Računalniško podprto načrtovanje programske opreme za postopkovno vodenje

Gregor Kandare
Institut Jožef Stefan
Jamova 39, 1000 Ljubljana
gregor.kandare@ijs.si

Automated development of control software

In the paper a computer-aided approach to design of control software is presented. A formal model of a domain-specific modelling language syntax is given. The modelling language is specialised for the procedural process control domain. Furthermore, a software modelling tool that enables model editing and automatic code and documentation generation is described. The tool's code generator makes use of the formal model of the modelling language.

1 Uvod

Programska oprema za procesno vodenje spada med najzahtevnejše tipe programske opreme. Sistemi za vodenje se uporabljajo v procesih, strojih in napravah. Če programska oprema za vodenje ni kakovostna in zanesljiva, lahko ti sistemi uidejo nadzoru in povzročijo materialno škodo ali celo ekološko katastrofo. Zaradi tega so najpomembnejše lastnosti programske opreme za vodenje zanesljivost, varnost, in odzivnost [1].

Eden od problemov programske opreme za vodenje je tudi njena kompleksnost, ki se z vedno večjimi in bolj zahtevnimi sistemi za vodenje še povečuje. Za obvladovanje kompleksnosti je potreben sistematičen inženirski pristop k razvoju. S takšnim pristopom se ukvarja programsko inženirstvo ([2]-[5]).

V literaturi najdemo poročila o uporabi metod programskega inženirstva pri razvoju programske opreme za vodenje ([6], [7]). Opazno pa je pomanjkanje namenskih programskih orodij (CASE), ki omogočajo avtomatizacijo procesa razvoja programske opreme za vodenje.

V naslednji sekciji članka je predstavljen domensko specifični modelirni jezik ProcGraph. Nadalje je podan opis postopka za preslikavo modelov v omenjenem jeziku v programsko kodo programirljivih krmilnikov (PLK). Sledi prikaz programskega orodja, ki omogoča modeliranje ter iz modelov avtomatsko generira programsko kodo in dokumentacijo. V zadnji sekciji je ilustrirana uporaba programskega orodja na konkretnem industrijskem problemu.

2 Modelirni jezik ProcGraph

ProcGraph je domensko specifičen modelirni jezik, ki je zasnovan za uporabo v načrtovanjeu programske opreme za postopkovno vodenje ([8], [9]). Modelirni jezik uporablja enake abstrakcije za opis sistema za vodenje, kot jih uporabljajo procesni inženirji za opis vodenega procesa. Abstrakcije in njihova hierarhija se ohranjajo skozi vse faze razvoja programske opreme. Prednost tovrstnega pristopa je lažja komunikacija med akterji v razvojnem procesu (procesni inženirji, analitiki/načrtovalci, programerji). Z ohranitvijo abstrakcij je prehod med fazami razvoja lažji (progresivni in regresivni), kar omogoča lažje vzdrževanje konsistentnosti modelov posameznih faz.

ProcGraph obsega naslednje tri tipe diagramov:

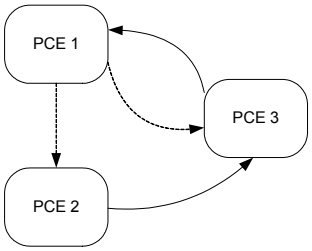
- *Entitetni diagram* upodablja sočasno in hkrati konceptualno dekompozicijo

sistema kot tudi povezave med konceptualnimi komponentami.

- *Diagram prehajanja stanj* opisuje dinamični vidik (obnašanje) konceptualnih komponent.
- *Diagram odvisnosti entitet* prikazuje vzročne in posledične odnose med konceptualnimi komponentami.

2.1 Entitetni diagram

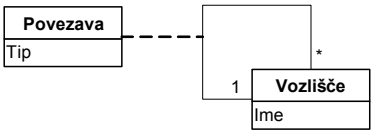
Ta diagram upodablja hkrati sočasne postopke in arhitekturno dekompozicijo sistema za vodenje. Vozlišča v entitetnem diagramu predstavljajo konceptualne komponente – postopkovne entitete, povezave pa ponazarjajo odnose (odvisnosti) med postopkovnimi entitetami. Primer entitetnega diagrama je prikazan na sliki 1.



Slika 1: Primer entitetnega diagrama

Povezava s polno črto ponazarja, da je eden od prehodov med dvema stanjema entitete, ki je ponor povezave, odvisen od nekega stanja entitete, iz katere povezava izvira, povezava s črtkano črto pa označuje, da vstop entitete izvora v določeno stanje povzroči prehod entitete ponora v novo stanje.

Sintaktični model entitetnih diagramov (<ED>) lahko prikažemo s sliko 2.



Slika 2: Sintaksa entitetnih diagramov

Iz sintakse vidimo, da je lahko vsako vozlišče povezano z enim ali več vozlišči ter da ima vsaka povezava svoj tip (polna ali črtkana).

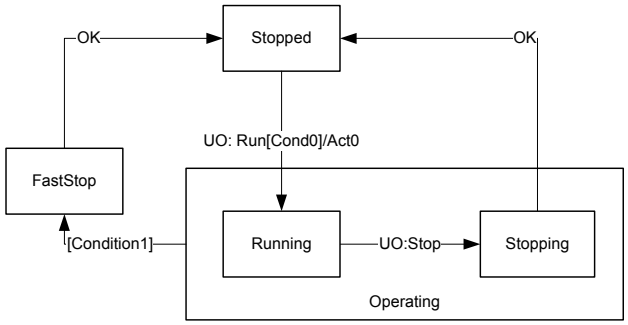
Drug način opisa sintakse entitetnih diagramov je s šesterico

$$<PCED> = \{V_E, P_E, \rho, iz, po, tp\} \quad (1)$$

kjer je V_E množica vozlišč, P_E množica povezav, ρ funkcija, ki preslika vozlišča v množico diagramov prehajanja stanj. Funkcije *iz*, *po* in *tp* določajo začetno, končno vozlišče ter tip povezave.

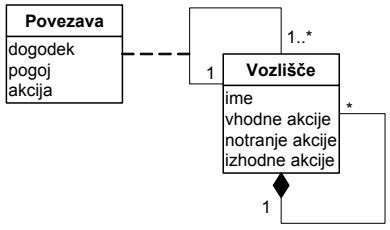
2.2 Diagram prehajanja stanj

Sistemi za postopkovno vodenje morajo reagirati na zunanje signale v realnem času zato jih prištevamo med reaktivne sisteme. Najprimernejši model za opis tovrstnih sistemov je diagram prehajanja stanj [10]. Slika 3 prikazuje primer diagrama prehajanja stanj, ki opisuje dinamiko postopkovne entitete PCE 3 iz slike 1.



Slika 3: Primer diagrama prehajanja stanj

Sintakso diagramov prehajanja stanj lahko opišemo z razrednim diagramom na sliki 4.



Slika 4: Sintaksa diagramov prehajanja stanj

Vsako vozlišče, ki predstavlja stanje je lahko povezano z enim ali več ostalimi vozlišči s povezavami, ki ponazarjajo prehode med stanji. Vsaka povezava ima še tri atribute, ki opisujejo dogodek ki prehod aktivira, pogoj za izvedbo ter akcijo, ki se ob prehodu izvede. Vozlišča lahko vsebujejo podvozlišča.

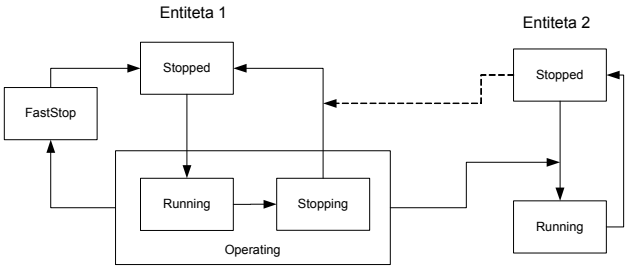
Drug način za opis sintakse diagramov prehajanja stanj je z deveterico

$$\langle STD \rangle = \{V_S, P_S, iz, po, nad, I, P, O, \lambda\}, \quad (2)$$

kjer V_S predstavlja množico vozlišč, ki predstavljajo stanja, P_S množico povezav (prehodov), *iz* in *po* funkciji, ki vrneta izvorno oziroma končno vozlišče povezave. Funkcija *nad* vrne vsebujoče vozlišče. *I* je množica opisov vhodnih signalov, *P* množica opisov parametrov, *O* množica opisov izhodnih signalov ter λ funkcija, ki vsaki povezavi iz množice P_S priredi logičen izraz, v katerem so vsebovani vhodni in izhodni signali kot tudi akcije, ki se izvršijo ob proženju prehoda.

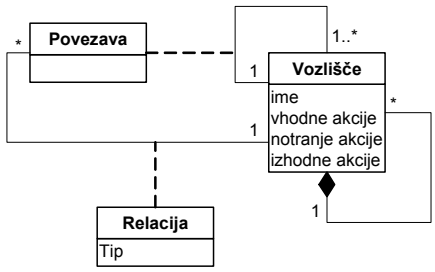
2.3 Diagram odvisnosti entitet

Diagrami odvisnosti entitet vsebujejo diagrame prehajanja stanj dveh ali več postopkovnih entitet skupaj z odvisnostmi med diagrami prehajanja stanj. Medtem ko entitetni diagram prikazuje zgolj obstoj vzročnih in posledičnih odnosov med entitetami, diagram odvisnosti entitet točno specificira, katera stanja ene entitete povzročijo oziroma pogojujejo prehod v drugi entiteti. Primer diagrama odvisnosti entite prikazuje slika 5.



Slika 5: Primer diagrama odvistnosti entitet

Sintakso diagramov odvisnosti entitet prikazuje slika 6.



Slika 6: Sintaksa diagrama odvistnosti entitet

Razlika od sintakse diagrama prehajanja stanj je v tem, da poleg povezav med vozlišči obstajajo tudi povezave, k izvirajo v vozliščih in se končajo na povezavah.

Sintakso diagrama odvisnosti entitet določa tudi izraz

$$\langle EDD \rangle = \{V_S, P_S, Q, iz, po, izp, pop, nad\}. \quad (3)$$

V primerjavi z izrazom (2), izraz (3) vsebuje množico pogojnih povezav (Q) in funkciji *izp* ter *pop*, ki vrneta izvorno vozlišče oziroma končno povezavo. Po drugi strani pa diagram odvisnosti entitet ne vsebuje informacij v zvezi z prehodi, ker so te že vsebovane v diagramu prehajanja stanj.

3 Preslikava modelov ProcGraph v programsko kodo krmilnikov

V razvojnem procesu programske opreme fazi modeliranja sledi faza izvedbe (programiranja). Kot smo že omenili, je dobro uporabljati skozi vse razvojne faze čimbolj podobne abstrakcije. Ker implementacijsko platformo sistema za vodenje predstavljajo programirljivi krmilniki mora biti izvedbeni programski jezik eden od jezikov standarda IEC 61131-3. Najprimernejši je funkcijski bločni diagram (FBD), saj omogoča direktno preslikavo abstrakcij in njihove hierarhije iz modelirnega jezika. Glavni elementi FBD so namreč funkcijski bloki, v katere lahko preslikamo postopkovne entitete in stanja. Nadalje jezik FBD omogoča hierarhično dekompozicijo programske kode (znotraj funkcijskih blokov lahko kličemo druge funkcijske bloke in tako zgradimo enako hierarhijo kot v modelu).

Da bi lahko izvedli preslikavo modelov v programsko kodo, moramo izdelati natančna pravila preslikave, ki bazirajo na sintaksi in semantiki modelirnega (ProcGraph) in programskega (FBD) jezika.

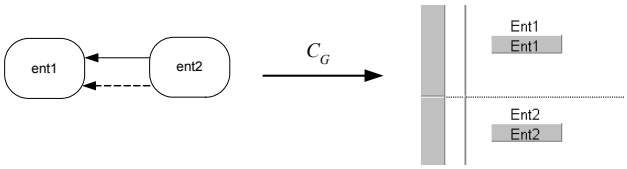
Modelirni jezik ProcGraph sestoji iz treh tipov diagramov, kar lahko zapišemo kot

$$<ProcGraph> = <PCED> + <STD> + <EDD>. \quad (4)$$

Preslikavo ProcGraph modelov v programsko kodo funkcijskega bločnega diagrama definiramo kot

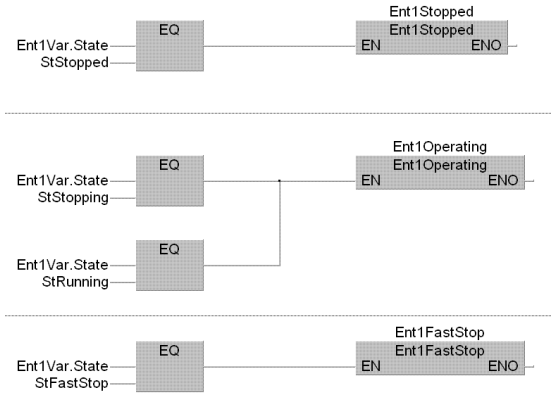
$$C_G: <ProcGraph> \rightarrow <FBD>. \quad (5)$$

Vsako vozlišče (postopkovna entiteta) entitetnega diagrama se preslika v pripadajoč funkcijski blok (slika 7).



Slika 7: Preslikava postopkovnih entitet

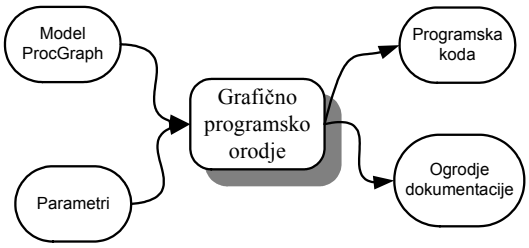
Dinamika vsake postopkovne entiete je opisana z diagramom prehajanja stanj. Zato funkcijski blok, ki predstavlja postopkovno entiteto, vsebuje algoritem, ki izvaja njen diagram prehajanja stanj. Tudi ta algoritem je izveden v funkcijskem bločnem diagramu – funkcijskih bloki predstavljajo stanja. Mehanizem prehajanja stanj je implementiran znotraj funkcijskih blokov stanj, kjer se kličejo tudi funkcijski bloki, ki predstavljajo podstanja. Primer implementacije diagrama prehajanja stanj s slike 3 je prikazan na sliki 8.



Slika 8: Preslikava diagrama prehajanja stanj

4 Modelirno programsko orodje

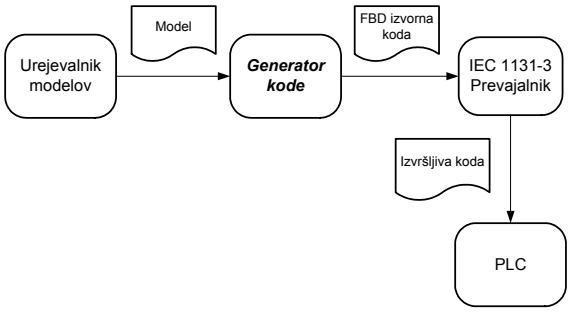
Za avtomatizacijo procesa razvoja programske opreme za vodenje smo razvili programsko orodje, ki vsebuje grafično/tekstovni urejevalnik modelov ProcGraph iz katerih avtomatsko generira FBD programsko kodo in dokumentacijo v obliki dokumenta MS Word. Slika 9 shematsko prikazuje vhode in izhode programskega orodja.



Slika 9: Vhodi in izhodi programskega orodja

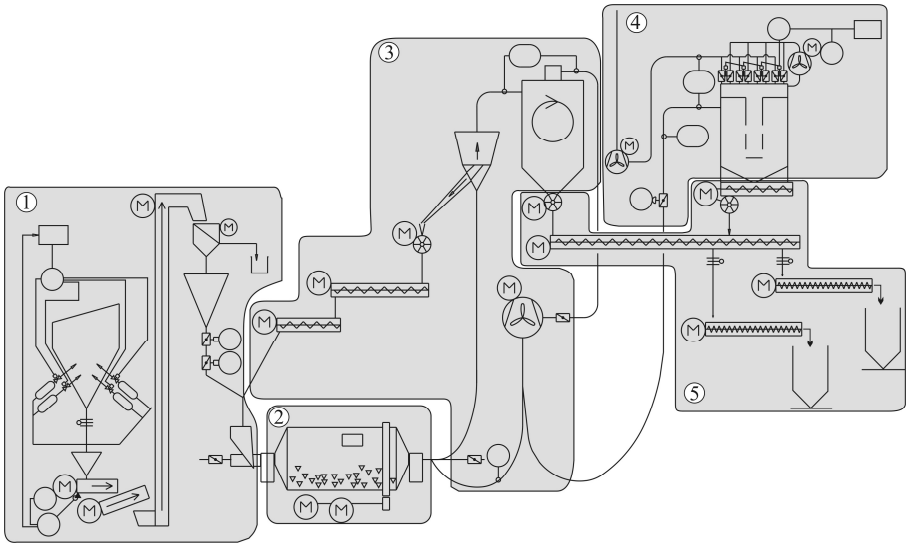
Ena najpomembnejših lastnosti orodja je avtomatsko generiranje FBD programske kode. Funkcijski bločni diagram je grafičen programski jezik. Generiranje kode v grafičnih jezikih je bolj zahtevno kot v tekstovnih jezikih kot sta C ali Java. Pri grafičnih jezikih je namreč potrebno generirati ne samo pravilno vsebino, temveč tudi obliko programske kode. Tipičen FBD program sestoji iz funkcijskih blokov in povezav med njimi. Pri generiranju kode je pomemben tako položaj blokov v shemi kot tudi potek povezav med bloki.

Slika 10 prikazuje proces generiranja kode. Najprej uporabnik v grafičnem urejevalniku ustvari ProcGraph model. Ko je urejanje modela končano in model vsebuje vse potrebne informacije, uporabnik aktivira generiranje programske kode. Rezultat je tekstovni dokument, ki opisuje FBD kodo. Ta dokument je potrebno uvoziti v programsko orodje za programiranje krmilnikov ter ga prevesti v izvršljivo kodo.



Slika 10: Proces avtomatskega generiranja kode

V primerjavi z ročnim kodiranjem je avtomatsko generiranje kode veliko hitrejše in manj podvrženo napakam.



Slika 11: Proces mletja

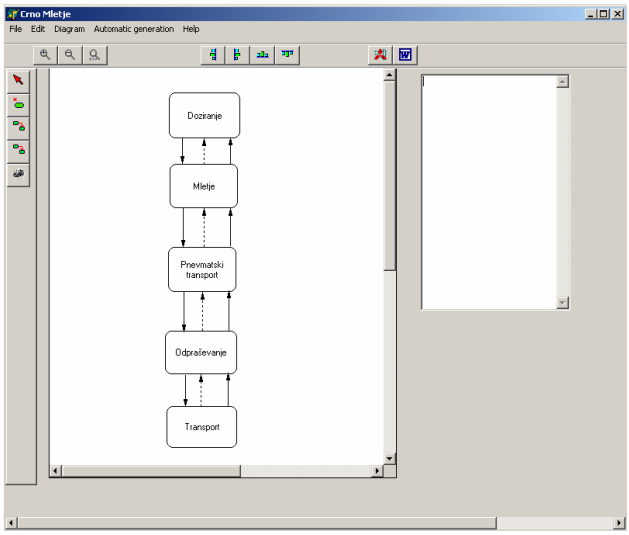
5 Primer uporabe modelirnega programskega orodja

Za prikaz uporabe modelirnega orodja uporabimo sistem za vodenje mletja rud, ki je shematsko prikazan na sliki 11.

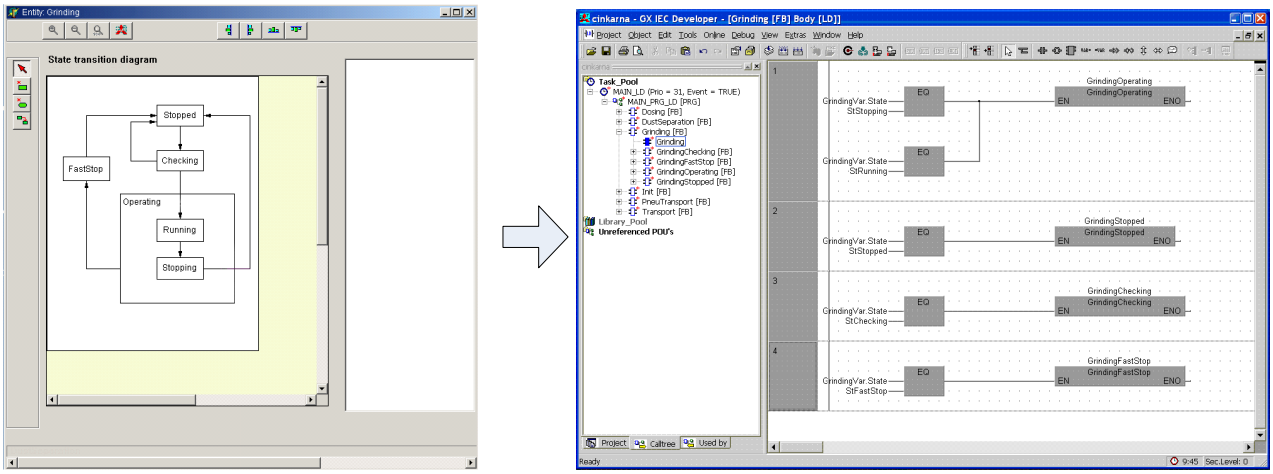
V skladu z principi problemsko orientirane dekompozicije izberemo konceptualne entitete sistema za vodenje tako, da odražajo dekompozicijo procesa, ki ga vodimo.

Slika 12 prikazuje posnetek ekrana glavnega okna modelirnega orodja, v katerem je urejevalnik entitetnih diagramov. V njem vidimo pet postopkovnih entite, ki ustrezajo petim podprocesom procesa mletja. Če uporabnik klikne na katero od entitet, se mu odpre novo okno z urejevalnikom diagramov prehajanja stanj.

Ko uporabnik zaključi z urejanjemo modela, lahko aktivira avtomatsko generiranje kode. Rezultat prikazuje slika 13. Na levi strani slike vidimo urejevalnik diagramov prehajanja stanj modelirnega orodja, ki vsebuje diagram prehajanja stanj ene od postopkovnih entitet. Iz diagrama prehajanja stanj se generira del FBD kode ki ga vidimo na desni polovici slike 13. Na primeru vidimo, da se vsako vsaki postopkovni entiteti pripada svoj funkcijski blok. Tudi stanja se preslikajo v funkcijske bloke in sicer tako, da je hierarhija blokov enaka hierarhiji stanj v modelu. Na ta način v programski kodi nastopajo enake abstrakcije in njihova hierarhija kot jih uporabljajo procesni inženirji za opis vodenega procesa.



Slika 12: Urejevalnik entitetnih diagramov



Slika 13: Urejevalnik diagramov prehajanja stanj in generirana koda

6 Povzetek

Razvoj programske opreme za vodenje postaja vedno bolj kompliciran zaradi kompleksnosti vodenih procesov in opreme ter zaradi nizkega nivoja programskih jezikov za krmilnike. Proces programiranja (kodiranja) zahteva veliko časa, poleg tega pa je zelo podvržen napakam. V članku je pokazano, da je mogoče natančno določati pravila za preslikavo programskih modelov v programsko kodo in da je mogoče to preslikavo avtomatizirati. To lahko implementiramo z domensko specifičnim generatorjem programske kode. V procesu generiranja generator uporablja vzorce kode, ki jih napolni z ustrezno vsebino. Avtomatsko generiranje kode bistveno pohitri razvoj programske opreme hkrati pa izboljša njeno kvaliteto.

7 Literatura

[1] Frey, G. and L. Litz (2000). Formal methods in PLC programming, *Proc. IEEE Conference on Systems Man and Cybernetics SMC 2000*, Nashville.

[2] DeMarco, T. (1978). *Structured Analysis and System Specification.* Prentice Hall, Englewood Cliffs.

[3] Wieringa, R. (1998). A Survey of Structured and Object-Oriented Software Specification Methods and Techniques, *ACM Computing Surveys*, **30**(4), 459-527.

[4] Booch G., J. Rumbaugh and I. Jacobson (1999). *The Unified Modeling Language User Guide*, Addison Wesley, Boston.

[5] Ludewig, J. (2003). Models in Software Engineering – an Introduction. *Software and Systems Modeling*, **2**(1), 5-14.

[6] Edan, Y. and N. Pliskin (2001). Transfer of Software engineering Tools from Information Systems to Production Systems. *Computers & Industrial Engineering*, **39**(1), 19-34.

[7] Davidson, C. M., J. McWhinnie and M. Mannion (1998). Introducing Object Oriented Methods to PLC Software Design, *Proc. International Conference and Workshop: Engineering of Computer-Based Systems (ECBS '98),* Jerusalem.

[8] Kandare, G. (2004). *Computer aided design of procedural control software for programmable logic controllers.* PhD Thesis, University of Ljubljana.

[9] Godena, G. (2004). ProcGraph: a procedure-oriented graphical notation for process-control software specification. *Control Engineering Practice*, **12**(1), 99-111.

[10] Harel, D. (1987). Statecharts: A Visual Formalism for Complex Charts. *Science of Computer Programming*, **8**(3), 231-274.