Programiranje sistemov industrijske avtomatizacije z domensko specifičnimi jeziki

Tomaž Kos

Dewesoft d.o.o.
Gabrsko 11a, 1420 Trbovlje
E-pošta: tomaz.kos@dewesoft.com

Programming industrial automation systems with domain-specific languages

Abstract. *We can say that today there is almost no industry that does not have at least one process automatic. It is possible to automate almost anything. There are various programming languages that can automate processes. For that reason, the correct chose of programming language is important. In this article two domain-specific visual programming languages are presented that can be used for this purpose. The first is Dewesoft Sequencer and the second is Ladder Diagram. Each has its own advantages and disadvantages, but which programming language to choose is always a question.*

1 Uvod

Avtomatizacija v industriji pomeni uporabo nadzornih sistemov, kot so računalniki in roboti, ter uporabo informacijskih tehnologij za upravljanje različnih procesov in strojev v industriji, ki nadomestijo človeka. Dandanes je avtomatizacija del vsakega proizvodnega ali tehnološkega procesa, tako pri izdelavi avtomobila, letala ali pa v kontinuiranem procesu kot je na primer nadzor celotnega sistema elektrarne. Tekoči trakovi in roboti so le del avtomatizacije. Poleg omenjenega, med avtomatizacijo štejemo tudi obdelavo procesnih podatkov, komunikacijo med porazdeljenimi sistemi, vizualizacijo, sledljivost, končno kontrolo, vzdrževanje kakor tudi vodenje podjetja. Prvotni namen industrijske avtomatizacije je bil povečati produktivnost in zmanjšati stroške povezane z zaposlenimi operaterji, saj lahko avtomatizirani sistemi delujejo 24 ur na dan. Danes se avtomatizacija osredotoča tudi na povečanje kakovosti in prožnosti v proizvodnem procesu.

V zadnjem desetletju se je na področju krmilne tehnike precej spremenilo. V šestdesetih letih prejšnjega stoletja je avtomatizacija večine industrijskih procesov vsebovala analogne ožičene krmilnike, kar so jih kasneje zamenjali mikrokrmilniki in klasični programirljivi logični krmilniki (angl. Programmable Logic Controllers - PLC) [1]. Ti krmilniki se uporabljajo še danes, vendar se vse več industrije odloča za krmilnike, ki temeljijo na splošno namenskih računalnikih (npr. Acontis, Beckoff, Dewesoft, LabVIEW). Takšni krmilniki prinašajo večjo fleksibilnost in razširljivost, vendar po drugi strani zaradi kompleksnosti lahko privedejo tudi do manjše stabilnosti sistema. Z njim lahko zajamemo, vizualiziramo in analiziramo procesne podatke, izvajamo krmiljenja in regulacije, varujemo procese v konfliktnih situacijah in opozarjamo na kritične situacije.

Za reševanje problemov se uporabljajo različni programski jeziki in pristopi. Poznamo tako imenovane splošno namenske jezike (angl. General-Purpose Languages - GPL). Med te jezike prištevamo C, C++, Visual Basic, C# itd. S temi jeziki lahko razvijamo aplikacije iz različnih problemskih področji. Nasprotje pa predstavljajo t.i. domensko specifični jezi (angl. Domain-Specific Lanuages – DSL) [2]. Ti jeziki so se razvili predvsem zaradi velikih potreb po hitrem razvoju programov za specializirane problemske domene. V ospredje postavljajo domeno abstrakcije in semantiko, ki omogoča domenskim strokovnjakom programiranje direktno z domenskimi koncepti.

Zaradi visokega nivoja abstrakcije so domenski specifični jeziki zelo primerni za načrtovanje kompleksnih sistemov industrijske merilne avtomatizacije. Programski paket Dewesoft [3], ki se uporablja za zajem, obdelavo in kasneje tudi analizo izmerjenih meritev, uporablja domensko specifičen jezik Sekvencer za definiranje merilnih postopkov [4]. S tem domensko specifičnim jezikom lahko uporabnik na zelo enostaven način, brez strokovnega računalniškega znanja, izdela poljubno avtomatizacijo merilnega procesa, ki se vrši na industrijskem sistemu.

Članek je zgrajen iz 6 poglavij. V drugem poglavju predstavimo osnovno zgradbo industrijskega krmilnika. Sledi tretje poglavje kjer sta opisana dva DSL jezika, ki se uporabljata za programiranje avtomatizacije procesov. V četrtem poglavju je narejena primerjava med jezikoma Sekvencer in lestvičnim diagramom. Za boljše razumevanje je v 5 poglavju predstavljen praktični primer uporabe. Zadnje poglavje povzame vsebino članka.

2 Zgradba industrijskega krmilnika

Splošno namenski krmilniki za industrijske procese so običajno industrijski računalniški nadzorni sistemi, ki nenehno spremljajo stanje vhodnih modulov. Ti moduli so lahko digitalni vhodi, analogni vhodi, in komunikacijski ali posebni funkcijski moduli (npr. časovni števci). Na osnovi vhodnih stanj in programa napisanega s strani uporabnika, sprejema odločitve in upravlja z delovanjem izhodnih naprav, ki direktno komunicirajo in krmilijo izhodne periferije. Tovrstni krmilniki so tipičen primer realno-časovnih sistemov,

saj je zakasnitev v odzivnem času glede na spremembo vhodov manjša od določenega majhnega časa [5].

Krmilni program se v krmilniku izvaja znotraj krmilnega cikla (angl. scan cycle) in se odvija v štirih ponavljajočih se fazah: branje vhodov, izvajanje programske kode, pisanje na izhode in komunikacija ter diagnostika sistema. Tovrstno izvajanje predstavlja tudi nekatere nevarnosti. Lahko se zgodi, da pride do sprememb na vhodu signala v času, ki je manjši od odzivnega časa krmilnika. V tem primeru krmilnik teh sprememb ne zazna. V ta namen imajo različni krmilniki vgrajene dodatne mehanizme, ki zaznavajo spremembe tudi v tem času. Nekateri podpirajo tudi možnost delovanja s pomočjo prekinitev.

Vsak program je običajno zapisan v posebni aplikaciji namenjeni za programiranje in simulacijo na osebnem računalniku. Ko je program zaključen, se prenese na krmilnik kjer se izvaja. Program je shranjen v PLC-ju, v pomnilniku (RAM-u), ki se napaja iz baterije ali v drugem trajnem pomnilniku.

Nekateri krmilniki med delovanjem potrebujejo interakcijo z ljudmi kar opravijo s pomočjo vmesnika človek-stroj (angl. Human-Machine Interface - HMI). Z grafičnim vmesnikom lahko spreminjamo različne parametre krmilnika, spremljamo opozorilo o alarmih ali nadzorujemo sistem. Za nadzor na višjem nivoju in povezovanje različnih sistemov v celoto, se uporabljajo računalniški nadzorni sistemi (angl. Supervisory Control and Data Acquisition - SCADA). Nadzorni sistemi komunicirajo s krmilniki preko standardnih protokolov kot je na primer protokol OPC-UA [6].

3 Programiranje logičnih krmilnikov

Naloga avtomatizacije merilnih procesov je nadomestiti človeško odločanje z uporabo mehanske opreme in ukazov za logično programiranje. Za opisovanje logike delovanja krmilnika so se razvili številni DSL jeziki.

V nadaljevanju poglavja si bomo podrobneje pogledali dva vizualna programska jezika. Prvi je Sekvencer podjetja Dewesoft, ki ga primerjamo z enim izmed bolj uporabljenih programskih jezikov, lestvičnim diagramom (angl. Ladder Diagram - LD).

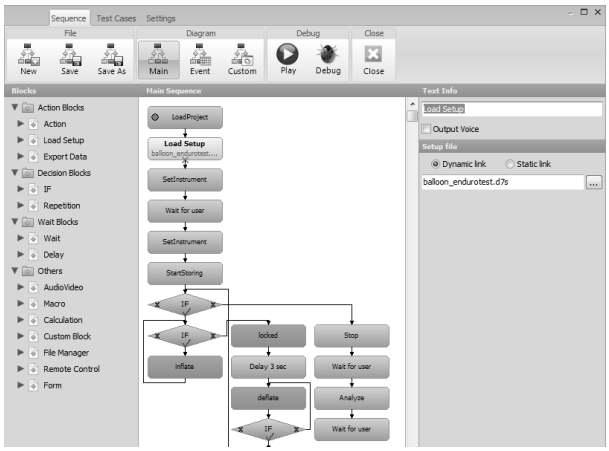
3.1 Dewesoft Sekvencer

Jezik Sekvencer je vključen v programski paket Dewesoft X3 (trenutna verzija X3 SP4). V osnovi je bil načrtovan v merilni industriji za kreiranje testnih postopkov med samim testiranjem. Danes pa se uporablja tudi v industrijski avtomatizaciji na najrazličnejših področjih (avtomobilska, letalska, elektro industrija itd.).

Krmiljenje najrazličnejših postopkov lahko kreiramo v vizualnem ali tekstovnem modelirnem okolju. Okolji sta prilagojeni za izgradnjo logike procesov v merilni domeni. Vizualne gradnike premikamo po metodi »primi in spusti« in jim določamo pripadajoče lastnosti.

Programski gradniki predstavljajo akcije, ki se izvajajo. Akcije se izvajajo od začetnega gradnika (označen s krogcem) in se nadaljuje na naslednjem gradniku kamor kaže puščica. Gradniki lahko vsebujejo tudi lokalne in globalne spremenljivke, ki so namenjeni za hranjenje trenutne vrednosti. Ker lahko ob velikem številu gradnikov vizualno programiranje postane neobvladljivo, modelirni jezik omogoča tudi vgnezdene modele (angl. custom blocks) s pomočjo katerih lahko združimo gradnike v celoto in se nanje sklicujemo v preostalem delu programa.

Slika 1 prikazuje razvojno orodje. Na levi strani najdemo gradnike domenskega jezika. Na sredini je delovna površina, kjer končni uporabniki programirajo proces. Na desni strani zaslona pa najdemo lastnosti, ki jih lahko uporabnik izbira za izbrani gradnik iz generiranega modela.



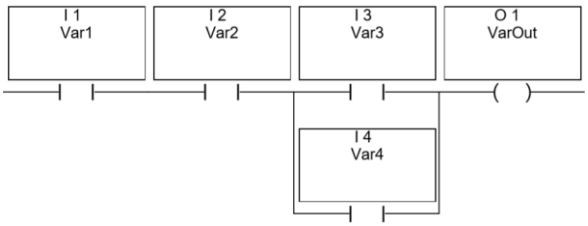
Slika 1: Primer programa zapisanega v jeziku Sekvencer

3.2 Standard IEC 61131-3: lestvični diagram

Standard IEC 61131-3 [7] govori o smernicah PLC programskih jezikov. Definira strukturo programov, podatkovne tipe in spremenljivke, standardne funkcije in funkcijske bloke, ter definira štiri osnovne programske jezike: seznam ukazov (angl. Instruction List - IL), strukturiran tekst (angl. Structured Text - ST), lestvični diagram (angl. Ladder Diagram - LD) ter funkcijski blokovni diagram (angl. Function Block Diagram - FBD). Vsak program je lahko zapisan v enem od omenjenih jezikov, možno pa jih je med seboj tudi kombinirati. Dodaten programski jezik, ki je namenjen strukturiranju opravil in programskih modulov je tako imenovani sekvenčni funkcijski diagram (angl. Sequential Function Chart - SFC).

Najstarejši in najbolj uporaben od osnovnih programskih jezikov je lestvični diagram. Jezik izhaja iz relejske tehnike, kjer so bili koncepti jezika razviti iz električnih shem in so namenjeni za procesiranje logičnih digitalnih signalov. Osnovni simboli, ki se uporabljajo so: normalno odprt kontakt, normalno zaprt kontakt in tuljava oz. izhodni kontakt. Izvajanje lestvičnega diagrama je iz leve proti desni in od zgornje do spodnje vrstice.

Slika 2 prikazuje enostaven program zapisan v lestvičnem diagramu. Normalno odprta kontakta Var1 in Var2 sta povezana zaporedno in predstavljata logično operacijo »IN«. Nato sledita elementa Var3 in Var4, ki sta povezana vzporedno in predstavljata logično operacijo »ALI«. Elementi od Var1 do Var4 so vhodni signali, kjer se izhodna vrednost zapiše v VarOut element ali izhodni kontakt. Spodnje grafične elemente bi lahko zapisali tudi v obliki OutVar = Var1 IN Var2 IN (Var3 ALI Var4).

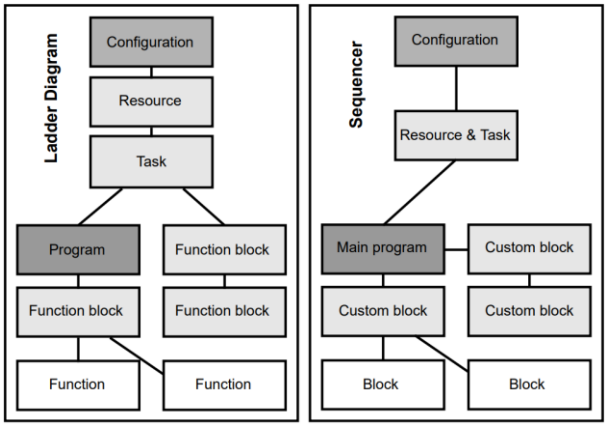


Slika 2: Primer programa zapisanega z lestvičnim diagramom

4 Primerjava Sekvencer jezika z lestvičnim diagramom

Jezika Sekvencer in lestvični diagram spadata v skupino domensko specifičnih vizualnih jezikov. Vendar sta si tako po zgradbi, kakor tudi po tipu programskih jezikov različna.

Struktura lestvičnega diagrama, ki je definirana po standardu, je zasnovana v več nivojih (Slika 3). Na najvišjem nivoju je konfiguracija (angl. configuration), ki določa kako in kje se koda izvaja. Nato sledi vir (angl. resource), ki je dejansko krmilnik oz. procesna enota. Znotraj vira so definirana opravila (angl. tasks). Opravilo je enota, ki vsebuje informacije o tem kako in kdaj naj se nek program izvede. Nato hierarhično sledi izvajalni program, ki je sestavljen iz programa (angl. program), funkcijskih blokov (angl. function block) in funkcij (angl. function). Tudi programski jezik Sekvencer ima podobno hierarhično strukturo. Na vrhu tako kot pri lestvičnem diagramu je konfiguracija sistema. Ker programski jezik Sekvencer ne podpira večopravilnosti in je vir vezan na trenutni sistem, sta naslednja nivoja enakovredna. Nato sledi glavni program (angl. main program), ki je kasneje razdeljen na vgnezdene (angl. custom blocks) in osnovne konstrukcijske bloke (angl. blocks).



Slika 3: Struktura programskih orodij

Programski jezik Sekvencer se izvaja koračno. Akcije se izvajajo sosledno in se prekinjajo glede na krmilni cikel. Programski jezik omogoča mehanizem dogodkov (angl. events), ki prinaša nekatere prednosti [8]. Lestvični diagram ne podpira dogodkov, izvaja se v krmilnem ciklu, ki v splošnem namenskem jeziku predstavlja neskončno zanko, ki se jo programerji skušamo izogniti zaradi slabega performančnega izkoristka.

Lestvični diagram se razlikuje od običajnega postopkovnega programiranja kot je Sekvencer na več načinov. Jezik je precej primitiven, vendar zahteva dobro znanje Boole-ove algebre. Večina uporabnikov ne uporablja podprogramov, zato se lestvični diagram uporablja skupaj s sekvenčnim funkcijskim diagramom, kar predstavlja dvojno poznavanje in učenje jezikov.

Prenosljivost programske kode obeh jezikov je med sistemi istega proizvajalca dobra. Problem se pojavi pri prenosu med različnimi proizvajalci, saj ni definiranega standardnega formata za izmenjavo programske kode.

Čeprav je lestvični diagram zasnovan na bitnih vrednostih, omogoča in podpira tudi druge podatkovne tipe kot so celoštevilčna števila in števila s plavajočo vejico. Podobno velja za programski jezik Sekvencer, ki poleg omenjenih tipov podpira tudi kompleksna števila in tekstovne nize. Oba jezika omogočata nekatere kompleksne tipe kot so digitalni in časovni števci. Spremenljivke omenjenih tipov pri obeh jezikih so lahko zunanje ali notranje, globalne ali lokalne, omogočajo pa tudi uporabo konstant. Posebnost jezika Sekvencer je v tem, da spremenljivke v osnovi ne hranijo samo trenutne vrednosti, ampak tudi zgodovino za določeno obdobje, ki se lahko kasneje uporabi pri analizi obnašanja sistema.

Ker je lestvični diagram standardiziran obstaja na trgu ogromno različnih razvojnih orodij. Prav tako obstajajo različni odprtokodni projekti kot je na primer OpenPLC, Beremiz. Razvojna orodja omogočajo različne funkcionalnosti in delujejo v večini samo za specifičen tip naprav določenega proizvajalca. Ker programski jezik Sekvencer ni standariziran, je v omenjenem jeziku mogoče programirati samo znotraj produkta Dewesoft in njihovih sistemov. Orodje omogoča tudi nekatere dodatne funkcije kot so na primer preoblikovanje programa (angl. refactoring) [9].

Splošno je znano, da napake, ki se pojavijo znotraj kontrolnega sistema, težko najti in odpraviti. Še posebej je problem lokalizirati napako v realno časovnih sistemih. V teh primerih uporabniki za odpravo problema uporabljajo dodaten sistem s pomočjo katerega zajemajo in shranijo vhodne ter izhodne podatke (signale) iz realnega sistema in jih kasneje analizirajo. V ta namen je bil razvit razhroščevalnik, ki pomaga odkriti napake, ki se pojavijo bodisi na strojni ali programski opremi (sekvenčnem programu). Pri tem orodju se lahko uporabijo različne tehnike, kot so koračno izvajanje, prekinitvene točke, animacija, pregled spremenljivk in pregled sledenja modela, s katerimi hitreje lokaliziramo in odpravimo napake.

Pri samem razvoju programov za avtomatizacijo je prav tako zelo pomembno testiranje zgrajenih programov. V ta namen se največkrat uporabljajo

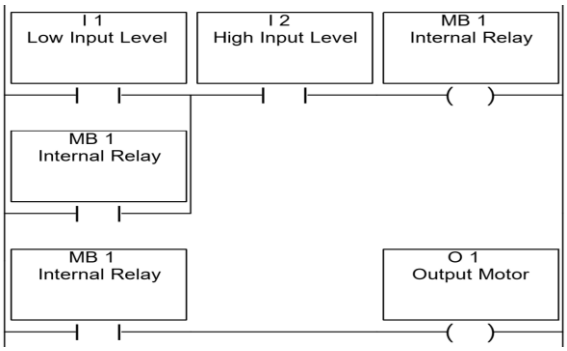
različne simulacije, ki pripomorejo k pravilnem delovanju. Pri orodju Sekvencer je bilo razvito orodje, ki omogoča kreiranje testnih primerkov tako za strojno kakor tudi programsko opremo.

Takšni vizualni jeziki imajo tudi nekatere slabosti. Jezike je mogoče uporabljati le na grafičnih operacijskih sistemih. Pri kreiranju programa zahtevajo več pomnilnika, večji prostor za shranjevanje in hitrejši procesor v primerjavi s tekstovnimi jeziki. Ena od slabosti je tudi razvoj takšnega jezika, ki je težak, saj je potrebno dobro poznavanje domene kot tudi znanje iz gradnje programskih jezikov. Zaradi omenjenih slabosti je cena načrtovanja, implementacije in vzdrževanja višja v primerjavi s splošno namenskimi jeziki.

5 Praktičen primer uporabe

Uporabo omenjenih jezikov bomo prikazali na primeru regulacije nivoja olja v rezervoarju. Za detekcijo nivoja olja bomo uporabili dva senzorja. Prvi senzor (High Input Value) bo nameščen na vrhu in bo določal visoki nivo, drugi senzor (Low Input Value) pa na dnu rezervoarja in bo določal nizki nivo olja. Senzorja sta aktivna (logična »1«), ko je tekočina pod nivojem. Dokler senzor ne doseže določenega praga, črpalka (Output Motor) polni olje v posodo. Ko se bo vklopil senzor visokega nivoja, bomo izklopili motor in počakali dokler nivo olja ne pade pod senzor nizkega nivoja. Ko pade pod ta nivo, ponovno vklopimo motor in ponavljamo postopek.

Slika 4 prikazuje program zapisan z lestvičnim diagramom. Na začetku je tank prazen (I1 in I2 sta aktivna), zato je interna spremenljivka (MB1) aktivna in motor prižgan (O1). Ko doseže olje nizek nivo, postane neaktivna L1 in pri visokem nivoju še L2. Po tem dogodku postane neaktivna MB1 in posledično se motor ustavi.

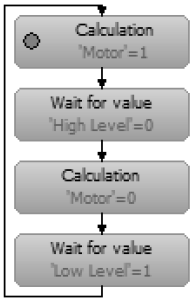


Slika 4: Program zapisan z lestvičnem diagramom

Na Sliki 5 je predstavljen algoritem zapisan z jezikom Sekvencer. Kontrola tanka je zgrajena iz štirih blokov. Najprej se vklopi motor za polnjene tanka (začeten blok je označen s krogcem). Nato se počaka, da doseže visoki nivo ('High Level'=0). Po tem dogodku ustavimo motor in ponovno počakamo, da se tank sprazni do nizkega nivoja ('Low Level'=0).

6 Zaključek

Industrijska avtomatizacija prihaja vse bolj in bolj v ospredje in postaja vse bolj pomembna zaradi velikih prednosti kot so povečanje produktivnosti, kvalitete in zmanjšanje cene. Zato je kratek čas izdelave produkta zelo pomemben. Pri tem moramo izbrati pravi jezik, ki je visokonivojski, enostaven in razumljiv. V članku sta predstavljena domensko specifična jezika lestvični diagram in Sekvencer. Prvi je zaradi zgodovinskih lastnosti močno uporabljen, standariziran in primeren za programiranje krmilnih sistemov v realnem času (angl. hard real-time systems). Drugi jezik Sekvencer je zaradi njegove enostavnosti namenjen za avtomatizacijo, vendar zaradi arhitekturne zasnove uporabljen za mehke sisteme v realnem času (angl. soft real-time systems), kjer majhni odziven čas ni nujno potreben.



Slika 5: Program zapisan v jeziku Sekvencer

Literatura

[1] G. Dunning. Introduction to Programmable Logic Controllers, 3rd Edition. Delmar Thomson Learning, 2006.

[2] M. Mernik, J. Heering, A. Sloane. When and how to develop domain-specific language. ACM computer surveys, 2005, štev. 37, zv. 4, str. 316-344.

[3] Dewesoft, https://www.dewesoft.com

[4] T. Kos, T. Kosar, M. Mernik, Development of data acquisition systems by using a domain-specific modeling language, Computers in Industry, 2012, štev. 63, str. 181-192.

[5] H. Kopetz, Real-Time Systems Design Principles for Distributed Embedded Applications, 2nd Edition, U.K., London:Kluwer, 2011.

[6] W. Mahnke, S. Leitner, M. Damm, OPC Unified Architecture, 2nd Edition, Springer, 2011.

[7] K.-H. John, M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, 2nd Edition, Springer, 2011.

[8] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazières, and R. Morris., Event-driven programming for robust software. In Proceedings of the 10th workshop on ACM SIGOPS European workshop (EW 10). ACM, NY, 2002, USA, str. 186-189.

[9] T. Kos, T. Kosar, M. Mernik, Princip preoblikovanja domensko specifičnih modelov s programskim paketom DEWESoft, Zbornik dvajsete mednarodne konference Elektrotehniške in računalniške konference, 2011, štev. 20, str. 119-122.