Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 621.317
DOI: https://doi.org/10.24867/09BE04Blagojevic

REALIZACIJA *LabVIEW* PROGRAMA KORIŠĆENJEM OBJEKTNO ORIJENTISANOG PROGRAMIRANJA

REALIZATION OF *LabVIEW* PROGRAM USING OBJECT ORIENTED PROGRAMMING

Milan Blagojević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Tema ovog master rada je korišćenje savremenih alata iz softverskog inženjeringa u cilju realizacije kvalitetnih programa za merenje i analizu signala u LabVIEW programskom paketu. Ova sinergija je nastala iz tesnog povezivanja metroloških i informacionih tehnologija prvenstveno korišćenjem PC računara u mernim aplikacijama. U radu će biti prikazano menjanje svojstava (properties) elemenata LabVIEW programa, zatim generisanje i korišćenje DLL biblioteka u LabVIEW programu i na kraju korišćenje ActiveX komponenti u LabVIEW programu. Takođe će biti prikazan i jedan program koji na jasan način prikazuju korišćenje ovih funkcija.*

Ključne reči: *LabVIEW, Virtualna instrumentacija, DLL biblioteke, ActiveX*

Abstract – *The topic of this paper is the use of modern tools in software engineering in order to realize quality programs for signal measurement and analysis in the LabVIEW software package. This synergy arose from the close integration of metrology and information technology primarily through the use of PCs in measurement applications. This paper will show how to change the properties of LabVIEW program elements, then generate and use DLLs in the LabVIEW program, and finally use ActiveX components in the LabVIEW program. There will also be one program that clearly demonstrates the use of these features.*

Keywords: *LabVIEW, Virtual instrumentation, DLL libraries, ActiveX*

1. UVOD

Krajem prošlog veka dolazi do razvoja računara i sve veće upotrebe u komercijalne svrhe. Ubrzo počinje i razvoj mernih sistema koji su zasnovani na softveru. Ovi sistemi se sastoje od računara, mernih kartica pomoću kojih se vrši merenje i softvera koji služi za izradu aplikacija i obradu podataka.

Ovakvi softveri daju veću slobodu korisniku da definiše merni softver prema svojim potrebama.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Josif Tomić, vanr. prof.

Merenja su osnova prirodnih nauka i svuda prisutan faktor moderne automatske proizvodnje, trgovine, saobraćaja, zaštite čovekove sredine, medicinske dijagnostike i lečenja. Merenja su takođe i ključni element istraživanja i razvoja u svakoj vrsti praktične delatnosti. Osnovne karakteristike savremenih merenja su:

- realizacija merenja gotovo svih fizičkih veličina električnim putem,
- pojava novih inteligentnih metoda merenja, koje su nastale iz tesnog povezivanja metroloških i informacionih sistema,
- tehnološki prelazak na sredstva merenja kod kojih softver postaje dominantan i ključni resurs razvoja i proizvodnje, umesto sredstava merenja sa hardverom kao dominantnim elementom, i
- korišćenje izuzetno složenih matematičkih algoritama, iz domena digitalne obrade signala u cilju dobijanja veoma preciznih rezultata merenja.

Virtualni instrumenti predstavljaju kombinaciju hardverskih i softverskih elemenata, najčešće korišćenih u varijanti sa personalnim računarom, u cilju realizacije klasičnog instrumenta. Oni kao takvi uzimaju prednosti iz savremene tehnologije ugrađene u personalne računare, koja se veoma brzo menja i usavršava zbog svoje ogromne masovnosti. Ova masovna proizvodnja sa svoje strane dovodi do veoma kvalitetnih uređaja po veoma niskim cenama. Virtualni instrument se sastoji od PC računara ili radne stanice, opremljen moćnim softverom za izradu aplikacija, jeftinim ugradnim ili USB mernim karticama i upravljačkim softverom, što zajedno čini funkcije tradicionalnog instrumenta.

Virtualni instrumenti predstavljaju pomeranje od tradicionalnih hardverskih instrumentacionih sistema u softverske sisteme koji eksploatišu računarsku snagu, produktivnost, izuzetno veliku mogućnost vizualizacije i grafičke prezentacije i mogućnosti povezivanja popularnih desktop kompjutera i radnih stanica. Mada su PC računari i tehnologija integrisanih kola značajno napredovali u poslednje tri decenije, softver je taj koji ima moć da postavi temelje za kreiranje virtualnih instrumenata, obezbeđujući bolje načine za inoviranje i značajno smanjujući troškove. Sa virtualnim instrumentima inženjeri i naučnici prave sisteme za merenje i automatizaciju koji tačno odgovaraju njihovim potrebama (*user-defined*) umesto da se ograniče tradicionalnim instrumentima sa fiksnim funkcijama (*vendor-defined*) [1].

2. *LABVIEW*

LabVIEW programski paket nastao je 1986. godine od strane američke kompanije *National Instruments*, sa ciljem da se obezbedi programski alat koji bi omogućio inženjerima da razvijaju svoje specifične aplikacije bez velikog poznavanja programiranja. Ovaj, takozvani G (grafički) jezik za inženjere, bio je odgovor na *spreadsheet* programske pakete koji su omogućavali ekonomistima da analiziraju finansijske podatke. *National Instruments* je na neki način pionir u virtualnoj instrumentaciji, novom načinu merenja, testiranja i automatizacije procesa. Ovakav način rada značajno je smanjio troškove i povećao efikasnost rada bez žrtvovanja tačnosti i kvaliteta merenja. Skraćenica *LabVIEW* potiče iz naziva *Laboratory Virtual Instrumentation Engineering Workbench*.

LabVIEW je grafički programski jezik koji za formiranje aplikacija koristi ikone umesto tekstualnog koda. Za razliku od tekstualnih programskih jezika, gde se izvršavanje programa određuje instrukcijama, u *LabVIEW*-u se za izvršavanje programa koristi protok podataka. Programi pisani, možda je bolje reći pravljeni (jer je tekstualni kod sveden na minimum), u *LabVIEW*-u, nazivaju se Virtualnim Instrumentima (skraćeno VI). Razlog tome je što njihova pojava i operacije imitiraju fizičke instrumente kao što su, npr. osciloskopi i multimetri. Pri tome treba biti obazriv sa korišćenjem termina *virtualni*, jer se pod njim u merenjima ne podrazumeva samo simulacija rada, već realan instrument koji poseduje neophodan hardver (najčešće akvizicionu karticu opšte ili posebne namene), a način rada se definiše programom. Kod objektno orijentisanih programskih jezika (kao *LabVIEW, Delphy, Visual Basic* i dr.) prednja ploča instrumenta se zamenjuje tzv. front panelom na ekranu računara. U daljem radu, pod terminom *Virtualni Instrument,* podrazumeva se program kreiran u *LabVIEW* programskom paketu, kome su pridružena dva prozora: front panel (*front panel*) i blok dijagram (*block diagram*).

U *LabVIEW*-u, korisnički interfejs tj. front panel se gradi posredstvom alatki nad skupom objekata. U *LabVIEW*-u objekti mogu biti upravljački i indikatorski, a putem njih korisnik može interaktivno da učestvuje u izvršavanju programa. Upravljački objekti predstavljaju ulazne terminale koji mogu biti razne varijante prekidača, preklopnika, klizača, potenciometara i ostalih ulaznih uređaja. Indikatorski objekti predstavljaju izlazne terminale koji mogu biti razne vrste grafika, LED dioda i ostalih vrsta prikazivača [2].

Nakon izgradnje front panela, dodaje se kod kojim se vrši upravljanje nad postavljenim objektima. Kod je grafička prezentacija funkcija kojim se to upravljanje obavlja. Blok dijagram je prozor koji se simultano otvara sa otvaranjem front panela i sadrži izvorni kod programa u grafičkom obliku. Grafički oblik je taj, koji daje akcenat na protok podataka i samim tim se problem postavlja u ravan jasnog fizičkog poimanja. Zapravo, grafički kod sadrži sve one elemente koje i tradicionalni programski jezici, promenljive, podatke, objekte, petlje i strukture, kao i mogućnost upravljanja greškama.

3. OBJEKTNO ORIJENTISANO PROGRAMIRANJE U *LabVIEW* PROGRAMU

Objektno programiranje pokazalo je svoju superiornost nad proceduralnim programiranjem. Prilikom rada sa objektno orijentisanim programom lakše je otklanjati greške koje nastaju prilikom izrade programa, a takođe omogućava lakšu izradu softvera u velikim programerskim timovima. Ideja objektnog programiranja je da se računarski program može posmatrati kao da sadrži zbir pojedinačnih jedinica ili objekata koji deluju jedni na druge, za razliku od tradicionalnog pogleda u kojem se program može posmatrati kao skup funkcija ili jednostavno kao lista uputstava za računar. Svaki objekat može da primi poruke, obrađuje podatke i šalje poruke drugim objektima. Objektno programiranje promoviše veću fleksibilnost i održivost u programiranju i popularno je u izradi velikih softverskih projekata. Ovaj način programiranja se smatra jednostavnijim, kako za učenje, tako i za razvoj i održavanje.

LabVIEW objektno programiranje koristi pojmove iz drugih objektno orijentisanih programskih jezika kao što su C++ i Java uključujući strukturu klase, enkapsulaciju i nasledstvo. Ove koncepte je moguće koristiti za kreiranje koda koji je lakši za održavanje i modifikaciju bez uticaja na druge delove koda unutar aplikacije. Moguće je koristiti objektno programiranje u *LabVIEW*-u da bi se kreirali korisnički definisani tipovi podataka. Ono što *LabVIEW* čini moćnim alatom je mogućnost da složene programerske tehnike koriste klijenti koji nemaju prethodno iskustvo sa programiranjem [6].

Menjanje svojstava (properties) elemenata *LabVIEW* programa odvija se preko čvorova (*Property Nodes*). U nekim aplikacijama, postoji potreba da programski izmenite izgled predmeta na front panelu kao odgovor na određene ulaze. Na primer, ako korisnik unese neispravnu lozinku, moguće je aktivirati crvenu LED diodu da počne da treperi ili da u zavisnosti od ulaza menjamo boju signala na grafikonu. Kada su vrednosti podataka iznad određene granice, moguće je da podesimo crveni trag umesto zelenog. Čvorovi omogućavaju da se ove modifikacije izvrše programski. Takođe, oni se mogu koristiti za promenu veličine predmeta na front panelu, sakrivanje delova panela, za programsko dodavanje kursora na grafikone i tako dalje. Čvorovi u *LabVIEW*-u su veoma moćni i imaju mnogo upotreba.

4. KORIŠĆENJE *DLL* BIBLIOTEKA

U operativnom sistemu *Windows*, deljena biblioteka se zove *DLL* (eng. *Dynamic Link Libraries - biblioteka dinamičke veze*). *DLL* je modul koji sadrži funkcije i podatke koje može koristiti drugi modul (aplikacija ili DLL). *DLL* može definisati dve vrste funkcija: eksportnu i unutrašnju. Eksportne funkcije su predviđene za pozivanje od strane drugih modula, kao i iz *DLL*-a gde su definisane. Unutrašnje funkcije se obično pozivaju samo iz *DLL*-a gde su definisane. Iako *DLL* može izvoziti podatke, njegovi podaci se uglavnom koriste samo u funkcijama. Međutim, ništa ne sprečava drugi modul da pročita ili napiše tu adresu. *DLL*-ovi pružaju način za modularizaciju aplikacija tako da se njihova funkcional-

nost lakše ažurira i ponovo koristi. *DLL* takođe pomažu u smanjenju zauzimanja memorije kada nekoliko aplikacija koristi istu funkciju istovremeno [4, 5].

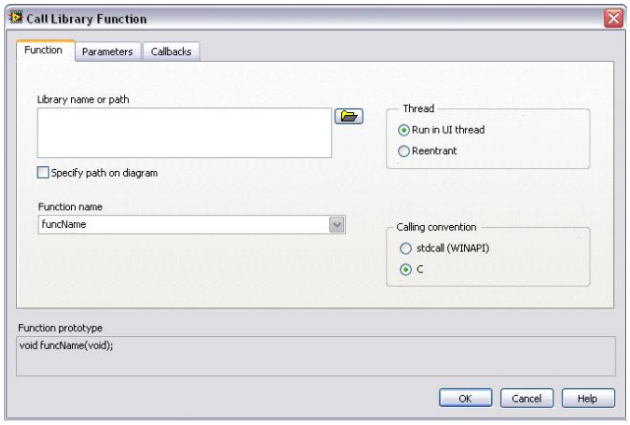
Većina standardnih deljenih biblioteka može se pozvati pomoću čvora funkcije poziva biblioteke (*Call Library Function Node*). U *Windows*-u, te deljene biblioteke su *DLL*-ovi, na Mac OS-u, to su *Frameworks*, a na *Linux*-u su *Shared Libraries*. Čvor funkcije poziva biblioteke uključuje veliki broj tipova podataka i konvencije pozivanja.

DLL ima sledeće prednosti:

- Može se menjati bez promene bilo koje od VI koje su povezane sa *DLL*-om tako da se ne menjaju ni prototipovi funkcija
- Praktično sva moderna razvojna okruženja omogućuju podršku za izradu *DLL*-a.

Ako je potrebno izvršenje nekog zadatka u tačno određeno vreme na raspolaganju je čvor funkcije poziva biblioteke. Čvor funkcije poziva biblioteke najprikladniji je kada imate postojeći kod koji želite da pozovete ili ako ste upoznati sa procesom stvaranja standardnih deljenih biblioteka. Budući da biblioteka koristi standard formata među nekoliko razvojnih okruženja, možete koristiti skoro svako razvojno okruženje da biste stvorili biblioteku koju *LabVIEW* može pozvati. *LabVIEW* kompajler u većini slučajeva generiše kod dovoljno brzo za potrebe programskih zadataka. Pozovite deljene biblioteke iz *LabVIEW*-a da biste izvršili zadatke na kojima tekstualni programski jezik može lakše da se izvrši, kao što su zadaci kod kojih je vreme ključni faktor.

National Instruments preporučuje korišćenje čvora funkcije poziva biblioteke za kreiranje interfejsa u *LabVIEW*-u da biste pozvali postojeće biblioteke ili nove biblioteke posebno napisane za upotrebu sa *LabVIEW*-om. Upotrebom dijalog prozora Funkcije poziva biblioteke, koji je prikazan na Slici 1, izaberite biblioteku, funkciju, parametre, povratnu vrednost za čvor, konvenciju poziva i povratne funkcije u sistemu *Windows*.



Slika 1. *Dijalog prozor Funkcije poziva biblioteke*

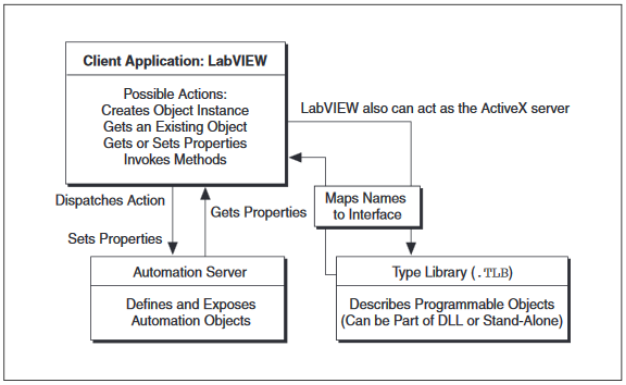
Konvencija o pozivanju definiše način prenošenja informacija iz dela koda u funkciju. Upotrebom kontrole Konvencija poziva na kartici Funkcija u dijalog prozoru Funkcija poziva biblioteke bira se konvencija poziva za funkciju. Podrazumevana konvencija poziva je C (u *Windows*-u je moguće koristiti i standardnu *Windows* konvenciju poziva, *stdcall*).

U operativnom sistemu sa više tredova moguće je istovremeno upućivati više poziva *DLL*-u ili deljenoj biblioteci. Svi čvorovi funkcije poziva biblioteke pokreću se u tredu korisničkog interfejsa. Kontrola treda na funkcijskoj kartici u dijalog prozoru Funkcija poziva biblioteke odražava vaš izbor *Run in UI Thread* ili *Reentrant*.

Pre nego što se konfiguriše čvor funkcije poziva biblioteke za ponovnu upotrebu, potrebno je proveriti da li više tredova (niti) može istovremeno pozvati funkciju. Sledeće karakteristike su osnovne karakteristike tred sigurnosnog koda u deljenoj biblioteci.
Kod je siguran za korišćenje tredova kada:
• ne pohranjuje nikakve globalne podatke, kao što su globalne promenljive, datoteke na disku i slično.
• ne pristupa hardveru, tj. kod ne sadrži programiranje na nivou registra.
• ne poziva funkcije, deljene biblioteke ili upravljačke programe koji nisu bezbedni za korišćenje u nitima.
• poziva ga samo jedan *non-reentrant* VI.

5. KORIŠĆENJE *ActiveX* KOMPONENTI

LabVIEW pruža pristup drugim *Windows* aplikacijama pomoću *.NET* ili *ActiveX* tehnologija.
.NET se odnosi na *Microsoft*-ovu *.NET* tehnologiju. *.NET Framework* je osnova programiranja u *.NET* okruženju koje koristite za pravljenje, implementaciju i pokretanje veb aplikacija, pametnih klijentskih aplikacija i XML veb usluga.
ActiveX se odnosi na *Microsoft*-ovu *ActiveX* tehnologiju i OLE tehnologiju. Pomoću *ActiveX Automation*, *Windows* aplikacija, poput *LabVIEW*, pruža javni skup objekata, naredbi i funkcija kojima druge *Windows* aplikacije mogu pristupiti. Moguće je koristiti *LabVIEW* kao *ActiveX* klijent za pristup objektima, svojstvima, metodama i događajima povezanim sa drugim aplikacijama koje podržavaju *ActiveX*. *LabVIEW* takođe može da se koristi kao *ActiveX* server, tako da druge aplikacije mogu pristupiti *LabVIEW* objektima, svojstvima i metodama [4, 7]. Na Slici 2. Prikazano je kako *LabVIEW* vrši interakciju kada se koristi kao *ActiveX* klijent.



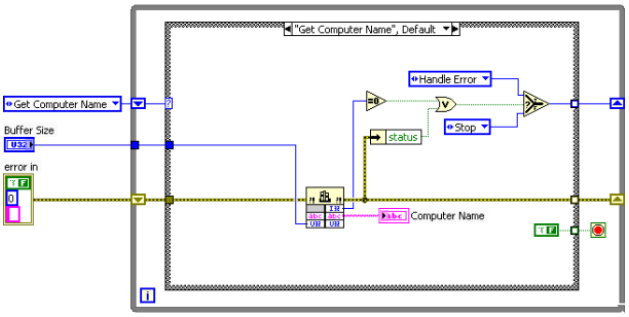
Slika 2. *Prikaz kako LabVIEW vrši interakciju kada se koristi kao ActiveX klijent*

6. PRIMER ODREĐIVANJA IMENA RAČUNARA

Programsko određivanje imena *Windows* računara neophodno je u mnogim situacijama. Na primer, ako postoji nekoliko računara koji beleže podatke na centralnom serveru, trebalo bi odrediti ime računara za svaki skup podataka da bi podaci mogli da se prate. Ime računara takođe je korisno za čuvanje ili preuzimanje podataka preko *Windows File Share*-a i koristan je podatak koji treba da se obuhvati prilikom generisanja izveštaja. U *LabVIEW* nema matične funkcije za preuzimanje računarskog imena. Međutim, *Windows* operativni sistem obezbeđuje API u obliku *DLL*-ova koji vam omogućava interfejs sa operativnim sistemom. Jedna uobičajena upotreba ovog API-ja je pronalaženje korisnih informacija o operativnom sistemu ili računaru, kao što je ime računara. Na Slici 3. prikazan je front panel realizovanog programa.

Slika 3. *Prikaz Front panela*

Kada pozivate funkcije iz *DLL*-a, *DLL* često definiše metod prijavljivanja grešaka. *Windows* API vraća brojčanu vrednost iz svakog poziva funkcije koji pokazuje da li je funkcija pravilno izvedena. U slučaju pogrešnog izvršenja, možete pozvati dodatne *Windows* API funkcije da biste identifikovali kod greške i pretvorili kod u razumljiv string. Da bi ispravno identifikovali grešku, funkcije za rukovanje greškama moraju se pozvati odmah nakon poziva originalne funkcije, jer će se informacije o grešci izgubiti ako se bilo koji drugi *Windows* API poziv izvrši iz istog programa. Ovaj mehanizam za upravljanje greškama funkcioniše drugačije od uobičajenog mehanizma za upravljanje greškama u *LabVIEW*, gde svaki pod VI izveštava o sopstvenim greškama, a greške se mogu povezati i obraditi na kraju programa ili sekcije. Da biste implementirali ovaj mehanizam za upravljanje greškama u *LabVIEW*, koristite mašinu stanja za proveru povratne vrednosti svakog poziva funkcije. Na Slici 4. prikazan je blok diagram realizovanog programa.



Slika 4. *Prikaz Blok dijagrama*

Za ovaj program koristite mašinu stanja (*State Machine*) sa tri stanja: Odredi ime računara (*Get Computer Name*), rukovanje greškom (*Handle Error*) i zaustavljanje (*Stop*). Prvo stanje preuzima ime računara. Ako je uspešno, program prelazi u stanje *Stop* i završava se. Ako *DLL* poziv za dobijanje računarskog imena ne uspe ili ako se dogodi normalna *LabVIEW* greška, program ulazi u stanje rukovanja greškom, gde pristupa kodu greške i prevodi kod greške u poruku o grešci.

Windows API obezbeđuje funkciju koja se u *Kernel32.dll* zove *GetComputerName*. Morate dodeliti bafer dovoljno velik da sačuva niz karaktera koji je vratila funkcija, srećom, čvor poziva biblioteke u *LabVIEW* verziji 8.2 ili novijoj može automatski dodeliti bafer na osnovu parametra veličine koji je prosleđen funkciji. Ako je bafer previše mali da biste sačuvali ime računara ili se dogodi druga greška, ova funkcija vraća vrednost nula. Koristite povratnu vrednost iz ove funkcije kao i klaster greške iz čvora poziva biblioteke da biste kontrolisali logiku promene mašine stanja.

7. ZAKLJUČAK

U ovom radu prikazan je rad sa softverskim paketom *LabVIEW*. U radu je prikazano menjanje svojstava (*properties*) elemenata *LabVIEW* programa, zatim generisanje i korišćenje *DLL* biblioteka u *LabVIEW* programu i na kraju korišćenje *ActiveX* komponenti u *LabVIEW* programu. Ovakav način rada znatno povećava mogućnosti i fleksibilnost *LabVIEW* programa jer omogućava komunikaciju sa drugim programima kao i prikaz podataka u programu koji korisniku najviše odgovara. Kao izuzetno moćan softverski alat, *LabVIEW* pruža velike mogućnosti za razvoj složenih programa koji se mogu koristiti prilikom merenja, praćenja i upravljanja složenim procesima.

8. LITERATURA

[1] Josif Tomić, Miodrag Kušljević, Merenje i analiza signala primenom *LabVIEW* programa, FTN-GRID Novi Sad, 2016.
[2] *LabVIEW* Basics I: Introduction Course Manual, National Instruments Corporation, Ostin, Teksas, USA, 2006.
[3] *LabVIEW* Basics II: Development Course Manual, National Instruments Corporation, Ostin, Teksas, USA, 2003.
[4] *LabVIEW* Intermediate II Connectivity Course Manual, National Instruments Corporation, Ostin, Teksas, USA, 2007.
[5] Dynamic-Link Libraries, https://docs.microsoft.com, preuzeto 26.11.2019.
[6] Milan Blagojević, Objektno programiranje u *LabVIEW* programu, diplomski rad, FTN Novi Sad, 2018.
[7] R. Bitter, T. Mohiuddin, M. R. Nawrocki, *LabVIEW* advanced programming techniques, Taylor & Francis Group, USA 2007.

Kratka biografija:

Milan Blagojević rođen je u Bačkoj Topoli 1990. godine. Osnovne akademske studije na Fakultetu tehničkih nauka završio je 2018. godine. Master studije upisao je iste godine.