UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Vitorovič

Kontrolni sistem pospeševalnika delcev v okolju LabVIEW

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Saša Divjak

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01785/2011
Datum: 02.11.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA VITOROVIČ**

Naslov: **KONTROLNI SISTEM POSPEŠEVALNIKA DELCEV V OKOLJU LABVIEW**

CONTROL SYSTEM FOR PARTICLE ACCELERATOR IN LABVIEW ENVIRONMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Izdelajte kontrolni sistem pospeševalnika delcev v programskem okolju LabVIEW. Najprej opišite delovanje sinhrotronskega pospeševalnika in njegovih sestavnih delov. Podajte, kako kontrolni sistem nadzoruje delovanje pospeševalnika. Opišite programsko okolje LabVIEW, grafični jezik G in način programiranja v le-tem. Opredelite omejitve jezika. Opišite konkretno implementacijo kontrolnega sistema, njegovo delovanje, sestavo in razložite posamezne komponente. Dokumentirajte vmesnik API in pomožne knjižnice, ki jih boste razvili ter naštejte nekaj aplikacij, ki so del kontrolnega sistema pospeševalnika.

Mentor:

prof. dr. Saša Divjak

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Miha Vitorovič,

z vpisno številko 24015452,

sem avtor diplomskega dela z naslovom:

Kontrolni sistem pospeševalnika delcev v okolju LabVIEW

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Saše Divjaka;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 2.11.2011 Podpis avtorja: Miha Vitorovič

Zahvala

Za pomoč pri izdelavi diplomske naloge se zahvaljujem mentorju prof. dr. Saši Divjaku, podjetju Cosylab d.d. in sodelavcem, ki sodelujejo z mano na projektu MedAustron, predvsem Jožetu Dediču in Mateju Šekoranji.

Zahvaljujem se tudi svoji ženi Poloni za vzpodbudo pri zadnjih izpitih in pisanju diplomskega dela in svoji mami za podporo v letih študija in potrpežljivost po tem.

Kazalo

Seznam uporabljenih kratic in simbolov

API – application programming interface

DOM – document object model

DPE – data point element

FEC – front end controller

FECOS – front end control system

FED – front end device

FIFO – first in, first out

FPGA – field programmable gate array

HTTP – hypertext transfer protocol

MAPS – MedAustron publish/subscribe

MTS – master timing system

OPC – OLE for process control

PCC – power converter controller

PXI – PCI extensions for instrumentation

RF – radio frequency

RMS – repository management system

SCADA – supervisory control and data acquisition

SIM – simple interface messaging

STM – simple TCP/IP messaging

SV – shared variable

SV-PVA – shared variable – public variable access

TCP – transmission control protocol

URL – universal resource location

UUID – universally unique identifier

VAA – virtual accelerator allocator

VI – virtual instrument

XML – extensible markup language

Oznake v besedilu

V besedilu se uporabljajo naslednje oznake:

- *poševno* so označene besede s posebnim pomenom ali izrazi, ki jih želimo posebej izpostaviti in originalni (angleški) izrazi, ki se pojavljajo v besedilu;
- **krepko** so označena imena funkcij, tipov okolja LabVIEW in imena parametrov v funkcijah;
- `s pisavo enakomerne širine` so označena imena razredov, razrednih metod, lastnosti in tipov sistema FECOS.

Povzetek

Diplomska naloga obravnava izdelavo kontrolnega sistema pospeševalnika delcev v programskem okolju LabVIEW. V prvem poglavju podaja pregleden opis delovanja sinhrotronskega pospeševalnika in njegovih sestavnih delov. Drugo poglavje opisuje kako kontrolni sistem nadzoruje delovanje pospeševalnika. V tretjem poglavju je opisano programsko okolje LabVIEW, grafični jezik G in način programiranja v le-tem. Našteva tudi nekatere omejitve jezika in kako smo jih zaobšli. Peto poglavje opisuje konkretno implementacijo kontrolnega sistema, njegovo delovanje, sestavo in natančneje razloži posamezne komponente. Zajame tudi vmesnik API in pomožne knjižnice, ki smo jih razvili ter našteje nekatere aplikacije, ki so del kontrolnega sistema pospeševalnika.

Ključne besede: kontrolni sistem, pospeševalnik delcev, LabVIEW

Abstract

The thesis presents the implementation of a control system for particle accelerator in the LabVIEW development environment. The first chapter gives an overview of the operation of the synchrotron accelerator and its parts. The second chapter describes how control system controls the accelerator. The third chapter gives an overview of the LabVIEW development environment, graphical language G and explains how graphical programs are written. It also lists some limitations of the language and how we worked around them. The fifth chapter implements our implementation of the control system framework, how it works, its structure and a detailed description of its components. It also describes the API and various support libraries developed for the system. The thesis concludes with a brief description of some of applications written for the control system.

Keywords: control system, particle accelerator, LabVIEW

1 Uvod

Pospeševalniki delcev se uporabljajo v raziskovalne in zadnjem času vse pogosteje medicinske namene. Glede na to, da so v uporabi že od začetka dvajsetega stoletja vidimo, da se pospeševanje delcev lahko uravnava tudi z analognimi napravami. Seveda v modernem času pospeševalnike krmilimo izključno z računalniškimi kontrolnimi sistemi.

Ker je pospeševalnik delcev sestavljen iz mnogo različnih sestavnih delov je naloga kontrolnega sistema tudi, da upravlja z vsakim od njih. Zato o kontrolnem sistemu lahko govorimo na dva načina. V enem pogledu gre za množico aplikacij, ki skrbijo vsaka za svoj tip končne naprave. Te aplikacije so ponavadi napisane s strani več različnih skupin, zato bi bilo neučinkovito, da bi vsaka od skupin napisala samostojno aplikacijo in na svoj način reševala določene skupne probleme.

Kontrolni sistem pospeševalnika v ožjem smislu pa je sistem in ogrodje, ki aplikacijam za nadzor končnih naprav nudi skupno osnovo za delovanje in skrbi za osnovne storitve na standarden način. Lahko rečemo, da je kontrolni sistem v tem smislu neke vrste operacijski sistem končnih naprav pospeševalnika.

Cilj te diplomske naloge je razvoj kontrolnega sistema v drugem, ožjem pomenu. Kot razvojno okolje je izbrano okolje LabVIEW podjetja National Instruments. Z vsemi končnimi napravami pospeševalnika upravljajo industrijski računalniki National Instruments PXI, na katerih teče operacijski sistem v realnem času ETS Phar Lap z moduli, ki podpirajo delovanje aplikacij napisanih v okolju LabVIEW.

Delo predstavljeno v tej diplomski nalogi je narejeno v okviru krovne pogodbe med podjetjema Cosylab d.d. in EBG MedAustron GmbH za izdelavo celotnega kontrolnega sistema raziskovalno-medicinskega pospeševalnika.

Za nekatere od aplikacij kontrolnega sistema, ki upravljajo s konkretnimi končnimi napravami pospeševalnika, so poskrbeli sodelavci iz podjetja Cosylab d.d..

2 Sinhrotronski pospeševalnik delcev

Za lažje razumevanje teme diplomske naloge poglavje podaja kratek opis delovanja sinhrotronskega pospeševalnika - sinhrotrona.

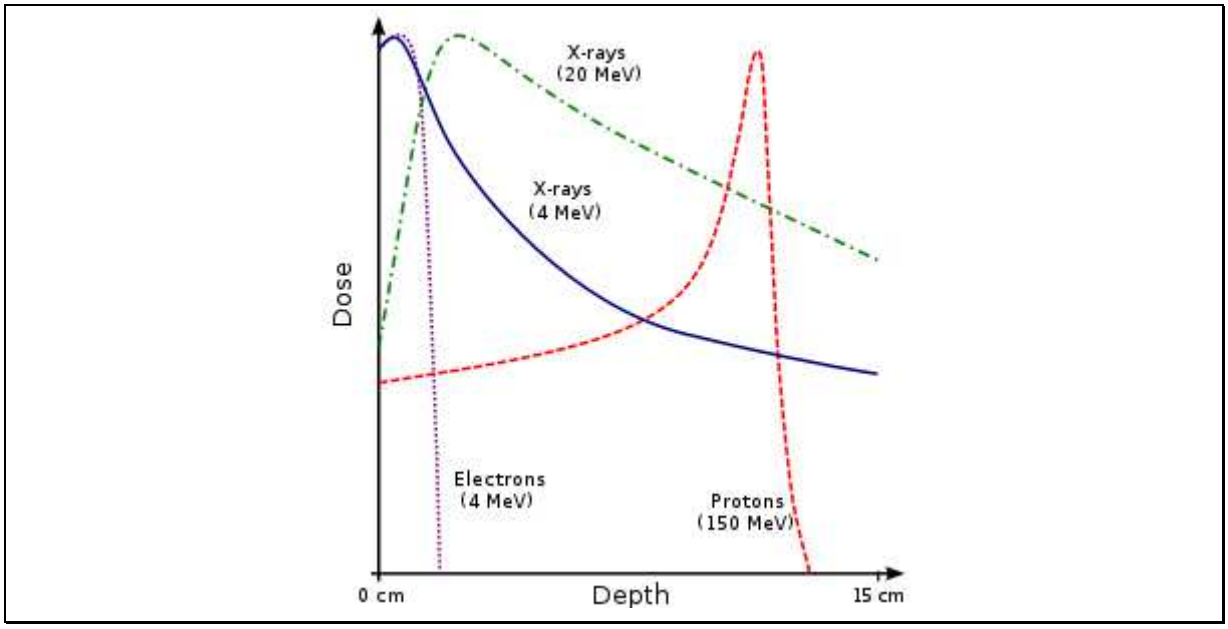
2.1 Namen in uporaba pospeševalnika

Sinhrotron je najmodernejša oblika pospeševalnika delcev, v katerem pospešujemo nabite delce, kot so elektroni, protoni ali ioni do skoraj svetlobne hitrosti. Pri raziskovalnih pospeševalnikih je torej želja po čim večji hitrosti oziroma energiji, saj lahko na ta način pri trku delcev pridejo do zanimivih rezultatov.

Sinhrotron je lahko tudi vir t.i. sinhrotronske svetlobe. Sinhrotronska svetloba je sicer stranski produkt delovanja pospeševalnika, vendar trenutno večino pospeševalnikov gradijo prav z namenom pridobivanja le-te, saj ima izjemno zanimive lastnosti in je primerna za veliko različnih raziskav in eksperimentov.

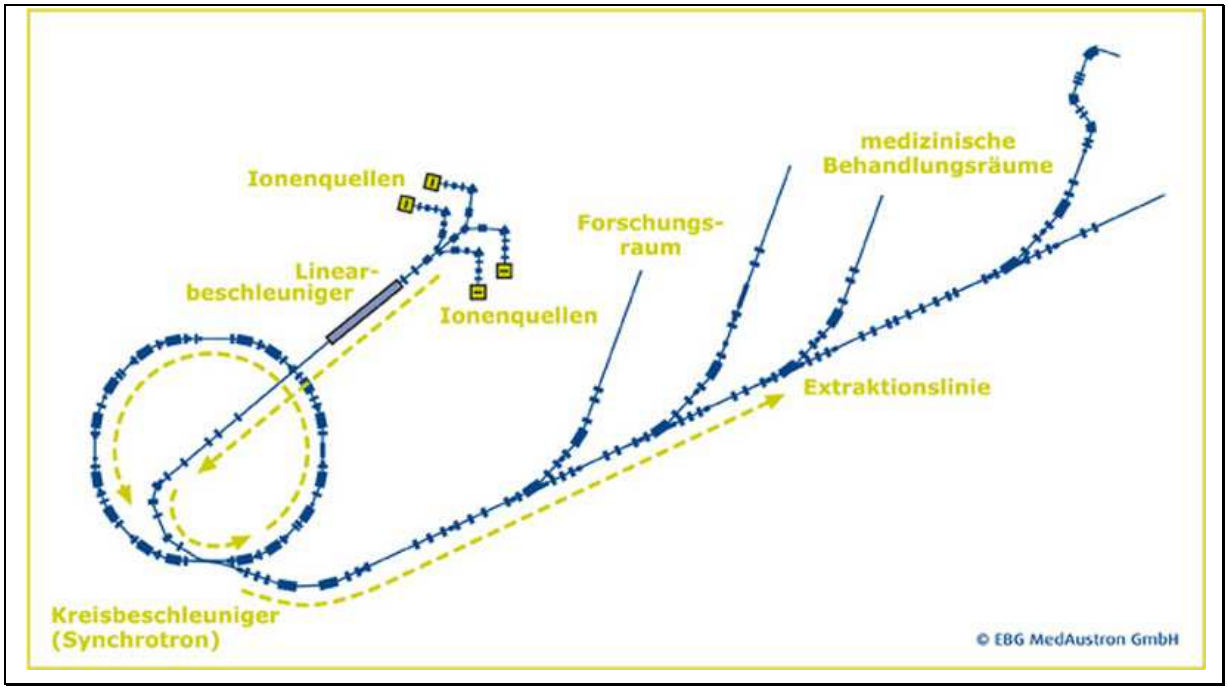
V zadnjem času se vedno bolj razvija tudi uporaba sinhrotronov za ionsko terapijo pri zdravljenju rakavih obolenj [1]. Pri obsevanju se poleg rentgenske svetlobe lahko uporablja tudi elektrone, protone ali ione. Obsevanje z rentgensko svetlobo je manj primerno od obsevanja z delci, saj je težko omejiti globino učinka. Elektroni se zaradi majhne mase lahko uporabljajo le za obsevanje kože, obsevanje s protoni ali ioni pa je najprimernejše, saj le-ti največ energije sprostijo na točno določeni globini, ki je odvisna od energije.

Slika 1 [1] prikazuje kako se sprošča energija v telesu glede na tip sevanja in lepo prikaže, da protoni z energijo 150MeV sprostijo največ energije na globini okrog 13 cm. Po tem energija hitro pade na 0, pred tem pa se je tudi sprosti relativno malo. Na ta način se lahko s prilagajanjem energije lokalizira učinek na globino tumorja.



Slika 1: Sproščena energija glede na tip obsevanja

V pospeševalniku MedAustron se bo za obsevanje uporabljalo protone ali ione ogljika. Protoni bodo dosegali energije med 60 in 250 MeV, ioni ogljika pa med 120 in 400 MeV. Sam sinhrotron bo imel premer približno 25m, celoten kompleks skupaj z linearnim pospeševalnikom, sobami za obsevanje in fizikalne raziskave pa bo meril približno 100x200m. Shemo pospeševalnika MedAustron prikazuje Slika 2 [2].



Slika 2: Shema pospeševalnika EBG MedAustron

2.2 Komponente pospeševalnika

Pospeševalnik je sestavljen iz več delov in kontrolni sistem upravlja z vsemi, da dobimo v pospeševalniku t.i. »žarek«. Žarek je v bistvu sestavljen iz več t.i. *gruč* delcev, ki pa se obnašajo podobno kot svetloba, zato tudi govorimo o žarku.

Na začetku pospeševalnika je vir delcev, oziroma v pospeševalniku MedAustron sta to dva vira ionov [3]. Pri obeh je postopek proizvajanja ionov dokaj zapleten ter zahteva dolgo pripravo naprav (do 30 minut), pri čemer so zelo pomembni napetost, tok, temperatura naprave in še veliko drugih parametrov, ki vplivajo na delovanje vira ionov.

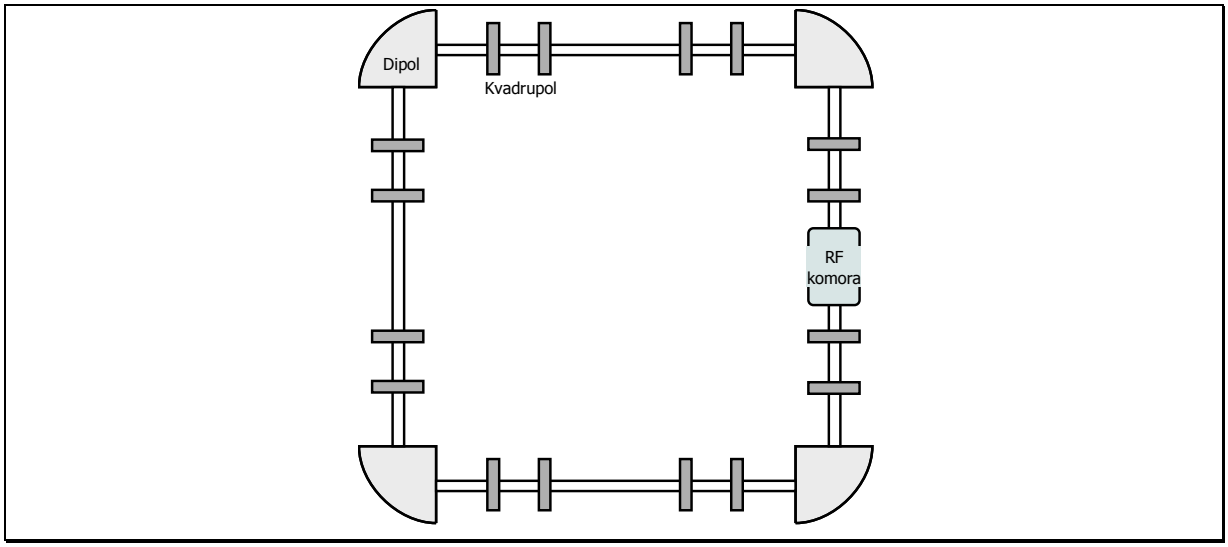
Delci potem vstopijo v linearni predpospeševalnik, kjer gredo skozi nekaj radio-frekvenčnih komor, ki ustvarjajo potreben potencial za pospeševanje delcev. Pri tem pridobijo že del energije, tako da v ciklotron vstopijo že delno pospešeni [4].

Pri delovanju sinhrotrona moramo poskrbeti za:

- odstranitev ovir na poti;
- pospeševanje delcev;
- vodenje delcev po sklenjeni poti.

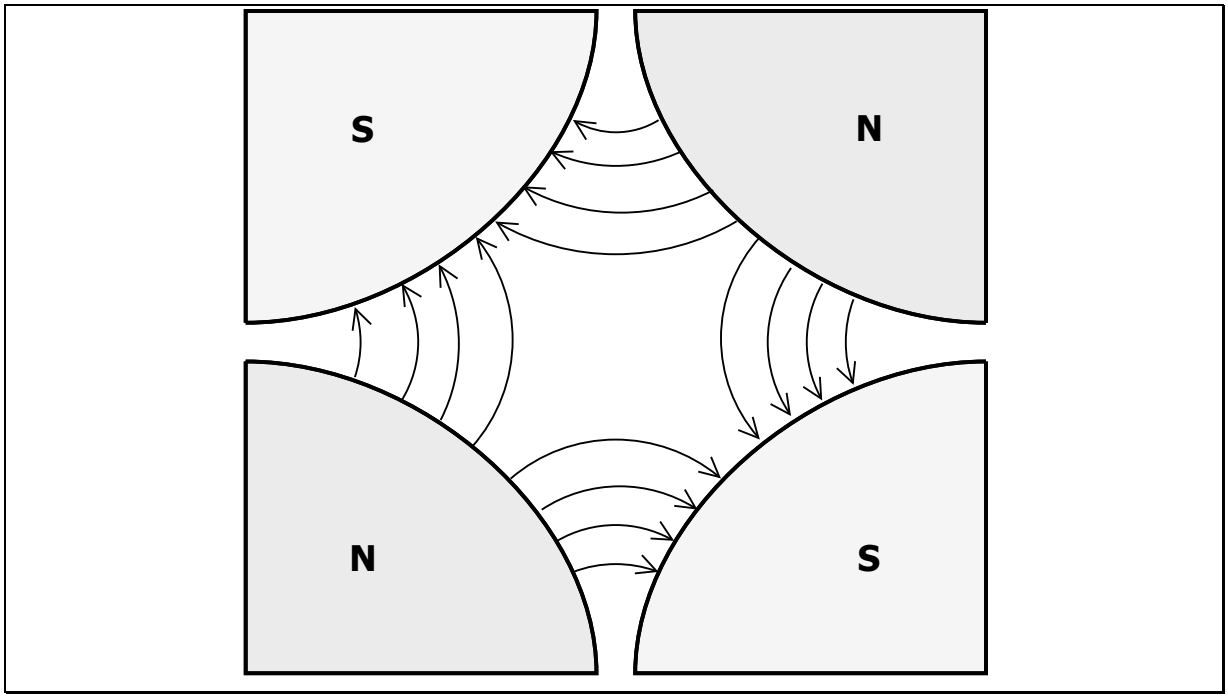
Za odstranitev ovir na poti delcev poskrbimo z ustvarjanjem zelo dobrega vakuuma. Delci se zato ponavadi gibljejo znotraj evakuirane cevi iz nerjavečega jekla.

Za pospeševanje in usmerjanje delcev se uporablja serijo magnetov in RF komora, kot prikazuje Slika 3.



Slika 3: Shema sinhrotrona

Delci potujejo skozi RF komoro, ki jih pospeši. Da potujejo v krogu gredo skozi dipolne magnete, v katerih je smer magnetnega polja pravokotna na ravnino pospeševalnika, zato delci zavijejo ravno prav, da vstopijo v naslednjo cev. Ker pa nimajo vsi delci enake energije, se ne uklonijo vsi enako; rečemo, da žarek izgubi fokus. Gručo delcev potem fokusiramo tako, da potuje skozi kvadrupolni magnet, ki ga prikazuje Slika 4 [5]. Ker gručo fokusira le v eni ravnini, je potrebno kvadrupolne magnete postaviti v parih, pri čemer je drugi magnet glede na prvega obrnjen za 90°.



Slika 4: Kvadrupolni magnet

Za pospeševanje delcev večamo frekvenco v RF komori pospeševalnika in zaradi večje energije delcev moramo povečati tudi magnetno polje magnetov. Ker to počnemo usklajeno (sinhrono) z RF komoro, je tak tip pospeševalnika imenovan *sinhrotron*.

Ko delci v sinhrotronu dosežejo ciljno energijo se RF komoro izključi, kar prekine pospeševanje, magneti pa ob tem ostanejo na zadnjih nastavitvah. Delci potem krožijo po sinhrotronu brez izgube energije. Za obsevanje se žarek iz obroča preusmeri v sistem za obsevanje.

3 Kontrolni sistem pospeševalnika

V poglavju 2.2 je opisano kako je pospeševalnik sestavljen, to poglavje pa opisuje, na kakšen način se upravlja njegovo delovanje.

3.1 Namen kontrolnega sistema

Kontrolni sistem pospeševalnika mora upravljati z vsemi končnimi napravami pospeševalnika, kar pomeni, da centralni strežnik pošilja ustrezne ukaze končnim napravam, ki potem s spremembo parametrov delovanja določajo kako pospeševalnik deluje.

Pospeševalnik ni centraliziran sistem v smislu, da centralni strežnik neposredno nadzira delovanje končnih naprav, ampak je distribuiran sistem, kjer je ob pospeševalniku razporejenih več kontrolnih računalnikov končnih naprav, ki prejemajo in izvajajo ukaze centralnega strežnika.

Pri kontrolnem sistemu pospeševalnika je najbolj pomembno dejstvo, da morajo biti vsi sestavni deli kontrolnega sistema zelo dobro časovno usklajeni. Za primer, obhodni čas gruče protonov v pospeševalniku MedAustron je pri energiji 60 MeV 0,75 µs. Najpomembnejši sestavni del pospeševalnika je torej časovni sistem. Le-ta preko optičnih povezav enake dolžine distribuira sistemsko uro, kar omogoča, da vsi kontrolni računalniki končnih naprav delujejo popolnoma usklajeno. Poleg tega časovni krmilnik po navodilih centralnega strežnika kontrolnim računalnikom končnih naprav pošilja ukaze ob točno določenih trenutkih. Celoten sistem deluje z natančnostjo, ki je boljša od 10 µs.

Kontrolni sistem pospeševalnika skrbi za vakuum v pospeševalniku in za to, da se prižge in izbere izvor delcev. Nadalje nadzoruje koliko gruč vstopi v linearni predpospeševalnik in kdaj. Na koncu linearnega pospeševalnika je potrebno gruče delcev speljati v sinhrotron, pri čemer so v sinhrotronu v tistem času že prisotni delci. Delce je potrebno nato pospešiti do določene energije, kar pomeni, da je potrebno spreminjati nastavitve RF komore in vseh magnetov v ciklotronu.

Ko je pospeševanje končano, je potrebno v pravem trenutku delce spet speljati iz sinhrotrona v sistem za obsevanje. Kot že rečeno, gre za distribuiran sistem, tako da je implementacija opisanih postopkov v aplikacijah na kontrolnih računalnikih končnih naprav. Kontrolni sistem končnih naprav (FECOS) preko optičnih povezav sprejema ukaze od centralnega strežnika, ki jih posreduje aplikacijam. Aplikacije potem v pravem trenutku

pošljejo potrebne ukaze sistemom za napajanje magnetov, kar spremeni magnetno polje na priključenem magnetu in s tem vpliva na pot delcev v pospeševalniku.

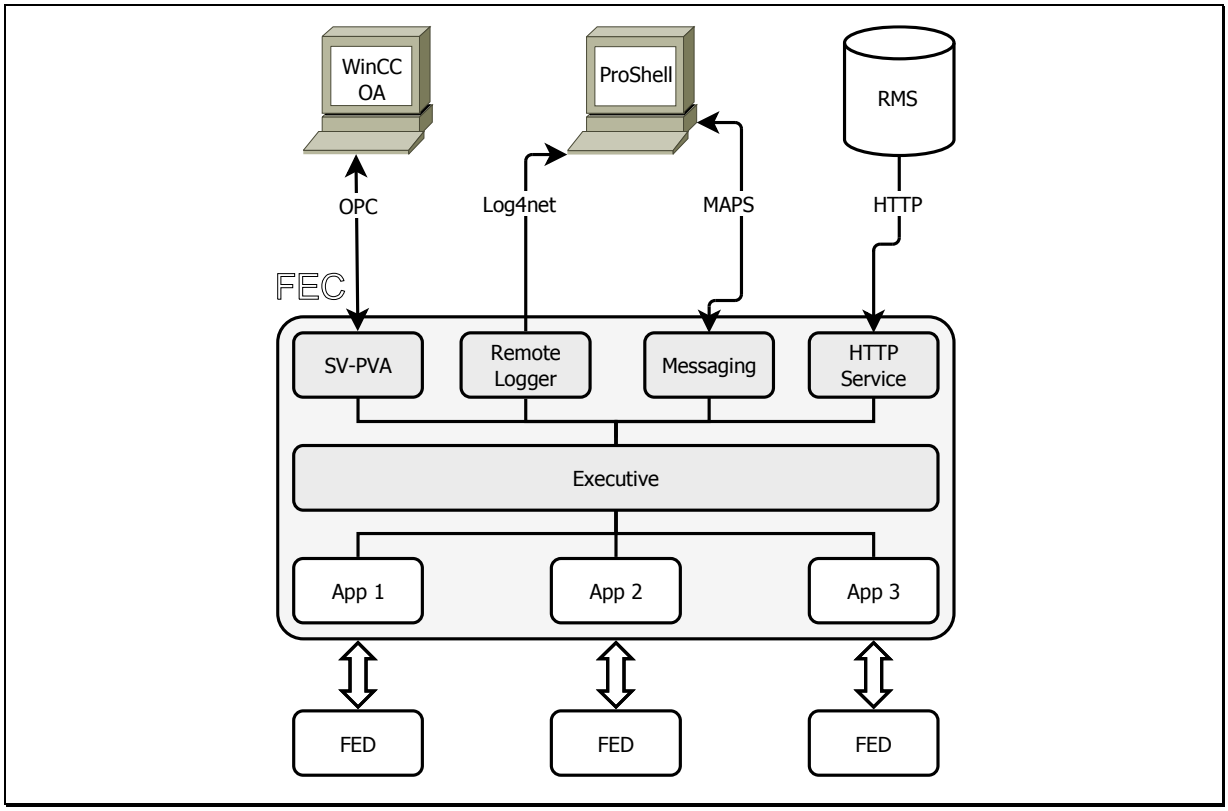
FECOS je operacijski sistem na kontrolnih računalnikih končnih naprav in hkrati ogrodje v katerem so napisane vse aplikacije za upravljanje s končnimi napravami in tudi aplikacija za sprejem časovnih dogodkov.

Kontrolni sistem ne skrbi za to, da delci ostanejo v pospeševalniku, saj pri obhodnem času 0,75 μs to ni mogoče. Za to skrbi samostojen specializiran sistem, kontrolni sistem skrbi le za »visoko nivojsko« delovanje, kot na primer, kdaj začeti s pospeševanjem, do kakšne energije pospešiti delce in podobno. Vendar je tudi za kontrolo na tem nivoju potrebno delovanje v območju nekaj μs.

3.2 Sestavni sklopi kontrolnega sistema

Pospeševalnik nadzoruje strežnik SCADA *WinCC OA* [6], ki pošilja ukaze komponentam sistema. Konfiguracije končnih naprav, strežnikov in vsi ostali podatki, ki jih vsi sistemi pospeševalnika potrebujejo za delovanje, so shranjeni v centralni bazi sistema – *bazi RMS*. Kontrolni računalniki končnih naprav (FEC) do potrebnih podatkov iz baze RMS dostopajo preko protokola HTTP.

Slika 5 prikazuje povezavo kontrolnega računalnika končnih naprav s strežniki kontrolnega sistema pospeševalnika. Preko teh povezav končne naprave dobijo konfiguracijo, ukaze in vse potrebne podatke, ki jih potrebujejo za delovanje. Kontrolni računalniki končnih naprav po istih povezavah pošiljajo tudi vse podatke, ki so potrebni za nadzor in delovanje pospeševalnika. Poleg spodaj prikazanih poti kontrolni računalniki končnih naprav najpomembnejše ukaze za delovanje dobijo po optičnih povezavah časovnega sistema.



Slika 5: Umeščenost sistema FECOS v kontrolni sistem

4 Razvojno okolje LabView

Razvojno okolje LabView [7] je grafično programsko orodje [8], ki se uporablja v raziskovalnih ustanovah in industriji. Omogoča razvoj programov z grafičnimi elementi v stilu diagramov poteka ali vezij. V okolje je poleg splošnih programskih elementov in struktur integrirano veliko število knjižnic in gonilnikov strojne opreme za meritve ter zajem podatkov, podpora za integrirane elemente temelječe na tehnologiji FPGA [9] ter podpora za sisteme v realnem času Phar Lap ETS [10] in VxWorks [11].

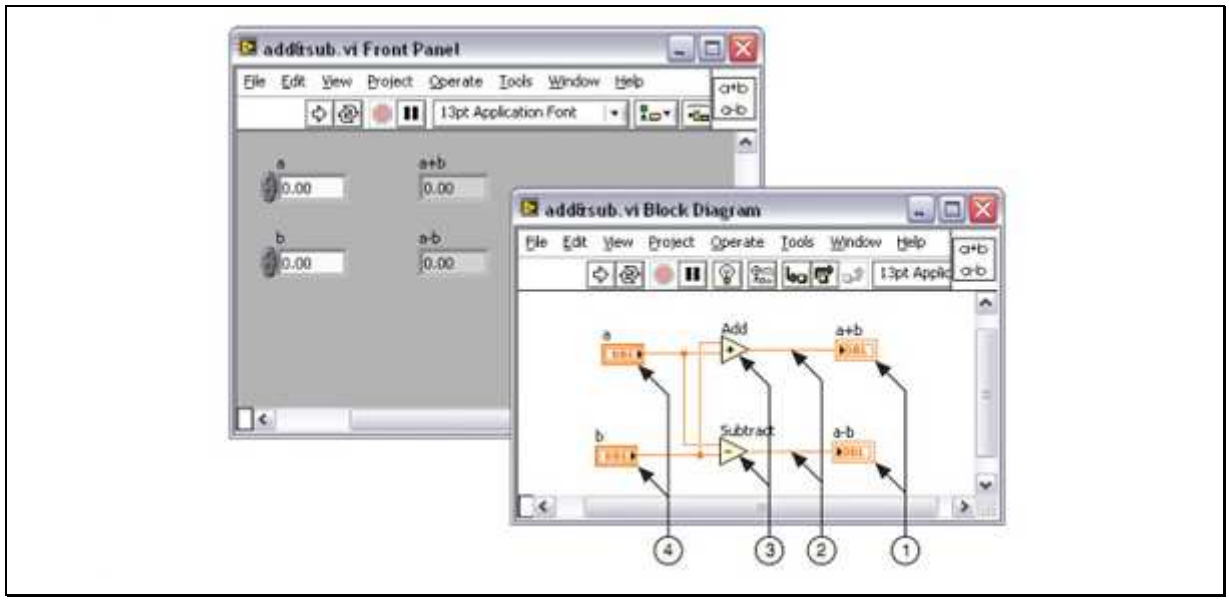
4.1 Sestavljanje programa

Zaporedje izvajanja programa v okolju LabVIEW temelji na toku podatkov, kar pomeni, da se vsaka funkcija v programu izvede v trenutku, ko so za njo na voljo vsi vhodni podatki. Izvor podatkov je lahko uporabniško vnosno polje, grafična kontrola ali strojna oprema.

Po izgledu program še najbolj spominja na shemo vezja ali diagrama poteka. V programu se lahko posamezne podatkovne linije delijo in v večini primerov to pomeni novo kopijo podatkov za vsako vejo. Prav tako velja, da so vzporedne podatkovne linije neodvisne in da se funkcije z neodvisnimi podatkovnimi linijami lahko izvajajo istočasno. Grafični

programski jezik, ki ga implementira okolje LabVIEW se imenuje **G**. Ponor podatkov so lahko funkcije ali pa uporabniški prikazovalniki.

Vsaka LabVIEW aplikacija ali funkcija je shranjena v datoteki VI, ki je sestavljena iz dveh delov. Prvi del je namenjen komunikaciji z uporabnikom (kontrolna plošča), drugi del pa je bločni diagram, ki vsebuje programsko logiko. Primer datoteke VI prikazuje Slika 6 [12].



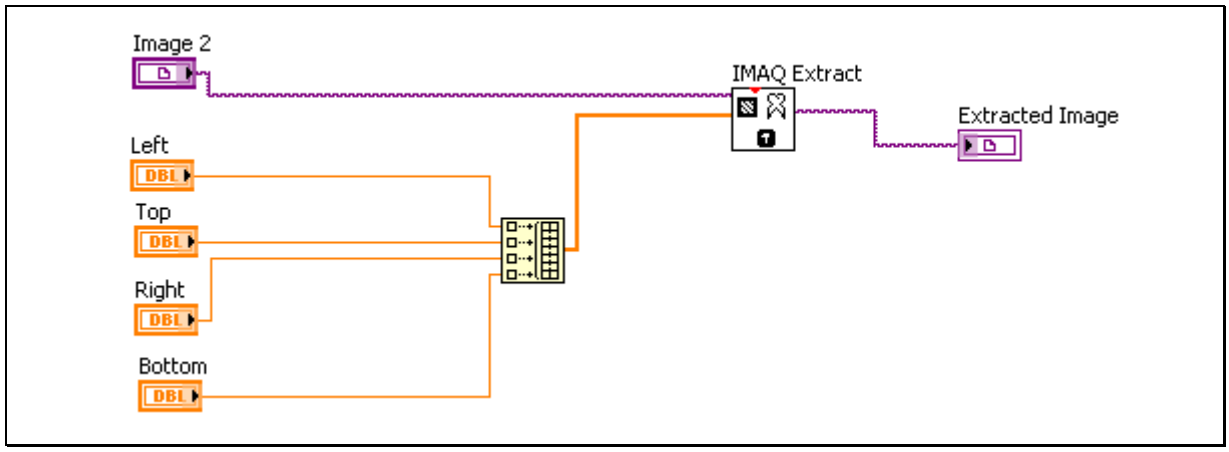
Slika 6: Kontrolna plošča in bločni diagram

Vsaka datoteka VI je lahko samostojna aplikacija ali funkcija, ki se lahko kliče iz drugih VI datotek. Datoteko VI se spremeni v funkcijo tako, da se za njo definira priklopno ploščo, ki določa tip in mesto podatkovnih linij, ki jih lahko priklopimo na funkcijo. Vhodne parametre funkcije določajo uporabniška vnosna polja, izhodne parametre pa uporabniški prikazovalniki.

Vsaka funkcija na bločnem diagramu je predstavljena z ikono datoteke VI v kateri je definirana. Slika 7 [14] prikazuje preprost primer bločnega diagrama, s petimi vnosnimi polji, enim prikazovalnikom in dvema funkcijama. V primeru, da bi ta diagram definiral funkcijo, bi le-ta lahko imela 5 vhodnih parametrov (**Image 2**, **Left**, **Top**, **Right** in **Bottom**), ter en izhodni parameter (**Extracted Image**). Ker jezik G temelji na toku podatkov so v veliko primerih žice določenih podatkovnih tipov speljane »skozi« funkcijo. To velja v primeru, ko funkcija deluje na podatkih tega tipa in jih spreminja, ali pa je to le referenca, ki jo funkcija potrebuje za delovanje. Taka funkcija je ponavadi ena iz družine sorodnih funkcij in se kliče v verigi z drugimi funkcijami. Take so na primer funkcije za delo z vrstami;

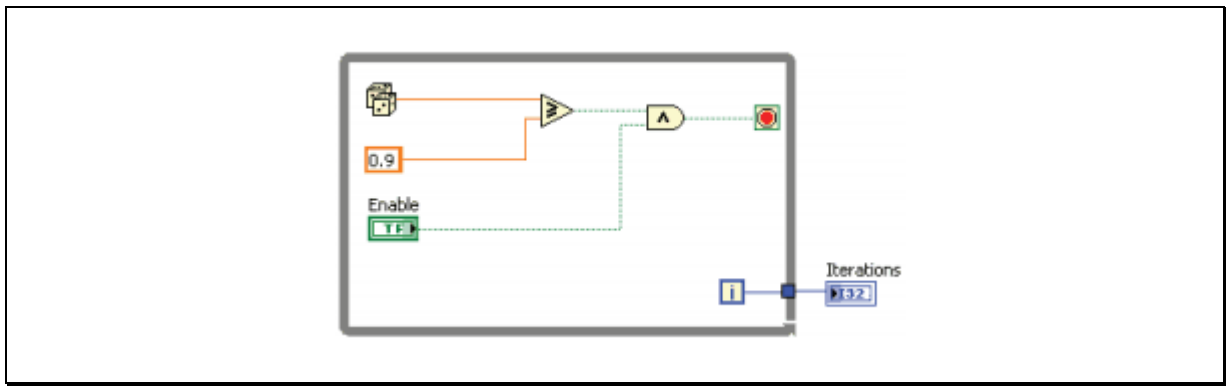
najprej se vrsto ustvari, potem se v njo dodaja in iz nje briše elemente, na koncu pa se vrsto uniči.

Skozi večino funkcij v okolju LabVIEW je speljana žica podatkovnega tipa »napaka« (**error cluster**), zato se uporablja za določanje vrstnega reda izvajanja programa. Če funkcije delujejo na povsem različnih podatkih, jih lahko okolje LabVIEW izvaja v poljubnem vrstnem redu. V trenutku, ko skozi njih napeljemo žico enega tipa, postane vrstni red izvajanja enolično določen. Žica podatkovnega tipa »napaka« ima ponavadi tudi to lastnost, da prisotnost napake na njej prepreči izvajanje funkcije. V primeru, da se je v eni od predhodnih funkcij zgodila napaka se trenutna funkcija ne izvede, saj so vhodni podatki lahko povsem nesmiselni.

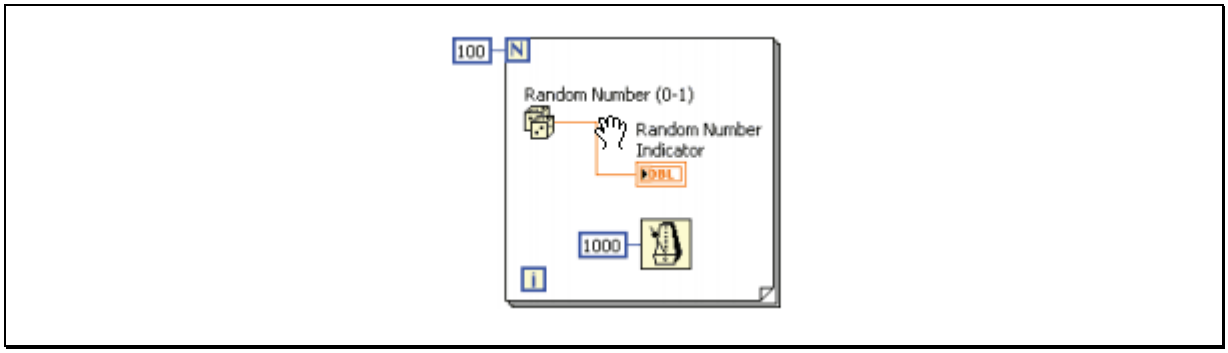


Slika 7: Primer preprostega LabVIEW programa

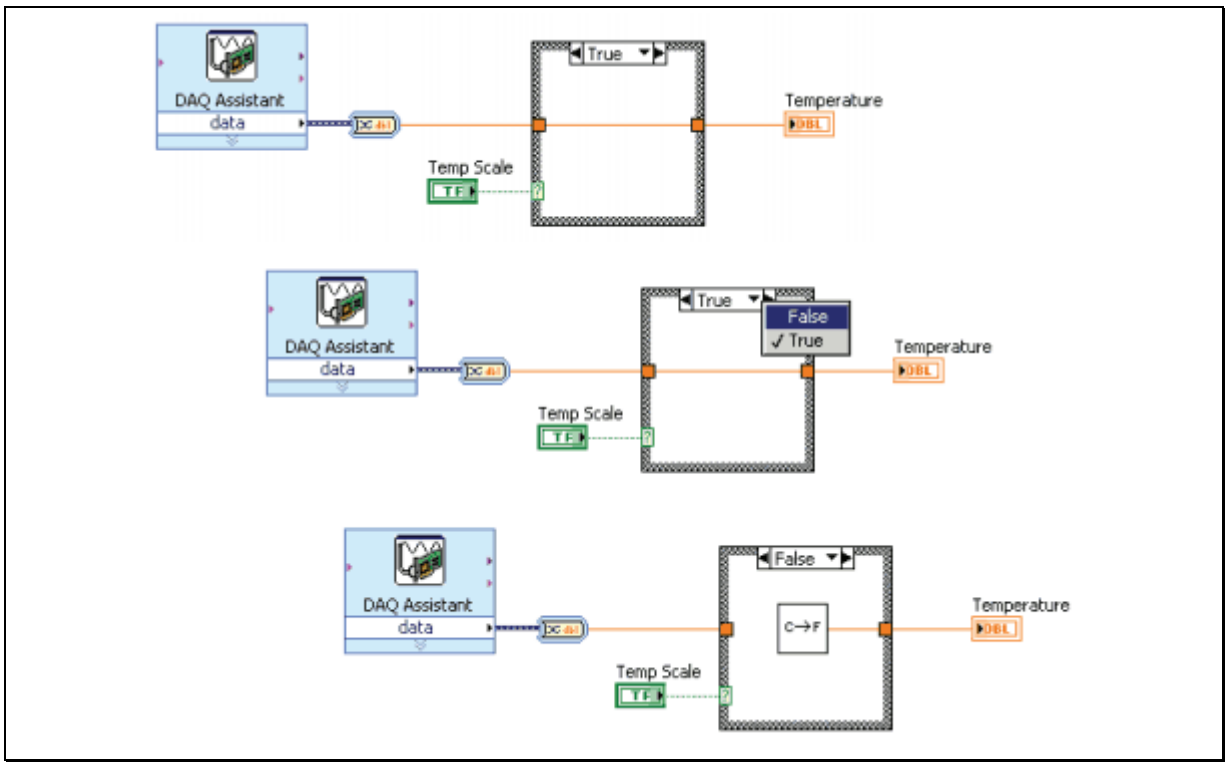
Poleg funkcij ter vhodnih in izhodnih polj programski jezik G vsebuje tudi programske strukture za kontrolo poteka izvajanja programa. To sta na primer zanki **while** (Slika 8 [13]) in **for** (Slika 9 [13]) ter odločitvena struktura (Slika 10 [13]), ki lahko predstavlja stavek **if-then-else** ali **case**.



Slika 8: Zanka "while"



Slika 9: Zanka "for"



Slika 10: Odločitvena struktura »if-then-else«

Jezik G podpira vse standardne enostavne podatkovne tipe, pa tudi sestavljene. Od sestavljenih so to eno in večdimenzionalna polja in strukture. Tipe v okolju LabVIEW se definira kot datoteke posebnega tipa, t.i. kontrole. Kontrola je posebne vrste datoteka VI brez bločnega diagrama, namenjena definicijam novih vnosnih polj ali podatkovnih tipov.

Poseben tip v okolju LabVIEW je tip **Variant**, ki lahko sprejme vrednost kateregakoli drugega tipa. Kasneje lahko to vrednost iz spremenljivke tipa **Variant** preberemo, vendar moramo poznati podatkovni tip shranjen v njej. Poskus branja vrednosti napačnega tipa v večini primerov povzroči napako.

Tip **Variant** ima še eno vlogo, in sicer lahko v spremenljivko takega tipa shranjujemo pare *ključ/vrednost*. Spremenljivka tipa **Variant** deluje torej kot razpršena tabela, pri čemer je lahko ključ le tipa **String**, vrednost pa je lahko poljubnega tipa.

4.2 Objektno orientirano programiranje v okolju LabView

Objektno orientirano programiranje je postalo del okolja LabVIEW z verzijo 8.2 leta 2006. Objektno orientirano programiranje v okolju LabVIEW je povsem enako objektno orientiranemu programiranju v kateremkoli drugem jeziku, vsebuje le nekaj posebnosti razloženih v tem poglavju.

Razred LabVIEW je na nek način knjižnica, ki združuje podatkovne tipe in metode. Razred LabVIEW implicitno vsebuje kontrolo namenjeno lastnostim razreda. V jeziku G so lahko lastnosti razreda le zasebne. Razred pa lahko vsebuje metode in definicije podatkovnih tipov kateregakoli dostopnega razreda, ki so naslednji:

- zasebni – klic metode je mogoč le iz istega razreda;
- zaščiteni – klic metode je mogoč le iz istega razreda ali njegovih potomcev;
- družbeni – klic metode je mogoč le iz prijateljskih razredov ali aplikacij;
- javni – klic metode ni omejen.

Za razliko od ostalih objektno orientiranih jezikov, jezik G ne vsebuje metod tipa *konstruktor* ali *destruktor*.

V ostalih jezikih enostavni konstruktorji inicializirajo vse lastnosti objekta, bolj kompleksni pa jih izračunajo. V jeziku G so prvotne vrednosti lastnosti objekta določene z njihovo deklaracijo, kar pomeni, da preprosti konstruktor ni potreben. Konstruktorjev, ki izračunajo začetne vrednosti lastnosti objekta, jezik G ne pozna zato, ker nimajo smisla v jeziku zasnovanem na toku podatkov [15].

Destruktorjev jezik G ne pozna iz enakega razloga kot niso definirani v Javi. Življenjska doba objekta je določena z njegovo uporabo in objekt se samodejno uniči, ko se ga ne potrebuje več.

Metode večine objektnih jezikov imajo poleg definiranih parametrov tudi implicitno podan skriti parameter, ki je kazalec na objekt na katerem metoda deluje (v Javi in C++ je to kazalec *this*). Metode jezika G imajo referenco do objekta na katerem delujejo podano eksplicitno, in sicer je žica reference speljana »skozi« metodo (glej poglavje 4.1).

V okolju LabVIEW je mogoče narediti dva tipa razrednih metod, *dinamične* in *statične*. Gre za drugo ime metod, ki so v objektno orientiranem programiranju znane kot *virtualne* (dinamične) in *ne-virtualne* (statične). V okolju LabVIEW so drugačno ime izbrali iz preprostega razloga, ker VI pomeni *virtual instrument* in niso želeli uporabiti enakega izraza še za drug pomen. V okolju LabVIEW statičnih metod pri dedovanju sploh ni mogoče redefinirati.

Potrebno je omeniti tudi, da se v okolju LabVIEW imena razrednih metod zapisuje skupaj z razredom in dvopičjem, ki obe imeni ločuje; npr. `Razred:Metoda`. Ker gre pri vseh elementih (knjižnicah, razredih in metodah) za datoteke na disku, so presledki lahko del imena kateregakoli od elementov.

4.3 Posebnosti okolja LabVIEW

Okolje LabVIEW ima več načrtovalskih odločitev, ki vplivajo na pisanje programa. Najpomembnejše od njih in rešitve, ki jih negirajo, so opisane spodaj.

Ena od najpomembnejših omejitev programov v jeziku G je, da vsaka žica predstavlja svojo vrednost podatka. To pomeni, da žica, ki se razdeli na tri dele, do treh funkcij pripelje tri ločene kopije iste vrednosti. Pri preprostih podatkovnih tipih, kot so števila, to ni problem. Pri objektih, ki nosijo neko interno informacijo o stanju je seveda drugače. V sistemu FECOS se lahko en objekt uporablja na več mestih, celo v več nitih. In če klicane metode spreminjajo notranja stanja več kopij enega objekta, sistem ne more delovati.

Na srečo se je mogoče zgoraj opisani odločitvi izogniti s t.i. podatkovnimi referencami, ki delujejo kot neke vrste kazalec na podatek, pri čemer je lokacija podatka v pomnilniku nespremenljiva. Torej se preko vseh kopij te reference dostopa do istih podatkov. V sistemu FECOS opisano značilnost izkoristimo tako, da so lastnosti objekta zapisane v posebni strukturi, sam objekt pa vsebuje le referenco na strukturo. Na ta način vse kopije objekta delujejo na istih podatkih.

Ker je potrebno podatkovno referenco ustvariti s klicem posebne funkcije in jo kasneje zbrisati, je bilo potrebno v sistemu FECOS eksplicitno implementirati metodi tipa *konstruktor* in *destruktor*, kar sicer ni del objektno orientiranega programiranja v jeziku G. Ti dve metodi sta zaščiteni dinamični metodi `Class Initialize` in `Class Cleanup`, ki sta v sistemu FECOS klicani implicitno. Objekte iz hierarhije sistema FECOS se sicer ustvari z eksplicitnim klicem metode `Create`.

Datoteke VI so privzeto implementirane tako, da se jih ne da klicati rekurzivno (*non-reentrant*). V praksi to pomeni, da se v času izvajanja funkcije le-te ne da klicati ponovno. Posebnost takšne implementacije v okolju LabVIEW je, da funkcija med klici ohrani zadnje vrednosti vhodnih in izhodnih parametrov oziroma vseh vnosnih polj in prikazovalnikov na kontrolni plošči. Ob naslednjem klicu funkcije je te vrednosti mogoče prebrati in prilagoditi delovanje. To pomeni, da si funkcija lahko shrani notranje stanje, ki se ohrani med dvema klicema. Takšen način pisanja funkcij se v okolju LabVIEW uporablja kar pogosto.

Pri objektno orientiranem programiranju je opisana lastnost okolja LabVIEW popolnoma neuporabna, saj dedovanje prinese kot posledico dejstvo, da se nekatere funkcije kličejo neprestano v kontekstu različnih objektov. Še posebej je to neprimerno v primeru, če objekti »živijo« v različnih procesih oziroma nitih, kar v sistemu FECOS velja za vse objekte. Istočasni klic take funkcije iz dveh niti pomeni, da mora ena od njiju počakati dokler se prva ne izvede do konca. Zato so v sistemu FECOS vse metode in funkcije definirane kot *reentrant*.

5 Opis kontrolnega sistema končnih naprav

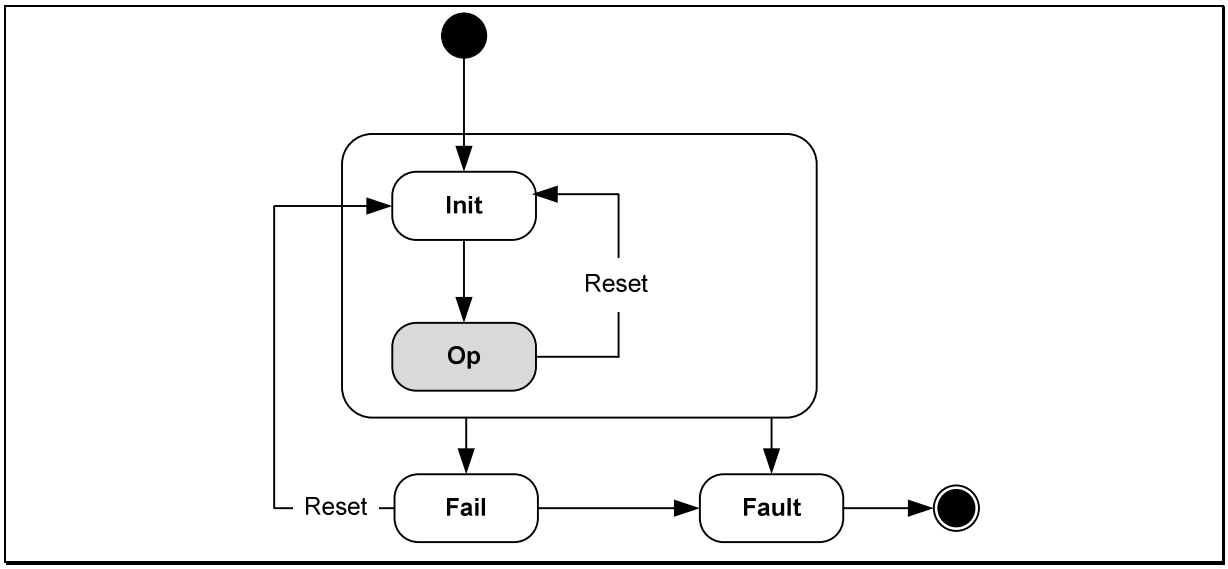
Poglavje podaja pregleden opis sestavnih delov sistema FECOS, kako se povezujejo in na kratko opisuje njihov namen.

5.1 Zahteve naročnika glede delovanja sistemskih komponent in aplikacij

Na zahtevo naročnika smo razvili večopravilen sistem za nadzor končnih naprav, ki se povezuje s strežnikom SCADA, centralno bazo in nadzornim sistemom. Sistem teče na kontrolnem računalniku končnih naprav. V našem primeru gre za industrijske računalnike *National Instruments PXI*. Uporabljeni strežnik SCADA je *Siemens SIMATIC WinCC Open Architecture (WinCC OA)* [6], ki s kontrolnim računalnikom končnih naprav komunicira preko protokola OPC.

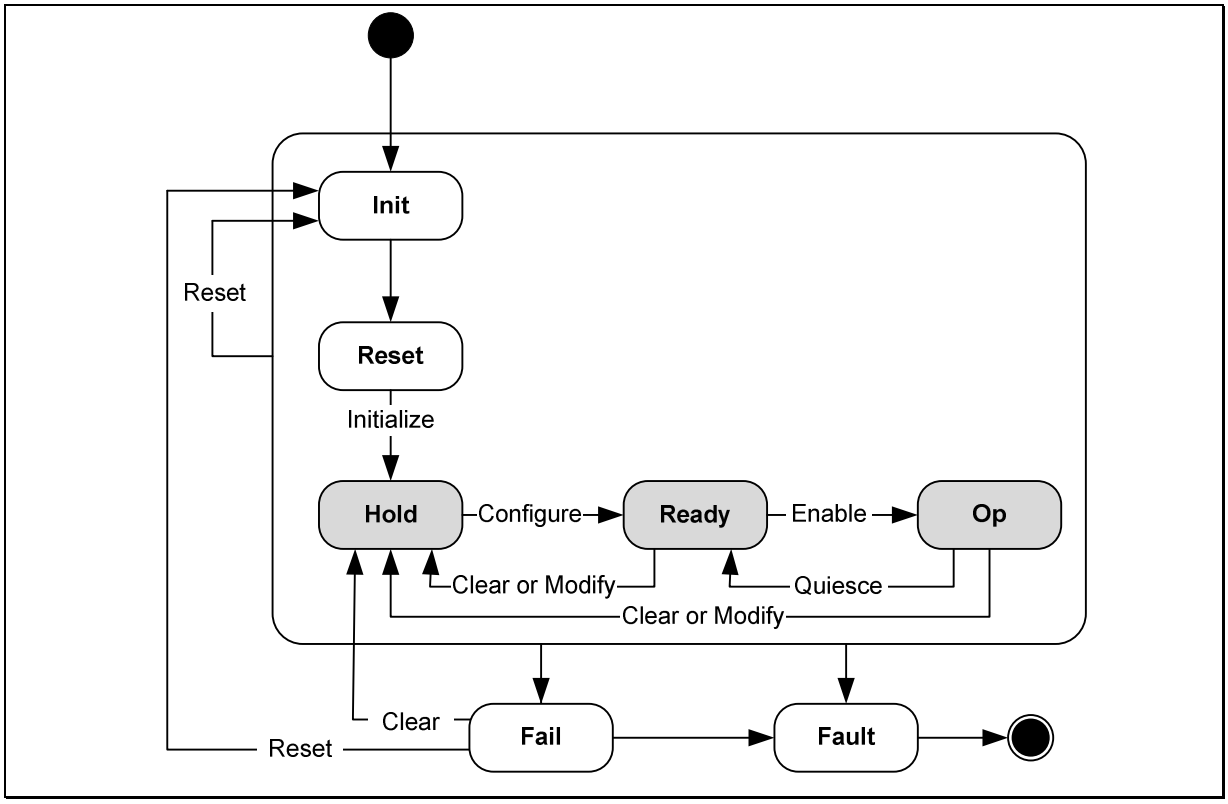
Vse aplikacije, ki tečejo na kontrolnem računalniku končnih naprav, morajo interno delovati kot določene vrste končni avtomat, kar pomeni, da imajo implementirano množico logičnih stanj in je aplikacija lahko v določenem trenutku v enem od njih. Stanja definirajo kaj aplikacija takrat lahko počne, določeno pa je tudi kako se lahko med stanji prehaja in pod kakšnimi pogoji. Na ta način vse aplikacije delujejo na vnaprej definiran način.

Sistem pozna dva tipa aplikacij. Prvi implementira preprosto napravo, torej končni avtomat, ki ga prikazuje Slika 11, drugi pa implementira kompleksen končni avtomat, ki ga prikazuje Slika 12.



Slika 11: Preprosta naprava

Preprosta naprava je namenjena enostavnim aplikacijam, ki nimajo zapletene funkcionalnosti; na primer, aplikacija bere neko vrednost končne naprave in jo posreduje centralnemu strežniku. Ima 4 stanja in med njimi prehaja samodejno glede na situacijo; na primer, če aplikacija zgubi stik s končno napravo, se lahko pomakne v stanje *Fail* ali *Fault*.



Slika 12: Kompleksen končni avtomat

Kompleksen končni avtomat uporabljajo vse zahtevnejše aplikacije v sistemu FECOS. Med stanji avtomata se prehaja z ukazi, ki v vsakem stanju definirajo prehod v neko drugo stanje. Edina izjema je stanje *Init*, iz katerega se končni avtomat samodejno premakne v stanje *Reset*. Ukaze za premik med stanji aplikacija prejme od centralnega strežnika. Kratek opis stanj končnega avtomata in ukazov prikazujeta Tabela 1 in Tabela 2.

Stanje	**Definicija stanja**
Fail	Prišlo je do napake v delovanju aplikacije. Potreben je ukaz *Reset* ali *Clear*.
Fault	Prišlo je do napake v delovanju naprave zaradi česar aplikacija ne more nadaljevati.
Hold	Stanje sistema ni veljavno in sprejema le sistemske ukaze.
Init	Začasno stanje, aplikacija ne sprejema nobenih ukazov.
Op (*Operational*)	Aplikacija opravlja svoje naloge in sprejema sistemske in uporabniške ukaze.

Ready	Aplikacija je konfigurirana in sprejema le sistemske ukaze.
Reset	Aplikacije je inicializirana, vendar ni v veljavnem stanju in pričakuje sistemske ukaze.

Tabela 1: Opis stanj kompleksnega končnega avtomata

Ukaz	**Opis ukaza**
Initialize	Prehod iz stanja *reset* v stanje *hold*.
Configure	Prehod iz stanja *hold* v stanje *ready*.
Enable	Prehod iz stanja *ready* v stanje *op*.
Quiesce	Prehod iz stanja *op* v stanje *ready*.
Clear	Ukaz pove aplikaciji naj zbriše vse trenutno veljavne konfiguracijske parametre in se premakne v stanje *hold*. Ukaz *clear* lahko aplikacija sprejme v stanjih *ready*, *op* in *fail*.
Modify	Ukaz je podoben ukazu *clear*. Aplikaciji pove naj zbriše vse veljavne konfiguracijske parametre. Istočasno pa aplikacija prejme nove parametre skupaj z ukazom. Končni avtomat se premakne v stanje *hold*. Ukaz *modify* lahko aplikacija sprejme v stanjih *ready* in *op*.
Reset	Ukaz prestavi končni avtomat v stanje *init*.

Tabela 2: Opis ukazov kompleksnega končnega avtomata

Ukaze za prehod med stanji končnega avtomata komponente sprejemajo od centralnega strežnika *WinCC OA*, preko sistemske komponente *SV-PVA*.

Sistem FECOS za vsa stanja končnih avtomatov opisanih v tem poglavju ponuja prototipne metode, ki se avtomatsko kličejo v primernem trenutku. Razvijalec pri pisanju aplikacije preprosto implementira potrebne metode. V večini primerov je to metoda *Op*.

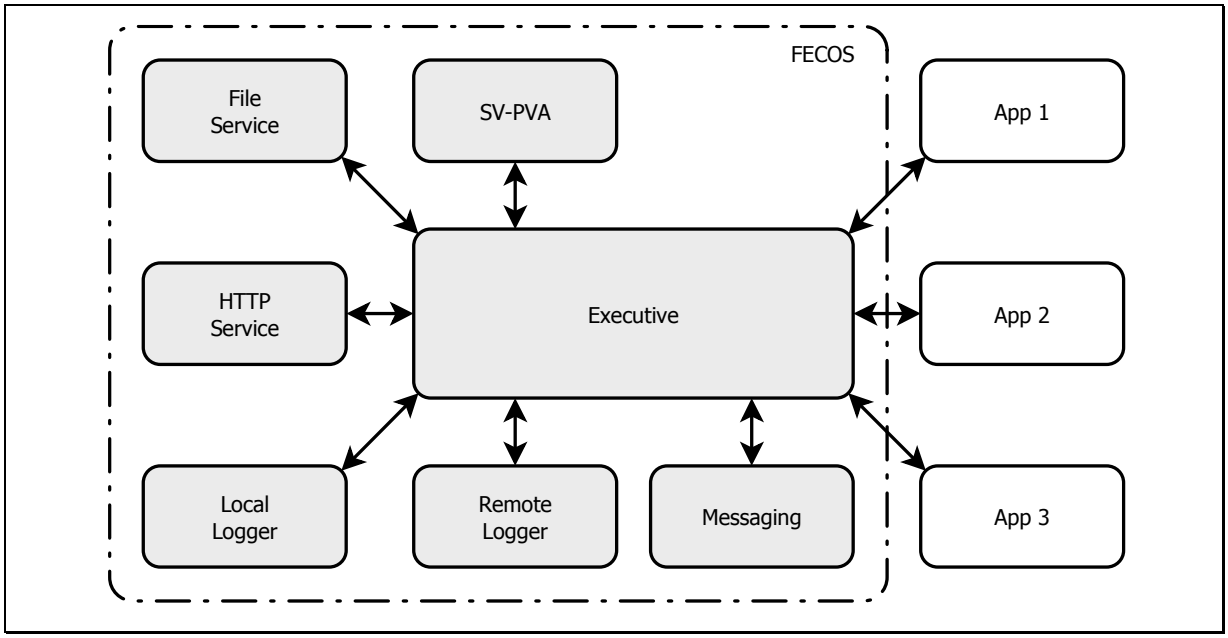
Razvijalec ima prav tako na voljo metode, ki se kličejo avtomatsko, ko končni avtomat zapusti kakšno stanje ali preden v kakšno stanje vstopi. Ob tem je kot parameter podan ukaz s katerim je prišlo do zamenjave stanja. V metodah, ki se kličejo ob zapustitvi stanja, lahko razvijalec zavrne menjavo stanja končnega avtomata s klicem metode `Reject State Transition`. Klic te metode v kateremkoli drugem trenutku se ignorira.

Drugi dve mesti kjer razvijalec navadno implementira želeno funkcionalnost sta metodi `User Consume Event` in `Consume Event`. Metodo `Consume Event` sistem FECOS avtomatsko kliče za vse dogodke, ki jih komponenta prejme, medtem ko se metoda `User Consume Event` avtomatsko kliče le za neznane dogodke (take, ki jih je definiral uporabnik) in le v stanju *Op*, ki je edino stanje v katerem komponenta opravlja svojo nalogo.

5.2 Shematski prikaz sistema na kontrolnem računalniku končne naprave

Osnovna razreda sistema FECOS sta `Component` in `Basic Device`. Razred `Component` implementira kompleksen končni avtomat, razred `Basic Device` pa implementira preprosto napravo. Sistem FECOS ima nekaj sistemskih razredov, ki temeljijo na razredu `Component` in to so:

- `Executive`;
- `SV-PVA`;
- `File Service`;
- `HTTP Service`;
- `Local Logger`;
- `Remote Logger`;
- `Messaging`.



Slika 13: Shema sistema FECOS

Sistemske komponente nudijo sistemske usluge ostalim aplikacijam sistema FECOS. Lahko so lokalne narave (pisanje dnevnika, branje ali zapisovanje datotek na disk) ali pa skrbijo za povezavo s strežniki kontrolnega sistema preko mreže.

Za splošno ime vseh sistemskih in uporabniških aplikacij bomo v nadaljnjem besedilu uporabljali izraz *komponenta*. Natančnejši opis sistemskih komponent se nahaja v poglavju 5.4.

5.3 Komunikacija med procesi

Komponente v sistemu FECOS lahko med seboj komunicirajo s pošiljanjem asinhronih sporočil, ki jih imenujemo *dogodki*.

5.3.1 Dogodki

Dogodek je objekt razreda `Event`, ki nosi določeno sporočilo. Tip sporočila definira *ime* dogodka, dogodek sam pa lahko nosi še dodatne informacije, kot so vrednost in *lastnosti*. Lastnosti dogodka so določene z imenom, ki je vedno tipa **String** in vrednostjo, ki je lahko kakršnegakoli tipa.

Dogodek je načeloma enosmerno sporočilo, vendar lahko komponenta na poslano sporočilo pričakuje tudi odgovor. Tip odgovora in njegova vsebina je odvisna od komponente, ki na dogodek odgovarja.

Sistem FECOS pozna nekaj sistemskih dogodkov, ki se od uporabniških razlikujejo le po imenu. Sistemske dogodke komponenta obdela samodejno in v večini primerov se aplikacija ne zaveda, da je prejela tak dogodek. Če je potrebno, je mogoče prestrezati tudi sistemske dogodke.

Polja razreda `Event` so

Polje	**Tip**	**Opis lastnosti**
Name	**String**	Ime oziroma tip dogodka.
Event ID	**Int32**	Identifikacijska številka dogodka, ki povezuje dogodke kjer je to potrebno (zahteva/odgovor).
Source	Address	Identifikacijsko ime komponente, ki dogodek pošilja.
Destination	Address	Identifikacijsko ime komponente ali ime storitve, ki je prejemnik dogodka.

Priority	**UInt16**	Prioriteta dogodka.
Status	Status	Status dogodka, ki je lahko *Completed*, *Waiting*, *Executing*, *Blocked* in *Error*.
User	**String**	Uporabnik, ki je dogodek poslal (lahko prazno).
Value	**Variant**	Poljuben podatek. Tako pošiljatelj kot prejemnik dogodka morata implicitno vedeti kakšni podatki se pošiljajo.
Time Stamp	**Timestamp**	Čas pošiljanja dogodka ali druge vrste časovni zaznamek. Na primer, čas zadnje spremembe vrednosti omrežne spremenljivke.
Error Code	**Int32**	Numerična vrednost v primeru napake (koda napake). Vrednost 0 pomeni brez napake.
Error data	**Array of String**	Podatki o kontekstu napake: • UUID; • Time Stamp; • Message; • Source.
Property	Key/Value	Vsak dogodek lahko nosi neomejeno število s strani uporabnika definiranih lastnosti, ki so pari ključ/vrednost, pri čemer je ključ vedno tipa **String**, vrednost pa je lahko kakršnegakoli tipa.

Tabela 3: Polja razreda `Event`

Kot že omenjeno, je dogodek načeloma enosmerno sporočilo, vendar v določenih primerih komponenta na določen dogodek pričakuje odgovor. V tem primeru se za povezavo zahteve z odgovorom uporabi polje **Event ID**. Komponenta v zahtevi napolni polje **Event ID** s poljubnim pozitivnim celim številom in dogodek z odgovorom dobi enako vrednost polja **Event ID**. To omogoča, da komponenta poveže odgovor z zahtevo, saj načeloma lahko pošlje več zahtev istega tipa in dobi odgovore na njih v poljubnem vrstnem redu.

Kot primer uporabe dogodka naj navedemo dogodek GET, ki ga komponenta lahko pošlje sistemski komponenti `File Service` in pomeni zahtevo po branju podatkov z diska.

Komponenta kot odgovor dobi prav tako dogodek GET, ki ima enako vrednost polja **Event ID**, polje **Value** pa vsebuje vsebino prebrane datoteke.

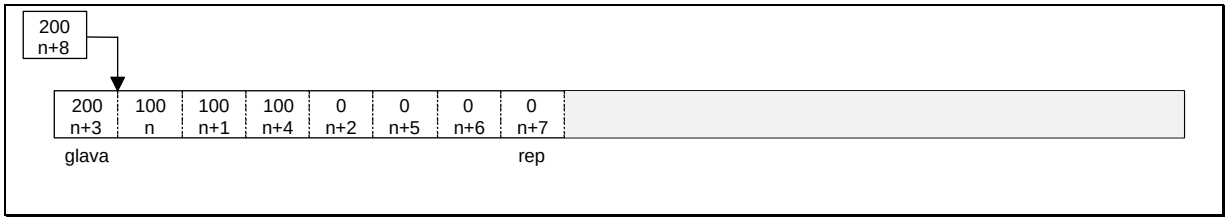
5.3.2 Prioritetna vrsta

Za komunikacijo z dogodki ima vsaka komponenta definirani dve vrsti, in sicer za dogodke, ki jih komponenta prejema in pošilja.

Okolje LabVIEW načeloma vsebuje sistemsko implementacijo vrste, vendar gre za preproste vrste FIFO, sistem FECOS pa potrebuje vrste, ki podpirajo prioriteto. V praksi to pomeni, da morajo biti dogodki z višjo prioriteto v vrsto vrinjeni pred dogodke z nižjo prioriteto, dogodki z enako prioriteto pa morajo biti v vrsto vstavljeni v vrstnem redu FIFO. Dogodki se iz vrste *vedno* jemljejo pri glavi vrste.

V sistemu FECOS je prioriteta dogodka celo število, pri čemer večje število pomeni višjo prioriteto dogodka. Vrsta je implementirana kot polje, v katerega se na pravo mesto vstavi nov dogodek. Najprej se preveri ali je dogodek mogoče dodati na konec vrste. Če ima višjo prioriteto se preveri ali ga je mogoče vstaviti na začetek vrste. V primeru, da glede na prioriteto to ni mogoče, se vrsta preišče linearno in dogodek vstavi na primerno mesto. Slika 14 prikazuje vstavljanje osmega dogodka po vrsti s prioriteto 200. Dogodek bo vrinjen na drugo mesto, torej na konec dogodkov s prioriteto 200 in pred dogodke s prioriteto 100.

Glede na predvideno število dogodkov je takšna implementacija prioritetne vrste dovolj hitra za potrebe sistema.



Slika 14: Vstavljanje dogodka v prioritetno vrsto

5.4 Opis posameznih komponent (razredov) sistema

Sistem FECOS sestavlja množica sistemskih razredov, ki temeljijo na osnovnem razredu sistema FECOS, torej na razredu `Component`. Poglavje opisuje njihovo funkcijo in storitve, ki jih nudijo.

5.4.1 Prikaz hierarhije razredov

Hierarhijo sistemskih razredov sistema FECOS prikazuje Slika 15.

Razred `State Machine` implementira kompleksen končni avtomat, kot ga prikazuje Slika 12. Je tako imenovani *aktivni objekt*, kar pomeni, da se ob kreaciji objekta ustvari nov proces določene prioritete. Razred definira prototipne funkcije za posamezna stanja in prehode med stanji glede na ukaze. Je osnovni razred sistema FECOS in ni predviden, da se ga uporablja kot prednika uporabniško definiranim razredom.

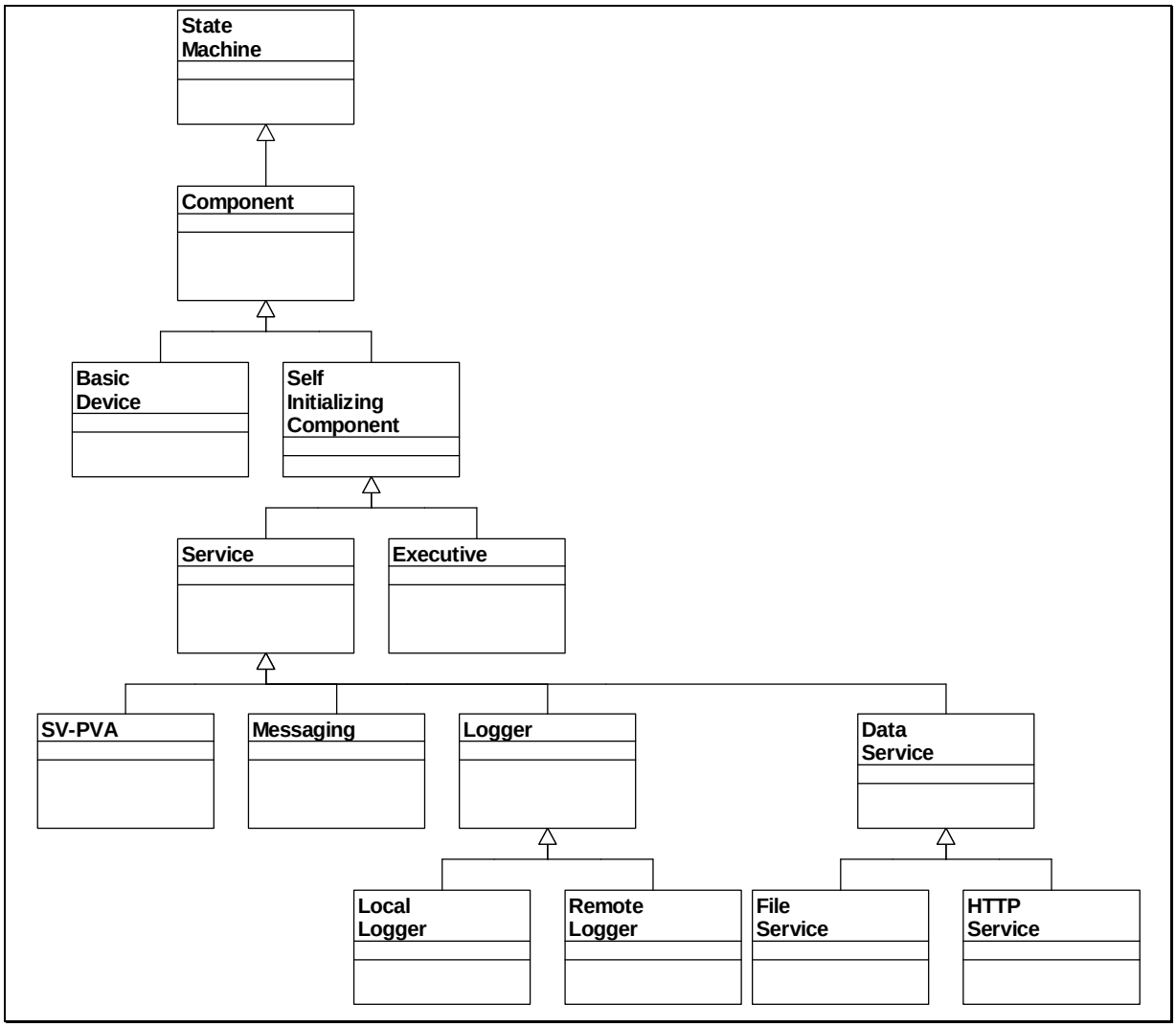
Razred `Component` je osnovni *uporabni* razred sistema FECOS. Je prvi razred sposoben pošiljati in sprejemati dogodke. Objekti razreda imajo že svoj *identifikator* (enolično ime v celotnem kontrolnem sistemu pospeševalnika), konfiguracijo, podpirajo uporabo funkcij v realnem času in vsebujejo množico storitvenih funkcij, ki jih potrebujejo uporabniške aplikacije v sistemu. Večina uporabniških aplikacij temelji na razredih, ki neposredno dedujejo iz razreda `Component`.

Logično gledano je najbolj preprost razred `Basic Device`. Lahko bi bil osnovni razred sistema FECOS, vendar je bil v razvoju definiran precej pozno, tako da dejanska implementacija izhaja iz razreda `Component`, ki smo mu odvzeli večino stanj.

Ker vse komponente opravljajo svojo primarno funkcijo v stanju `Op` in ker sistem ne more delovati, ne da bi sistemske komponente opravljale svojo primarno funkcijo, je bil definiran razred `Self Initializing Component`. Objekti tega razreda si sami pošljejo tri ukaze, ki so potrebni, da končni avtomat pride v stanje `Op`.

Za vse sistemske komponente (razen komponento `Executive`) je osnovni razred `Service`. Definira prototipa dveh metod, ki ju `Executive` kliče vsakič, ko se v sistem FECOS registrira nova aplikacija. To sta metodi `Registration Notification` in `Unregistration Notification`. Objekti razreda `Service` imajo lahko poleg *identifikatorja* tudi poljubno število storitvenih imen. To so splošna imena, ki se lahko uporabijo kot naslovnik dogodka namesto *identifikatorja* in so lokalna na posameznem kontrolnem računalniku končne naprave. To omogoča, da lahko posamezne aplikacije

uporabljajo sistemske storitve, ne da bi poznale dejanski *identifikator* sistemske komponente na kontrolnem računalniku končne naprave.



Slika 15: Hierarhija sistemskih razredov sistema FECOS

5.4.2 Executive

Glavna komponenta sistema je `Executive`. Komponenta se ob zagonu sistema naloži prva in je poglavitna za njegovo delovanje. Skrbi za nalaganje vseh preostalih komponent sistema in uporabniških aplikacij, z izjemo štirih osnovnih komponent, ki se naložijo med postopkom zagona naprave (poglavje 5.6).

Komponenta `Executive` skrbi za nalaganje vseh sistemskih in uporabniških komponent, komunikacijo med njimi in nadzoruje njihovo pravilno delovanje. Posledično mora imeti komponenta najvišjo prioriteto v sistemu, da morebitna napaka v razvoju aplikacije ne ogrozi osnovnega delovanja celotnega sistema.

Po osnovnem zagonu sistema komponenta `Executive` naloži konfiguracijo kontrolnega računalnika končnih naprav in z izjemo osnovnih štirih komponent začne nalagati komponente in aplikacije v vrstnem redu, ki je določen v konfiguraciji (poglavje 5.4.2.2). Komponenta `Executive` programsko kodo vsake aplikacije pričakuje na točno določenem mestu na disku naprave, od koder jo naloži in zažene s prioriteto določeno v konfiguraciji.

Istočasno se zgodi tudi registracija aplikacije, kar je potrebno za usmerjanje sporočil. V primeru, da naložena komponenta izhaja iz razreda `Service`, se registrira tudi z vsemi storitvenimi imeni, ki so za to komponento definirana. To pomeni, da komponenta prejema tudi vsa sporočila naslovljena na katero od storitvenih imen. V primeru, da ima več komponent enako storitveno ime, uspe le registracija storitvenega imena prve komponente, ostale poskuse registracije pa se ignorira.

Ob registraciji katerekoli komponente se za vse registrirane storitvene komponente kliče tudi metoda `Registration notification`, za primer če storitvena komponenta vsem ostalim komponentam sistema nudi svoje storitve avtomatsko.

Komponenta `Executive` skrbi za komunikacijo med registriranimi komponentami, kar pomeni, da periodično preiskuje izhodne vrste vseh registriranih komponent in vse najdene dogodke prestavi v vhodne vrste ciljnih komponent. To počne v t.i. *round-robin* načinu.

Komponenta `Executive` nadzoruje tudi pravilno delovanje vseh registriranih komponent. Podsistem za nadzor delovanja imenujemo *pes čuvaj*.

5.4.2.1 Podsistem za nadzor delovanj komponent

Pes čuvaj ne more zaznati vseh napak aplikacij. Omejen je na zaznavanje slabe odzivnosti posamezne aplikacije. Čeprav skrito pred razvijalcem, je proces obdelovanja prejetih dogodkov del glavnega toka izvajanja aplikacije. V primeru, da je aplikacija napisana slabo ali je njeno delovanje popolnoma blokirano (na primer z neskončno zanko), le-ta počasi obdeluje prejete dogodke ali pa s tem popolnoma preneha.

Pes čuvaj vsako minuto vsem registriranim aplikacijam pošlje dogodek visoke prioritete (*PING*), na katerega pričakuje odgovor v določenem času. Zaradi visoke prioritete je tak dogodek obdelan pred večino ostalih. Aplikacija nanj odgovori samodejno, in sicer z dogodkom *PONG*. Če komponenta `Executive` odgovor prejme pravočasno dobi oznaka pravilnega delovanja komponente vrednost 0.

V primeru, da je odgovor prispel do 2x počasneje kot v prvem primeru, se oznaka pravilnega delovanja komponente poveča za 2.

V primeru, da odgovor pride še počasneje ali pa ga sploh ni, se oznaka pravilnega delovanja komponente poveča za 5.

Za aplikacije, ki imajo oznako pravilnega delovanja med vrednostma 1 in 10 komponenta `Executive` v dnevnik sporoči, da se aplikacija odziva slabo. Če pa oznaka pravilnega delovanja preseže vrednost 10, se aplikacija razglasi za *mrtvo*.

5.4.2.2 Konfiguracija kontrolnega računalnika končnih naprav

Kontrolni računalnik končnih naprav ima poleg sistema FECOS instalirano tudi kodo vseh uporabniških aplikacij, ki se pojavljajo v kontrolnem sistemu računalnika. Delovanje kontrolnega računalnika končnih naprav določa konfiguracija, ki na posameznem računalniku definira vrsto in vrstni red naloženih aplikacij in podaja njihove nastavitve. Konfiguracija je zapisana v formatu XML in vsebuje seznam vseh sistemskih in uporabniških aplikacij, ki morajo teči na kontrolnem računalniku. Poleg seznama vsebuje sistemske in uporabniške nastavitve za vsako aplikacijo.

Sistemske nastavitve so enake za vse aplikacije v sistemu. Komponente jih ob zagonu upoštevajo avtomatsko, uporabniške nastavitve pa so prilagojene za vsako aplikacijo posebej in jih mora aplikacija predelati eksplicitno. Kot prikazuje Slika 16 lahko razdelek `<componentConfig>` vsebuje poljuben XML tekst, naloga razvijalca aplikacije pa je, da napiše kodo, ki se pomika po drevesu DOM in se odziva na naštete nastavitve. Načeloma ima lahko vsaka aplikacija konfiguracijo sestavljeno iz povsem svojega nabora XML elementov, vendar je večina potrebnih elementov XML za nastavitve standardizirana.

```
<?xml version="1.0"?>

<fecos version="1.0" id="FEC_ID" ip="10.5.2.111">
  <component id="CMP_ID_1" class="ComponentClass">
    <frameworkConfig>
      <priority>200</priority>
      <logLanguage>en_US</logLanguage>
      <localLogLevel>info</localLogLevel>
      <remoteLogLevel>error</remoteLogLevel>
      <queueSize>48</queueSize>
    </frameworkConfig>
    <componentConfig>
      <!-- Any XML -->
    </componentConfig>
  </component>
```

```
  ...
</fecos>
```

Slika 16: Primer konfiguracijske datoteke

Komponenta Executive ob zagonu prebere konfiguracijsko datoteko in jo s pomočjo knjižnice za delo z datotekami XML (poglavje 5.5.2) predela v drevo DOM. Nato komponenta Executive potuje po elementih <component> in za vsak element naloži objekt razreda, ki ga določa atribut XML class. Ob kreiranju objekta se uporabi tudi atribut XML id, vsaka komponenta pa prejme še kazalec na svoj del konfiguracije, ki ga obdela kot je opisano zgoraj.

V trenutku, ko komponenta Executive prebere konfiguracijo, so štiri osnovne komponente sistema FECOS že naložene. To so torej edine štiri komponente sistema, ki svoje konfiguracije ne prejmejo ob zagonu, ampak se prilagodijo konfiguraciji kasneje. Vendar pa za razliko od ostalih komponent njihova konfiguracija vsebuje samo sistemske nastavitve v elementu <frameworkConfig>.

5.4.3 SV-PVA

SV-PVA, kar pomeni *shared variable – public variable access*, je razred sistemske komponente za povezavo z glavnim nadzornim strežnikom *WinCC OA*. Storitveno ime komponente je *pva* in nudi storitve implicitne in eksplicitne povezave z nadzornim strežnikom SCADA. Ker to povezavo potrebujejo vse komponente sistema, je to tudi druga od štirih osnovnih komponent sistema in se naloži takoj za komponento Executive.

Je tudi ena od redkih komponent, ki implementira metodi Registration Notification in Unregistration Notification. Komponenta SV-PVA za vsako novo komponento, ki se registrira v sistem FECOS, odpre povezavo na kontrolne omrežne spremenljivke definirane za posamezno komponento. Preko njih komponenta sprejema ukaze za menjavo stanja končnega avtomata, pridobiva podatke za avtorizacijo in avtentikacijo uporabnikov, sporoča v katerem stanju je in ali deluje normalno, ter sporoča napake, ki se lahko zgodijo pri izvajanju sistemskih ukazov.

Komponenta SV-PVA nadzoruje sistemske mrežne spremenljivke vseh registriranih komponent in ob spremembi ustvari dogodek, ki ga pošlje komponenti.

Poleg implicitne povezave na mrežne spremenljivke komponenta SV-PVA nudi tudi storitve povezave na poljubno mrežno spremenljivko ostalim komponentam sistema FECOS. Komponente se lahko povežejo s spremenljivko in berejo ali nastavljajo njeno

vrednost s pomočjo dogodkov. Po ustvarjeni povezavi lahko komponenta pošlje dogodek za branje ali pisanje vrednosti spremenljivke.

Poleg tega zna komponenta za določene mrežne spremenljivke ustvariti tudi t.i. *monitor*. Pri tem komponenta SV-PVA nadzoruje čas zadnje spremembe vrednosti spremenljivke in vsakič, ko zazna spremembo, komponenti pošlje dogodek z novo vrednostjo spremenljivke. Komponenta dogodke o spremembah vrednosti mrežne spremenljivke dobiva le v stanju *Op*.

Pri zgoraj opisanih dogodkih mora komponenta, ki storitev potrebuje, obvezno določiti vrednost polja dogodka **Event ID**, katero potem vsebujejo vsi dogodki, ki jih komponenta SV-PVA pošlje v odgovor.

5.4.4 Razredi za branje podatkov

Zadnji dve od štirih osnovnih sistemskih komponent sta komponenti, ki nudita storitve branja podatkov z lokalnega diska ali glavnega podatkovnega strežnika kontrolnega sistema pospeševalnika.

5.4.4.1 File Service

`File Service` je tretja osnovna sistemska komponenta. Namenjena je branju in pisanju podatkov z diska kontrolnega računalnika končnih naprav. Čeprav samo branje in pisanje podatkov na disk v okolju LabVIEW ni nič bolj zapleteno kot kje drugje, smo za ta namen razvili posebno komponento iz več razlogov:

- enoten vmesnik do vseh storitev sistema (pošiljanje dogodkov);
- komponenta `File Service` že poskrbi za podatke v XML obliki;
- komponenta `File Service` samodejno prebere podatke s pravega mesta glede na izbrani tip konfiguracije (poglavje 5.6).

Kot omenjeno, komponenta podpira dostop do podatkov na dva načina. Če podatke vrne kot *surove*, so podatki shranjeni v spremenljivki tipa **String**. Prav tako lahko storitev odpre in že prevede datoteke tipa XML. V tem primeru vrne podatke kot drevo DOM, po katerem aplikacija išče z osnovnimi operacijami za iskanje po drevesu.

Komponenta `File Service` omogoča tudi zapisovanje podatkov na trdi disk računalnika, vendar je storitev omejena zgolj na *surove* podatke, saj sistem FECOS ne omogoča spreminjanja ali ustvarjanja drevesa DOM.

5.4.4.2 HTTP Service

Zadnja od osnovnih sistemskih storitev je namenjena branju podatkov z glavnega podatkovnega strežnika kontrolnega sistema pospeševalnika in jo imenujemo HTTP `Service`.

Za prenos podatkov je bil izbran protokol HTTP, ki je zelo razširjen in preizkušen ter ponuja veliko izbiro kvalitetnih strežnikov. Okolje LabVIEW vsebuje vse funkcije za omenjeni protokol.

Komponenta HTTP `Service` je izpeljana iz enakega razreda kot `File service`, torej `Data service`. V osnovi ponuja enake storitve, vendar storitev zapisovanja podatkov ni implementirana, ker ta možnost ni predvidena v kontrolnem sistemu pospeševalnika.

Komponenta omogoča branje podatkov z glavnega strežnika kontrolnega sistema, katerega naslov je zapisan v konfiguraciji komponente. V tem primeru je za dostop do podatkov potreben relativni naslov (URL). V primeru, da je uporabljen absolutni naslov, pa komponenta omogoča branje s poljubnega strežnika HTTP na omrežju.

Komponenta omogoča aplikaciji zahtevo za prenos podatkov neposredno na trdi disk kontrolnega računalnika končnih naprav, pri čemer aplikacija prejme zgolj potrdilo o prenosu podatkov in ne podatkov samih. To je zlasti uporabno za prenos velikih datotek, arhivov ZIP ali podatkov, ki se jih med delovanjem pospeševalnika potrebuje na lokalnem disku za hiter dostop, v trenutku prenosa pa ne.

5.4.5 Pisanje dnevnika

Sistem FECOS omogoča zapisovanje dnevnikov delovanja aplikacij, pri čemer se zgleduje po knjižnicah *log4j* in *log4net* fundacije Apache. Razreda za vodenje dnevnika omogočata klasifikacijo sporočil glede na 5 stopenj:

- *Debug;*
- *Info;*
- *Warning;*
- *Error;*
- *Fatal.*

Oba razreda omogočata tudi lokalizacijo sporočil dnevnika. Lokalizirana sporočila so zapisana v posebnih datotekah na lokalnem disku kontrolnega računalnika končnih naprav. Sporočila omogočajo parametrizacijo za zapisovanje konkretnih informacij o napaki ali obvestilu.

Pomembna lastnost razredov za vodenje dnevnika je še filtriranje sporočil glede na njihovo pomembnost. Oba razreda sporočila samodejno opremita z informacijo o času zapisa sporočila, kar omogoča lažjo rekonstrukcijo dogodkov, ki so pripeljali do napake. Vsako sporočilo dnevnika vsebuje naslednje podatke:

- stopnjo sporočila;
- ID komponente;
- lokalizirano sporočilo;
- mesto napake v aplikaciji (*stack trace*);
- številsko kodo napake;
- UUID napake;
- mrežno ime naprave;
- čas zapisa sporočila;
- ime metode zapisa sporočila;
- uporabniško ime.

5.4.5.1 Local Logger

Razred `Local Logger` skrbi za vodenje dnevnika na lokalnem disku kontrolnega računalnika končnih naprav. Podpira tri možne načine vodenja dnevnika, in sicer:

- pisanje v eno datoteko v vsem času delovanja kontrolnega računalnika končnih naprav (**append**);
- pisanje v prazno datoteko ob vsakem zagonu sistema FECOS (**rewrite**);
- pisanje v datoteko omejene dolžine (**rollover**).

V primeru *rollover* dodaten parameter konfiguracije določi omejitev velikosti datoteke. V primeru, da datoteka dnevnika preseže določeno velikost, komponenta ustvari novo datoteko v katero nadaljuje z zapisovanjem sporočil. Komponenta poleg aktive datoteke hrani le še prejšnjo datoteko, starejše pa samodejno zbriše.

5.4.5.2 Remote Logger

Razred `Remote Logger` skrbi za pošiljanje sporočil dnevnika centralnemu strežniku. Za komunikacijo s centralnim strežnikom se uporablja serializacijo XML [17] log4net [16] sporočil, kar pomeni, da je centralni strežnik lahko katerikoli log4net strežnik, ki potem sporočila zapisuje v bazo ali pa kam drugam.

Razred podpira dva načina komunikacije s strežnikom; pošiljanje sporočil v paketu in takojšnje pošiljanje sporočil. Ker je pošiljanje XML sporočil dokaj počasna operacija se predvideva, da komponenta večino sporočil shrani v lokalni pomnilnik in jih pošlje

centralnemu strežniku le občasno (pošiljanje v paketu). Takojšnje pošiljanje sporočil se uporabi za pomembna sporočila, ki kažejo na hudo napako v delovanju sistema. Nastavitev, ki določa mejo med pomembnimi in navadnimi sporočili, se nastavi za vsako komponento sistema posebej.

Vsaka komponenta torej pozna dve nastavitvi:

- *minRemoteLevel* – minimalno stopnjo sporočila, ki se pošlje centralnemu strežniku;
- *urgentRemoteLevel* – sporočila enake ali višje stopnje se centralnemu strežniku pošlje takoj.

Sama komponenta `Remote Logger` ima nastavitve, ki določajo njeno delovanje pri komunikaciji s centralnim strežnikom:

- *cacheSize* – ko število shranjenih sporočil doseže vrednost nastavitve, poskuša komponenta sporočila poslati centralnemu strežniku. V primeru, da to ni mogoče komponenta sprejema nova sporočila in poskuša sporočila vedno znova poslati centralnemu strežniku;
- *maxCacheSize* – ko število shranjenih sporočil doseže vrednost nastavitve, se do takrat zbrana sporočila zavržejo ali shranijo v datoteko na lokalnem disku. V dnevnik se doda obvestilo, da so bila sporočila zavržena;
- *flushTimeout* – čas oziroma število sekund po katerem se shranjena sporočila pošljejo centralnemu strežniku, ne glede na število zbranih sporočil;
- *loggerAddress* – naslov centralnega strežnika.

5.4.6 Razredi za komunikacijo

Komponente na istem kontrolnem računalniku končnih naprav lahko komunicirajo preko dogodkov, načeloma pa direktna komunikacija med komponentami na različnih kontrolnih računalnikih končnih naprav ni potrebna. V veliko primerih komponente zbirajo podatke, ki jih potrebuje ena aplikacija, ali obratno ena aplikacija pošlje sporočilo, ki ga mora prejeti več komponent sistema. Za takšno uporabo je najbolj primeren protokol objavi/naroči [18], katerega glavna prednost je, da omogoča objavljanje in sprejemanje podatkov poljubnemu številu sodelujočih, ne da bi bila med njimi potrebna povezava *vsak-z-vsakim*.

5.4.6.1 Messaging

Razred `Messaging` nudi ostalim komponentam sistema storitev objavljanja podatkov ali naročanja na podatke, ki jih objavlja ena ali več aplikacij kontrolnega sistema MedAustron. Za komunikacijo se uporablja protokol MAPS, ki je bil namensko razvit za potrebe kontrolnega sistema MedAustron in je natančneje opisan v poglavju 5.5.1.2.

Protokol sam po sebi omogoča objavljanje podatkov poljubnega tipa in ne omejuje tipa podatkov na kanal. To pomeni, da lahko različne komponente pod istimi oznakami objavljajo povsem različne podatke in naloga prejemnika podatkov je, da jih zna pravilno interpretirati. Zaradi tega vsak naročnik poleg identifikacije kanala, na katerega se naroča, sporoči tudi tip podatkov, ki jih želi po tem kanalu sprejemati. Komponenta `Messaging` potem naročnikom pošilja zgolj podatke tipa, na katerega so se naročili.

Edini dve nastavitvi, ki jih komponenta `Messaging` potrebuje, sta naslov in številka vrat centralnega strežnika protokola MAPS. Pri tem naj omenim še, da centralni strežnik ni razvit v okolju LabVIEW, ampak je napisan v jeziku C#.

Komponenta vzpostavi povezavo s centralnim strežnikom MAPS kontrolnega sistema MedAustron šele, ko prejme prvi podatek za objavo ali zahtevo po naročnini. Komponenta torej skrbi tako za objavo podatkov, kot za naročanje na podatke, ki jih objavljajo drugi.

Pri protokolih tipa objavi/naroči se večkrat uporablja tako imenovane *kanale* oziroma *teme*, ki objavljane informacije razdelijo na logične enote. Naročniki se posledično lahko naročijo le na podatke, ki jih zanimajo. Pri protokolu MAPS smo namesto tega uvedli *oznake*. Tudi njihova uporaba je namenjena ločevanju podatkov glede na namen, razlikujejo pa se v tem, da podatkov ne ločujejo povsem. Vsak objavljeni podatek lahko vsebuje poljubno množico oznak in naročnik se lahko pri ustvarjanju naročnine odloči za poljubno (pod)množico oznak, s katero izbere želene podatke. Pri pravilni izbiri oznak lahko prejema točno določeno vrsto podatkov, ali pa z ustrezno izbiro oznak podatke zbira od več virov hkrati.

Pri objavi ni potrebno vzdrževati nikakršnega konteksta in objava podatkov je v osnovi anonimna, kar pomeni, da objavljeni podatki implicitno ne nosijo informacije o tem, katera komponenta jih objavlja. Če bi bilo potrebno, lahko podatki to informacijo vsebujejo eksplicitno. Podatki, ki so implicitni del vsake objave, so ime tipa podatkov in verzija tipa. Verzija je namenjena temu, da lahko identificira spremembo v tipu podatkov, če do nje pride. Če bi se na primer med delovanjem ugotovilo, da podatki potrebujejo tudi informacijo o času, se lahko ta podatek doda v tip in verzija se spremeni z *1.0* na *2.0*. S tem bi lahko nekatere komponente še vedno objavljale starejšo verzijo podatkovnega tipa in prejemnik bi lahko med obema razlikoval oziroma prilagodil delovanje.

Pri naročnini komponente določijo množico *oznak*, na katere se želijo naročiti, definirajo pa tudi tip podatkov, ki ga želijo prejemati. Pri naročilu na podatke komponenta prejme *oznako naročila*, ki je lahko enaka za več naročil, saj se prav lahko zgodi, da se več

komponent naroči na enake podatke. V tem primeru komponenta `Messaging` podatke prejme le enkrat in njena naloga je, da jih razdeli med naročnike.

5.4.7 Opis API

V sistemu FECOS je API realiziran v dveh oblikah. V prvi gre za javne in zaščitene metode, ki jih podpirajo razredi v hierarhiji FECOS razredov, v drugi pa za dogodke, ki jih podpirajo posamezne sistemske komponente in preko njih nudijo storitve aplikacijam.

5.4.7.1 Opis metod, ki jih posamezni razredi podpirajo

Kot prikazuje hierarhija objektov (Slika 15) sta osnovna razreda sistema FECOS `State Machine` in `Component`. Vsebujeta večino metod, ki jih ostali razredi lahko uporabljajo za delovanje. Poglavje podaja kratek opis le-teh z namenom predstavitve celotnega sistema, čeprav nekatere pojme in koncepte zaradi širine pušča nepojasnjene.

Metode, ki jih podpira razred `State Machine` so:

Metoda	**Opis metode**
`State To String`	Vrne ime trenutnega stanja, da se ga lahko uporabi za izpis na zaslon.
`Create`	Ustvari nov objekt razreda `State Machine` in je v bistvu konstruktor tega razreda, kot je opisano v poglavju 4.3.
`Send Command`	Končnemu avtomatu, ki ga implementira objekt, pošlje enega od ukazov, ki jih našteva Tabela 2.
`Force Stop`	Ustavi izvajanje procesa, ki je bil ustvarjen s klicem metode `Create`.
`Is Running`	Vrne **Boolean** vrednost, ki kaže ali je proces objekta še aktiven.
`Read State`	Vrne stanje, v katerem je končni avtomat, ki ga implementira objekt.
`Read Sleep Value`	Vrne čas neaktivnosti procesa (v milisekundah) med iteracijami izvajanja kode.

`Write Sleep Value`	Nastavi čas neaktivnosti procesa (v milisekundah) na novo vrednost.
`Read Priority`	Vrne prioriteto procesa objekta.
`Read Language`	Vrne jezik uporabljan za lokalizacijo sporočil dnevnika.
`Write Language`	Nastavi jezik uporabljan za lokalizacijo sporočil dnevnika.

Tabela 4: Javne metode razreda `State Machine`

Metoda	**Opis metode**
`Reject State Transition`	Omogoča programerju, da eksplicitno zavrne menjavo stanja končnega avtomata pod določenimi pogoji. Zavrnitev se zapiše v dnevnik.
`Wake Up`	Prekine stanje neaktivnosti procesa.
`Read Mode`	Vrne *mode* v katerem trenutno teče končni avtomat.
`Set Next Mode`	Nastavi *mode* v katerega se bo prestavil končni avtomat v naslednji iteraciji.
`Write Priority`	Nastavi prioriteto procesa objekta.
`Get Error Message`	Vrne lokalizirano sporočilo, ki pripada napaki določene kode.
`Read Failure`	Pove ali je potrebno ob naslednji spremembi stanja preiti v stanje napake – *Fail* ali *Fault*.
`Write State`	Nastavi stanje na novo vrednost, ne da bi se upoštevalo shemo končnega avtomata, ki jo prikazuje Slika 12.
`Read Sleep Helper`	Vrne referenco na objekt, ki prekine neaktivnost procesa objekta.
`Read Block Transition`	Interna metoda, ki pove če je prehod v novo stanje zavrnjen.

Write Block Transition	Nastavi vrednost, ki zavrne menjavo stanja končnega avtomata objekta.
Read Conf Locked	Vrne **Boolean** vrednost, ki pove ali je konfiguracija objekta zaklenjena.
Write Conf Locked	Zaklene ali odklene konfiguracijo objekta.

Tabela 5: Zaščitene metode razreda State Machine

Kot je videti s seznama se večina metod razreda State Machine ukvarja z osnovnimi operacijami končnega avtomata, s procesi v katerih tečejo in lokalizacijo dnevniških sporočil.

Daleč največ metod prinaša razred Component, ki je osnovni razred sistema FECOS in temelj vsem ostalim razredom.

Metoda	**Opis metode**
Log Message	Sporočilo ali napako obdela in glede na nastavitve uporabi primeren dnevnik. V primeru, da objekti za pisanje dnevnika še niso aktivni, vsaka komponenta zapiše sporočilo v lastno datoteko. Ta mehanizem se uporablja le na začetku, preden prvi od objektov za vodenje dnevnika postane aktiven in preprečuje, da bi se sporočila ob zagonu sistema FECOS izgubila.
Dispatch Event	Vstavi dogodek v izhodno vrsto objekta. S tem ga preda sistemu FECOS, ki ga dostavi izbranemu objektu.
Create With ID	Ustvari nov objekt razreda Component in se od metode Create razreda State Machine razlikuje v tem, da komponenti določi tudi identifikacijsko ime (ID) in konfiguracijo.
Receive Event	Vstavi dogodek v vhodno vrsto objekta in objekt ga kasneje obdela. S to metodo Executive dostavlja dogodke objektom.

`Reload Log Messages`	Ponovno naloži sporočila dnevnika, lokalizirana za izbrani jezik.
`Real-time Action`	Prototipna metoda, ki jo razvijalec napolni s kodo, ki mora teči v realnem času. Metoda mora biti pravilno napisana, da se jo lahko uporablja v sistemu FECOS.
`Read Device Name`	Vrne ime naprave – *hostname*.
`Read ID`	Vrne identifikacijsko ime komponente.
`Get Value`	Prebere vrednost omrežne spremenljivke.
`Put Value`	Zapiše vrednost omrežne spremenljivke.
`Get Min Local Level`	Vrne minimalno stopnjo pri kateri se sporočila še zapisujejo v dnevniško datoteko.
`Get Min Remote Level`	Vrne minimalno stopnjo pri kateri se dnevniška sporočila pošiljajo centralnemu strežniku.
`Get Urgent Remote Level`	Vrne minimalno stopnjo pri kateri se dnevniška sporočila pošiljajo centralnemu strežnika. Ta stopnja je višja od stopnje `Remote Level`.
`Set Min Local Level`	Nastavi minimalno stopnjo pri kateri se sporočila še zapisujejo v dnevniško datoteko.
`Set Min Remote Level`	Nastavi minimalno stopnjo pri kateri se dnevniška sporočila pošiljajo centralnemu strežniku.
`Set Urgent Remote Level`	Nastavi minimalno stopnjo pri kateri se dnevniška sporočila pošiljajo centralnemu strežnika. Ta stopnja mora biti višja od stopnje `Remote Level`.
`Set Property`	Nastavi lastnost objekta s poljubnim imenom. Lastnosti niso enake zasebnim atributom objekta in se jih lahko nastavlja v poljubnem trenutku brez predhodne deklaracije. Uporabne so za dinamične in začasne spremenljivke.
`Get Property`	Vrne vrednost prej nastavljene lastnosti objekta s poljubnim imenom.

`Is Component Ready`	Pove, če je komponenta pripravljena za delovanje.

Tabela 6: Javne metode razreda `Component`

Metoda	**Opis metode**
`Set Failure`	Nastavi, da je potrebno ob naslednji spremembi stanja preiti v stanje napake – *Fail* ali *Fault*.
`Add Monitor`	Doda monitor (opisano v poglavju 5.4.3) na določeno omrežno spremenljivko.
`Remove Monitor`	Odstrani prej nastavljen monitor na omrežno spremenljivko.
`Start RT Action`	Zažene metodo, ki mora teči v realnem času.
`Set Status`	Nastavi mrežno spremenljivko (*WinCC OA* DPE) komponente *statusOut*.
`Read Configuration`	Vrne konfiguracijo komponente, ki je bila nastavljena ob kreiranju.
`Write ID`	Omogoča spremembo identifikacije komponente.
`Get VI Reference`	Vrne referenco na VI (metodo) na lokalnem disku.
`Read Outbound Queue`	Vrne referenco na izhodno realno-časovno vrsto. Ta se po določenih lastnostih razlikuje od navadne vrste in se uporablja v delih kode, ki teče v realnem času. Njena glavna uporaba je za komunikacijo med deli kode, ki morajo teči v realnem času in deli kode, ki nimajo te zahteve.
`Read Inbound Queue`	Vrne referenco na vhodno realno-časovno vrsto.
`Read RT Active`	Pove, ali je metoda, ki se izvaja v realnem času, trenutno aktivna.
`Set Scratchpad Enabled`	Aktivira t.i. *tablo*.
`Scratchpad Enabled`	Pove, če je t.i. *tabla* aktivna.

`Read CCV`	Vrne trenutno vrednost nastavitve, ki je odvisna od trenutnega cikla pospeševalnika – *current control value*.
`New Scratchpad Notification`	Metoda, ki jo sistem FECOS kliče ob spremembi vrednosti t.i. *table*. Razvijalec to metodo napolni s kodo v primeru, da mora na nove vrednosti reagirati preden stopijo v veljavo.

Tabela 7: Zaščitene metode razreda `Component`

5.4.7.2 Opis dogodkov, ki jih posamezni razredi podpirajo

Poleg metod, ki jih vsebujeta razreda `State Machine` in `Component` in so preko dedovanja dostopne vsem razredom sistema, sistemske komponente ponujajo tudi storitve, ki so ostalim objektom sistema na voljo preko dogodkov. Seznam in opis le-teh se nahaja spodaj.

Razred	**Dogodek**	**Opis**
`Executive`	GET_CMPNT	Dogodek z odgovorom vsebuje referenco na zahtevano komponento. V primeru, da zahtevana komponenta še ni bila registrirana pri objektu `Executive`, si zahtevo shrani in jo izpolni kasneje.
`SV-PVA`	PVA_CONN	Za pošiljatelja dogodka odpre povezavo na omrežno spremenljivko.
`SV-PVA`	PVA_DISC	Zapre povezavo do zahtevane omrežne spremenljivke.
`SV-PVA`	PVA_GET	V določeno omrežno spremenljivko zapiše vrednost. Pred tem je potrebno odpreti povezavo z dogodkom PVA_CONN.
`SV-PVA`	PVA_PUT	Vrne vrednost izbrane omrežne spremenljivke. Pred tem je potrebno odpreti povezavo z dogodkom PVA_CONN.
`SV-PVA`	PVA_MON	Odpre monitor (poglavje 5.4.3) do izbrane omrežne spremenljivke. Zahteva implicitno odpre tudi novo povezavo. Naročnik kot odgovor prejme trenutno vrednost spremenljivke, kasneje pa ob vsaki spremembi novo vrednost.
`SV-PVA`	PVA_DSRY	Zapre monitor do izbrane omrežne spremenljivke.

File Service	GET	Vrne vsebino izbrane datoteke. V primeru datoteke XML storitev vrne podatke v obliki drevesa DOM. V vseh ostalih primerih storitev vrne neobdelane podatke kot niz znakov – **String**.
File Service	PUT	Zapiše vrednost dogodka v datoteko. V primeru, da dogodek specificira tip podatkov kot XML, storitev drevo DOM zapiše v datoteko, v nasprotnem primeru gre za neobdelane podatke, ki so podani kot niz znakov – **String**.
HTTP Service	GET	Vrne vsebino izbrane datoteke iz naslova podanega kot URL ter po potrebi zahtevane podatke zapiše na izbrano mesto na lokalnem disku kontrolnega računalnika končnih naprav. Če je pričakovana količina podatkov prevelika, lahko naročnik zahteva, da se podatki sami ne vrnejo v dogodku z odgovorom.
Messaging	MSG_P2O, MSG_P2M	Komponenta z dogodki tega tipa objavlja podatke na strežniku MAPS (glej poglavje 5.4.6.1). Pred objavo podatkovnega tipa objavljenih podatkov ni potrebno registrirati.
Messaging	MSG_S2O, MSG_S2M	Komponenta se z dogodki tega tipa naroči na podatke, ki se objavljajo na strežniku MAPS. Dogodek mora vsebovati tudi tip podatkov, na katerega se komponenta naroča, s čemer se izvede registracija tipa.

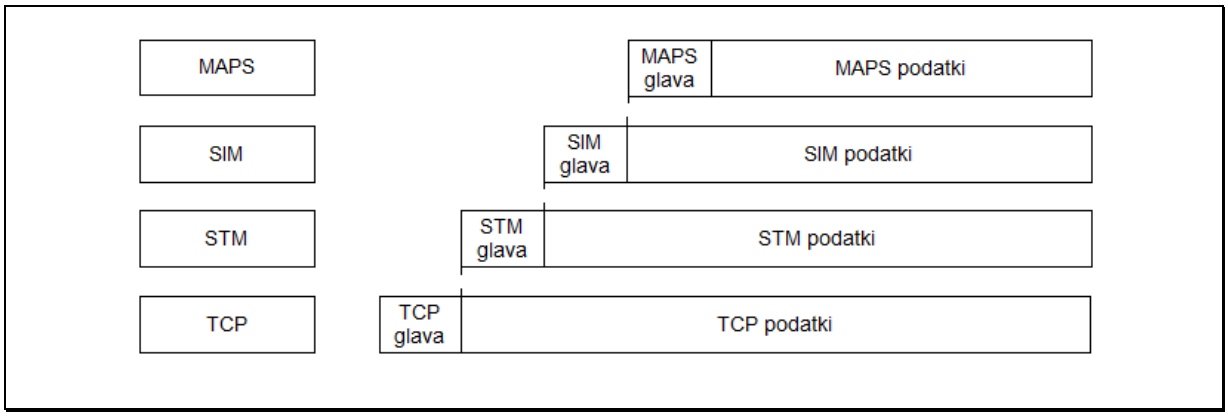
Tabela 8: Dogodki storitvenih razredov sistema FECOS

5.5 Pomožne knjižnice

Poleg samih komponent so bili za delovanje sistema razviti tudi namenski komunikacijski protokoli in pomožne knjižnice za obdelavo podatkov XML.

5.5.1 Knjižnici za mrežno komunikacijo

Za mrežno komunikacijo med komponentami sistema in s strežnikom MAPS smo razvili dva protokola, in sicer protokol SIM in njegovo nadgradnjo protokol objavi/naroči – MAPS. Hierarhija komunikacijskih protokolov je prikazana na spodnji sliki.



Slika 17: Shema hierarhije komunikacijskih protokolov

5.5.1.1 Knjižnica SIM

Protokol SIM kot osnovo uporablja preprost protokol STM [19]. Namenjen je izmenjavi eno ali dvosmernih sporočil preko omrežja. Uporablja ga več aplikacij, pri čemer ima vsaka aplikacija svoj vmesnik. Vsi podatki v protokolu SIM so zapisani v t.i. *little endian* obliki. Trenutno definirani vmesniki so *MTS*, *VAA* in *MAPS*. Glavo protokola SIM sestavljajo naslednja polja:

Polje	**Velikost v bitih**	**Opis**
Identifikacija protokola	16	*Magično število* 4D41 v šestnajstiškem zapisu, ki predstavlja ASCII kodi znakov **M** in **A**.
Verzija protokola	8	Verzija protokola SIM predstavljena z dvema 4-bitnima številoma. Trenutna verzija je 1.0.
Oznaka vmesnika	6	Število, ki opisuje vmesnik sporočila. Trenutno definirani vmesniki so: • Nedovoljena vrednost – 0; • MAPS – 1;

		• MTS – 2; • VAA – 3; • Test – 63.
Zahteva	1	Zastavica, ki pri dvosmernih sporočilih določa ali gre za zahtevo ali odgovor.
Dvosmerno sporočilo	1	Zastavica, ki določa ali je sporočilo eno ali dvosmerno.
Oznaka sporočila	8	Oznaka, ki enolično identificira sporočilo znotraj vmesnika.
Verzija vmesnika	8	Verzija vmesnika predstavljena z dvema 4-bitnima številoma. Trenutna verzija je 1.0.

Tabela 9: Polja glave protokola SIM

Tako protokol kot vmesnik sta predstavljena z verzijo. Verzija omogoča, da se lahko definicija protokola ali vmesnika spremeni, na kar se lahko strežnik in klient pravilno odzoveta.

Knjižnica za delo s protokolom SIM vsebuje razred, ki ponuja metode za komunikacijo in metode za manipulacijo polj glave protokola. Pri pošiljanju se vsakemu sporočilu doda glava protokola in sporočilo se v serializirani obliki pošlje s pomočjo protokola STM.

Metode razreda, ki jih implementira razred SIM so:

Ime metode	**Opis**
`Write SIM Interface`	Nastavi polje *vmesnik SIM* na želeno vrednost. Vhodni parameter za nastavitev vmesnika je enumeracija, kar pomeni, da razvijalec ne more nastaviti nedefinirane vrednosti.
`Read SIM Interface`	Vrne vrednost *vmesnika SIM*.
`Write Connection Info`	Nastavi strukturo za opis povezave, ki jo uporablja protokol STM.
`Read Connection Info`	Vrne strukturo za opis povezave, ki jo uporablja protokol STM.
`Read Default Port Number`	Vrne privzeto številko vrat TCP, ki se uporablja za izbrani vmesnik. Predvideno je, da se za sporočila na različnih vmesnikih uporablja različne povezave TCP.

`Read Port Number`	Vrne številko vrat TCP, ki jih uporablja strežnik SIM.
`Connect`	Odpre povezavo SIM do strežnika na določenih vratih TCP.
`Accept`	Sprejme vhodno povezavo TCP. Ta metoda se uporablja v primeru, da se implementira strežnik, ki sprejema povezave.
`Close`	Zapre povezavo SIM.
`Read`	Prebere naslednje sporočilo, ki je prišlo po povezavi SIM.
`Write`	Pošlje sporočilo po povezavi SIM.

Tabela 10: Metode razreda `SIM Client`

5.5.1.2 Knjižnjica MAPS

Protokol MAPS je preprost protokol objavi/naroči, pri katerem odjemalci objavljajo in se naročajo na podatke poljubnega tipa. Podatki se objavljajo s t.i. oznakami, protokol sam pa temelji na protokolu SIM.

Protokol pozna štiri tipe sporočil: *publish*, *subscribe*, *unsubscribe* in *heartbeat*. Tip sporočila določa vrednost polja SIM glave *Oznaka sporočila* (glej Tabela 9).

Sporočila tipa *heartbeat* se uporabljajo za detekcijo stanja povezave. Odjemalec poskuša občasno poslati sporočilo tipa *heartbeat* tudi če ne prejema ali objavlja podatkov in s tem lahko zazna morebitne probleme na omrežni povezavi.

Pri sporočilu tipa *subscribe* odjemalec strežniku MAPS pošlje enodimenzionalno polje oznak. S tem začne prejemati sporočila z vsemi naštetimi oznakami. Pri sporočilu tipa *unsubscribe* je vsebina enaka in s tem strežnik MAPS preneha pošiljati podatke s poslanimi oznakami.

Pri sporočilu tipa *publish* odjemalec v sporočilu pošlje enodimenzionalno polje oznak, čemur sledita ime in verzija tipa. Oba podatka sta predstavljena kot niza. Šele nato sledijo sami podatki.

Funkcije za uporabo protokola MAPS so zbrane v razredu `MAPS Client` in opisane v spodnji tabeli.

Ime metode	**Opis**
`Connect`	Metoda omogoča povezavo z strežnikom MAPS. Vhodni parametri so naslov strežnika, vrata TCP na katerih deluje in čas v sekundah, v katerem je potrebno strežniku poslati sporočilo tipa *heartbeat*.
`Disconnect`	Zapre povezavo do strežnika.
`Is Connected`	Pove, če je povezava trenutno vzpostavljena.
`Read STM Options`	Vrne nastavitve protokola STM, ki se uporabljajo pri komunikaciji.
`Write STM Options`	Spremeni nastavitve protokola STM, ki se uporabljajo pri komunikaciji.
`Register Data Def`	Prijavi nov tip podatkov z imenom in verzijo za objavo ali naročilo.
`Unregister Type Def`	Zbriše aktivno prijavo za določen tip podatkov.
`Publish`	Objavi podatke z določenimi oznakami. Oznake so podane kot enodimenzionalno polje imen.
`Subscribe`	Ustvari naročnino na poljubne podatke objavljene z določenimi oznakami. Metoda vrne identifikacijo naročnine, ki se uporablja pri metodah `Unsubscribe` in `Get Messages`.
`Unsubscribe`	Zbriše naročilo z izbrano identifikacijo.
`Get Messages`	Vrne vsa sporočila, ki so v preteklem času prišla od strežnika, oziroma počaka določeno število milisekund na nova sporočila. Metoda vrne dve enodimenzionalni polji. V prvem so identifikacije naročnin prejetih sporočil in indeks na podatke v drugem polju. Drugo polje pa vsebuje prejete podatke. Predstavitev z dvema poljema je izbrana, ker lahko isti podatki pripadajo več naročninam in na ta način imamo za vse naročnine le eno kopijo podatkov.

`Send Heartbeat`	Sporočilo tipa *heartbeat* pošlje strežniku, če je od prejšnjega sporočila minilo dovolj časa. To metodo implicitno kličejo tudi metode `Publish`, `Subscribe`, `Unsubscribe` in `Get Messages`.

Tabela 11: Metode razreda MAPS `Client`

5.5.2 Knjižnica za branje XML

Okolje LabVIEW ima svojo implementacijo knjižnice za delo z datotekami XML. Implementacija sicer popolnoma ustreza standardom, ki jih definira ustanova W3C in je enakovredna implementacijam v drugih jezikih, kot je na primer Java, vendar je ta funkcionalnost na voljo le v aplikacijah LabVIEW, ki tečejo v okolju Windows.

Ker potrebujemo sposobnost branja datotek XML tudi na napravah PXI smo napisali svojo knjižnico za branje datotek XML, ki implementira potrebno funkcionalnost. Knjižnica ne podpira celotne specifikacije jezika XML, ampak le funkcionalnost, ki jo potrebujemo. Med drugim ne podpira imenskih prostorov, kakor tudi ne vozlišč, ki hkrati vsebujejo podvozlišča in tekstno vrednost.

Knjižnica prav tako ne prepozna vseh mogočih napak, ki se lahko pojavijo v datotekah XML in lahko sprejme tudi datoteko XML, ki bi jo povsem pravilna implementacija XML razčlenjevalnika zavrnila. Primer najbolj očitne razlike je, da knjižnica uspešno predela datoteko XML, ki ima dve korenski vozlišči, kar po XML standardu ni dovoljeno. Drugo korensko vozlišče preprosto ignorira.

Rezultat razčlenjevalnika je poenostavljeno drevo DOM, sestavljeno iz objektov razreda `Node`. Razred `Node` vsebuje štiri lastnosti:

- ime vozlišča;
- atribute, ki jih vozlišče vsebuje in so shranjeni v razpršeni tabeli;
- tekstno vrednost vozlišča, če jo le-to vsebuje;
- enodimenzionalno polje vozlišč XML, ki so nasledniki trenutnega.

Knjižnica vsebuje le nekaj osnovnih funkcij, ki jih potrebujemo v sistemu FECOS, oziroma nekaj specializiranih funkcij, ki iz definirane sheme dokumenta XML izluščijo konfiguracijske vrednosti.

Ime metode	**Opis**
`Parse XML`	Funkcija razčleni datoteko XML in v primeru, da ni prišlo do napake vrne poenostavljeno drevo DOM.

`Write XML`	Zapiše drevo DOM v datoteko XML.
`Find First Child`	Med nasledniki vozlišča poišče prvo z določenim imenom.
`Find Child By Attribute`	Med nasledniki vozlišča najde prvo z izbrano vrednostjo določenega atributa.
`Get Property Data`	V XML konfiguraciji komponente poišče vozlišče določene nastavitve in vrne njeno vrednost, tip in nekaj dodatnih atributov. Funkcija vrne tudi referenco na najdeno vozlišče, ki se uporablja v spodnjih funkcijah.
`Get Property Value Metadata`	Vrne metapodatke nastavitve, če jih le-ta ima: • fizikalna enota; • največja dovoljena vrednost; • najmanjša dovoljena vrednost.
`Get Property Cycle Dependent Data`	Vrne vrednosti nastavitve, ki so odvisne od trenutnega cikla pospeševalnika.
`Get Property Data From Node`	Funkcija je po vrnjenih vrednostih enaka funkciji `Get Property Data`, le da deluje na izbranem vozlišču in ga ne poišče med vsemi vozlišči konfiguracije.

Tabela 12: Funkcije knjižnice za delo z datotekami XML

5.6 Opis zagona naprave

Sistem FECOS je t.i. operacijski sistem kontrolnih računalnikov končnih naprav v pospeševalniku MedAustron, vendar teče le kot aplikacija na napravah PXI. Kljub temu se sistem FECOS srečuje s podobnimi problemi kot pravi operacijski sistemi ob zagonu, oziroma še dodatnimi, saj so sistem FECOS in njegove aplikacije odvisne od glavnega strežnika SCADA *WinCC OA*. Aplikacije, ki tečejo na kontrolnih računalnikih končnih naprav potrebujejo vsaka svoje nastavitve in lokalne podatke. Za vse to mora poskrbeti sistem FECOS.

Zaradi odprave napak se aplikacije s časom spreminjajo ali pa se razvijajo nove. Prav tako se spreminjajo tudi podatki, ki jih nekatere aplikacije uporabljajo. Zaradi hitrega dostopa morajo biti podatki že prisotni na disku kontrolnega računalnika končnih naprav. Posledično je sistem FECOS je zasnovan tako, da se na napravo naloži le osnovni sistem in FECOS ob zagonu avtomatsko naloži vse aplikacije, konfiguracijo in podatke s strežnika baze RMS. Sistem FECOS ob vsakem zagonu preveri ali se je verzija podatkov in aplikacij na strežniku RMS spremenila in v primeru spremembe naloži novo verzijo.

Strežnik baze RMS ima shranjeno celotno zgodovino konfiguracij pospeševalnika, vendar so tri označene s posebnimi imeni. Trenutno aktivna konfiguracija pospeševalnika se imenuje *pro*, verzija pred trenutno aktivno konfiguracijo pospeševalnika se imenuje *pre*, verzija, ki je trenutno v razvoju in bo postala aktivna ob koncu razvoja pa se imenuje *dev*.

Preko strežnika *WinCC OA* je mogoče izbrati katero od omenjenih konfiguracij naj kontrolni računalniki končnih naprav uporabijo ob naslednjem zagonu. Kontrolni računalniki vse tri konfiguracije hranijo na disku na ločenih lokacijah, tako da menjava konfiguracije iz *pro* v *pre* ne pomeni ponovnega prenosa podatkov. Ponovni prenos podatkov se zgodi le, če se je spremenila vsebina izbrane konfiguracije. V primeru nespremenjene vsebine sistem FECOS le zamenja mapo iz katere bere konfiguracijo.

Poleg sistema FECOS je na kontrolnem računalniku končnih naprav instalirana tudi datoteka z osnovnimi nastavitvami – *default.fml*. Le-ta je enaka na vseh napravah in vsebuje samo osnovne nastavitve potrebne za zagon sistema:

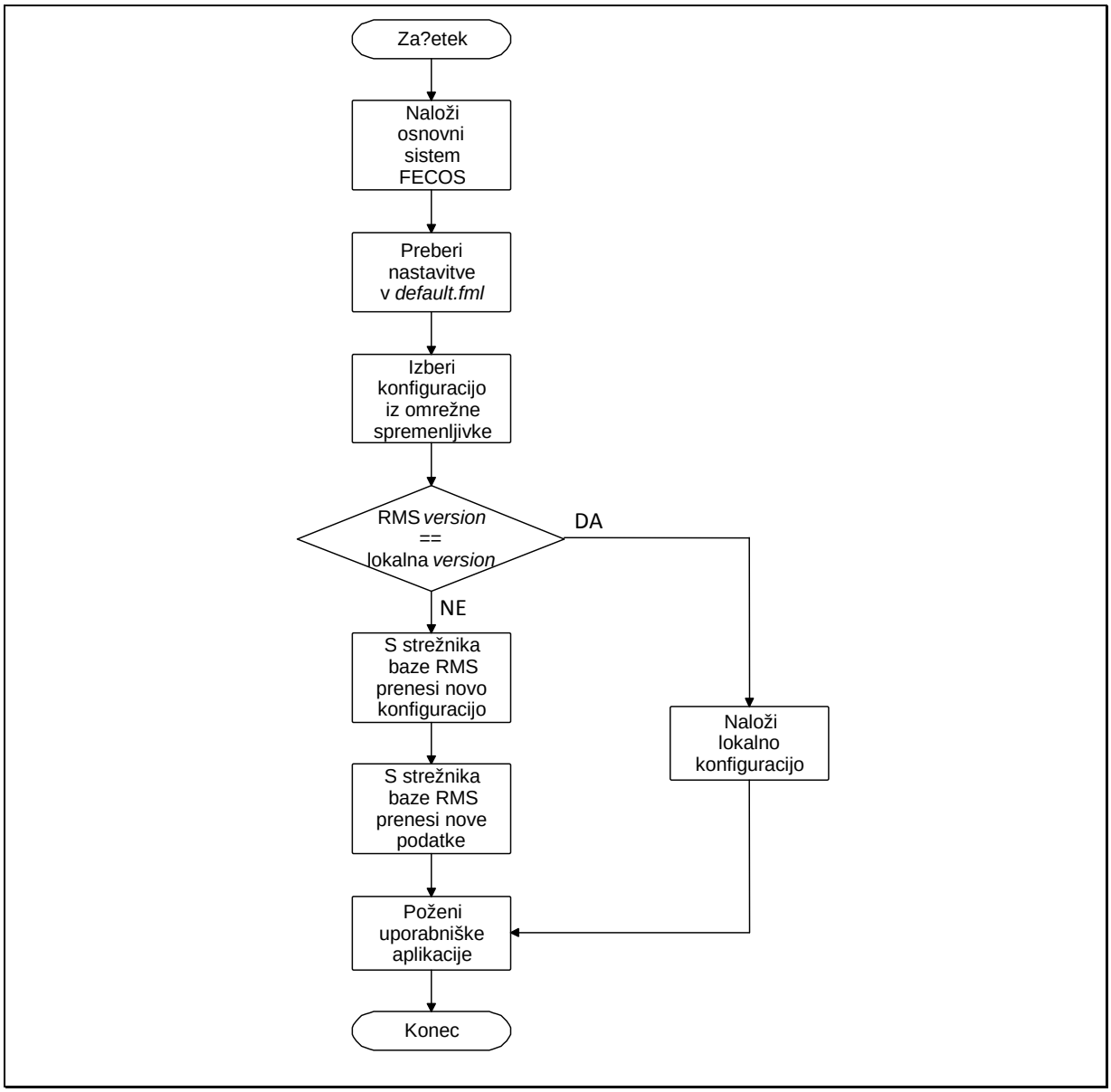
- naslov strežnika *WinCC OA*;
- naslov strežnika baze RMS;
- lokacijo korenskih map za konfiguracije *pro*, *pre* in *dev*;
- ime omrežne spremenljivke iz katere sistem FECOS prebere trenutno aktivno konfiguracijo (*pro*, *pre* ali *dev*).

Vsaka konfiguracija v korenski mapi vsebuje datoteko *version*, katere vsebina določa verzijo konfiguracije in se spremeni z vsako novo konfiguracijo. Postopek zagona naprave prikazuje Slika 18.

Ko se sistem FECOS naloži se najprej prebere osnovne konfiguracijske podatke iz datoteke *default.fml*. Iz omrežne spremenljivke strežnika *WinCC OA* se prebere trenutno aktivno konfiguracijo, nato se lokalno datoteko *version* iz mape trenutno aktivne konfiguracije primerja z datoteko *version* v mapi trenutno aktivne konfiguracije strežnika RMS. V primeru, da sta datoteki enaki, sistem FECOS naloži vse uporabniške aplikacije z lokalnega diska kontrolnega računalnika končnih naprav.

Če datoteki nista enaki sistem FECOS pobriše mapo trenutno izbrane konfiguracije. Nato s strežnika RMS prenese novo datoteko *version* in novo datoteko s konfiguracijo kontrolnega računalnika končnih naprav. Ta vsebuje tudi podatke o vseh arhivih ZIP, ki jih naprava potrebuje. Arhivi vsebujejo tako uporabniške aplikacije kot podatke, ki jih aplikacije potrebujejo. Sistem FECOS prenese arhive ZIP na napravo, jih razpakira na lokalni disk in

nadaljuje z nalaganjem uporabniških aplikacij. S tem so aplikacije in podatki na kontrolnem računalniku končnih naprav osveženi in ob naslednjem zagonu naprave menjava podatkov ni potrebna.



Slika 18: Zagon kontrolnega računalnika končnih naprav

5.7 Pregled nekaterih nesistemskih (uporabniških) aplikacij

Sistem FECOS predstavlja okolje v katerem bodo tekle uporabniške aplikacije, ki bodo upravljale končne naprave. Za celovitost pregleda bomo v tem poglavju našteli in na kratko opisali nekatere od njih. Njihova implementacija ni bila neposredno povezana z izdelavo te naloge.

Kot sem že omenil v poglavju 3.1 je ena izmed najpomembnejših aplikacij v kontrolnem sistemu pospeševalnika časovni sistem. Pri generiranju in kontroliranju žarka je potrebno različne naprave upravljati na določen način v točno določenih trenutkih in sekvencah. Za to skrbi časovni sistem. Operater pospeševalnika sestavi sekvence dogodkov na različnih napravah pospeševalnika in določi časovne razmike med njimi. Ko so take sekvence sestavljene, jih časovni krmilnik razpošlje različnim kontrolnim računalnikom končnih naprav. Končne naprave se lahko s temi sekvencami pripravijo na delovanje in potrdijo pripravljenost. Nato začne časovni krmilnik pošiljati dogodke, ki prožijo pripravljene sekvence v točno določenih časovnih intervalih.

Na kontrolnih računalnikih končnih naprav časovni odjemalci prejemajo dogodke po optiki in po namenskih vodilih pošiljajo signale ostalim krmilnikom, ki izvajajo v naprej pripravljene akcije. Tako časovni krmilnik kot odjemalci so aplikacije napisane za sistem FECOS.

Druga pomembna aplikacija upravlja z magneti pospeševalnika. V bistvu aplikacija upravlja z različnimi napajalniki, ki posledično spreminjajo stanje magnetov. Za upravljanje z magneti morajo napajalniki na izhodnih linijah predvajati t. i. valovne datoteke. Le-te določajo kako napajalnik skozi čas spreminja tok in napetost na izhodnih linijah, s čimer se uravnavata smer in jakost magnetnega polja. Valovne datoteke vsebujejo večjo količino podatkov, zato jih je potrebno naložiti v pomnilnik pred časovnim dogodkom (informacijo o sekvenci dogodkov pošlje časovni krmilnik) in jih poslati napajalniku ob prejetju prožilnega signala preko namenskih vodil.

Za delovanje pospeševalnika je prav tako potrebno upravljati z virom delcev. Sam vir je dokaj zahtevna naprava, ki potrebuje tudi do 30 minut za dosego operativnega stanja, operativne parametre pa je potrebno tudi med delovanjem neprestano prilagajati. Raziskovalno-medicinski pospeševalnik MedAustron ima dva taka vira, ker omogoča obsevanje z dvema tipoma delcev. Aplikacija mora torej poleg zagona in nadzora med delovanjem omogočati tudi menjavo tipa delcev.

Za delovanje pospeševalnika bodo potrebne še aplikacije za nadzor kvalitete žarka, ustvarjanje in nadzor vakuuma ter mnoge druge. Večina le-teh bo tekla na sistemu FECOS.

6 Zaključek

Izdelava kontrolnega sistema pospeševalnika delcev je zahteven projekt, ki potrebuje sodelovanje velike skupine ljudi. Poleg sestave osnovnega kontrolnega sistema je potrebno istočasno razvijati tudi večino ostalih aplikacij, ki bodo del razširjenega kontrolnega sistema. Nujno je neprestano sodelovanje vseh ekip, hkrati pa mora kontrolni sistem tudi v času razvoja delovati čim bolj stabilno, da ne ovira razvoja ostalih aplikacij.

Kontrolni sistemi pospeševalnikov delcev so v današnjem času že uveljavljena panoga. Obstaja kar nekaj kontrolnih sistemov v klasičnih programskih jezikih. Dva izmed bolj znanih sta EPICS [20] in TANGO [21]. Diplomsko delo opisuje prvi kontrolni sistem v razvojnem okolju LabVIEW. Z izdelavo kontrolnega sistema se je pokazalo, da sta okolje LabVIEW in programski jezik G povsem enakovredna klasičnim programskim jezikom, kot je C++.

Od začetka razvoja je kontrolni sistem v pogledu funkcionalnosti in stabilnosti že zelo napredoval in vstopa v zaključno fazo projekta. Sistem bo prešel v prvo uporabo na samem pospeševalniku predvidoma v prvi polovici leta 2013. Testiranja obstoječe kode na prototipnih napravah, ki se razvijajo za pospeševalnik, potekajo že sedaj. Sistem bo v popolnosti zaživel šele, ko bodo v centru MedAustron v avstrijskem mestu Wiener Neustadt obsevali prve paciente.

Viri

[1] Particle therapy, http://en.wikipedia.org/wiki/Particle_therapy

[2] EBG MedAustron Media Center, http://www.medaustron.at/en/service/presse/bilder/

[3] Ion source, http://en.wikipedia.org/wiki/Ion_source

[4] G. Pajor, Krmiljenje vakuumskega sistema predpospeševalnika avstralskega sinhrotrona, Diplomsko delo, FMF, Ljubljana, 2005

[5] Quadrupole magnet, http://en.wikipedia.org/wiki/Quadrupole_magnet

[6] SIMATIC WinCC Open Architecture, http://www.pvss.com/index_e.asp

[7] Product Information: What is LabVIEW, http://www.ni.com/labview/whatis/

[8] The benefits of Programming Graphically in NI LabVIEW, http://www.ni.com/labview/whatis/graphical-programming/

[9] FPGA, Field-programmable gate array, http://en.wikipedia.org/wiki/Field-programmable_gate_array

[10] IntervalZero Phar Lap ETS, http://www.intervalzero.com/ets.htm

[11] Wind River VxWorks, http://www.windriver.com/products/vxworks/

[12] Elementi datoteke VI, http://www.ni.com/images/gettingstarted/neutral/lv_env_6.jpg

[13] Execution Structures in NI LabVIEW, http://www.ni.com/gettingstarted/labviewbasics/exestructures.htm

[14] Bločni diagram s strani *Extract Rectangular ROI from Image in VBAI*, http://decibel.ni.com/content/servlet/JiveServlet/download/8595-1-11164/Block%20diagram.PNG

[15] LabVIEW Object-Oriented Programming: The Decisions Behind the Design, http://zone.ni.com/devzone/cda/tut/p/id/3574

[16] Apache log4net logging service, http://logging.apache.org/log4net/

[17] Apache log4net XmlLayout class, http://logging.apache.org/log4net/release/sdk/log4net.Layout.XmlLayout.html

[18] Publish/subscribe protokol, http://en.wikipedia.org/wiki/Publish/subscribe

[19] Simple Messaging Reference Library (STM), http://zone.ni.com/devzone/cda/epd/p/id/2739

[20] Experimental Physics and Industrial Control System, http://www.aps.anl.gov/epics/

[21] TANGO, http://www.tango-controls.org/

Slovar izrazov

baza RMS: repository management system

bločni diagram: block diagram

časovni krmilnik: timing master

časovni odjemalec: event receiver

časovni sistem: timing system

datoteka VI: virtual instrument

dinamične metode: dynamic dispatch methods

dogodek: event

DOM: document object model

dostopni razred: access scope

DPE: data point element

družbeni: community

enumeracija: enumeration

FECOS: Front End Control System - kontrolni sistem končne naprave

FIFO: First In, First Out

gruča: bunch

identifikator: ID

imenski prostor: namespace

ionska terapija: ion therapy, particle therapy

kanal: channel

kompleksen končni avtomat: state driven device

končna naprava: front-end device

kontrola: control

kontrolna plošča: front panel

kontrolni računalnik končnih naprav: FEC - front end controller

linearni predpospeševalnik: linear accelerator, linac

MAPS: MedAustron Publish/Subscribe

MTS: main timing system

nit: thread

objavi/naroči: publish/subscribe

ogrodje: framework

omrežna spremenljivka: shared variable

OPC: OLE for Process Control

oznake: tags

podatkovna referenca: data value reference

polje: array

preprosta naprava: basic device

priklopna plošča: connector pane

prožilni signal: trigger

radiofrekvenčna komora: radio-frequency chamber, RF chamber, RF cavity

razčlenjevalnik: parser

razpršena tabela: hash table

realno-časovna vrsta: real-time queue

realno-časovno vrsto: real-time queue

SCADA: supervisory control and data acquisition

SIM: Simple Interface Messaging

sistem za obsevanje: beam delivery system

statične metode: static dispatch methods

struktura: cluster

surovi podatki: raw data

tabla: scratchpad

tema: topic

tip podatkov: content-type

uporabniški prikazovalniki: LabVIEW indicator

uporabniško vnosno polje: LabVIEW control
usluga: service
UUID: universally unique identifier
VAA: virtual accelerator allocator
valovna datoteka: waveform
vir ionov: ion source
vrsta: queue, FIFO
žarek: beam
zaščiteni: protected
zasebni: private

Seznam slik

Seznam tabel