunosti eliminisati. Dodavanje novih uređaja u sistem takođe neće uticati na funkcionalnost postojećih djelova sistema.

Tokom izrade softverskog dijela sistema za nadzor proizvodnje, naročita pažnja posvećena je izboru vrste podataka koje sistem treba da prikuplja, tako da menadžment kompanije može imati na raspolaganju kompletnu informaciju, kako o učinku i efikasnosti mašina u procesu, tako i o efikasnosti pojedinih radnika i njihovom učinku, bilo zbirnom ili onom vezanom za pojedinačnu mašinu.

Intenzivni razvoj “interneta stvari” (Internet of Things) i njegova široka zastupljenost u posljednjih nekoliko godina, otvara neke nove mogućnosti proširivanja sistema. Odranije široko zastupljene i provjereno pouzdane tehnologije, kroz Internet of Things pronalaze novu svrhu. Tako i rješenje koje sam razvio u drugom dijelu ovog rada i koje je, takođe, uspješno primijenjeno u praksi, na novi način koristi e-mail kao stabilnu i široko zastupljenu tehnologiju, omogućavajući da sistem, za proširenje na udaljene djelove prostora, može da iskoristi najrazličitije tipove mrežnih infrastruktura koje su mu na raspolaganju u datim okolnostima. Otkrivanjem drugačijih strana e-mail tehnologije, pogodnih za njenu primjenu u novom kontekstu komunikacije između uređaja, ovaj dio rada predstavlja i doprinos razvoju na polju interneta stvari.

Iako su se opisana rješenja pokazala dobra u praksi, ostao je otvoren širok prostor za dalje unapređenje sistema. Do sada sam uočio mogućnost poboljšanja na sljedećim poljima:

- Nadzor proizvodnje je trenutno napravljen kao desktop aplikacija. Umjesto toga, web orijentisana aplikacija bi povećala udobnost za korisnika, jer bi mogao pristupiti podacima sa bilo kojeg računara i sa bilo kojim internet pregledačem.

- Sistem je sada posvećen prikupljanju podataka sa mašina, uz tek malo iskorišćenu mogućnost daljinskog upravljanja mašinama. Daljom razradom ovih izlaznih funkcija ostvarila bi se mogućnost povećane automatizacije cjelokupnog procesa proizvodnje. Na primjer, zavisno od rezultata rada nekih mašina, mogla bi se optimizovati proizvodnja na drugim mašinama i tome slično.

- U dijelu e-mail sinhronizacije značajne mogućnosti unapređenja vidim u povećanju bezbijednosti komunikacije metodama autentifikacije i kriptovanja poruka.

- Postoji još puno mogućnosti unapređenja u hardveru, a kako vrijeme prolazi, na raspolaganju je sve više novih opcija.

Dodatak

Izvorni kod aplikacija opisanih u radu nalazi se na priloženom disku.

Literatura

[1] V. Manojlović, Z. Mijanović: *E-mail script tunneling*, 2nd Mediterranean Conference on Embedded Computing (MECO-2013), Budva, jun 2013.

[2] *Juran's quality handbook* / Joseph M. Juran, co-editor-in-chief, A. Blanton Godfrey, co-editor-in-chief. — 5th ed. McGraw-Hill, 1998.

[3] David Tran: *Factors in the successful implementation of six sigma in canadian manufacturing firms*, Eric S prott S chool of Business Carleton University Ottawa, Ontario, May 2006.

[4] Carr, Lawerence P. "Applying Cost of Quality to a Service Business." Sloan Management Review. Summer 1992: 72-77.

[5] Feigenbaum, Armand V. "*No Pain, No Gain*." Chief Executive. March 1997: 36-39.

[6] Gupta, M. and Campbell, V. S., (1995), "*The Cost of Quality*", Production and Inventory Management Journal, Vol. 36 No. 3, pp. 43-50.

[7] Mark De Feo: *The Juran Institute Research on Cost of Poor Quality*, August, 2005.

[8] https://en.wikipedia.org/wiki/Walter_A._Shewhart

[9] https://en.wikipedia.org/wiki/Control_chart

[10] https://en.wikipedia.org/wiki/Genichi_Taguchi

[11] https://en.wikipedia.org/wiki/Taguchi_methods

[12] http://asq.org/learn-about-quality/history-of-quality/overview/total-quality.html

[13] T. E. Kissell: *Industrial electronics: applications for programmable controllers, instrumentation and process control, and electrical machines and motor controls*, Third Edition, Prentice Hall, 2003.

[14] Telecommunications Industry Association: *TIA-232-F Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*, 1997.

[15] *TIA/EIA STANDARD, Electrical Characteristics of Balanced Voltage Digital Interface Circuits, TIA/EIA-422-B*, May 1994.

[16] ATmega16 datasheet, Atmel Corporation, 2010. (http://www.atmel.com/Images/2466s.pdf)

[17] LPC23xx User Manual, NXP Semiconductors, 2009.

http://www.keil.com/dd/docs/datashts/philips/lpc23xx_um.pdf

[18] J. Pipes , M. Kruckenberg: *Pro MySQL*, Apress, July 26, 2005.

[19] J. Blanchette; M. Summerfield: *C++ GUI Programming with Qt 4*, Second Edition, Prentice Hall, February 04, 2008.

[20] https://en.wikipedia.org/wiki/Internet_of_Things

[21] A Guide to the Internet of Things Infographic
(http://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html)

[22] https://projectmeshnet.org/

[23] J. F. Kurose, K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed., Pearson, Mar. 2009.

[24] R. K. Michael, *Mastering Unix Shell Scripting*, 2nd ed., Willey, Sep. 2011.

[25] J. Clensin, "Simple Mail Transfer Protocol," RFC5321, Oct. 2008.

[26] R. Crispin, "Internet Message Access Protocol – version 4rev1," RFC3501, Mar. 2003.