MODELI PROGRAMSKE OPREME ZA VODENJE ŠARŽNIH PROCESOV

Giovanni Godena

Doktorska disertacija
Mednarodna podiplomska šola Jožefa Stefana
Ljubljana, Slovenija, maj 2016

Komisija za oceno doktorske disertacije:

Prof. dr. Đani Juričić, predsednik, Institut »Jožef Stefan«, Jamova 39, 1000 Ljubljana

Prof. dr. Gašper Mušič, član, Fakulteta za elektrotehniko, Tržaška cesta 25, 1000 Ljubljana

Prof. dr. Juš Kocijan, član, Univerza v Novi Gorici, Vipavska 13, 5000 Nova Gorica

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL



Giovanni Godena

MODELI PROGRAMSKE OPREME ZA VODENJE ŠARŽNIH PROCESOV

Doktorska disertacija

MODELS OF BATCH PROCESS CONTROL SOFTWARE

Doctoral Dissertation

Mentor: Prof. dr. Stanko Strmčnik

Ljubljana, Slovenija, maj 2016

Kazalo

Povzetek ..... **IX**

Abstract ..... **XI**

Seznam kratic ..... **XIII**

1 Uvod **1**

1.1 Namen disertacije 1

1.2 Cilji disertacije 3

1.3 Hipoteza 3

1.4 Utemeljitev znanstvene relevantnosti 3

1.5 Metodologija 3

1.6 Okvir za preizkušanje rezultatov disertacije 4

1.7 Struktura disertacije 4

2 Šaržni procesi in njihova programska oprema **7**

2.1 Opis šaržnih procesov 7

2.2 Programska oprema za vodenje šaržnih procesov 8

2.3 Standardi, modeli in orodja za vodenje šaržnih procesov 8

2.4 Opis standarda ISA 88.00.01 9

2.4.1 Vrste vodenja 9

2.4.2 Modeli in terminologija standarda ISA 88.00.01 9

2.4.2.1 Procesni model 10

2.4.2.2 Fizični model 10

2.4.2.3 Model postopkovnega vodenja 11

2.4.2.4 Relacije med modeli 12

2.4.3 Recepti 13

2.4.4 Razvojni proces šaržnega vodenja 14

2.5 Težave pri uporabi standarda ISA 88.00.01 15

3 Abstrakcija receptov **19**

3.1 Analiza abstrakcije receptov 19

3.2 Nova abstrakcija receptov 21

3.2.1 Preslikava SFC zapisa recepta v tabelo ... 22
3.2.2 Algoritem izvajanja tabelarnega recepta ... 24
3.2.3 Omejitve strukture receptov ... 26
3.2.4 Mehanizmi upravljanja toka izvajanja tabelarnih receptov ... 28
3.2.4.1 Sistemska faza »Pogoj« ... 28
3.2.4.2 Sistemska faza »Razvejitev« ... 29
3.2.4.3 Sistemska faza »Skok« ... 29
3.2.4.4 Sistemska faza Sporočilo ... 29
3.2.4.5 Sistemska faza »Vnos operaterja« ... 30
3.3 Preizkušanje nove abstrakcije receptov in zaključki ... 30

4 Model obnašanja faz ... 33

4.1 Analiza aktualnega modela obnašanja faz ... 33
4.1.1 Abstrakcija avtomatov stanj in nekateri problemi ... 33
4.1.2 Abstrakcija avtomatov stanj pri vodenju šaržnih procesov ... 35
4.2 Nov model obnašanja faz ... 44
4.2.1 Obnašanje faz ... 44
4.2.2 Vrste stanj in prehodov ... 45
4.2.3 Kompleksni prehodi ... 46
4.2.4 Sekvence fazne logike ... 48
4.2.5 Nadaljnja redukcija modela obnašanja faz ... 51
4.2.6 Primer modeliranja obnašanja faze ... 51
4.2.7 Zaključki ... 54

5 Objektni model opreme in postopkovnega vodenja ... 55

5.1 Analiza aktualnega objektnega modela opreme in postopkovnega vodenja ... 55
5.1.1 Uvod ... 55
5.1.2 Modeli vodenja šaržnih procesov ... 56
5.1.3 Objektna tehnologija pri vodenju šaržnih procesov ... 62
5.1.4 Standardni objektni model opreme in postopkovnega vodenja ... 64
5.1.4.1 Struktura standardnega objektnega modela ... 64
5.1.4.2 Pomanjkljivosti standardnega objektnega modela ... 65
5.2 Nov objektni model opreme in postopkovnega vodenja ... 66
5.3 Uporaba novega koncepta ... 70
5.3.1 Opis študije primera novega koncepta ... 70
5.3.2 Ugotovitve preizkušanja novega koncepta ... 78
5.4 Vrednotenje novega koncepta ... 78

6 Zaključki ... 81

7 Zahvale 83

8 Literatura in viri 85

Kazalo slik 91

Kazalo tabel 93

Priloga A: Seznam lastnih publikacij 95

Povzetek

Ob nedvomnih koristih, ki jih je v sisteme vodenja šaržnih procesov prinesel standard ISA-88 in na njem temelječa orodja, obstajajo v zvezi z njimi tudi nekatere težave, med katerimi se v disertaciji osredotočamo na naslednje tri. Prva je v recepturnem sistemu, ki je zaradi kompleksne abstrakcije receptov dokaj zahteven za operaterje. Druga je v nezadostnem obvladovanju kompleksnosti razvoja šaržnih postopkov zaradi preveč enostavnega modela obnašanja teh postopkov. Tretja težava, s katero se ukvarjamo, je prevelika stopnja podvajanja informacij v receptih in njihova prenizka ponovna uporaba. Vzrok te težave je v prenizkem nivoju abstrakcije objektnega modela opreme in postopkovnega vodenja.

V disertaciji izhajamo iz hipoteze, da je mogoče zasnovati nove modele programske opreme za šaržno vodenje, ki bodo olajšali delo operaterjem, poenostavili implementacijo postopkov vodenja ter zmanjšali podvajanje informacij v receptih in povečali njihovo ponovno uporabo.

V delu najprej podamo uvod v šaržne procese in programsko opremo za njihovo vodenje. Pojasnimo motivacijo za nastanek ter podamo pregled standarda ISA-88, predvsem njegovih treh modelov – procesnega modela, fizičnega modela in modela postopkovnega vodenja – ter predstavimo relacije med temi modeli. V nadaljevanju opišemo težave pri uporabi standarda ISA88, od katerih bomo tri, ki so po naši presoji najbolj pereče, poskusili rešiti v nadaljevanju dela.

V jedru dela se najprej ukvarjamo z abstrakcijo receptov. Začnemo z analizo obstoječe abstrakcije, kot jo določa standard ISA-88 in je implementirana v orodjih na osnovi standarda, ter izpostavimo težave pri njeni implementaciji in uporabi. Nato predstavimo naš predlog nove abstrakcije receptov, ki temelji na tabelarni predstavitvi. Novo abstrakcijo nekoliko bolj podrobno predstavimo, in sicer najprej način preslikave klasičnih receptov v tabelarni zapis, nato algoritem izvajanja tabelarnega recepta, omejitve tabelarne abstrakcije receptov v primerjavi s klasično abstrakcijo receptov ter na koncu mehanizme upravljanja toka izvajanja tabelarnih receptov, ki izhajajo iz same tabelarne strukture receptov.

Naslednje področje, ki ga v disertaciji obravnavamo, je model obnašanja elementarnih entitet postopkovnega vodenja. Najprej se posvetimo pregledu abstrakcij na osnovi avtomatov stanj. Nato analiziramo aktualni model obnašanja faz, kot ga opredeljuje standard ISA 88 in kot je implementiran in dokumentiran v prototipnem orodju za vodenje šaržnih procesov. Pri analizi ugotovimo pomanjkljivosti modela, in sicer tako v predstavitvi modela v standardu in v dokumentaciji orodja kot tudi v sami strukturi modela. V

nadaljevanju predstavimo predlog novega modela obnašanja faz, ki omogoča odpravljanje ugotovljenih pomanjkljivosti, ter nov model tudi opišemo. Na koncu podamo še ilustracijo modeliranja obnašanja faz z novim modelom na primeru iz realnega industrijskega projekta.

Zadnje področje, ki ga v disertaciji obravnavamo, je objektni model opreme in postopkovnega vodenja. Najprej ugotovimo, da model, ki ga določa standard ISA 88 in je implementiran v orodjih za recepturno vodenje šaržnih procesov, ni uspešen pri izogibanju podvajanja informacij v receptih in zagotavljanju visoke stopnje njihove ponovne uporabe. Potem predstavimo idejo, da je možna rešitev teh problemov v uvedbi bolj sofisticiranega objektnega modela, ki temelji na konceptu večkratnega dedovanja oziroma konkretno na prekrivajočih se razredih opreme. V nadaljevanju podamo podroben opis novega objektnega modela opreme in postopkovnega vodenja skupaj z njegovo implementacijo v lastnem, doma razvitem orodju za vodenje šaržnih procesov. Uporabo novega koncepta ilustriramo na primeru realnega industrijskega projekta vodenja procesa sinteze smol. Na koncu predstavimo še rezultat vrednotenja novega koncepta na omenjenem primeru, ki je pokazal, da nov pristop zagotavlja pomembno izboljšanje na področju izogibanja podvajanja informacij in povečanja stopnje ponovne uporabe receptov.

Abstract

Beside the undoubted benefits brought into the batch-process control systems by the introduction of the standard ISA-88 and the tools based on it, there are also some problems concerning the standard and the related tools. In the dissertation, we focus on the following three problems. The first problem comes from the recipe system, which is rather demanding for the operators due to its complex abstraction. The second problem lies in the unsatisfactory management of the complexity of batch procedures development due to the too simple behaviour model of these procedures. The third problem in our concern is the too high level of repetition of information in the recipes and in their too low reusability. This problem stems from the too low abstraction level of equipment and procedural control object model.

In the dissertation, we proceed from the hypothesis that it is possible to concieve new models of the batch-process control software, which will alleviate the operators' work, simplify the implementation of the control procedures, and reduce the repetition of information in the recipes and increase their reuse.

In the work, we first introduce batch processes and their control software. We explain the motivation for introducing the ISA-88 standard and present its overview, with the emphasis on its three models, namely the process model, the physical model, and the procedural control model, as well as the relations between them. In the following we describe the difficulties in using the ISA-88 standard and point out three of them that we consider the most troubling and that we try to resolve in the dissertation.

The first topic we deal with in the core part of the dissertation is the recipes abstraction. We begin with the analysis of the actual abstraction as defined by the ISA-88 standard and implemented in tools based on the standard. In the analysis, we point out some difficulties in the implementation and the use of this abstraction. Then we present our proposition for a new recipe abstraction based on a tabular notation of the recipes. In the sequel, we give a more detailed presentation of the new abstraction. We start with the definition of the mapping from the classical recipe notation to the tabular notation. Then we present the algorithm for executing the tabular recipes and point out some limitations of the tabular recipe notation compared to the classical notation. Finally, we present the mechanisms for controlling the execution flow of the tabular recipes, which stem from the tabular structure of the recipes.

The next topic we deal with is the behaviour model of the elementary entities of

procedural control. We begin with an overview of state machine based abstractions. Then we analyze the actual phase behaviour model, as defined by the ISA-88 standard, and as implemented and documented in a prototype tool for batch process control. The analysis shows some deficiencies of the actual phase behaviour model, namely in its representation in the standard and in the tool documentation, as well as in its very structure. In the following, we present a proposition for a new phase behaviour model, which renders it possible to get rid of the stated deficiencies, and we give a thorough description of the new model. Finally, the phase behaviour modeling using the new model is illustrated on an example from a real industrial project.

The last topic we deal with is the object model of equipment and procedural control. First, we state that the current model, as defined by the ISA-88 standard and implemented in the related batch process control tools, fails in avoiding the repetition of information in the recipes and in assuring a high degree of recipes reuse. Then we present an idea that a possible solution to these problems could be in the introduction of a more sophisticated object model, based on a concept of multiple inheritance, i.e. on overlapping equipment classes. In the sequel, we give a detailed description of the object model of equipment and procedural control, including its implementation in a home-developed batch-process control tool. We illustrate the use of a new concept by means of a real industrial case study, performed on a resin-synthesis batch-process control system. Finally, we present results of a validation of the new concept, which shows that the new approach assures a significant improvement in avoiding the repetition of information in the recipes and increasing the degree of their reuse.

Seznam kratic

MBA	=	(ang. *modular batch automation*) modularna šaržna avtomatizacija
MES	=	(ang. *manufacturing execution systems*) informacijski sistemi za spremljanje in nadzor proizvodnje
ERP	=	(ang. *enterprise resource planning*) informacijski sistemi za planiranje virov
PC	=	(ang. *personal computer*) osebni računalnik
SFC	=	(ang. *sequential function chart*) sekvenčni funkcijski diagram
PLK	=	programirljivi logični krmilnik, krmilnik na osnovi programirne logike
PLC	=	(ang. *programmable logic controller*) programirljivi logični krmilnik
MDE	=	(ang. *model-driven engineering*) modelno usmerjen razvoj, modelno usmerjeno inženirstvo
PLI	=	(ang. *phase logic interface*) vmesnik fazne logike
UML	=	(ang. *unified modelling language*) grafični modelirni jezik
FB	=	(ang. *function block*) funkcijski blok
ECC	=	(ang. *execution control chart*) diagram vodenja izvajanja (funkcijskega bloka)

1 Uvod

Pri šaržnih procesih končne količine produkta dobimo z obdelavo končne količine vhodnih materialov, pri čemer sama obdelava sestoji iz sekvence večjega števila zveznih procesiranj. Dve glavni značilnosti šaržnih procesov sta fleksibilnost in kompleksnost. S ciljem boljšega obvladovanja omenjene kompleksnosti je nastal (prvotno ameriški) standard ISA-88 [1-4], ki določa modele, terminologijo in koncepte za opis šaržne proizvodnje in njenega vodenja. Standard določa tudi strukturo za povezovanje posameznih nivojev proizvodnje [5, 6]. Leta 1997 je standard povzela tudi mednarodna organizacija IEC in ga izdala kot mednarodni standard IEC 61512 [7-10]. Standard ISA-88 je omogočil razvoj večjega števila računalniških orodij za razvoj sistemov vodenja šaržnih procesov in izvajanje tega vodenja na platformi programirljivih logičnih krmilnikov (s kratico PLK; ang. *programmable logic controller*, s kratico PLC). PLK-ji so sicer najpogosteje uporabljena platforma za izvedbo sistemov vodenja industrijskih procesov [11].

Standard ISA-88 je nedvomno prinesel bistven napredek na področju vodenja šaržnih procesov, vendar pa v povezavi s standardom in na njem temelječih orodjih obstajajo tudi nekateri problemi [12]. Po naših izkušnjah lahko glavne težave standarda in z njim povezanih orodij razdelimo v dve skupini. Po eni strani obstaja prevelika kompleksnost nekaterih konceptov za končne uporabnike sistemov vodenja, npr. kompleksna abstrakcija receptov. Po drugi strani obstajajo prevelike poenostavitve pri uporabljenih modelih določenih segmentov. Prvi tak segment je model obnašanja osnovnih entitet recepturnega vodenja (faz), kjer bi lahko uporaba modela, ki bi bil bolj razčlenjen in prilagojen domenskim značilnostim, prinesla razvijalcem še boljše obvladovanje kompleksnosti razvoja sistemov šaržnega vodenja in s tem tudi njihovo boljšo kakovost. Drugi segment je objektni model opreme in postopkovnega vodenja, kjer bi lahko uporaba bolj sofisticiranih višjenivojskih abstrakcij uporabnikom prinesla manjše podvajanje informacij v posameznih elementih sistemov vodenja in večjo ponovno uporabo teh elementov.

1.1 Namen disertacije

Namen dela je z uporabo konceptov modelno usmerjenega razvoja programske opreme definirati nove pristope, metode in modele, s pomočjo katerih bi bilo mogoče odpraviti ali omiliti zgoraj omenjene probleme vodenja šaržnih procesov. Raziskave se bodo osredotočile na naslednja področja dela:

Abstrakcija receptov

Na osnovi izkušenj iz preteklih projektov izvedbe sistemov vodenja šaržnih procesov, ko smo operaterjem na različne načine pojasnjevali način delovanja (semantiko) SFC zapisa receptov, pri čemer smo si pogosto pomagali s tabelami, se nam poraja ideja, da bi določene vrste tabelarni zapis receptov lahko bil tako enostavnejši za implementacijo na PLK kot tudi razumljivejši operaterjem v primerjavi z zapisom SFC.

Namen tega področja dela je torej iskanje abstrakcije receptov za izvajanje na platformi programirljivih logičnih krmilnikov, ki bi po eni strani imela izrazno moč podobno izrazni moči jezika SFC, po drugi strani pa bi bila bolj enostavna tako za razumevanje in uporabo s strani operaterjev kot tudi za implementacijo na omenjeni platformi.

Modeli obnašanja faz

Model obnašanja faz, ki ga priporoča standard ISA-88 in ga uporabljajo današnja komercialna orodja, je v osnovi dober, vendar je po našem prepričanju nekoliko preveč enostaven oziroma na prenizkem nivoju abstrakcije. To ima za posledico preveč kompleksno in nepregledno aplikacijsko kodo (inherentna kompleksnost sistema pač nekje mora priti do izraza), kar pa ima negativen vpliv na obvladovanje razvoja in posledično na kakovost samega sistema vodenja. Rešitev je v uvedbi bolj sofisticiranega modela obnašanja faz, in sicer v skladu s sodobnimi koncepti programskega inženirstva, predvsem s konceptom modelno usmerjenega razvoja, pri katerem obstaja večje število abstrakcij, ki lahko služijo kot izhodišče pri razvoju novega modela obnašanja [13-26].

Namen tega področja dela je torej raziskava možnosti za razvoj novega modela obnašanja faz v obliki končnih avtomatov z večjo izrazno močjo od aktualnega modela, ki bi omogočal boljše obvladovanje procesa razvoja postopkovnega vodenja šaržnih procesov.

Objektni modeli opreme in postopkovnega vodenja

Objektna tehnologija je omogočila velik napredek na področju razvoja programske opreme nasploh. Dve pomembni pozitivni posledici uvedbe objektne tehnologije so v izboljšanju ponovne uporabe [27, 28] in zmanjševanju podvajanja informacij [29]. Čeprav naj bi take pozitivne posledice uvedbe objektne tehnologije pričakovali tudi na področju programske opreme za vodenje šaržnih procesov, temu žal ni tako. Po našem mnenju je glavni vzrok danes prisotne visoke stopnje podvajanja informacij v receptih in njihove nizke ponovne uporabe v pomanjkljivostih standardnega objektnega modela šaržnega vodenja, ki izvirajo iz fiksne strukture razredov enot in preveč enostavne razredne hierarhije. Rešitev vidimo v uvedbi bolj sofisticiranega objektnega modela opreme in postopkovnega vodenja, ki bi vseboval najsodobnejše koncepte objektne tehnologije, vključno s konceptom večkratnega dedovanja [30-41]. Le-ta bi omogočil doseganje fleksibilne strukture razredov opreme in na ta način manjše podvajanje informacij v receptih in povečanje njihove ponovne uporabe.

1.2 Cilji disertacije

V skladu z namenom doktorskega dela naj bi bili doseženi naslednji cilji:

1. Definirana in preizkušena nova abstrakcija receptov;
2. Definiran in preizkušen novi model obnašanja faz;
3. Definiran in preizkušen nov objektni model opreme in postopkovnega vodenja.

1.3 Hipoteza

Pričakujemo, da je mogoče zasnovati nove modele programske opreme za šaržno vodenje, ki bodo poenostavili implementacijo na krmilnikih, olajšali uporabo operaterjem, izboljšali obvladovanje razvojnega procesa in omogočili bistveno zmanjšanje podvajanja informacij v receptih in njihovo večjo ponovno uporabo.

1.4 Utemeljitev znanstvene relevantnosti

Programska oprema je osrednji del sodobnih sistemov vodenja, saj se za njen razvoj porabi 60–80 % celotnega razvojnega napora [31]. Zato so zelo pomembni tisti faktorji, ki imajo velik vpliv na lastnosti programske opreme glede njene kakovosti in ekonomičnosti njenega razvoja in vzdrževanja. Med pomembnejše tovrstne faktorje sodijo (i) abstrakcije, ki jih uporabljamo za modeliranje obnašanja sistemov in njihovih komponent, ter (ii) mehanizmi za doseganje čim nižje stopnje podvajanja informacij in čim večje stopnje ponovne uporabe komponent programske opreme.

Velik pomen teh faktorjev se kaže v intenzivnih raziskavah, ki segajo od zelo splošnih konceptov pa do povsem konkretnih rešitev za posamezna specifična področja ali celo primere (glej npr. [42-58]). Omenjeni faktorji so ključnega pomena tudi pri specifični programski opremi za vodenje šaržnih procesov in zato se v tej disertaciji ukvarjamo z raziskovanjem možnosti njihovega izboljšanja. Pri tem si bomo pomagali s koncepti modelno usmerjenega razvoja (model-driven engineering – MDE) [59], ki sodijo med najsodobnejše pristope programskega inženirstva. Ti koncepti razvoja programske opreme temeljijo na gradnji modelov na visokem nivoju abstrakcije [60] in njihovi rutinski (ali avtomatski) preslikavi v izvršljivo kodo [61], poudarek pa je na uporabljenih abstrakcijah in na ponovni uporabi.

1.5 Metodologija

Metodološki pristop bo temeljil na tehnikah programskega inženirstva, predvsem na elementih metodološkega pristopa modelno usmerjenega inženirstva in na hevristiki.
Pri definiciji nove abstrakcije receptov bomo uporabili hevrističen pristop.

Pri analizi in definiciji novega modela obnašanja faz bomo uporabili tehnike modeliranja obnašanja na osnovi končnih avtomatov.

Pri analizi in definiciji novega objektnega modela opreme in postopkovnega vodenja bomo uporabili tehnike objektnega modeliranja.

Oceno novega objektnega modela opreme in postopkovnega vodenja bomo opravili s študijo primera na osnovi realnega industrijskega procesa.

1.6 Okvir za preizkušanje rezultatov disertacije

Kot smo omenili v podpoglavju 1.2 je v ciljih za vsa tri področja, s katerimi se v disertaciji ukvarjamo, predvideno tudi preizkušanje rezultatov disertacije. Vsi novi koncepti, zasnovani v okviru disertacije, so bili preizkušeni v okviru orodja za recepturno vodenje šaržnih procesov po standardu ISA 88.00.01, imenovanega PLCbatch [62, 63, 64], ki smo ga razvili v sodelovanju s podjetjem INEA d.o.o. Omenjeno orodje in novi koncepti so bili do sedaj uporabljeni v štirih projektih izvedbe sistemov vodenja srednje velikih šaržnih procesov, in sicer:

- ogljičenje in kaljenje izdelkov iz jekla [65],
- izboljšava plastičnih polizdelkov v avtoklavih [66],
- mešanje barv in impregnacijskih sredstev za industrijski filc [67] in
- sinteza smol [68].

1.7 Struktura disertacije

Besedilo disertacije je strukturirano na naslednji način. V drugem poglavju podamo opis šaržnih procesov in programske opreme za njihovo vodenje. Nadaljevanje poglavja posvečamo standardom in modelom za vodenje šaržnih procesov, konkretno standardu ISA 88.00.01. Najprej pojasnimo vzroke njegovega nastanka, potem pa podamo pregled njegove vsebine. Sledi opis razvojnega procesa, ki ga standard določa, ter na koncu opredelitev težav pri uporabi standarda.

Tretje poglavje obravnava abstrakcijo receptov za vodenje šaržnih procesov. Najprej podamo analizo glavne abstrakcije, ki se danes uporablja za zapis receptov šaržnega vodenja, in sicer abstrakcije sekvenčnega funkcijskega diagrama. Abstrakcijo opišemo, nato pa opredelimo težave ob njeni uporabi pri vodenju šaržnih procesov, ki so bile motiv za snovanje nove abstrakcije receptov. V osrednjem delu poglavja podamo podroben opis nove, tabelarne abstrakcije receptov, opišemo preslikavo iz receptov iz zapisa v sekvenčnem funkcijskem diagramu v tabelarni zapis ter definiramo algoritem izvajanja tabelarnega recepta. V nadaljevanju opredelimo omejitve strukture receptov, katerih vzrok je nižja izrazna moč tabelarnega zapisa glede na zapis v sekvenčnem funkcijskem diagramu. Na koncu opišemo še način izvedbe nekaterih strukturnih elementov receptov v tabelarni obliki.

Četrto poglavje obravnava model obnašanja faz, ki so elementarne entitete postopkovnega vodenja. Najprej se posvetimo pregledu abstrakcij na osnovi avtomatov

stanj. Nato analiziramo aktualni model obnašanja faz, kot ga opredeljuje standard ISA 88.00.01 in kot je implementiran in dokumentiran v vzorčnem orodju za vodenje šaržnih procesov. Pri analizi ugotovimo pomanjkljivosti modela, in sicer tako v predstavitvi modela v standardu in v dokumentaciji orodja kot tudi v sami strukturi modela. V nadaljevanju predstavimo predlog novega modela obnašanja faz, ki naj bi omogočil odpravljanje ugotovljenih pomanjkljivosti, ter nov model podrobno opišemo. Na koncu podamo še ilustracijo modeliranja obnašanja faze z novim modelom na primeru iz realnega industrijskega projekta.

Peto poglavje obravnava objektni model opreme in postopkovnega vodenja. Najprej ugotovimo, da model, ki ga določa standard ISA 88.00.01 in je trenutno v uporabi v vseh nam znanih orodjih, ni uspešen pri izogibanju podvajanja informacij v receptih in zagotavljanju visoke stopnje njihove ponovne uporabe ter da možno rešitev teh problemov vidimo v uvedbi bolj sofisticiranega objektnega modela. V naslednjem podpoglavju podamo uvodni pregled modelov, ki jih standard ISA 88.00.01 uporablja za opis šaržnega vodenja. V nadaljevanju podamo širši pregled objektne tehnologije pri vodenju šaržnih procesov in ugotovimo neuspešnost te tehnologije pri zagotavljanju izogibanja podvajanja informacij v receptih in zagotavljanja visoke stopnje njihove ponovne uporabe. Nato opredelimo pomen mehanizma dedovanja pri uporabi objektne tehnologije za doseganje omenjenih ciljev, poseben poudarek pa namenimo konceptu večkratnega dedovanja, ki ima še posebej velik potencial pri doseganju teh ciljev in je po našem mnenju zelo obetaven tudi na področju šaržnega vodenja. V nadaljevanju bolj podrobno opišemo strukturo in pomanjkljivosti standardnega objektnega modela opreme in postopkovnega vodenja. Sledi podroben opis novega objektnega modela opreme in postopkovnega vodenja. Nato opišemo implementacijo novega koncepta v lastnem orodju za vodenje šaržnih procesov ter ilustriramo njegovo uporabo na primeru realnega industrijskega projekta vodenja procesa sinteze smol. Na koncu predstavimo še rezultat vrednotenja novega koncepta, ki je izvedeno v kontekstu omenjenega industrijskega projekta in je pokazalo, da nov pristop zagotavlja pomembno izboljšanje na področju izogibanja podvajanja informacij v receptih in povečanja stopnje njihove ponovne uporabe.

V šestem, zadnjem, poglavju je sklep, v katerem na kratko povzamemo problematiko modelov šaržnega vodenja, in sicer na treh področjih, s katerimi se v delu ukvarjamo in ki smo jih obravnavali v tretjem, četrtem in petem poglavju te disertacije. Na koncu navedemo originalne prispevke disertacije ter ugotovimo, da smo z rezultati disertacije potrdili hipotezo, podano v uvodu te disertacije.

2 Šaržni procesi in njihova programska oprema

2.1 Opis šaržnih procesov

Proces je zaporedje kemičnih, fizikalnih ali bioloških aktivnosti, ki nastopajo pri pretvorbi (preoblikovanju), transportu, in/ali skladiščenju materiala, energije in/ali informacije. Industrijske procese lahko razvrščamo na več načinov. Možni kriteriji za klasifikacijo so: tip elementa, ki ga transformirajo, tip spremembe stanja, način predelave snovi in tip procesnih spremenljivk [69]. Glede na prevladujoči tip spremenljivk delimo procese na:

- zvezne procese, kjer se snov, energija ali informacija zvezno pretakajo skozi proces, npr. predelava naftnih derivatov;
- sekvenčne procese, pri katerih nastopajo zaporedja različnih med seboj ločljivih stanj, npr. vožnja dvigala;
- diskretne procese, pri katerih nastopajo posamezni objekti, ki jih preoblikujemo, sestavljamo, transportiramo in skladiščimo, pri čemer objekti ohranjajo svojo identiteto, npr. sestavljanje radijskih aparatov iz sestavnih delov;
- šaržne procese, ki so kombinacija zveznih in sekvenčnih procesov, npr. proizvodnja PVC.

Šaržni proces je definiran kot proces, pri katerem dobimo končne količine produkta tako, da končno količino vhodnega materiala obdelujemo z urejenim zaporedjem procesnih aktivnosti preko omejenega časovnega intervala in pri tem uporabimo enega ali več kosov opreme. Produkt, proizveden v šaržnem procesu, se imenuje šarža, podatki o postopku pa so zbrani v receptu. Šaržni proces je torej sestavljen iz sekvence zveznih procesov, npr. polnjenje, mešanje, segrevanje, reakcija, ohlajanje, praznjenje, kar pomeni, da sta oba tipa postopkov zelo prepletena. Pri vodenju šaržnih procesov moramo tako uporabljati kombinacijo zveznega vodenja posameznih enot ter sekvenčnega vodenja šarž, kar je eden od razlogov, zaradi katerih je vodenje šaržnih procesov zahtevnejše od vodenja čistih zveznih ali čistih sekvenčnih procesov. Šaržne procese srečujemo na nekaterih področjih farmacevtske industrije in biotehnologije, kemične, kovinske in prehranske industrije. Šaržni procesi so primerni pri pogostih spremembah specifikacij proizvoda in pri proizvodnji manjših količin proizvodov, ki imajo tipično veliko dodano vrednost.

2.2 Programska oprema za vodenje šaržnih procesov

Dobra lastnost šaržnih procesov je njihova velika fleksibilnost, saj je mogoče z isto opremo proizvajati širši spekter proizvodov. Seveda pa moramo za to fleksibilnost plačati ustrezno ceno – ti sistemi so bolj kompleksni, še posebej njihova programska oprema. Razvijalci in uporabniki sistemov vodenja šaržnih procesov so dolga leta plačevali visoko ceno te kompleksnosti, kar je pogosto imelo težke posledice skozi celotni življenjski cikel teh sistemov, od analize potreb (težavna komunikacija med analitiki in uporabniki zaradi pomanjkanja univerzalnega modela šaržnega vodenja), preko njihove verifikacije in validacije, do uporabe (uporabnikom nerazumljivo in pogosto nedeterministično obnašanje sistemov vodenja). Ti problemi so imeli za posledico sisteme vodenja, ki niso izpolnjevali pričakovanj uporabnikov in ki jih je bilo zelo težko (če ne celo nemogoče) vzdrževati, predvsem zaradi prepletanja dela programske opreme, ki obravnava izdelek (recept), dela, ki obravnava opremo, na kateri se proizvaja izdelek (fazna logika oziroma posamezni proizvodni postopki), in dela, ki obravnava krmiljenje naprav na najnižjem nivoju.

2.3 Standardi, modeli in orodja za vodenje šaržnih procesov

Zaradi zgoraj omenjenih težav se je pojavila potreba po izdelavi konsistentnih modelov za vodenje šaržnih procesov in standardizaciji le-teh. Rezultat tega je standard ISA 88.00.01 [1], ki uvaja referenčne modele, terminologijo in koncepte za opis šaržne proizvodnje in vodenja šaržnih procesov na osnovi koncepta tako imenovane modularne šaržne avtomatizacije (ang. *Modular Batch Automation*, s kratico MBA). ISA 88.00.01 definira hierarhični okvir za segmentiranje procesa in za upravljanje receptur, s ciljem ločevanja izdelka in procesa, ki proizvaja ta izdelek. Standard omogoča ponovno uporabo in fleksibilnost procesne in programske opreme ter z uporabo koncepta receptov omogoča tudi nestrokovnjakom za programsko opremo spreminjanje obnašanja sistemov glede na proizvodne zahteve različnih izdelkov. Standard poleg tega tudi daje strukturo za koordinacijo in integracijo informacije o šaržah in receptih skozi vse nivoje proizvodnje, od fizičnega, preko MES (ang. *Manufacturing Execution Systems*, informacijski sistemi za spremljanje in nadzor proizvodnje), do ERP (ang. *Enterprise Resource Planning*, informacijski sistemi za planiranje virov) nivoja [5, 6]. Zelo pomemben del standarda je tudi tisti, ki obravnava izjemne situacije v procesu, kar predstavlja enega izmed najzahtevnejših vidikov šaržnega vodenja. Standard v tem smislu uvaja koncept reverzibilnega varnega stanja (koncept Zadržanje-Nadaljevanje), ki omogoča avtomatski prehod procesa v varno stanje ob nastopu izjemne situacije in prav tako avtomatsko sinhronizacijo procesa in recepta ob nadaljevanju izvajanja po odpravljanju izjemne situacije. Standard ISA 88.00.01 torej predstavlja močno orodje, ki omogoča obvladovanje dokaj kompleksnega področja človekovega delovanja s pomočjo vpeljanega abstraktnega modela tega področja. Standard ISA 88.00.01 je leta 1997 povzela tudi mednarodna

organizacija IEC in ga izdala kot mednarodni standard za področje šaržnega vodenja IEC 61512-1 [7-10]. V nadaljevanju tega dela bomo uporabljali originalni naziv ISA 88.00.01, in sicer zato, ker je ta naziv prevladujoč v svetovnem merilu. Standard ISA 88.00.01 je omogočil razvoj večjega števila računalniških orodij za vodenje šaržnih procesov (ang. *Batch tools*), npr. Proficy Batch Execution [70], FactoryTalk Batch [71], Wonderware InBatch [72], SIMATIC BATCH [73]. Omenjena orodja omogočajo tako razvoj sistemov šaržnega vodenja, kot tudi samo izvajanje tega vodenja, večinoma na platformi osebnih računalnikov. Danes obstaja med proizvajalci opreme za avtomatizacijo, ponudniki storitev na tem področju in končnimi uporabniki širok konsenz glede potrebe in smiselnosti upoštevanja standarda ISA 88.00.01 pri vodenju šaržnih procesov ne glede na stopnjo avtomatizacije tega vodenja.

2.4 Opis standarda ISA 88.00.01

2.4.1 Vrste vodenja

Standard za vodenje šaržnih procesov ISA 88.00.01 določa naslednje vrste vodenja:

- osnovno vodenje (ang. *basic control*), ki je namenjeno vzpostavljanju in vzdrževanju določenega stanja procesa in procesne opreme in je običajno za vse pod-domene procesnega vodenja podobno;
- postopkovno vodenje (ang. *procedural control*), ki določa izvajanje procesno usmerjenih aktivnosti v urejenem vrstnem redu in je za vsako od pod-domen procesnega vodenja različno;
- koordinacijsko vodenje (ang. *coordination control*), ki na nivoju upravljanja proizvodnje zagotavlja arbitražo in alokacijo opreme posameznim postopkom, izbor postopkov, ki jih je mogoče izvesti, izbiro med različnimi režimi obratovanja in podobno.

Našteta delitev vrst vodenja ni lastna le šaržnim procesom, temveč jo je smiselno uporabiti kot način dekompozicije zaradi obvladovanja kompleksnosti tudi pri drugih vrstah procesov. Pri šaržnih procesih je ta delitev bolj pomembna zaradi posebej velike kompleksnosti njihovega postopkovnega in koordinacijskega vodenja.

2.4.2 Modeli in terminologija standarda ISA 88.00.01

Standard ISA 88.00.01 je zasnovan na treh hierarhično zgrajenih modelih:

- procesni model (ang. *process model*),
- fizični model (ang. *physical model*),
- model postopkovnega vodenja (ang. *procedural control model*).

2.4.2.1 Procesni model

Procesni model predstavlja možno porazdelitev šaržnih procesov v hierarhično strukturo, ki jo sestavljajo naslednje entitete:

- proces,
- procesna stopnja (ang. *process stage*),
- procesna operacija (ang. *process operation*),
- procesna akcija (dejavnost) (ang. *process action*).

Šaržni proces sestoji iz urejenega zaporedja ene ali več procesnih stopenj, ki se lahko izvajajo zaporedno, vzporedno ali kombinirano.

Procesna stopnja je del procesa, ki se lahko izvaja neodvisno od ostalih delov – stopenj. Običajno rezultira v načrtovano zaporedje kemičnih ali fizikalnih sprememb materiala v obdelavi. Vsako procesno stopnjo sestavlja urejeno zaporedje ene ali več procesnih operacij.

Procesna operacija predstavlja glavne procesne aktivnosti. Rezultat procesne operacije je navadno kemična ali fizikalna sprememba materiala v obdelavi. Vsaka procesna operacija je porazdeljena v urejeno zaporedje ene ali več procesnih akcij.

Procesna akcija je najmanjši sestavni del procesnega modela in predstavlja manjšo procesno aktivnost, ki se skupaj z ostalimi procesnimi akcijami združuje v procesno operacijo.

2.4.2.2 Fizični model

Fizični model opisuje dejansko opremo, uporabljeno za procesiranje šarž, in je oblikovan hierarhično. Vsak nivo vsebuje enega ali več elementov nižjega nivoja. Elementi fizičnega modela so:

- podjetje (ang. *enterprise*),
- podružnica (ang. *site*),
- obrat (ang. *area*),
- procesna celica (ang. *process cell*),
- enota (ang. *unit*),
- modul opreme (ang. *equipment module*),
- modul vodenja (ang. *control module*).

Podjetje je sestavljeno iz ene ali več podružnic podjetja in lahko vsebuje podružnice, obrate, procesne celice, enote, module opreme in module vodenja. Podjetje določa, kateri izdelki se bodo proizvajali in v kateri podružnici podjetja ter v splošnem tudi kako se bodo proizvajali.

Podružnica podjetja je fizična, geografska ali logična razvrstitev, določena s strani podjetja. Vsebuje lahko obrate, procesne celice, enote, module opreme in module vodenja.

Obrat je fizična, geografska ali logična razvrstitev, določena s strani sedeža podjetja.

Vsebuje lahko procesne celice, enote, module opreme in module vodenja.

Procesna celica vsebuje celotno opremo, ki je potrebna za izdelavo ene ali več šarž. Sestavljena je iz enot, modulov opreme in modulov vodenja. Glede na število produktov lahko procesne celice klasificiramo na eno- in več-produktne. Pri eno-produktnih celicah se v vsaki šarži proizvaja isti produkt, možne so le variacije postopkov in procesnih parametrov. V več-produktnih celicah se proizvajajo različni produkti na dva načina – z različnimi vhodnimi materiali in parametri ali z različnimi postopki. Procesne celice pa lahko klasificiramo tudi glede na fizično povezavo enot znotraj celice. Pri strukturi z eno potjo se vhodni material obdeluje postopno v zaporedno povezanih enotah. Od vhodnega materiala skozi enote do končnih produktov obstaja ena sama pot. Struktura z več potmi ima od vhodnega materiala do končnih produktov več vzporednih poti, ki se med seboj ne stikajo, na vsaki poti pa so enote povezane zaporedno. Vsaka enota je lahko del ene same poti. Mrežna struktura ima prav tako več poti skozi enote, vendar se poti med seboj stikajo in križajo, zato lahko posamezna enota nastopa na različnih poteh.

Enota je sestavljena iz modulov opreme in modulov vodenja, ki skrbijo za izvajanje ene ali več samostojnih procesnih aktivnosti, kot so reakcija, kristalizacija, priprava raztopine, filtriranje in podobno. Enota lahko deluje na največ eni šarži hkrati.

Modul opreme je funkcionalna skupina naprav, ki lahko opravlja manjše procesne aktivnosti, kot so mešanje, tehtanje in podobno. Modul opreme lahko vsebuje module vodenja in druge module opreme. Lahko je del enote ali samostojna skupina naprav znotraj procesne celice. Če je samostojen, se enoti dodeli ali kot izključni ali kot deljeni vir.

Modul vodenja izvaja manipulacijo z regulacijskimi napravami, drugimi osnovnimi napravami, kot so regulatorji, senzorji, aktuatorji, ventili, črpalke, ter ostalimi moduli vodenja, ki v smislu procesnega vodenja nastopajo kot en objekt. Vsebuje elemente osnovnega vodenja in včasih tudi elemente za upravljanje z izjemnimi dogodki.

Gornji trije nivoji fizičnega modela so običajno definirani v poslovnih sistemih in v nadaljevanju standarda nadalje niso obravnavani. Vse entitete fizičnega modela vsebujejo osnovno vodenje, vendar standard ne določa načina implementacije vodenja opreme. Ne glede na to, ali je vodenje opreme ročno ali avtomatsko, lahko oprema sodeluje v šaržnem vodenju samo preko izvajanja akcij vodenja.

2.4.2.3 Model postopkovnega vodenja

Model postopkovnega vodenja določa, kako – na kakšen način – pridemo z uporabo elementov fizičnega nivoja (procesne opreme) do produkta. Vsebuje elemente, ki vodijo in usmerjajo izvajanje procesno usmerjenih akcij v urejenem vrstnem redu:

- postopek (ang. *procedure*),
- postopek enote (ang. *unit procedure*),
- operacija (ang. *operation*),
- faza (ang. *phase*).

Postopek je najvišji nivo v hierarhiji postopkovnega modela in določa strategijo za izvedbo pomembnih procesnih aktivnosti (npr. izdelava šarže) znotraj procesne celice. Postopek sestavlja urejeno zaporedje enega ali več postopkov enote.

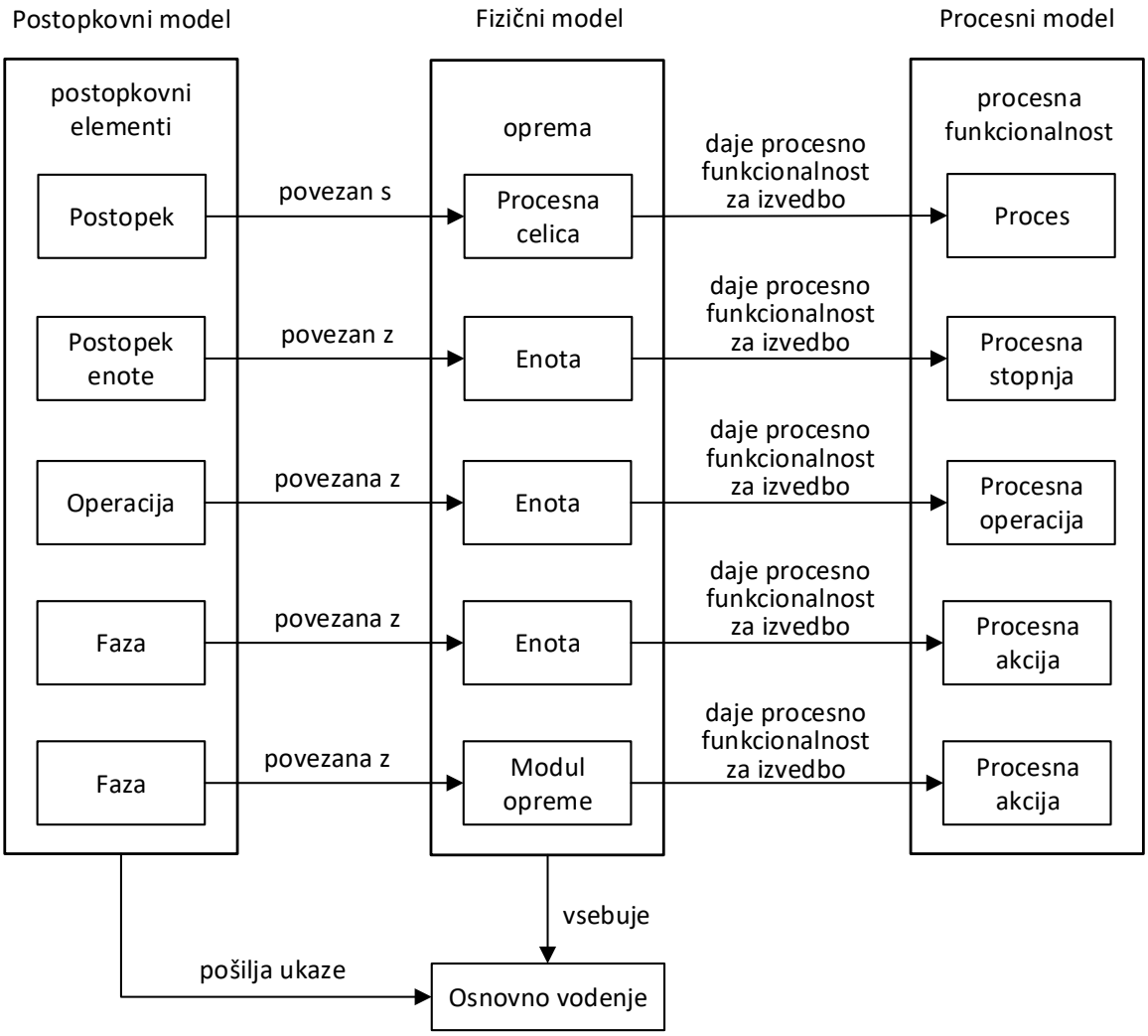
Postopek enote je sestavljen iz urejene množice zaporednih operacij znotraj ene enote. Hkrati se sme izvajati več postopkov, vendar v različnih enotah. Postopek enote sestoji iz ene ali več operacij.

Operacija je algoritem, potreben za zagon, organizacijo in nadzor posameznih faz. Operacija običajno pripelje do spremembe fizikalnega ali kemičnega stanja obdelovanega materiala in se mora izvesti v celoti v isti enoti. Sestavljena je iz posameznih faz.

Faza predstavlja najnižji nivo v postopkovnem modelu in je najmanjši element, ki lahko izvrši procesno orientirano aktivnost (npr. gretje, mešanje, polnjenje). Fazo sestavljajo posamezni koraki, ki izvajajo procesno orientirane akcije s pošiljanjem ukazov osnovnemu vodenju, ki je vsebovano v fizičnem modelu. Faza lahko pošilja ukaze tudi elementom postopkovnega modela, na primer drugi fazi. Ker so faze najmanjši gradniki postopkovnega modela, jih pri uporabi sistema vodenja sestavljamo v hierarhično višje elemente postopkovnega modela.

2.4.2.4 Relacije med modeli

Preslikava med vsemi tremi modeli je hierarhično organizirana po načelu, da postopek, realiziran na določeni opremi, rezultira v proces. Medsebojne relacije modelov si zato predstavljamo na naslednji način: izvajanje faze, ki je element postopkovnega modela, na ustreznem modulu opreme daje procesno funkcionalnost za izvedbo procesne akcije, ki pa je entiteta procesnega modela. Pri tem elementi postopkovnega modela pošiljajo ukaze osnovnemu vodenju, ki je vsebovano v entitetah fizičnega modela. Relacije med vsemi tremi modeli prikazuje Slika 1.



Slika 1: Relacije med modeli ISA 88.00.01.

2.4.3 Recepti

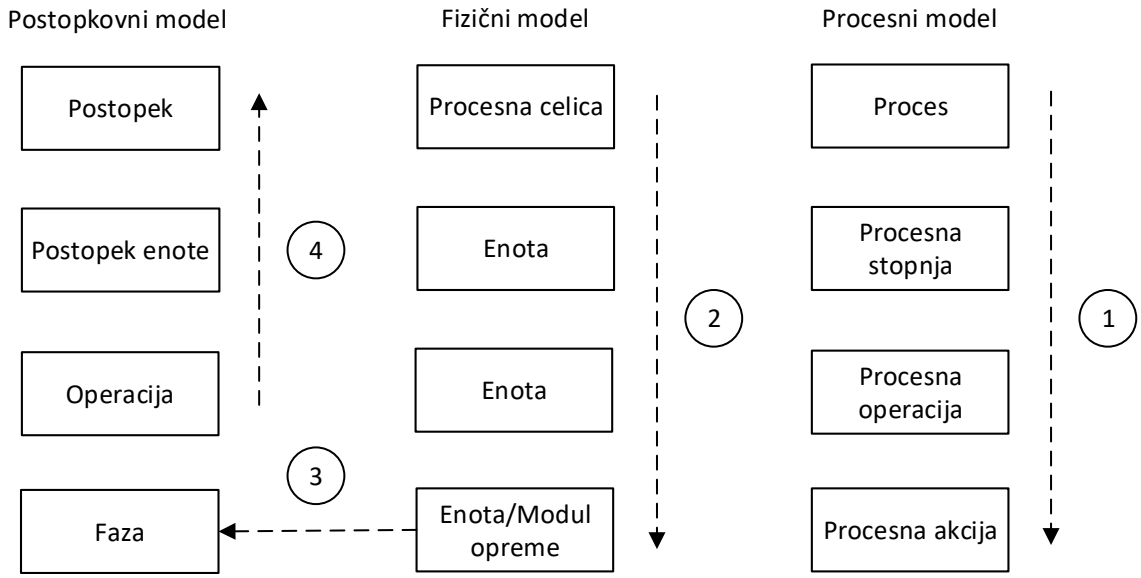
Poleg treh zgoraj opisanih modelov standard določa še recepte (ang. *recipe*), ki vsebujejo vse informacije, ki enolično določajo proizvodne zahteve za določen produkt. Standard predvideva štiri tipe receptov, ki se razlikujejo glede na to, v kakšni meri so informacije, ki jih vsebujejo, splošne (neodvisne od konkretnega proizvodnega okolja) ali konkretne (prilagojene konkretnemu proizvodnemu okolju). Tako se na najvišjem nivoju nahaja glavni ali splošni recept (ang. *general recipe*), ki je popolnoma neodvisen od opreme, na kateri se bo izdelek proizvajal. Na drugem nivoju se nahaja podružnični recept (ang. *site recipe*), ki vsebuje tako splošne informacije, kot tudi informacije značilne za določeno podružnico podjetja. Na tretjem nivoju je temeljni recept (ang. *master recipe*), ki je že na takem nivoju podrobnosti, da vsebuje vso potrebno informacijo, potrebno za njegovo izvajanje na konkretni procesni celici ali celo podmnožici opreme procesne celice. Temeljni recept pa še vedno vsebuje tudi določeno splošno informacijo, saj v njem še vedno ni

opredeljeno, na katerih konkretnih enotah znotraj procesne celice se bo izvajal, prav tako pa tudi ni informacije glede časovnega razporejanja izvajanja. Nivo temeljnega recepta je prvi nujni nivo (višje nivoje se lahko izpusti), saj brez njega ni mogoče izvajati recepturnega vodenja šaržne proizvodnje. Na zadnjem nivoju se nahaja recept vodenja (ang. *control recipe*), ki v bistvu predstavlja kopijo pripadajočega temeljnega recepta, v kateri so preostalim splošnim informacijam prirejene konkretne vrednosti (ob tem je treba omeniti, da lahko tudi tukaj določene informacije ostanejo splošne in da konkretno vrednost dobijo šele med izvajanjem recepta, npr. izbira konkretne opreme v času izvajanja, glede na trenutno razpoložljivost, ali spremembe procesnih parametrov med izvajanjem, ali spremembe postopka zaradi izjemnih dogodkov).

Recepti vsebujejo več kategorij informacij, zato so strukturirani, oziroma razdeljeni na naslednje dele: glava (ang. *header*), formula (ang. *formula*), zahteve po opremi (ang. *equipment requirements*) in postopek recepta (ang. *recipe procedure*). V kontekstu naših raziskav je pomemben postopek recepta, zato se bomo v naslednjih poglavjih z njim bolj podrobno ukvarjali.

2.4.4 Razvojni proces šaržnega vodenja

Razvojni proces šaržnega vodenja shematsko prikazuje Slika 2. Tako fizični kot procesni model (puščici 1 in 2) se med snovanjem samega procesa razvijata iz najvišjega nivoja proti najnižjemu, medtem ko se postopkovni model gradi od najnižjega nivoja proti najvišjemu. Najnižji gradnik postopkovnega modela je faza in njena izdelava je odvisna od tehnološke opreme, definirane v fizičnem modelu. Puščice 1, 2 in 3 predstavljajo razvojne aktivnosti, puščica 4 pa predstavlja delo uporabnika. Uporabnik sestavlja recepte iz postopkov enot, operacij in faz tako, da se izvrši želen postopek. Recept torej predstavlja mehanizem preslikave (grupiranja) eno-nivojske strukture osnovnih gradnikov kode v krmilniku, to je faz, v štiri-nivojsko strukturo recepta. Ta koncept predstavlja mehanizem za doseganje fleksibilnosti, ki nam ga ponujajo orodja za vodenje šaržnih procesov na osnovi standarda ISA 88.00.01.



Slika 2: Razvojni proces šaržnega vodenja.

2.5 Težave pri uporabi standarda ISA 88.00.01

V zvezi z uporabo standarda ISA-88 in na njem temelječih orodjih pa obstajajo tudi nekateri problemi [12]. Po naših izkušnjah so glavne težave standarda in z njim povezanih orodij naslednje:

1. *Previsoka cena teh orodij za večino manjših in srednjih projektov*. Cene licenc orodij za vodenje šaržnih procesov so pri vseh nam znanih orodjih (primerjali smo štiri orodja [70, 71, 72, 73]) odvisne od največje velikosti procesa, ki ga lahko z določeno verzijo licence orodja vodimo. Mera velikosti je povsod število enot in to verjetno zaradi tega, ker je število enot po eni strani dobra mera velikosti in kompleksnosti šaržnega procesa ter s tem tudi dobra mera stroška izdelave sistema za njegovo vodenje, po drugi strani pa je pri številu enot, ki jih uporabimo, izredno težko, če ne nemogoče, goljufati (prikazati, oziroma kupiti licenco za manjše število enot, kot jih sistem dejansko ima), saj bi s tem več izgubili s stroški dela in stroški zaradi nižje kakovosti sistema vodenja, kot pa bi pridobili zaradi nakupa licence za manjše število enot, kot jih dejansko imamo. Funkcija cene licence za posamezna od štirih obravnavanih orodij glede na število enot je pri vseh orodjih sestavljena iz začetne cene, ki pokrije eno do tri enote, ter dodatka glede na število enot, ki je stopničasta funkcija z različnimi točkami preskoka glede na število enot pri posameznih orodjih. Rezultirajoča cena na posamezno enoto pri vseh orodjih pada glede na število enot. V nadaljevanju podajamo območja cen pri štirih obravnavanih orodjih za nekaj tipičnih velikosti procesov:
   - zelo majhen proces (1 enota): cena od okoli 4.000 € do okoli 17.000 €
   - majhen proces (3 enote): cena od okoli 8.000 € do okoli 17.000 €

- srednje majhen proces (6 enot): cena od okoli 15.000 € do okoli 19.000 €
- srednje velik proces (12 enot): cena od okoli 16.000 € do okoli 27.000 €
- velik proces (več kot 20 enot): cena od okoli 26.000 € do okoli 42.000 €
- zelo velik proces (50 ali več enot): cena od okoli 27.000 € do okoli 49.000 €

Proces sinteze smol, ki smo ga omenili v poglavju 1.6 kot enega od procesov, na katerih smo preizkušali koncepte iz te disertacije, je sestavljen iz 12 enot, torej predstavlja srednje velik proces. Cena orodja za šaržno vodenje pa bi za ta proces bila med 16.000 € in 27.000 €, kar je glede na naše izkušnje zelo visoko in precej problematično za naročnika.

2. *Nezadovoljivo časovno obnašanje PC platforme, predvsem glede na hitrost in na časovno deterministočnost*. Ta problem sicer ni zelo izrazit, saj je za večino primerov šaržnega vodenja časovno obnašanje PC platforme v mejah sprejemljivega. So pa tudi primeri, ko obstaja zahteva po najdaljšem dovoljenem odzivnem času na določen procesni dogodek, ki ga je na PC platformi veliko težje in dražje zagotoviti kot na PLK platformi.
3. *Prenizka zanesljivost in varnost PC platforme.* Splošno znano je, da PC platforma, tako z vidika materialne opreme kakor tudi s stališča sistemske programske opreme, ne zagotavlja visoke stopnje zanesljivosti, ki je zahtevana pri sistemih vodenja procesov. Ta problem je sicer rešljiv z uporabo redundance, kar pa žal prinaša dodatne, precej visoke stroške. Pri tem se ti dodatni stroški razlikujejo pri posameznih proizvajalcih, praviloma pa povečajo strošek licence za orodje za okoli 50 %.
4. *Prevelika kompleksnost orodij v smislu, da praviloma ne potrebujemo vse funkcionalnosti, ki jo ponujajo*. Sodobna orodja za vodenje šaržnih procesov večinoma ponujajo zelo bogato funkcionalnost, kar pa ima za posledico tudi kompleksnost in dokaj visoko ceno. O ceni je sicer bilo več povedanega v točki 1 zgoraj, tukaj lahko dodamo še to, da ob uporabi vse razpoložljive funkcionalnosti orodij cena ne bi bila več pretirano visoka.
5. *Prenizek nivo abstrakcije in s tem premajhna izrazna moč na področju modela obnašanja faz kot avtomatov stanj*. To se kaže v različnih težavah, kot so npr.: a) prekompleksne sekvence, ki jih je težko obvladovati in zagotavljati njihovo kakovost, b) težave pri sinhronizaciji posameznih segmentov procesiranja, ki so jih programerji prisiljeni umeščati izven konstrukta avtomata stanj in jih je zato potrebno sinhronizirati z drugimi podobnimi konstrukti ali s samim avtomatom stanj in c) pogosta potreba po pomnjenju koraka v stanju ob prehodu v drugo stanje zaradi zagotavljanja vračanja v isti korak, kar je nepregleden nizkonivojski koncept in kot tak pogost vir napak.
6. *Nizka ponovna uporaba receptov in podvajanje informacij z velikim številom podobnih, a ne povsem enakih receptov.* Vzrok omenjene težave sta neustrezen objektni model opreme in postopkovnega vodenja (na prenizkem nivoju abstrakcije), njuni posledici

pa sta večji obseg dela pri razvoju in vzdrževanju receptov ter večja verjetnost napak v receptih.

Zaradi nekaterih od zgoraj naštetih problemov (predvsem previsoke cene orodij in njihove prevelike kompleksnosti, včasih pa tudi zaradi težav zaradi lastnosti PC platforme) se izvajalci projektov avtomatizacije šaržnih procesov odločajo za različne rešitve izvajanja receptov na platformi industrijskih krmilnikov, ki pa so večinoma preveč poenostavljene in zaradi tega ob večino izrazne moči, ki jo ponujajo na ISA 88.00.01 temelječa orodja. Najpogostejši primer je izvedba recepta kot sekvence korakov. Ker se v vsakem koraku izvaja le po en postopek, so ti postopki kompleksni oziroma vsak postopek vsebuje vso v določenem trenutku izvajanja procesa potrebno funkcionalnost. V tem primeru je edina teoretično možna fleksibilnost v spreminjanju zaporedja postopkov, kar pa v večini primerov pomeni nesmiselno zaporedje in zato ne moremo več govoriti o fleksibilnosti rešitve. Ta problem se v praksi rešuje na dva načina. Prvi je v doseganju fleksibilnega obnašanja obsežnih postopkov skozi njihovo parametriranje, drugi pa v uporabi recepta, sestavljenega iz drobnih postopkov, pri čemer še vedno v receptu nastopa le po en postopek na korak ob uporabi receptu sočasnih postopkov. Problem druge rešitve je v težavnem doseganju sinhronizacije med postopki glavne veje in sočasnimi postopki. To se v praksi v glavnem rešuje s programsko odvisnostjo posameznih postopkov, kar pa predstavlja očiten primer neustrezne (previsoke) sklopljenosti postopkov in je velik potencialni vir napak.

3 Abstrakcija receptov

3.1 Analiza abstrakcije receptov

Sodobna orodja za vodenje šaržnih procesov po standardu ISA 88.00.01 za predstavitev postopka receptov uporabljajo grafično notacijo, imenovano sekvenčni funkcijski diagram (ang. *Sequential Function Chart*, s kratico SFC). V nadaljevanju tega dela bomo uporabljali kratico SFC.

SFC je model, ki je nastal v osemdesetih letih dvajsetega stoletja na osnovi francoskega standarda Grafcet iz sedemdesetih let dvajsetega stoletja. Model SFC je bil leta 1988 definiran v standardu za specifikacijo in dokumentiranje postopkov v sistemih vodenja IEC 848 [74] kot strukturno orodje za specifikacijo sekvenčnega vodenja. Z manjšimi spremembami je bil ponovno definiran v standardu IEC 61131-3 [75] z namenom sekvenčnega strukturiranja algoritmov funkcijskih blokov in programov pri programiranju programirljivih logičnih krmilnikov. V SFC-ju so z usmerjenimi povezavami medsebojno izmenično povezani koraki (grafično predstavljeni s kvadrati ali pravokotniki), ki so jim pridružene akcije, in prehodi (grafično predstavljeni z vodoravnimi odebeljenimi črticami prek navpičnih povezav). Povezava vstopa v korak vedno navpično od zgoraj in izstopa iz koraka navpično navzdol.

Stanje sekvenčnega funkcijskega diagrama je določeno z aktivnostjo ali neaktivnostjo množice korakov diagrama. Stanje se spremeni, ko nek korak preneha biti aktiven oziroma se aktivira naslednji. To se zgodi ob sprožitvi prehoda med njima. Prehod se lahko sproži le, če sta izpolnjena naslednja pogoja:

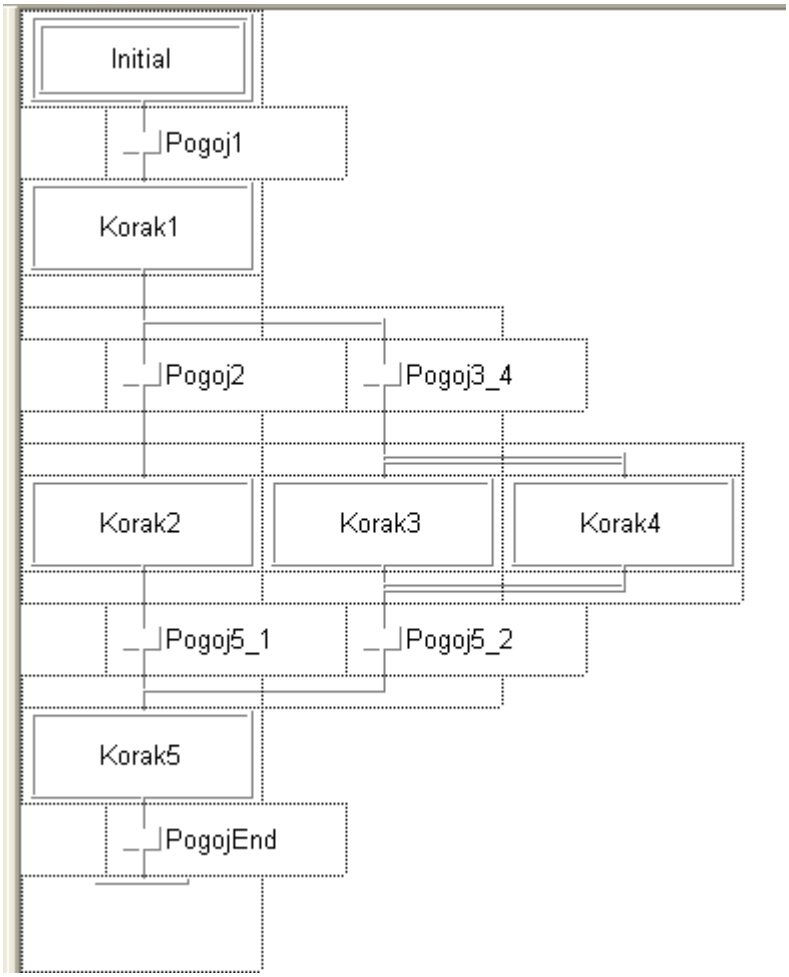
- prehod je omogočen, kar pomeni, da so vsi (sočasni) koraki pred njim aktivni, in
- prehod je dovzeten, kar pomeni, da je prehodni pogoj izpolnjen.

SFC diagram pozna tudi sočasne in izbirne razvejitve. Sočasna razvejitev je predstavljena s konstruktom, ki se nahaja za (pod) znakom za prehod, sestavljen pa je iz dveh (začetne in končne) dvojnih vodoravnih črt, med katerima poteka več vertikalnih poti, od katerih se vsaka začne in konča s korakom. Posamezne poti sočasne razvejitve se izvajajo neodvisno in sočasno. Izbirna razvejitev je konstrukt, ki se nahaja za (pod) korakom, sestavljen pa je iz dveh (začetne in končne) enojnih vodoravnih črt, med katerima poteka več vertikalnih poti, od katerih se vsaka začne in konča s prehodom. Od vseh poti izbirne razvejitve se izvaja le ena, in to tista, katere prehod je dovzeten. Če je izbirnih poti z dovzetnimi prehodi več, se izvede prva taka pot z leve strani razvejitve.

Primer SFC diagrama iz razvojnega okolja GX IEC Developer za PLKje podjetja

Mitsubishi Electric prikazuje Slika 3.

V kontekstu naše raziskave obstajata pri uporabi notacije SFC dve težavi. Prva težava nastopa v povezavi z našim ciljem, da naj bi se recepti izvajali na platformi PLC. Zaradi omejitev platforme ocenjujemo, da bi bila implementacija SFC receptov zelo zahtevna, kar velja tako za samo izvajanje, kot tudi (ali še bolj) za prikaz in animacijo SFC diagramov na prikazovalnikih, ki se uporabljajo na PLC platformah.



Slika 3: Primer SFC diagrama iz razvojnega okolja Mitsubishi GX IEC Developer.

Druga težava je v težavah operaterjev z razumevanjem in sledenjem prikazu receptov v notaciji SFC. S to težavo smo se srečali med razvojem in uvajanjem v uporabo nekaterih sistemov vodenja šaržnih procesov, v katerih smo uporabili komercialna orodja, pri katerih se recepti izvajajo na PC platformi, za zapis receptov pa se uporablja notacija SFC. Notacija SFC je relativno kompleksna za razumevanje, še posebej ob upoštevanju izobrazbene strukture uporabnikov (operaterjev). Razumevanje notacije in sledenje izvajanja receptov dodatno zakomplicira dejstvo, da je SFC prikaz receptov zelo raztegnjen (papirni izpis že relativno enostavnega recepta je tipično dolg nekaj metrov), zaradi česar morajo operaterji pogosto premikati zaslonski izpis navzgor in navzdol, pa tudi levo-desno, v primeru

obsežnejših sočasnih razvejitev.

Opisane težave z razumevanjem in sledenjem receptov v SFC notaciji s strani operaterjev smo občasno reševali tako, da smo skicirali strukturo receptov v obliki tabele in smo potem na osnovi te tabele pojasnjevali njihovo izvajanje. Posledično se je porodila tudi misel, da bi tabelarna notacija receptov utegnila biti za operaterje bolj primerna kot SFC notacija. Poleg tega bi bil tabelaren zapis receptov zagotovo bolj enostaven za implementacijo na PLC platformi, tako v segmentu izvajanja kot tudi v segmentu vizualizacije izvajanja. Posledično smo prišli do enega od ciljev te raziskave, ki je v določitvi preslikave SFC recepta v tabelo ter v zasnovi ter izvedbi programa za izvajanje te tabele na PLC platformi (interpreter tabele).

3.2 Nova abstrakcija receptov

Kot smo že omenili, je eden od ciljev raziskave razvoj recepturnega sistema, pri katerem naj se recepti v celoti izvajajo na platformi PLC. Na osebnem računalniku naj se izvajajo le funkcije izven realnega časa, kot npr. urejanje receptov, upravljanje receptov, upravljanje šarž in arhiviranje podatkov izvedenih šarž. Poleg tega naj bodo recepti predstavljeni v tabelarni obliki. Recepte se lahko na papirju razvija v SFC notaciji, ki pa jo je potem potrebno preslikati v tabelarno obliko, kjer jo nato izvaja interpreter tabelarnih receptov. Ker ni nujno, da je izrazna moč tabele enaka izrazni moči zapisa v SFC, je potrebno to razliko natančno opredeliti in v skladu z njo tudi določiti omejitve tabelarnih receptov ter pravila glede njihove strukture.

Zaradi omejitev platforme PLC bomo uporabili model opreme in postopkovnega vodenja, reduciran na samo 3 nivoje. Izpustili bomo nivo operacij, tako da ostanejo le nivo postopka, nivo postopkov enot in nivo faz. Tako bosta obstajala dva nivoja receptov oziroma dve tabeli – nivo postopka, ki bo vseboval postopke enot, in nivo postopkov enot, ki bo vseboval faze. Povedano v terminologiji tabelarnih receptov bosta obstajala dva nivoja tabel. Zgornji nivo bo tabela postopka, ki bo v celicah vsebovala postopke enot. Vsak postopek enote bo predstavljen s tabelo na nižjem nivoju, ki bo v celicah vsebovala faze. Z izpuščanjem nivoja operacij bomo žrtvovali nekaj ponovne uporabe in del strukture. Vendar pa ocenjujemo, da bi ta model moral zadoščati za večino realnih manjših in srednje velikih projektov razvoja in izvedbe vodenja šaržnih procesov.

Zaradi omejitev PLC platforme bomo omejili tudi dimenzijo tabel. Število vrstic tabele določa število korakov določenega nivoja recepta, pri čemer na osnovi izkušenj iz prakse ocenjujemo, da je ustrezna dimenzija dvajset do trideset korakov. Število stolpcev tabele določa največje število sočasnih entitet na določenem nivoju recepta (največje število sočasnih postopkov enot v postopku ali največje število faz v določenem postopku enote). Ocenjujemo, da je ustrezno število nekje od pet navzgor, vendar ne več kot deset.

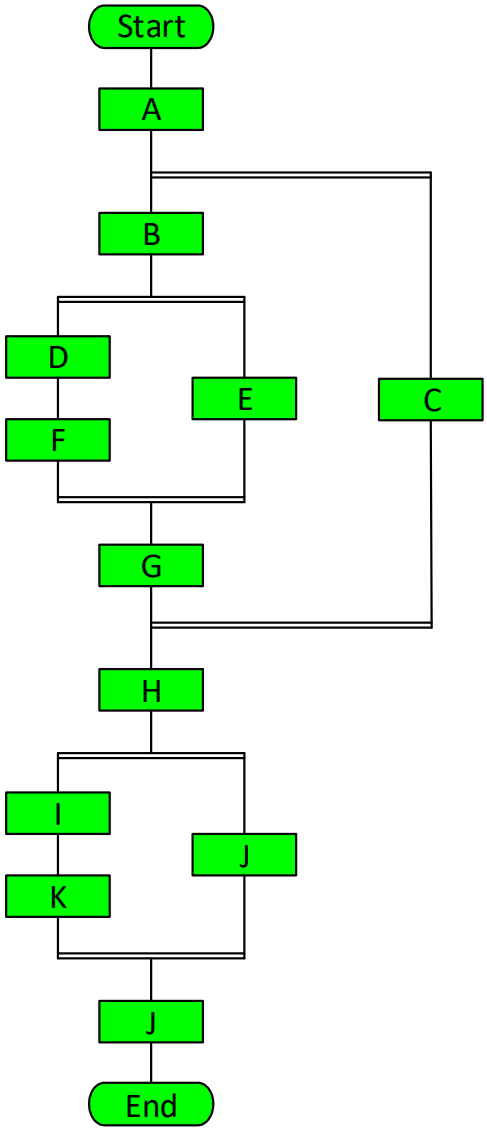
3.2.1 Preslikava SFC zapisa recepta v tabelo

SFC zapis recepta se preslika v tabelo na naslednji način. Prva vrstica tabele vsebuje tiste postopkovne entitete (postopke enot ali faze, odvisno od nivoja recepta), ki jih vsebuje začetno stanje SFC recepta. To pomeni, da se vsak aktiven korak začetnega stanja SFC zapisa recepta zapiše v prvo vrstico tabelarnega zapisa recepta. V nadaljevanju vsaka možna sprememba stanja SFC recepta (oziroma možna sprememba nabora aktivnih korakov recepta) pomeni novo vrstico tabelarnega zapisa recepta.

Preslikavo SFC diagrama v tabelo se izvede s sprehodom skozi SFC diagram. Sprehod gre od začetka do konca SFC diagrama. V delih, kjer obstaja le ena veja, gre sprehod po tej veji. V delih, kjer obstaja več sočasnih vej, gre sprehod vedno po veji, ki vsebuje največ zaporednih korakov. To vejo bomo imenovali glavna veja. Vsak korak glavne veje se zapiše v novo vrstico tabele, v isto vrstico pa se zapišejo tudi vsi koraki SFC diagrama, ki so lahko aktivni sočasno z omenjenim korakom. To pomeni, da se tisti koraki, ki se nahajajo v »dolgih« sočasnih vejah, pojavljajo v večjem številu zaporednih vrstic tabele. Preslikava izbirnih (»OR«) razvejitev SFC diagrama zaradi omejitev tabelarnega zapisa gre na drugačen način, ki bo opisan nižje.

Konkretno preslikavo si oglejmo na primeru SFC recepta, ki ga prikazuje Slika 4, njegovo preslikavo v tabelarni zapis pa prikazuje Tabela 1. Preslikava gre na naslednji način:

1. Prvi korak je korak (postopkovna entiteta) A. Ker ta korak nima nobenega sočasnega koraka, se preslika v prvo vrstico tabele kot njen edini korak.
2. Naslednji korak glavne veje je korak B. Ker ima ta korak sočasen korak C, se oba omenjena koraka preslikata vsak v svojo celico druge vrstice tabele.
3. Naslednji korak glavne veje je korak D, ki ima dva sočasna koraka, E in C, zato se ti trije koraki pojavijo v treh celicah tretje vrstice tabele.
4. Korak F ima dva sočasna koraka, E in C, zato se ti trije koraki pojavijo v treh celicah četrte vrstice tabele.
5. Korak G ima sočasen korak C, zato se ta dva koraka pojavita v dveh celicah pete vrstice tabele.
6. Korak H nima nobenega sočasnega koraka, zato se ta korak sam pojavi v šesti vrstici tabele.
7. Naslednji korak glavne veje je korak I, ki ima sočasen korak J, zato se ta dva koraka pojavita v dveh celicah sedme vrstice tabele.
8. Korak K ima sočasen korak J, zato se ta dva koraka pojavita v dveh celicah osme vrstice recepta.
9. Naslednji korak je spet J, ki se torej ponavlja v dveh zaporednih stanjih SFC zapisa recepta. Ker ta korak nima sočasnega koraka, se preslika v edino zasedeno celico devete vrstice tabele.



Slika 4: SFC recept.

Na tem mestu vpeljimo še en pojem, in sicer primerek oziroma instanco (ang. *instance*) postopkovne entitete v tabelarni predstavitvi recepta. Na zgoraj opisanem primeru preslikave SFC recepta v tabelo lahko vidimo, da se nekatere postopkovne entitete, ki v SFC diagramu nastopajo enkratno, v tabeli ponavljajo v več vrsticah. Temu je tako, ker vrstica tabele ustreza stanju SFC recepta. Tako npr. korak C v zgornjem primeru nastopa v štirih stanjih SFC diagrama (BC, DEC, FEC in GC), v tabeli pa ga najdemo v štirih zaporednih vrsticah. Ker pa dejansko gre za eno entiteto, rečemo, da v tabeli v štirih vrsticah nastopa isti primerek te entitete. Ta primer je potrebno razlikovati od primera, ko bi določena postopkovna entiteta že v SFC receptu nastopala v večjem številu zaporednih stanj ter bi se vsaka preslikala v eno (ali več) vrstic tabele, pri čemer bi tokrat šlo za različne primerke te entitete. Za ponazoritev si oglejmo zadnje tri vrstice tabele. V vsaki izmed njih nastopa entiteta J, pri čemer predstavljata prvi dve en primerek (ki ustreza prvi ponovitvi

entitete v SFC receptu), tretja pa drug primerek (ki ustreza drugi ponovitvi entitete v SFC receptu).

Tabela 1: Tabelarni recept.

1.	A				
2.	B	C			
3.	D	E	C		
4.	F	E	C		
5.	G	C			
6.	H				
7.	I	J			
8.	K	J			
9.	J				
⋮	⋮	⋮	⋮	⋮	⋮

3.2.2 Algoritem izvajanja tabelarnega recepta

Preden se lotimo opisa algoritma izvajanja tabelarnega recepta, je potrebno definirati še atribut (sistemski parameter) postopkovne entitete recepta, ki določa, ali je entiteta dominantna ali odvisna. Dominantna entiteta je tista, katere zaključek predstavlja pogoj za napredovanje SFC recepta v novo stanje oziroma za prehod tabelarnega recepta v novo vrstico. Na nivoju postopkov enot so entitete vedno dominantne, saj se postopek enote vedno mora zaključiti, ko se zaključijo vse njegove faze. Zaradi tega se v nadaljevanju tega poglavja obravnava dominantnosti nanaša izključno na faze.

Za dominantno fazo je značilno, da se v določeni točki zavestno (po lastni iniciativi) zaključi (primer take faze je doziranje določene surovine, ki se zaključi, ko je dozirana predpisana količina materiala), odvisna faza pa bi se inherentno (brez zunanjega vpliva) izvajala neskončno dolgo (primer take faze je mešanje, ki se praviloma izvaja sočasno s kakšno dominantno fazo in se zaključi hkrati z njo, čas tega zaključka pa ni predvidljiv in odvisna faza nanj nima vpliva).

Če je faza dominantna, lahko do prehoda tabelarnega recepta iz tekoče vrstice, v kateri se nahaja ta dominantna faza, v novo vrstico pride šele po zaključku izvajanja te faze. Če pa je faza odvisna, do prehoda vrstice recepta pride neodvisno od te faze in ob prehodu v novo vrstico jo, če je takrat še aktivna, sistem (interpreter tabele) ustavi.

Vrednost atributa dominantnosti inherentno dominantne faze lahko uporabnik spremeni v odvisno obnašanje, s čimer odločitev o zaključku izvajanja faze preide od entitete k

interpreterju tabelarnih receptov. Obratno pa ne drži – inherentno odvisna faza v nobenem primeru ne more imeti dominantnega obnašanja, saj ni sprogramirana na način, ki bi to omogočal.

Jedrnata definicija algoritma interpreterja tabelarnih receptov se glasi: *Pogoj za napredovanje tabelarnega recepta v naslednjo vrstico je, da se je končalo izvajanje vseh dominantnih faz tekoče vrstice, ki se v naslednji vrstici ne ponavljajo kot isti primerek. Tiste odvisne faze tekoče vrstice, katerih izvajanje v trenutku izpolnjevanja pogojev za prehod v novo vrstico še ni končano, bodo ustavljene s strani interpreterja tabelarnega recepta pred prehodom v novo vrstico*.

Izvajanje tabelarnega recepta poteka dokaj intuitivno. Algoritem bomo pokazali na primeru tabele recepta iz prejšnjega podpoglavja, v katero smo zaradi lažjega sledenja dodali informacijo o dominantnosti (»*«) in primerkih posameznih faz (številka ob imenu entitete), kot prikazuje Tabela 2. Izvajanje recepta poteka na naslednji način:

1. V prvi vrstici je le dominantna faza A, torej konec izvajanja te faze omogoči prehod recepta v novo vrstico.
2. V drugi vrstici recepta nastopata dve dominantni fazi, B in C, vendar pa le konec izvajanja faze B omogoči prehod recepta v novo vrstico, saj se faza C ponavlja v naslednji vrstici kot isti primerek. Prehod v novo vrstico ne vpliva na izvajanje faze C.
3. V tretji vrstici nastopata dve dominantni fazi, D in C, ter odvisna faza E. Prehod vrstice je odvisen le od dokončanja izvajanja faze D, saj je faza E odvisna, faza C pa se v naslednji vrstici ponavlja kot isti primerek. Prehod v novo vrstico ne vpliva na izvajanje faze C, pa tudi na izvajanje faze E ne, čeprav je le-ta odvisna, in sicer zato, ker se ponavlja v naslednji vrstici kot isti primerek.
4. V četrti vrstici nastopata dve dominantni fazi, F in C, ter odvisna faza E. Prehod vrstice je odvisen le od dokončanja izvajanja faze F, saj je faza E odvisna, faza C pa se v naslednji vrstici ponavlja kot isti primerek. Prehod v novo vrstico ne vpliva na izvajanje faze C, medtem ko fazo E interpreter tabelarnega recepta ustavi pred prehodom v novo vrstico (ker se faza E ne ponavlja v naslednji vrstici).
5. V peti vrstici nastopata dve dominantni fazi, G in C. Prehod vrstice je odvisen od dokončanja izvajanja obeh faz, saj se nobena od njiju ne ponavlja v naslednji vrstici kot isti primerek (v tem primeru se fazi sploh ne ponavljata).
6. V šesti vrstici je le dominantna faza H, torej konec izvajanja te faze omogoči prehod recepta v novo vrstico.
7. V sedmi vrstici recepta nastopata dve dominantni fazi, I in J, vendar pa le konec izvajanja faze I omogoči prehod recepta v novo vrstico, saj se faza J ponavlja v naslednji vrstici kot isti primerek. Prehod v novo vrstico ne vpliva na izvajanje faze J.

8. V osmi vrstici nastopata dve dominantni fazi, K in J. Prehod vrstice je odvisen od dokončanja izvajanja obeh faz, saj se nobena od njiju ne ponavlja v naslednji vrstici kot isti primerek (K se sploh ne ponavlja, J pa se ponavlja kot drugi primerek).
9. V deveti vrstici je le dominantna faza J, torej konec izvajanja te faze omogoči prehod recepta v novo vrstico.

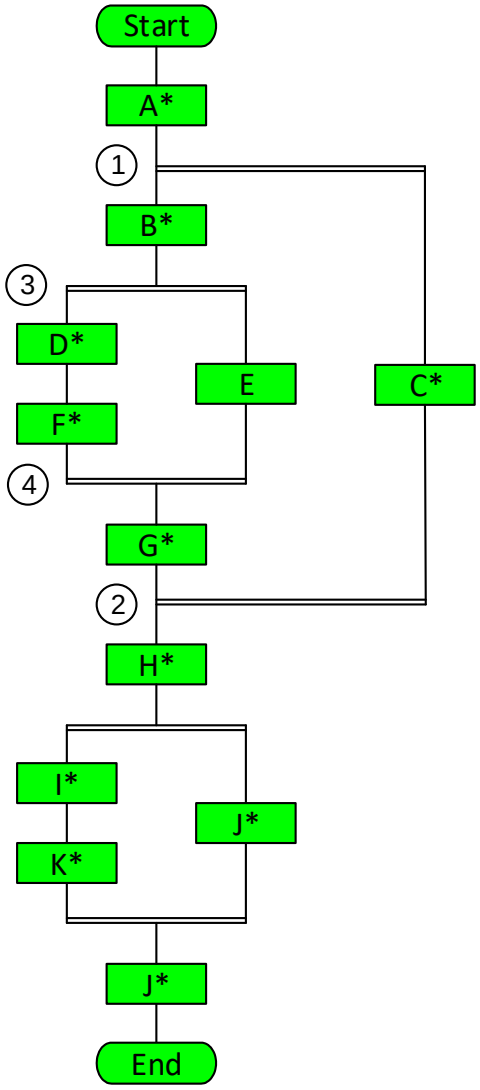
Tabela 2: Označena tabela recepta.

1.	A*:1				
2.	B*:1	C*:1			
3.	D*:1	E:1	C*:1		
4.	F*:1	E:1	C*:1		
5.	G*:1	C*:1			
6.	H*:1				
7.	I*:1	J*:1			
8.	K*:1	J*:1			
9.	J*:2				
	⋮	⋮	⋮	⋮	⋮

3.2.3 Omejitve strukture receptov

Že intuitivno lahko sklepamo, da ima predstavljen tabelarni zapis receptov nižjo izrazno moč kot SFC zapis, kar ima dve vrsti posledic. Prvič, nekaterih elementov strukture SFC receptov se v tabelarnem zapisu sploh ne da realizirati, kar pomeni, da obstajajo določene omejitve strukture SFC recepta, da bi le-tega bilo možno preslikati v tabelarni zapis. Drugič, nekaterih elementov strukture SFC se ne da preslikati v ekvivalentne elemente strukture tabele, jih je pa možno izvesti v tabelarnem receptu z modifikacijo njegove vsebine, konkretno z uporabo posebnih faz, ki jih imenujemo sistemske faze in ki bodo predstavljene v naslednjem podpoglavju.

Najprej definirajmo elemente strukture SFC receptov, ki jih sploh ni mogoče realizirati v tabelarnem receptu. Izvor omejitev je zahteva, da so pogoji za prehod vrstice tabelarnega recepta deterministični. Ta pogoj je lahko ogrožen v primeru bolj kompleksnih sočasnih razvejitev recepta. Omejitev sočasnih razmejitev je, da je le v eni sočasni veji (ki ji rečemo glavna veja) lahko več kot ena zaporedna postopkovna entiteta (ali sočasna podstruktura recepta), v vseh ostalih sočasnih vejah (ki jim rečemo pomožne veje) pa sme biti v vsaki le po ena entiteta. Če znova vzamemo naš primer SFC recepta, ki ga prikazuje Slika 4, na kateri smo dodatno označili nekaj začetnih in končnih točk razvejitev, kot to prikazuje Slika



Slika 5: Označen SFC recept.

5, lahko ugotovimo, da se med točkama 1 in 2 recept razveji na dve veji: glavna se začne z entiteto B in ima več zaporednih entitet (in novo razvejitev), pomožna pa vsebuje le entiteto C. Nadalje se glavna veja med točkama 3 in 4 znova razveji na dve veji, od katerih ima glavna dve zaporedni entiteti (D in F), pomožna pa zopet le eno entiteto, E. Torej, med izvajanjem korakov D in F se izvajajo tri sočasne veje: veja z entitetama D in F, veja z entiteto E in veja z entiteto C, le prva veja pa vsebuje več kot eno entiteto. Primera nedovoljenih struktur SFC diagrama sta z dodatno entiteto v veji z entiteto E (pred ali za njo) in dodatna entiteta v veji z entiteto C. Opisana omejitev po naših izkušnjah v večini realnih primerov ni problematična, saj praviloma v pomožnih vejah tudi želimo imeti le po eno entiteto. V primerih, ko pa bi jih želeli imeti več, so nam še vedno na voljo nekateri nekoliko bolj kompleksni prijemi, s katerimi lahko to omejitev zaobidemo. Če bi na primer hoteli imeti dodatno dominantno entiteto za entiteto C, recimo ji C', bi jo lahko izvedli kot entiteto izven recepta, ki bi jo sprožila entiteta C ob svojem zaključku. Nova entiteta bi se morala sinhronizirati z receptom v točki 2, kar bi lahko izvedli tako, da bi med entiteto G

in točko 2 dodali sinhronizacijsko sistemsko entiteto G', ki ne bi izvajala nobene funkcije, razen sinhronizacije zaključka z zaključkom entitete C'. Več o mehanizmih komunikacije in sinhronizacije med entitetami v tabelarnem receptu je mogoče prebrati v naslednjem poglavju.

Elementi strukture SFC, ki jih ni mogoče realizirati v strukturi tabelarnega recepta, jih pa je mogoče enostavno realizirati z vključitvijo raznih sistemskih faz v strukturo tabelarnega recepta, so izbirne oziroma ALI razvejitve, katerih namen, obnašanje in uporaba so opisani v naslednjem poglavju.

3.2.4 Mehanizmi upravljanja toka izvajanja tabelarnih receptov

Pri SFC receptih tok izvajanja recepta uravnavamo s pomočjo pogojev v prehodih, npr. določena vrednost izbrane procesne spremenljivke, in z ALI razvejitvami, s katerimi lahko izvajanje recepta usmerimo v eno izmed več vej, ki lahko gredo naprej ali nazaj, s čimer lahko izvedemo tudi zanko v izvajanju recepta. Ker pa tabelarni recepti inherentno omogočajo neposredno izvesti le sočasne (IN) razvejitve in je edini mehanizem kontrole toka recepta zgoraj opisan algoritem ugotavljanja pogojev za prehod vrstice recepta na osnovi dokončanja dominantnih faz, je za pogoje prehodov in za ALI razvejitve potrebno poiskati ustrezne mehanizme. Najboljši način za to je zasnova posebnega tipa faz, ki jih bomo imenovali sistemske faze (za razliko od tehnoloških faz), in sicer dveh različnih tipov: faze za kontrolo toka izvajanja recepta in faze za izvedbo interakcije z operaterjem (pošiljanje sporočil operaterju in sprejem vnosov operaterja). V naslednjih dveh podpoglavjih bomo predstavili zasnovo omenjenih sistemskih faz.

3.2.4.1 Sistemska faza »Pogoj«

Sistemska faza »Pogoj« služi za izvedbo obnašanja, ko logični izraz, ki izvaja relacijsko operacijo na določeni procesni spremenljivki, nastopa kot pogoj prehoda v SFC predstavitvi recepta. Faza deluje tako, da ciklično izvaja dano relacijsko operacijo in se zaključi, ko rezultat te operacije postane »TRUE«. Ker pa je faza dominantna, in če gre za edino dominantno fazo v tej vrstici, je s tem omogočen prehod recepta v novo vrstico. Na ta način torej dosežemo želeno obnašanje recepta, torej da do prehoda v novo vrstico pride, ko je izpolnjen pogoj oziroma ko relacijska operacija vrne vrednost »TRUE«.

Ker si želimo splošno uporabno fazo »Pogoj«, jo je treba zasnovati tako, da ob njeni uporabi lahko določimo vrsto primerjave (relacijski operator) in objekta primerjave (kaj primerjamo s čim). To lahko dosežemo z naslednjimi parametri faze:

- tip prvega operanda [*spremenljivka* | *konstanta*],
- identifikator/vrednost prvega operanda (glede na to, ali gre za spremenljivko ali konstanto),
- relacijski operator [= | <> | < | > | <= | >=],

- tip drugega operanda [*spremenljivka* | *konstanta*],
- identifikator/vrednost drugega operanda (glede na to, ali gre za spremenljivko ali konstanto).

3.2.4.2 Sistemska faza »Razvejitev«

Sistemska faza »Razvejitev« služi za izvedbo »OR« razvejitev recepta, pri katerih se pot možnega izvajanja recepta razveji v dve možni smeri glede na dani pogoj (vrednost logičnega izraza, podobno kot pri fazi »Pogoj«). Takšno obnašanje faze »Razvejitev« lahko dosežemo tako, da fazi omogočimo dostop do spreminjanja vrednosti sistemske spremenljivke z vrednostjo številke naslednje vrstice recepta. Faza se odloči glede na rezultat relacijske operacije, zato potrebuje enakih pet parametrov, kot jih ima faza »Pogoj«, poleg teh pa še dva parametra:

- Naslednji korak recepta v primeru, da je vrednost logičnega izraza »TRUE«,
- Naslednji korak recepta v primeru, da je vrednost logičnega izraza »FALSE«.

3.2.4.3 Sistemska faza »Skok«

Sistemska faza »Skok« služi za izvedbo brezpogojnega skoka na določeno vrstico tabele recepta. Torej, po zaključku tekoče vrstice (v kateri se nahaja le faza »Skok«) se izvajanje recepta ne nadaljuje z naslednjo vrstico tabele, temveč z vrstico, ki je podana s parametrom faze »Skok« (parameter *Naslednji korak*).

3.2.4.4 Sistemska faza Sporočilo

Sistemska faza »Sporočilo« služi za posredovanje tekstovnega sporočila operaterju. Vsebina sporočila je podana s parametrom faze. Predvideli smo tri vrste obnašanja faze »Sporočilo«, pri čemer se obnašanje konkretnega primerka faze v receptu opredeli s parametrom faze. Gre za naslednje tipe obnašanja:

1. Prikaz informacije brez zahtevane potrditve s strani operaterja. Sporočilo ima zgolj informativni namen in je operaterju prepuščena odločitev, ali bo sporočilo prebral ali ga ignoriral. Tako obnašanje fazo opredeli kot odvisno fazo, saj taka faza ne sme ovirati toka izvajanja recepta in se mora zaključiti, ko so izpolnjeni pogoji za prehod recepta v novo vrstico.
2. Prikaz informacije in zahteva potrditve s strani operaterja. Tako obnašanje fazo opredeli kot dominantno fazo, ki se zaključi po potrditvi operaterja.
3. Prikaz informacije, zahteva po potrditvi in nastavitev vrednosti izbrane sistemske sinhronizacijske spremenljivke na »TRUE«. Namen take faze je, da se potrditev operaterja preko sinhronizacijske spremenljivke prenese v katero koli drugo fazo v tej ali kakšni drugi tabeli recepta (npr. v drugem postopku enote). Sinhronizacijska spremenljivka je ena od spremenljivk, ki so elementi polja *Podatki šarže*. Kateri element polja se uporabi, določa parameter faze »Sporočilo«. Faza s takim

obnašanjem se obnaša kot dominantna faza.

Sistemska faza »Sporočilo« ima naslednje parametre:

- besedilo sporočila,
- tip obnašanja,
- indeks elementa polja *Podatki šarže*, ki se uporabi kot sinhronizacijska spremenljivka.

3.2.4.5 Sistemska faza »Vnos operaterja«

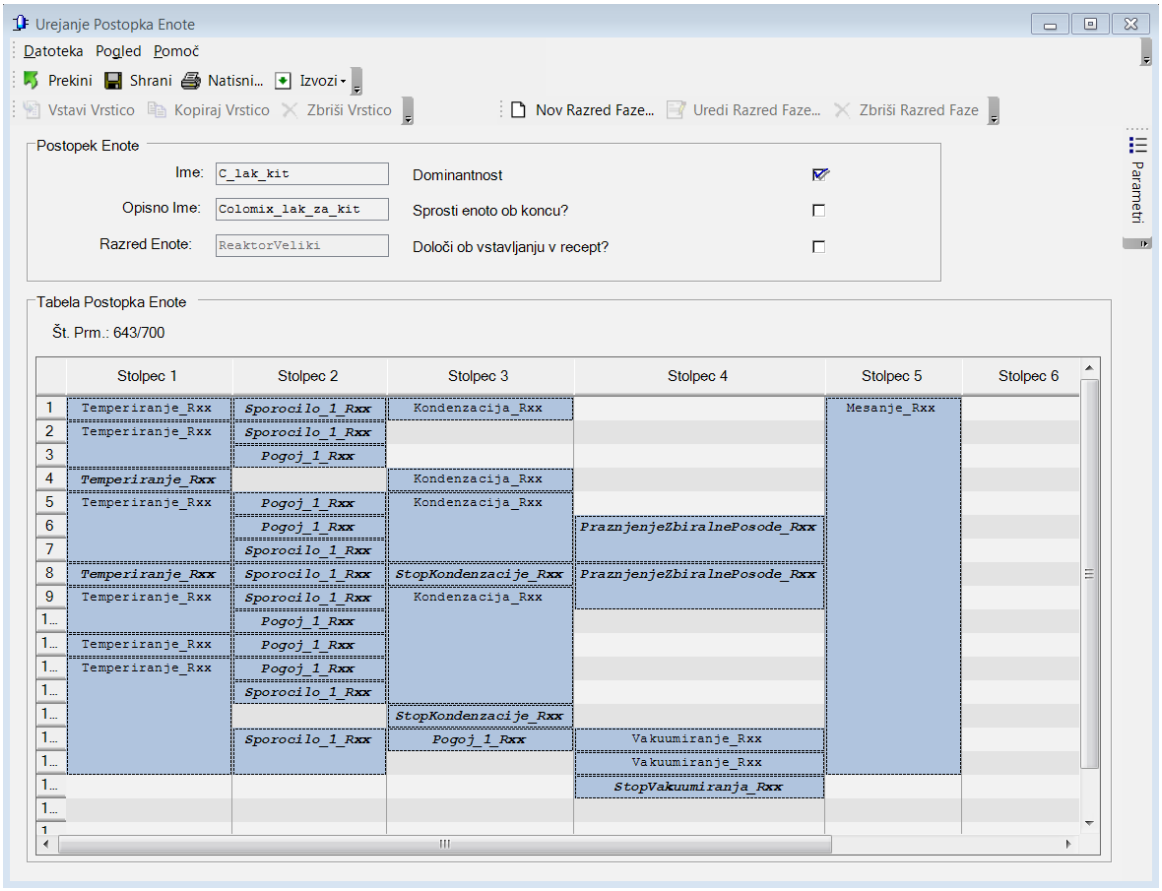
Sistemska faza »Vnos operaterja« posreduje operaterju sporočilo z zahtevo po vnosu določene informacije, sprejme vneseno informacijo in jo zapiše v določen element polja *Podatki šarže*. Faza »Vnos operaterja« ima naslednje parametre:

- besedilo sporočila zahteve po vnosu informacije,
- tip podatka za vnos,
- zgornja meja vrednosti podatka,
- spodnja meja vrednosti podatka,
- indeks elementa polja »Podatki šarže«, v katerega se zapiše vnesena vrednost.

3.3 Preizkušanje nove abstrakcije receptov in zaključki

Nova abstrakcija receptov je bila praktično preizkušena v sklopu orodja PLCbatch, in sicer v večjem številu realnih industrijskih projektov (glej poglavje 1.6 Okvir za preizkušanje rezultatov disertacije). Za ilustracijo podajamo nekoliko bolj obsežen recept iz projekta [68]. Recept prikazuje Slika 6.

Abstrakcija receptov, njena implementacija v omenjenem orodju in uporaba v projektih so bile predstavljene v večjem številu publikacij [62, 63, 64]. Izkušnje z uporabo nove abstrakcije receptov so zelo dobre, in sicer kljub manjši izrazni moči v primerjavi s SFC recepti (kot je bilo omenjeno v tem poglavju). Ta manjša izrazna moč v večini primerov ne pride do izraza, ko pride, pa je praviloma rešitev mogoče prilagoditi na način, da manjša izrazna moč ni moteča. Po drugi strani pa je sama abstrakcija izredno kompaktna, enostavna in razumljiva operaterjem, ki jo brez izjeme sprejemajo zelo pozitivno.



Slika 6: Primer recepta iz sistema vodenja procesa sinteze smol.

4 Model obnašanja faz

4.1 Analiza aktualnega modela obnašanja faz

Abstrakcija avtomatov stanj je splošno sprejeta kot zelo ustrezna za opis reaktivnih sistemov [13]. Aktualni model obnašanja faz po standardu ISA 88.00.01 prav tako temelji na abstrakciji avtomatov stanj. Zato bomo v nadaljevanju tega podpoglavja najprej podali kratek pregled in analizo te abstrakcije in nekaterih njenih pomanjkljivosti pri uporabi v sistemih vodenja procesov nasploh, nato pa se bomo še nekoliko bolj podrobno dotaknili značilnosti in nekaterih težav pri uporabi te abstrakcije v domeni vodenja šaržnih procesov, s katerim se ukvarjamo v tej disertaciji.

4.1.1 Abstrakcija avtomatov stanj in nekateri problemi

Avtomat stanj je model izvajanja, ki določa niz stanj, skozi katera gre določen objekt tekom svojega življenjskega cikla, do prehodov stanj pa prihaja zaradi dogodkov [14, Poglavje 2]. Avtomat stanj je v bistvu abstraktni stroj, ki ga določa množica stanj, začetno stanje, vhodna abeceda dogodkov in prehodna funkcija, ki preslikuje trenutna stanja in simbole dogodkov v nova stanja [15]. Pojem avtomat stanj se nanaša na model izvajanja in ne na diagram, ki ta model predstavlja; splošno sprejet naziv diagrama, ki predstavlja avtomat stanj, je *diagram stanj* ali *diagram prehajanja stanj*.

Originalni (prvotni) avtomat stanj generira izhode le, ko doseže končno stanje, vendar pa so kasneje nastale mnoge izpeljanke originalne abstrakcije, med katerimi so bili najbolj znani avtomati stanj tipa *Mealy* (ki generira izhode med prehodi stanj) [16], tipa *Moore* (ki proizvaja izhode v stanjih) [17] in avtomati stanj, ki so kombinacija obeh omenjenih tipov (*Moore-Mealy*, ki generirajo izhode tako med prehodi stanj kot tudi v stanjih).

Avtomati stanj sicer predstavljajo pomembno abstrakcijo za modeliranje obnašanja v realnem času, vendar pa zanje velja, da predstavljajo abstrakcijo na dokaj nizkem nivoju v smislu obvladovanja kompleksnosti razvoja velikih oziroma zahtevnih sistemov. V poznih osemdesetih letih dvajsetega stoletja je Harel definiral vizualni formalizem za opisovanje stanj in prehodov na modularen način z uporabo mehanizmov združevanja (ang. *clustering*), sočasnosti oziroma ortogonalnosti (ang. *orthogonality*) in prečiščevanja (ang. *refinement*) ter z omogočanjem »zumiranja« med posameznimi nivoji abstrakcije. Ta nov formalizem je poimenoval *Statecharts*, prvič pa ga je predstavil v [17]. Pri tej novi abstrakciji je dejansko šlo za diagrame prehajanja stanj, ki so bili nadgrajeni s koncepti hierarhije (oziroma globine), ortogonalnosti (oziroma sočasnosti) in izmenjave sporočil

(ang. *broadcast communication*) [17, 18]. Prvi predstavitvi novega modela so hitro sledile nove objave Harel-a in drugih avtorjev, v katerih je bila definirana preliminarna semantika formalizma *Statecharts* [19, 20]. Formalizem *Statecharts* še zdaleč ni bil končni izdelek, temveč se je z leti postopoma razvijal, ob čemer je nastalo več njegovih izpeljank. Tako je, na primer, leta 1994 že obstajalo najmanj 20 izpeljank abstrakcije *Statecharts* [21]. Prvotno verzijo abstrakcije se v literaturi večinoma poimenuje bodisi enostavno kot *Statecharts*, *Harel-ovi Statecharts* ali *Klasični Statecharts*. Klasični *Statecharts* so na primer implementirani v orodju *Statemate* [18, 22].

Danes obstaja večje število dialektov avtomatov stanj, ki so nastali bodisi kot izpeljanke abstrakcije Harel-ovih *Statecharts* bodisi vsaj pod močnim vplivom te abstrakcije, npr. avtomati stanj modelirnega jezika UML (kot so določeni v UML 2.4.1 [23]) ali novejša objektno usmerjena izpeljanka Harel-ovih *Statecharts* (implementirana v modelirnem in razvojnem okolju Rhapsody [24]). Čeprav so vsi omenjeni formalizmi na prvi pogled zelo podobni, obstajajo med njimi nekatere subtilne sintaktične in semantične razlike [13].

V domeni vodenja procesov se prav tako uporabljajo nekateri dialekti avtomatov stanj, ki so podobni zgoraj omenjenim dialektom. Najnovejši standard na področju sistemov vodenja industrijskih procesov in avtomatizacije v industriji je IEC 61499 [25], ki predstavlja razširitev idej iz standarda IEC 61131-3 ter podpira fazo načrtovanja sistemov in porazdeljene sisteme procesnega vodenja. Programi, skladni z IEC 61499, sestojijo iz mrež funkcijskih blokov (ang. *Function Block*, s kratico FB). Vsak funkcijski blok ima podatkovne vhode in izhode, ki jih uporabljajo notranji algoritmi funkcijskega bloka, poleg tega pa tudi dogodkovne vhode in izhode, ki jih uporablja tako imenovani diagram vodenja izvajanja (ang. *Execution Control Chart*, s kratico ECC). ECC je posebna vrsta diagrama *statechart*, ki določa obnašanje funkcijskega bloka s sekvenciranjem klicev algoritmov funkcijskega bloka. Drug pristop pri sistemih procesnega vodenja, podoben pristopu IEC 61499 glede na abstrakcije, ki jih uporablja in glede na njegovo izrazno moč, je pristop iz okolja Matlab Simulink/Stateflow. Glede na [26] lahko govorimo o naravni komplementarnosti med okoljem Simulink/Stateflow in funkcijskimi bloki po IEC 61499, pri čemer Simulink/Stateflow predstavlja primerno okolje za modeliranje in simulacijo sistemov vodenja in vgrajenih sistemov, funkcijski bloki pa so dobri za načrtovanje porazdeljenih sistemov vodenja. Po drugi strani pa se lahko tudi Simulink/Stateflow uporabi v fazi načrtovanja, zahvaljujoč njegovi zmožnosti generiranja programske kode v jeziku C ali v jezikih za programiranje PLK-jev v skladu s standardom IEC 61131-3.

Vsi zgoraj omenjeni formalizmi avtomatov stanj so v bistvu enaki glede na njihov temeljni model izvajanja na osnovi avtomata stanj. V tem kontekstu je za našo obravnavo pomembno predvsem to, da je vsem omenjenim formalizmom skupno, da ima le procesiranje v stanjih (to je *loop* ali *do* procesiranje) lahko trajanje, medtem ko so vsa druga procesiranja (*entry* in *exit* procesiranji stanj in procesiranje prehodov) lahko le trenutna. Tak model, po naših izkušnjah iz prakse, ni najbolj primeren za modeliranje programske opreme sistemov vodenja industrijskih procesov, še posebej če gre za zvezne ali šaržne

procese, ki so v svojih reakcijah praviloma počasni (v smislu reaktivnih sistemov). Pri takih počasnih sistemih bi pogosto prišlo zelo prav, če bi *entry* ali *exit* procesiranje ali procesiranje prehodov imelo trajanje. Najboljši primer takega scenarija je sekvenca, ki med izvajanjem posreduje ukaz za spremembo stanja določeni napravi (osnovnemu vodenju) ter nato čaka na povratno informacijo iz osnovnega vodenja, da se je stanje naprave spremenilo. Za primer tega vzemimo scenarij, ko je ob vhodu v določeno stanje treba odpreti zaporni ventil in nato počakati, da se ventil dejansko odpre pred začetkom izvajanja *loop* akcij tega stanja, kar je dokaj pogosta zahteva. To zahtevo lahko izvedemo v *entry* sekvenci stanja, ki najprej posreduje ukaz za odpiranje zapornega ventila ter nato čaka na informacijo iz osnovnega vodenja, da je ventil prešel v stanje *odprto* (kar osnovno vodenje ugotovi na osnovi signalov končnih stikal ventila) ter se šele nato zaključi in s tem omogoči začetek izvajanja *loop* sekvence. Trajanje takih sekvenc tipično merimo v sekundah, saj večji ventil potrebuje toliko za prehod iz enega v drugi končni položaj. Obstajajo pa seveda tudi še bistveno počasnejše naprave oziroma elementi osnovnega vodenja, kjer prehodne čase merimo tudi v minutah. V trenutno aktualnih formalizmih avtomatov stanj se lahko omenjeno počasno procesiranje izvede skozi ločevanje tistih sekvenc procesiranja, ki naj bi imele trajanje, od samega avtomata stanj, ki potem zgolj proži omenjene sekvence iz svojih sekvenc brez trajanja oziroma iz trenutnih akcij, npr. *entry* akcij, *exit* akcij ali akcij prehodov. Očitno je, da tako ločevanje obnašanja in procesiranja pripelje do težav zaradi potrebe po sinhronizaciji zaključka teh, iz avtomata stanj izločenih procesiranj s procesiranji, ki so ostala znotraj avtomata stanj, npr. z *loop* procesiranjem. Druga možna rešitev v okviru obstoječih formalizmov je uvajanje dodatnega stanja, imenovanega npr. *Odpiranje ventila*, katerega negativna posledica je nižanje granularnosti elementov avtomata stanj in s tem tudi nižanje nivoja abstrakcije, kar pa je po našem prepričanju zelo nezaželeno. (Kakšna je primernost uvajanja stanja *Odpiranje ventila* v kontekstu procesno usmerjenega pogleda, ki bi mu želeli slediti?).

Zgoraj opisane težave v zvezi z uporabo trenutno aktualnega modela avtomatov stanj pri razvoju programske opreme sistemov vodenja so značilne za vse poddomene domene vodenja, torej tako za domeno šaržnih procesov, kot tudi domene zveznih, sekvenčnih ali diskretnih procesov. Obstajajo pa tudi še nekatere dodatne težave, ki so značilne za domeno šaržnega vodenja, in ki jih bomo predstavili in analizirali v naslednjem podpoglavju.

4.1.2 Abstrakcija avtomatov stanj pri vodenju šaržnih procesov

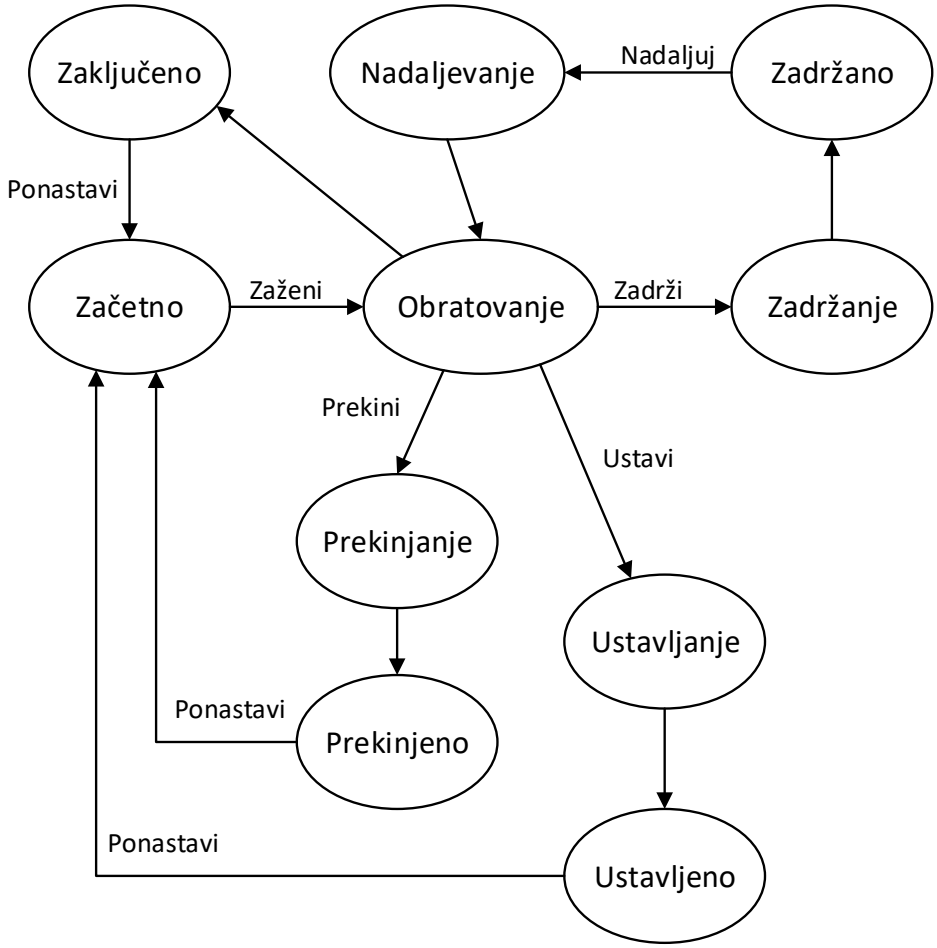
Danes je pri večini strokovnjakov na področju sistemov za vodenje šaržnih procesov prisotno mnenje, ki ga delimo tudi mi, da je zgoraj predstavljena abstrakcija avtomatov stanj zelo primerna tudi za opis obnašanja entitet postopkovnega vodenja šaržnih procesov. Zaradi tega standard ISA 88.00.01 priporoča uporabo modela obnašanja faz, ki temelji na abstrakciji avtomatov stanj. Omenjeni model obnašanja faz je v standardu prikazan na dva načina, in sicer z matriko prehajanja stanj in z diagramom prehajanja stanj. Za diagram

sicer v standardu piše, da se nanaša le na prve tri vrstice matrike, torej na stanja *Začetno* (ang. *Idle*), *Obratovanje* (ang. *Running*) in *Zaključeno* (ang. *Complete*), vendar pa po našem mnenju temu ni tako. Diagram vsebuje veliko več od navedenega in prej daje vtis neuspešnega poskusa prikaza celotnega modela obnašanja, zato ga bomo v nadaljevanju obravnavali kot takega. Tukaj naj omenimo še, da smo v matriki in diagramu prehajanja stanj izpustili stanji *Pavziranje* (ang. *Pausing*) in *Pavzirano* (ang. *Paused*), ker se v tem delu s tema stanjema ne bomo ukvarjali in njihovo izpuščanje ne vpliva na diskusijo o preostanku modela obnašanja. Stanji smo se odločili izpustiti, ker po našem prepričanju to v bistvu nista stanji, temveč dve podstanji stanja *Obratovanje,* ki se izmenjujeta med polavtomatskim (ang. *semi-auto*) načinom izvajanja fazne logike. Med stanjem *Pavziranje* se izvajanje fazne logike izvaja do prve vnaprej predvidene in sprogramirane točke pavziranja v fazni logiki, kjer se izvajanje ustavi in faza pride v stanje *Pavzirano*. V tem stanju ostane do ukaza *Obnovi* (ang. *Resume*), ko se izvajanje fazne logike nadaljuje do naslednje sprogramirane točke pavziranja. Število točk pavziranja je lahko poljubno. Omenjeno reducirano matriko prehajanja stanj iz standarda ISA 88.00.01 prikazuje Tabela 3, reducirani diagram prehajanja stanj pa Slika 7.

Model vsebuje deset stanj, med katerimi so tri končna stanja (ang. *final state*), šest prehodnih stanj (ang. *transient state*) in eno mirujoče stanje (ang. *quiescent state*). Končna stanja so *Zaključeno* (ang. *Complete*), Ustavljeno (ang. *Stopped*) in *Prekinjeno* (ang. *Aborted*). Prehodna stanja so *Začetno* (ang. *Idle*), *Obratovanje* (ang. *Running*), *Zadržanje* (ang. *Holding*), *Nadaljevanje* (ang. *Restarting*), *Ustavljanje* (ang. *Stopping*) in *Prekinjanje* (ang. *Aborting*). Mirujoče stanje je *Zadržano* (ang. *Held*).

Tabela 3: Matrika prehajanja stanj faz po ISA 88.00.01.

Ukaz		Zaženi	Ustavi	Zadrži	Nadaljuj	Prekini	Ponastavi
Izhodiščno stanje	Končno stanje brez ukaza	Matrika prehajanja stanj					
Začetno		Obratovanje					
Obratovanje	Zaključeno		Ustavljanje	Zadržanje		Prekinjanje	
Zaključeno							Začetno
Zadržanje	Zadržano		Ustavljanje			Prekinjanje	
Zadržano			Ustavljanje		Nadaljevanje	Prekinjanje	
Nadaljevanje	Obratovanje		Ustavljanje	Zadržanje		Prekinjanje	
Ustavljanje	Ustavljeno					Prekinjanje	
Ustavljeno						Prekinjanje	Začetno
Prekinjanje	Prekinjeno						
Prekinjeno							Začetno



Slika 7: Diagram prehajanja stanj faz po ISA 88.00.01.

Sistem vodenja šaržnega procesa se obnaša v skladu z zgoraj podanim modelom stanj, in sicer na naslednji način. Standard določa le nabor stanj, ne pa tudi strukture procesiranja, katere zasnova je prepuščena izdelovalcem komercialnih orodij za vodenje šaržnih procesov, ki pa so večinoma ubrali enostavne pristope. Večina orodij, oziroma vsa najbolj razširjena (npr. [70, 71, 72]), procesiranje umešča le v pet aktivnih stanj, in sicer za vsako aktivno stanje le po eno sekvenco. Aktivna stanja so vsa prehodna stanja, razen stanja *Začetno*, torej *Obratovanje*, *Zadržanje*, *Nadaljevanje*, *Ustavljanje* in *Prekinjanje*.

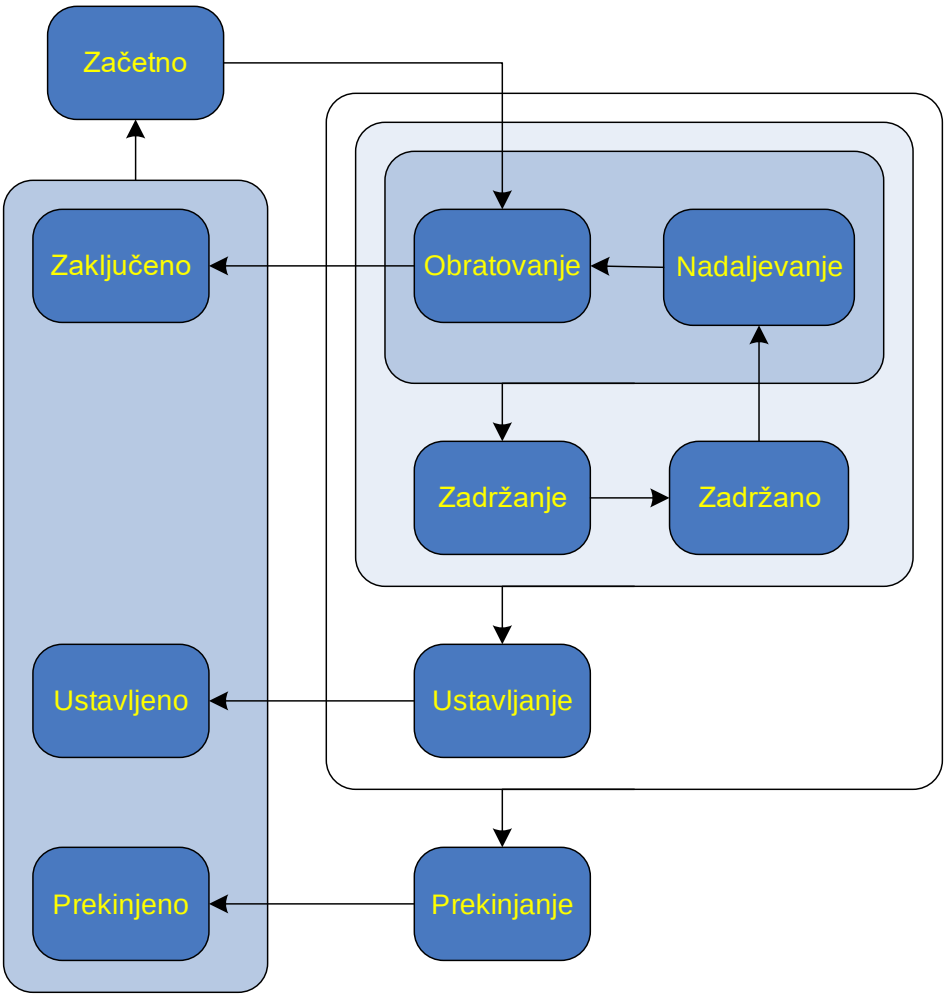
Program za vodenje na PLK je sestavljen iz dveh delov, sistemskega dela in aplikacijskega dela. Sistemski del predstavlja tako imenovani vmesnik fazne logike (ang. *Phase Logic Interface*, s kratico PLI), ki izvaja algoritem avtomata stanj za vsako posamezno fazo. Ta algoritem po eni strani preverja vzroke/pogoje za izvajanje prehodov stanj ter te prehode tudi izvaja, po drugi strani pa upravlja (zaganja in ustavlja) izvajanje sekvenc procesiranja, ki pripadajo posameznim stanjem. Aplikacijski del programa za vodenje sestavljajo posamezni programski bloki, kjer imamo po en blok za vsako aktivno stanje določene faze, torej po pet blokov za vsako fazo. Ti bloki so medsebojno nepovezani, iz česar sledi, da na nivoju programske kode ni nobenih integriranih entitet za vsako

posamezno fazo in da le PLI »vidi« posamezno fazo kot celoto oziroma kot konceptualno entiteto.

Čeprav se zgoraj opisani model obnašanja faz na prvi pogled morda zdi primeren, se ob njegovi uporabi v praksi pojavljajo tudi nekatere težave, katerih glavni vzrok je v pregrobi granulaciji posameznih modulov aplikacijske kode. Večina bi se verjetno strinjala, da je najboljši način za doseganje visoke kakovosti programske opreme za vodenje šaržnih procesov (kot tudi programske opreme nasploh) v taki dekompoziciji, pri kateri so posamezni gradniki kar se da enostavni in razumljivi. Po drugi strani pa so osnovni gradniki aplikacijske kode v skladu z zgoraj opisanim modelom celotne sekvence posameznih stanj, ki so lahko dokaj obsežne in kompleksne. Glavni vzrok te kompleksnosti je v premajhnem številu gradnikov kode, zaradi česar posamezni gradnik vsebuje več procesiranja, kot bi bilo optimalno z vidika obvladovanja kompleksnosti.

Omenjena težava po našem mnenju izvira iz uporabe skrajno enostavnega modela obnašanja faz, temelječega na elementarnih avtomatih stanj, brez gnezdenih stanj in z le po eno sekvenco za vsako aktivno stanje. Neustreznost tega modela bomo ilustrirali na nekaterih podrobnostih standarda ISA 88.00.01 in dokumentaciji enega od komercialnih orodij za vodenje šaržnih procesov.

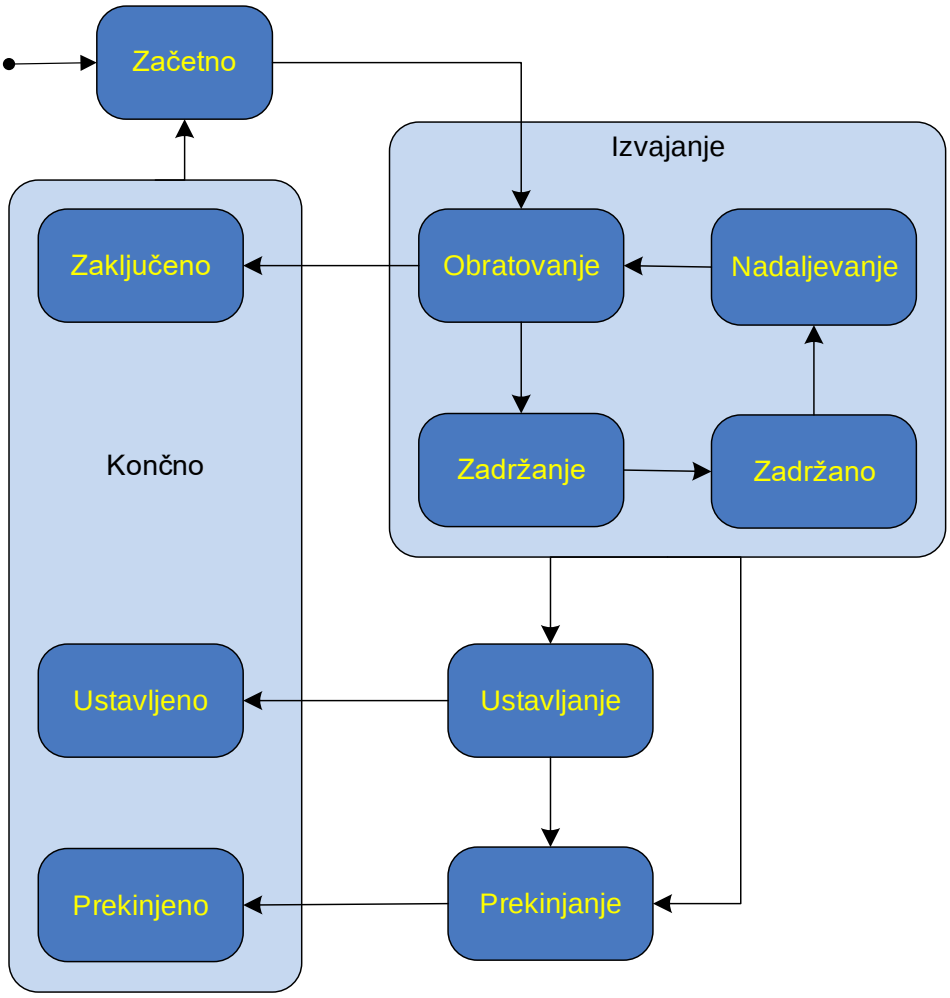
V standardu celoten model obnašanja faz ni, kot bi pričakovali, podan v obliki diagrama prehajanja stanj na ustreznem nivoju abstrakcije, temveč je podan v obliki tabele prehajanja stanj. Dodatno je podan tudi delni diagram prehajanja stanj, ki pa pokriva le kakšno tretjino celotnega obnašanja, podanega s tabelo. In zakaj je kot glavna notacija izbrana tabela in ne diagram prehajanja stanj? Razlog je očitno v tem, da bi opisovanje celotnega obnašanja faze le na osnovi enostavnega diagrama stanj brez nadstanj pripeljalo do zapletene mreže ponavljajočih se prehodov stanj, kar bi slabo vplivalo na razumljivost modela obnašanja. Po drugi strani pa je tabela polna podvajanja informacij: prehod v stanje *Prekinjeno* nastopa osemkrat, prehod v stanje *Ustavljeno* pa šestkrat, medtem ko bi v ustrezno zasnovanem strukturiranem diagramu stanj, kot ga prikazuje Slika 8, vsak od teh prehodov nastopal le enkrat. Imamo torej veliko mero podvajanja informacij pri prehodih, podvajanje informacij pa predstavlja velik potencialen vir napak in je zato ena od najhujših negativnih lastnosti programske opreme. Podobno kot pri prehodih je tudi pri procesiranju, saj obstaja velika verjetnost, da bodo deli procesiranja posameznih sekvenc medsebojno enaki oziroma da bo določen del procesiranja nastopal v več kot eni sekvenci. Tudi temu bi se seveda bilo mogoče izogniti, če bi model obnašanja vseboval nadstanja. Tukaj naj omenimo, da so tudi avtorji standarda prišli do podobnih zaključkov, zato se v zadnji verziji iz leta 2010 [76], ob drugih spremembah in dopolnitvah nahaja tudi diagram prehajanja stanj, ki je praktično enak temu, ki ga prikazuje Slika 8.



Slika 8: Strukturiran diagram prehajanja stanj faz po ISA 88.00.01.

Kot drugi primer si bomo ogledali tehnično dokumentacijo enega od komercialnih orodij za vodenje šaržnih procesov, ki je po našem vedenju eno izmed najbolj razširjenih [77]. V tem priročniku je model stanj predstavljen z diagramom prehajanja stanj, ki vsebuje tudi nadstanja: *Izvajanje* (ang. *Executing*) in *Končno* (ang. *Terminated*), kar očitno poveča izrazno moč modela. Vendar pa diagram ni optimalen, saj tudi tukaj nekateri prehodi nastopajo več kot enkrat: npr. prehod v stanje Aborting nastopa dvakrat (enkrat iz nadstanja Executing in enkrat iz stanja Stopping). Jasno je, da tukaj manjka en nivo v hierarhiji stanj. Omenjeni diagram prehajanja stanj prikazuje Slika 9, pri čemer smo tudi tukaj izpustili semi-auto način in pripadajoči podstanji *Pausing* in *Paused*.

Za ilustracijo zgoraj omenjene trditve, da je abstrakcija, ki vsebuje podvajanje informacij, potencialen vir napak, bomo pokazali na nekatere tovrstne slabosti v standardu in v dokumentaciji omenjenega orodja.



Slika 9: Poenostavljen diagram prehajanja stanj faz komercialnega orodja.

V standardu oziroma v njem definirani tabeli prehajanja stanj bomo omenili dve slabosti. Prvič, v tabeli je odvečni prehod iz stanja *Ustavljeno* v stanje *Prekinjanje*. Drugič, manjka prehod iz *Zadržano* v *Zadržanje*. Slednje sicer ne predstavlja nujno slabosti, ker ta prehod seveda ni obvezen, vendar pa je vsekakor koristen v praksi, pa tudi vsa nam znana orodja ga podpirajo.

V diagramu prehajanja stanj, prikazanem v tehnični dokumentaciji omenjenega orodja [77], manjka veliko število prehodov, npr. iz *Nadaljevanje* v *Zadržanje*, iz *Zadržano* v *Zadržanje*, iz večjega števila stanj (*Nadaljevanje*, *Zadržano*, *Zadržanje*) v stanje *Ustavljanje* ter iz večjega števila stanj (*Nadaljevanje*, *Zadržano*, *Zadržanje*, *Ustavljanje*) v stanje *Prekinjanje*. Pri tem je treba poudariti, da je večina teh prehodov podprtih v samem orodju in tudi omenjenih v drugih delih dokumentacije, v diagramu prehajanja stanj pa jih ni. Omenjena pomanjkljivost bi se morda lahko komu zdela nepomembna, saj navsezadnje gre le za napako v dokumentaciji, vendar pa smo mi prepričani v nasprotno. Gre za to, da je dokumentacija pomembna za razumevanje in pravilno uporabo orodja. Dokumentacija naj bi na višjem nivoju abstrakcije na razumljiv način opisovala tisto, kar je implementirano na nizkem nivoju v kodi orodja. Podobno je tudi pri razvoju – del sistema najprej zasnujemo

in modeliramo z uporabo visokonivojskih abstrakcij na način, da ga je čim lažje možno razumeti, preveriti njegovo pravilnost in preslikati v kodo, izdelek na nivoju kode pa je praviloma bistveno težje razumeti in preveriti njegovo pravilnost. Dejstvo, da je v omenjenem orodju obnašanje na nivoju kode pravilno, v dokumentaciji (in to v njenem najpomembnejšem delu, kar bi grafični visokonivojski model obnašanja vsekakor moral biti) pa napačno, kaže na to, da se je orodje razvijalo direktno na nivoju kode, naknadno pa se je, šele pri izdelavi dokumentacije, narisalo grafični model obnašanja, ki pa je napačen. In ravno to je paradoks – koda na nizkem nivoju je pravilna, visokonivojski model pa napačen. Za konec povejmo še to, da gre za eno najbolj razširjenih orodij, ki v raznih verzijah obstaja že več kot dvajset let, omenjena napaka pa se ponavlja v prav vseh verzijah njegove dokumentacije.

Do zgoraj opisanih slabosti je prišlo kljub temu, da sta oba omenjena dokumenta ([1] in [77]) prav gotovo bila zelo pozorno napisana in preverjena. Po našem mnenju je glavni vzrok teh slabosti v neustreznih abstrakcijah, ki se jih uporablja za opisovanje obnašanja faz.

Če pomanjkljivosti modela obnašanja faz in njihove posledice lahko zaznamo v zgoraj omenjenih pozorno napisanih in preverjenih dokumentih, je vpliv neustreznega modela na aplikativno programsko opremo lahko le še slabši. Težava je najbolj očitna pri stanju *Obratovanje*, ki ima različna obnašanja v odvisnosti od predhodnega stanja (stanja, iz katerega je prišlo do prehoda v stanje *Obratovanje*). Ilustrirajmo to s primerom iz prakse, vzemimo primer faze *Doziranje*. Pri tej fazi navadno mora biti ob prehodu v stanje *Obratovanje* iz stanja *Začetno* (torej ob zagonu faze) ponastavljen dozirni števec, medtem ko mora ob prehodu v stanje *Obratovanje* iz stanja *Nadaljevanje* vrednost tega števca ostati nespremenjena. Podobne primere bi lahko našli tudi pri drugih stanjih.

Omenjenih težav se razvijalci zavedajo in iščejo različne rešitve, ki pa so žal večinoma neustrezne. Za ilustracijo si še enkrat oglejmo prej omenjeno dokumentacijo orodja za vodenje šaržnih procesov, kjer se omenjata dve paradigmi programiranja faz z uporabo jezika SFC: sekvenčno paradigmo in paradigmo avtomata stanj. Vsak od teh pristopov ima tako prednosti kot pomanjkljivosti:

- Sekvenčna paradigma je bolj intuitivna in lažje razumljiva, ima pa premajhno izrazno moč zaradi nizkega nivoja abstrakcije, ki povzroča težave pri opisovanju bolj kompleksnih obnašanj, npr. izjem;
- Paradigma avtomatov stanj je na višjem nivoju abstrakcije in ima posledično večjo izrazno moč, kar olajšuje obvladovanje kompleksnih obnašanj, po drugi strani pa je težje razumljiva in lahko pripelje do prezapletenega aplikativnega programa.

Po našem prepričanju je bistvo težav v porazdelitvi funkcionalnosti med PLI in aplikativnim programom. S to delitvijo se poleg funkcionalnosti razdeli tudi kompleksnost. Celotno obnašanje ima pač določeno kompleksnost, ki je seveda odvisna od inherentne kompleksnosti realnega šaržnega sistema, ki ga aplikacija vodi. Trenutna rešitev pri večini

orodij je taka, da je model obnašanja faz, implementiran v PLI in izvajan s strani PLI, relativno enostaven. To seveda pomeni večjo kompleksnost aplikativnega programa, kar potem pogosto prinaša težave, še posebej v primeru aplikacij z bolj zahtevno funkcionalnostjo. Težave seveda izvirajo iz tega, da je v tem primeru velik del kompleksnosti prepuščen aplikativnemu programerju, kar pogosto – in nepredvidljivo – prinaša težave pri obvladovanju kompleksnosti projekta.

Možna rešitev zgoraj opisanih težav bi bila v zviševanju nivoja abstrakcije in posledično izrazne moči modela obnašanja faz na način, da bi bil aplikativni program sestavljen le iz enostavnih sekvenc. Torej, namen je na sistemskem nivoju »zakomplicirati« model obnašanja faz, ki ga izvaja PLI in ki se ga načrtuje in programira le enkrat, da bi kolikor se le da poenostavili aplikativne sekvence, ki se jih razvija za vsak projekt posebej (čeprav seveda tudi tukaj težimo k čim višji stopnji ponovne uporabe, vendar pa je to druga zgodba).

Zgoraj omenjene težave bi lahko zmanjšali z uvedbo modela obnašanja faz manjše granularnosti, kar pomeni, da bi bil sestavljen iz večjega števila manjših elementov dekompozicije obnašanja. Pri tem se kot najustreznejša izbira kaže tak model, ki bi se od trenutno prevladujočega modela razlikoval po treh pomembnih lastnostih.

Prva lastnost je v uvedbi hierarhije stanj (nadstanj in podstanj), katerih namen je odpravljanje podvajanja informacij, ki je ena izmed najslabših lastnosti programske opreme. Z uvedbo hierarhije stanj se podvajanju informacij izognemo tako, da nadstanjem priredimo obnašanje (akcije in prehode), ki je skupno vsem njihovim podstanjem.

Druga lastnost je bolj drobna granularnost akcij. Namesto uporabe le petih aktivnih sekvenc (po eno za vsako od petih aktivnih stanj v današnjih orodjih) predlagamo razširitev množice aktivnih stanj in uvajanje različnih tipov akcij, kot so npr. vhodne akcije (ang. *Entry*), akcije v zanki (ang. *Loop*), izhodne akcije (ang. *Exit*) in akcije, ki se izvajajo ves čas med aktivnostjo stanja (ang. *Always*), poleg omenjenih akcij stanj pa še akcije prehodov. Torej, z uvedbo modela obnašanja faz bolj drobne granularnosti bi dosegli boljšo modularizacijo aplikativne programske opreme in bi se približali cilju, da bi bila programska koda zgrajena zgolj iz enostavnih sekvenc.

Tretja lastnost je v tem, da imajo lahko posamezne sekvence trajanje. To se ne nanaša le na procesiranje v zanki (kot je to npr. pri notaciji *Statecharts*), temveč tudi na procesiranje v vhodnem (ang. *Entry*) in izhodnem (ang. *Loop*) procesiranju ter v procesiranju prehodov. To velja seveda za vsa procesiranja na vseh nivojih hierarhije stanj (vsa podstanja oziroma vsa nadstanja). Razlogi za uvajanje te lastnosti so podani v predhodnem podpoglavju (4.1.1 Abstrakcija avtomatov stanj).

Ideja in teoretične osnove za uvedbo omenjenih treh lastnosti v model obnašanja faz izhajajo iz našega domensko specifičnega modelirnega jezika ProcGraph [55], pri katerem so aktivnosti sestavni del avtomata stanj, kar zagotavlja bistveno lažjo sinhronizacijo posameznih delov procesiranja.

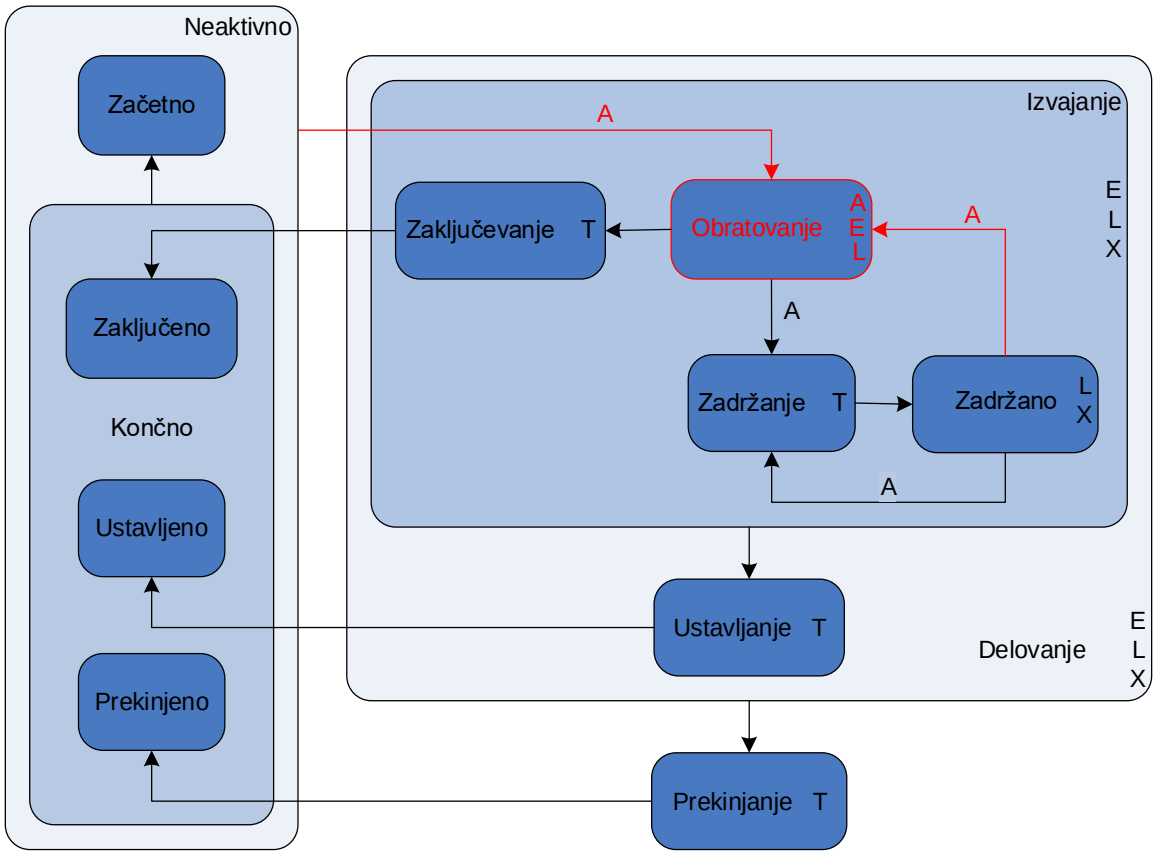
4.2 Nov model obnašanja faz

Predlagani nov model obnašanja faz izhaja iz modela, ki ga predlaga standard ISA 88.00.01. Model je razširjen z gnezdenimi stanji in drobnejše razčlenjenim procesiranjem z namenom doseganja boljših lastnosti, predvsem glede na enostavnost, transparentnost, razumljivost in zanesljivost aplikativne programske opreme. Pri nadgradnji modela obnašanja smo izhajali iz modela obnašanja entitet postopkovnega vodenja, ki smo ga določili pri našem domensko specifičnem modelirnem jeziku ProcGraph. Nov model obnašanja faz je predstavljen v [78].

4.2.1 Obnašanje faz

Nov model obnašanja faz prikazuje Slika 10. Razlike med novim modelom in modelom po standardu ISA 88.00.01 so naslednje:

1. Ukinitev stanja *Nadaljevanje*, saj procesiranje tega stanja lahko umestimo v druge elemente modela.
2. Uvedba gnezdenih stanj (elementarna stanja in več nadstanj).
3. Uvedba strukture procesiranja zelo fine granularnosti:
   a) Sekvence akcij stanj, za vsako stanje možnih pet različnih sekvenc:
      - Sekvenca *Vhod*, ki se izvede le enkrat ob prehodu v to stanje; na sliki (Slika 10) so stanja, ki imajo sekvenco *Vhod* označeno s črko E (od ang. *Entry*),
      - Sekvenca *Zanka*, ki vsebuje notranje procesiranje faze, ki se izvaja ciklično, z začetkom izvajanja po zaključku sekvence *Vhod* in zaključkom izvajanja pred začetkom izvajanja sekvence *Izhod*; na sliki (Slika 10) so stanja, ki imajo sekvenco *Zanka* označeno s črko L (od ang. *Loop*),
      - Sekvenca *Izhod*, ki se izvede le enkrat ob izhodu iz tega stanja; na sliki (Slika 10) so stanja, ki imajo sekvenco *Izhod* označeno s črko X (od ang. *eXit*),
      - Sekvenca *Vedno*, ki se izvaja ciklično ves čas, dokler je faza v tem stanju, sočasno z vsemi drugimi sekvencami stanja, vključno s sekvencami prehodov v to stanje; na sliki (Slika 10) so stanja, ki imajo sekvenco *Vedno* označeno s črko A (od ang. *Always*),
      - Sekvenca *Prehodno*, ki jo imajo prehodna stanja in ki se izvede le enkrat med prehodom skozi prehodno stanje; na sliki (Slika 10) so stanja, ki imajo sekvenco *Prehodno* označeno s črko T (od ang. *Transient*),
   b) Sekvence akcij prehodov, ki so v bistvu specifične vhodne sekvence stanja ponora (specifične zato, ker so odvisne od stanja izvora oziroma se razlikujejo glede na stanje izvora) in jih zato smatramo za procesiranje tega stanja, ki se izvede pred sekvenco Vhod tega stanja; na sliki (Slika 10) so prehodi, ki imajo sekvenco procesiranja označeno s črko A (od ang. *transition Actions*).



Slika 10: Nov model obnašanja faz.

V tej točki naj ponovimo, da imajo lahko posamezne sekvence trajanje. Še enkrat in glede na konkretne predlagane nove sekvence povejmo, da se to ne nanaša le na sekvence *Vedno* in *Zanka* (kot je npr. primer pri *Loop* sekvenci pri jeziku Statecharts ali pri diagramih stanj jezika UML), temveč tudi na sekvence stanj *Vhod*, *Izhod* in *Prehodno* ter sekvence prehodov.

Kot dodatno podrobnost naj omenimo še, da se v primeru, če so ob vhodu v stanje izpolnjeni pogoji za prehod iz njega v novo stanje, sekvenci *Vhod* in *Zanka* ne izvedeta, sekvenca *Izhod* pa se izvede v vsakem primeru.

4.2.2 Vrste stanj in prehodov

Nov model obnašanja faz vsebuje več vrst stanj, in sicer glede na dva kriterija. Glede na kriterij procesiranja stanja delimo na:

1. Neaktivna stanja, ki ne vsebujejo nobenega procesiranja in so prikazana na levem delu slike (Slika 10); pri tem gre za nadstanje *Neaktivno* in njegova podstanja.
2. Aktivna stanja, ki vsebujejo določeno procesiranje in so prikazana na desnem

delu slike (Slika 10); pri tem gre za nadstanje *Delovanje* in njegova podstanja ter stanje *Prekinjanje*.

Glede na kriterij trajanja stanja delimo na naslednji način:

1. Prehodna stanja so aktivna stanja, ki vsebujejo le eno sekvenco z določenim trajanjem; ko se ta sekvenca zaključi, pride do prehoda v novo stanje. Vsako prehodno stanje vsebuje le sekvenco *Prehodno*.
2. Trajajoča stanja so stanja, v katerih faza praviloma ostane dalj časa. Trajajoča stanja so lahko aktivna ali neaktivna. Za procesiranje aktivnih trajajočih stanj je značilno, da vsebujejo ciklično komponento, ki je vsebovana v sekvenci *Zanka*. Celotno procesiranje aktivnih trajajočih stanj je sicer razdeljeno v večje število sekvenc, od katerih je lahko po ena od tipov *Vhod*, *Zanka*, *Izhod* in *Vedno*.

Neaktivna stanja (ki so vsa hkrati tudi trajajoča stanja) so elementarna stanja *Začetno*, *Zaključeno*, *Ustavljeno* in *Prekinjeno* ter nadstanja *Končno* in *Neaktivno*.

Aktivna trajajoča stanja so elementarna stanja *Obratovanje* in *Zadržano* ter nadstanja *Izvajanje* in *Delovanje*; aktivna prehodna stanja pa so *Zadržanje*, *Ustavljanje* in *Prekinjanje* (vsa elementarna).

Prehode delimo na aktivne in neaktivne, pri čemer aktivni vsebujejo določeno procesiranje. Aktivni prehodi so:

- iz nadstanja *Neaktivno* v stanje *Obratovanje*,
- iz stanja *Obratovanje* v stanje *Zadržanje*,
- iz stanja *Zadržano* v stanje *Zadržanje*,
- iz stanja *Zadržano* v stanje *Obratovanje*.

Vsak aktiven prehod vsebuje po eno sekvenco procesiranja prehoda.

4.2.3 Kompleksni prehodi

Kompleksni prehodi so nizi sekvenc procesiranja stanj in prehodov, ki se izvedejo ob prehodu iz enega trajajočega v drugo trajajoče stanje. Ali z drugimi besedami, kompleksni prehodi so integrirane celote procesiranja (sestavljene iz večjega števila sekvenc) med izvajanjem avtomata stanj faze. Avtomati stanj posameznih faz se izvajajo v sklopu programa, ki ga imenujemo *vmesnik fazne logike* (ang. *Phase Logic Interface*, s kratico PLI), ki ciklično preverja potrebo po aktiviranju enega od kompleksnih prehodov iz trenutno aktivnega trajajočega stanja. Ko so pogoji (ali vzroki) za nastop kompleksnega prehoda izpolnjeni, PLI izvede ta kompleksni prehod skozi zaporedno aktiviranje posameznih sekvenc tega kompleksnega prehoda. Definirali smo naslednje kompleksne prehode:

1. Kompleksni prehod iz stanja *Neaktivno* v stanje *Obratovanje*, do katerega pride potem, ko faza dobi ukaz *Zaženi* (ang. *Start*). Izvor ukaza *Zaženi* je lahko PLI, ki zažene fazo v stanju *Začetno*, ali operater, ki zažene fazo v stanju *Končno*. V

slednjem primeru se faza izvaja izven recepta oziroma neodvisno od recepta. Kompleksni prehod se začne v neaktivnem stanju *Neaktivno*, konča pa se v trajajočem aktivnem stanju *Obratovanje*.

2. Kompleksni prehod iz aktivnega trajajočega stanja *Obratovanje* v aktivno trajajoče stanje *Zadržano*, do katerega pride, potem ko faza dobi ukaz *Zadrži* (ang. *Hold*) ali ob nastopu izjeme (odpovedi), medtem ko je faza v tako imenovanem *varovanem območju* (ang. *guarded region*), to je v stanju, v katerem se preverja nastop izjem za zadržanje faze. To stanje lahko opredelimo s pomočjo sekvenc, ki so takrat aktivne. To pa so vse sekvence, ki so sočasne sekvenci *Vedno* stanja *Obratovanje* in v katere sodijo sama sekvenca *Vedno*, poleg nje pa še sekvenci *Vhod* in *Zanka* stanja *Obratovanje* ter sekvenci obeh prehodov v stanje *Obratovanje* – iz stanja *Neaktivno* in iz stanja *Zadržano*. Torej, izjava, da je neka faza v varovanem območju, je ekvivalentna izjavi, da je sekvenca *Vedno* stanja *Obratovanje* te faze aktivirana. Naloga programerja v zvezi z izjemami in zadržanji določene faze je le polnjenje določenih sekvenc (glede na doseg posamezne izjeme) z logičnimi izrazi, od katerih vsak kaže na nastop določene izjeme te faze. PLI v vsakem ciklu ugotavlja, ali je prišlo do določene izjeme (izvede logično ALI operacijo med logičnimi izrazi, pripadajočimi posameznim izjemam), in če je temu tako, sproži kompleksni prehod iz stanja *Obratovanje* v stanje *Zadržano*. Povedano bolj podrobno, logične izraze za ugotavljanje nastopa izjem lahko programer umesti v naslednje sekvence:
   - logične izraze, ki obravnavajo izjeme najširšega dosega (aktivne ves čas aktivnega izvajanja faze), umesti v sekvenco *Vedno* stanja *Obratovanje*,
   - logične izraze, ki obravnavajo zagonske pogoje faze, umesti v sekvenco prehoda iz nadstanja *Neaktivno* v stanje *Obratovanje*,
   - logične izraze, ki obravnavajo pogoje za nadaljevanje faze, umesti v sekvenco prehoda iz stanja *Zadržano* v stanje *Obratovanje*,
   - logične izraze, ki obravnavajo pogoje, ki so tako pogoji za zagon kot tudi pogoji za nadaljevanje faze, umesti v sekvenco *Vhod* stanja *Obratovanje*,
   - logične izraze, ki obravnavajo izjeme z dosegom, omejenim na čas cikličnega procesiranja med obratovanjem faze, umesti v sekvenco *Zanka* stanja *Obratovanje*.
3. Kompleksni prehod iz aktivnega trajajočega stanja *Zadržano* skozi stanje *Zadržanje* nazaj v stanje *Zadržano*, do katerega pride ob povišanju stopnje odpovedi (ang. *failure increase*). Do tega pride, ko PLI ugotovi nastop izjeme višje stopnje, kot je stopnja odpovedi, zaradi katere je faza trenutno v stanju *Zadržano*. Logične izraze, ki obravnavajo povišanje stopnje odpovedi, se umesti v sekvenco *Zanka* stanja *Zadržano*, lahko pa tudi v sekvenco *Zanka* nadstanja *Izvajanje*, kar je enako, kot če bi bili umeščeni v sekvenci *Zanka* obeh stanj *Obratovanje* in *Zadržano*.

4. Kompleksni prehod iz aktivnega trajajočega stanja *Zadržano* v stanje *Obratovanje*, do katerega pride, potem ko faza v stanju *Zadržano* dobi ukaz *Nadaljuj* (ang. *Restart*).
5. Kompleksni prehod iz stanja *Obratovanje* v stanje *Zaključeno*, do katerega pride, ko faza zaključi izvajanje funkcije, za katero je programirana.
6. Kompleksni prehod iz nadstanja *Končno* v stanje *Začetno*, do katerega pride, ko faza v enem od podstanj nadstanja *Končno* (*Zaključeno*, *Ustavljeno*, *Prekinjeno*) dobi ukaz *Ponastavi* (ang. *Reset*). V stanju *Začetno* faza čaka na ukaz *Zaženi*, ki ji ga posreduje PLI. Ta kompleksni prehod ne vsebuje nobene sekvence procesiranja.
7. Kompleksni prehod iz nadstanja *Izvajanje* v stanje *Ustavljeno*, do katerega pride, ko faza v enem od elementarnih podstanj stanja *Izvajanje* (*Obratovanje*, *Zadržanje* ali *Zadržano*) dobi ukaz *Ustavi* (ang. *Stop*).
8. Kompleksni prehod iz nadstanja *Delovanje* v stanje *Prekinjeno*, do katerega pride, ko faza v enem od elementarnih podstanj stanja *Delovanje* (*Obratovanje*, *Zadržanje*, *Zadržano* ali *Ustavljanje*) dobi ukaz *Prekini* (ang. *Abort*).

4.2.4 Sekvence fazne logike

Sekvence fazne logike predstavljajo okvir za modularizacijo programske opreme za vodenje šaržnih procesov. Glede na model obnašanja faz, ki ga prikazuje Slika 10, je teoretično največje število sekvenc posamezne faze enako 55, in sicer:

- po štiri sekvence za vsako trajajoče stanje, ki jih je 10 (elementarnih stanj in nadstanj), torej skupaj 40 sekvenc,
- po ena sekvenca za vsako prehodno stanje, ki so 4, torej skupaj 4 sekvence, in
- po ena sekvenca za vsak prehod, ki jih je 11, torej skupaj 11 sekvenc.

Jasno je, da toliko sekvenc za uspešno zajemanje vseh možnih obnašanj ne potrebujemo in da zato lahko njihovo število bistveno zmanjšamo.

Prvi, trivialni nivo redukcije števila sekvenc se nanaša na vse sekvence neaktivnih stanj in prehodov med pari neaktivnih stanj. Nobene izmed teh sekvenc ne potrebujemo in so zato izključene iz modela. Drugi nivo redukcije števila sekvenc opravimo z uporabo naslednjega enostavnega pravila: vse sekvence, ki se pojavijo v določenem kompleksnem prehodu in zgolj v tem prehodu, lahko nadomestimo z le eno skupno sekvenco. Po izvedbi obeh omenjenih redukcij modela obnašanja faze v tem ostane še 19 sekvenc: 15 v stanjih in 4 v prehodih. Gre za naslednje sekvence:

1. Sekvenca *Vhod* nadstanja *Delovanje*, ki vsebuje kodo, ki naj se izvede enkratno ob zagonu faze, na primer zagon števca celotnega trajanja izvajanja faze.
2. Sekvenca *Zanka* nadstanja *Delovanje*, ki vsebuje kodo, ki naj se izvaja ciklično ves čas med izvajanjem faze.
3. Sekvenca *Izhod* nadstanja *Delovanje*, ki vsebuje kodo, ki naj se izvede enkratno,

neposredno pred prehodom faze v nadstanje *Končno* (oziroma v enega od njegovih podstanj), na primer ustavitev števca celotnega trajanja izvajanja faze.

4. Sekvenca *Vhod* nadstanja *Izvajanje*, ki vsebuje kodo, ki naj se izvede enkratno ob dejanskem začetku izvajanja funkcionalnosti faze, na primer ponastavitev dozirnega masnega števca.
5. Sekvenca *Zanka* nadstanja *Izvajanje*, ki vsebuje kodo, ki naj se izvaja ciklično ves čas med izvajanjem dejanske funkcionalnosti faze, kar vključuje stanja *Obratovanje*, *Zadržanje* in *Zadržano* ter prehode med njimi, na primer alarmiranje.
6. Sekvenca *Izhod* nadstanja *Izvajanje*, ki vsebuje kodo, ki naj se izvede enkratno, neposredno pred prenehanjem dejanske funkcionalnosti faze, na primer shranjevanje vrednosti dozirnega masnega števca.
7. Sekvenca *Vhod* stanja *Obratovanje*, ki vsebuje kodo, ki naj se v vsakem primeru (ne glede na izvorno stanje) izvede enkratno ob vhodu v stanje *Obratovanje*, na primer zagon fizičnih naprav.
8. Sekvenca *Zanka* stanja *Obratovanje*, ki vsebuje kodo, ki naj se izvaja ciklično med stabilnim trajanjem stanja *Obratovanje* (po zaključenem kompleksnem prehodu v stanje *Obratovanje* in pred začetkom kompleksnega prehoda iz stanja *Obratovanje*), na primer preverjanje pogojev za zaključek izvajanja faze ali izvajanje alarmiranja, specifičnega za stanje *Obratovanje*.
9. Sekvenca *Vedno* stanja *Obratovanje*, ki vsebuje kodo, ki naj se izvaja ciklično in časovno omejeno na unijo časa izvajanja vseh treh notranjih sekvenc stanja *Obratovanje* (*Vhod*, *Zanka* in *Izhod*) in njegove specifične vhodne sekvence, to je sekvence prehoda v stanje *Obratovanje* (*AKT_PR_Neaktivno-Obratovanje* ali *AKT_PR_Zadržano-Obratovanje*). To je pomembna sekvenca, ki predstavlja varovano območje faze oziroma doseg preverjanja izjem (za zadržanje) te faze. Na tem mestu naj povemo še, da niz »AKT_PR« nastopa pri imenih vseh sekvenc prehodov, predstavlja pa okrajšavo za »aktivnost prehoda«.
10. Sekvenca *Prehodno* prehodnega stanja *Zaključevanje*, ki vsebuje kodo, ki naj se izvede enkrat ob prehodu skozi to stanje in ki vsebuje procesiranje, ki naj se izvede ob regularnen zaključku izvajanja faze.
11. Sekvenca *Prehodno* prehodnega stanja *Zadržanje*, ki vsebuje kodo, ki naj se izvede enkrat ob prehodu skozi to stanje in ki vsebuje procesiranje, potrebno za prehod procesa v varno stanje.
12. Sekvenca *Zanka* stanja *Zadržano*, ki vsebuje kodo, ki naj se izvaja ciklično ves čas med trajanjem stanja *Zadržano*. Sekvenca lahko vsebuje preverjanje izjem oziroma nastop izjeme višjega nivoja od trenutno aktualne (ki je fazo pripeljala v stanje *Zadržano*); ob nastopu izjeme višjega nivoja se izvede cikel *Zadržano-Zadržanje-Zadržano*. Sekvenca poleg tega lahko izvaja tudi določeno reducirano funkcionalnost ali alarmiranje glede na zahteve določene faze.

13. Sekvenca *Izhod* stanja *Zadržano*, ki vsebuje kodo, ki naj se izvede enkrat ob izhodu faze iz stanja *Zadržano*.
14. Sekvenca *Prehodno* prehodnega stanja *Ustavljanje*, ki vsebuje kodo, ki naj se izvede enkrat ob prehodu skozi to stanje in vsebuje procesiranje, potrebno za izvedbo prehoda faze v stanje *Ustavljeno*. Sekvenca tipično ima določeno trajanje, saj v določenih točkah izvajanja čaka na povratno informacijo o uspešno izvedenem delu sekvence.
15. Sekvenca *Prehodno* prehodnega stanja *Prekinjanje*, ki vsebuje kodo, ki naj se izvede enkrat ob prehodu skozi to stanje in vsebuje procesiranje, potrebno za izvedbo prehoda faze v stanje *Prekinjeno*. Sekvenca tipično v nobeni točki ne čaka povratne informacije in posledično nima trajanja. Namen te sekvence je izvedba prehoda faze v končno stanje v primerih, ko sekvenca *Ustavljanje* ne more biti uspešno zaključena zaradi neuspešnega čakanja na povratno informacijo o uspešno zaključenem delu sekvence.
16. Sekvenca *AKT_PR_Zadržano-Obratovanje*, ki vsebuje kodo, ki naj se izvede enkrat ob izvajanju prehoda iz stanja *Zadržano* v stanje *Obratovanje*.
17. Sekvenca *AKT_PR_Obratovanje-Zadržanje*, ki vsebuje kodo, ki naj se izvede enkrat ob izvajanju prehoda iz stanja *Obratovanje* v stanje *Zadržanje*.
18. Sekvenca *AKT_PR_Neaktivno-Obratovanje*, ki vsebuje kodo, ki naj se izvede enkrat ob izvajanju prehoda iz nadstanja *Neaktivno* v stanje *Obratovanje*.
19. Sekvenca *AKT_PR_Zadržano-Zadržanje*, ki vsebuje kodo, ki naj se izvede enkrat ob izvajanju prehoda iz stanja *Zadržano* v stanje *Zadržanje*.

Za ilustracijo procesiranja (akcij vmesnika PLI), ki se zgodi med kompleksnim prehodom, si oglejmo korak za korakom, kaj se točno zgodi med (uspešnim) zagonom faze, to je med izvajanjem kompleksnega prehoda iz nadstanja *Neaktivno* v stanje *Obratovanje*. Med izvajanjem omenjenega kompleksnega prehoda PLI izvede naslednjo sekvenco akcij:

1. Izvedba (proženje in čakanje na povratno informacijo o dokončanju) sekvence *Vhod* nadstanja *Delovanje*.
2. Proženje sekvence *Zanka* nadstanja *Delovanje*.
3. Izvedba (proženje in čakanje na povratno informacijo o dokončanju) sekvence *Vhod* nadstanja *Izvajanje*.
4. Proženje sekvence *Zanka* nadstanja *Izvajanje*.
5. Proženje sekvence *Vedno* stanja *Obratovanje* (in s tem proženje detekcije odpovedi faze).
6. Izvedba (proženje in čakanje na povratno informacijo o dokončanju) sekvence prehoda iz stanja *Neaktivno* v stanje *Obratovanje* (*AKT_PR_Neaktivno-Obratovanje*).
7. Izvedba (proženje in čakanje na povratno informacijo o dokončanju) sekvence *Vhod*

stanja *Obratovanje*.

8. Proženje sekvence *Zanka* stanja *Obratovanje*.

V trenutku po zaključku zgoraj opisanih osmih korakov so sočasno aktivne štiri sekvence: sekvenca *Zanka* nadstanja *Delovanje*, sekvenca *Zanka* nadstanja *Izvajanje*, sekvenca *Vedno* stanja *Obratovanje* in sekvenca *Zanka* stanja *Obratovanje*.

4.2.5 Nadaljnja redukcija modela obnašanja faz

Opisani model obnašanja faz z 19 sekvencami predstavlja največji smiselni obseg elementov oziroma sekvenc, oziroma najdrobnejšo možno granularnost elementov izvajanja faz. V primeru če določen programer(ka) (ali organizacija) meni, da je 19 sekvenc preveč in bi raje uporabljal(a) enostavnejši model obnašanja faz (z manjšim številom sekvenc), lahko določi svoj nabor sekvenc, in to tako, da sekvence, ki jih ne želi uporabljati, enostavno konfigurira kot neaktivne. Med izvajanjem modela PLI take sekvence enostavno ignorira oziroma jih ne proži. Teoretično bi programer lahko aplikacije programiral z uporabo le ene sekvence, bi pa v tem primeru pri le nekoliko bolj kompleksnih primerih bila koda izredno nepregledna in bi bila verjetnost napak zelo velika. Programer bi lahko tudi uporabljal natanko tistih pet sekvenc, ki jih uporablja večina današnjih komercialnih orodij za vodenje šaržnih procesov po standardu ISA 88.00.01, vendar zopet na račun manjše preglednosti kode in slabšega obvladovanja njene kompleksnosti.

4.2.6 Primer modeliranja obnašanja faze

Kot primer modeliranja obnašanja faze na način, vpeljan v tem delu, si oglejmo zelo enostavno fazo iz sistema vodenja procesa proizvodnje smol v slovenski tovarni barv. Procesna celica sestoji iz šestih proizvodnih linij, od katerih vsaka vsebuje po dve enoti – reaktor in razredčevalno posodo. Naloga faze, ki jo bomo obravnavali, je transport reakcijske vode iz zbiralne posode reaktorske vode posamezne linije v skupno zbiralno posodo reaktorske vode za procesno celico. Fazo bomo poimenovali PraznjenjeZP_Rxx (Praznjenje zbiralne posode na reaktorju Rxx).

Med sistemsko analizo so bile ugotovljene naslednje zahteve glede faze PraznjenjeZP_Rxx:

Faza ima dva mogoča načina obnašanja. V prvem načinu se faza obnaša kot dominantna faza: izvede praznjenje zbiralne posode reaktorske vode, do ciljne vrednosti nivoja v posodi, ki je podana z recepturnim parametrom faze, nakar se njeno izvajanje zaključi. V drugem načinu se faza obnaša kot odvisna faza: prehaja med dvema notranjima stanjema stanja obratovanje, »Prazni« in »NePrazni«, s ciljem vzdrževanja nivoja v zbiralni posodi reaktorske vode med zgornjo in spodnjo ciljno vrednostjo, pri čemer je vrednost spodnje meje podana z istim parametrom kot pri prvem načinu, vrednost zgornje meje pa z drugim recepturnim parametrom faze.

Faza lahko, ali pa tudi ne, beleži količino transportirane vode v odvisnosti od še enega recepturnega parametra faze.

Faza izvaja svojo funkcijo z uporabo dveh elementov osnovnega vodenja: črpalke P1 in zapornega ventila V1. V sekvenci zagona praznjenja zbiralne posode naj bo izvedena zakasnitev med odpiranjem ventila in vklopom črpalke, prav tako naj bo zakasnitev v sekvenci ustavitve praznjenja med izklopom črpalke in zapiranjem ventila. Odpoved P1 ali V1 naj ima za posledico zadržanje faze. Odziv sistema na izpad napajanja sistema osnovnega vodenja naj bo odvisen od notranjega stanja faze: če je faza v stanju »NePrazni« in naj bi prešla v stanje »Prazni«, naj odpoved povzroči takojšnje zadržanje; če je faza v notranjem stanju »Prazni«, naj do zadržanja pride z zakasnitvijo (med praznjenjem dovolimo krajše odpovedi napajanja).

Izvajanje šestih faz »PraznjenjeZP_Rxx« posameznih linij naj bo medsebojno izključujoče (zbiralna posoda je skupni vir z omejitvijo enega uporabnika v določenem trenutku). Arbitraža alokacije skupnega vira se izvaja po principu »first-come-first-served«, in to neizpodrinjevalno (ang. non-preemptive).

Fazi »PraznjenjeZP_Rxx« in »Vakuumiranje_Rxx« (na isti reaktorski liniji) sta medsebojno izključujoči. Tukaj naj bo arbitraža izvedena po prioritetnem izpodrinjevalnem principu, pri čemer naj ima faza »Vakuumiranje« višjo prioriteto. Z drugimi besedami, faza »PraznjenjeZP« je prisiljena v zadržanje, če pride med njenim delovanjem do zagona faze »Vakuumiranje« ali če pride med delovanjem faze »Vakuumiranje« do zagona faze »PraznjenjeZP«.

Zgoraj podane zahteve so bile med implementacijo sistema vodenja preslikane v naslednjih deset sekvenc (torej 9 od 19 sekvenc v primeru te faze ni bilo potrebnih):

1. Sekvenca *Vhod* stanja *Delovanje*:
   ponastavi števec količine transportirane reakcijske vode

2. Sekvenca *Zanka* stanja *Delovanje*:
   IF vrednost katerega koli parametra spremenjena
   THEN preberi vrednost vseh parametrov ENDIF

3. Sekvenca *Vhod* stanja *Izvajanje*:
   aktiviraj preverjanje izjem v modulih osnovnega vodenja za V1 in P1

4. Sekvenca *Izhod* stanja *Izvajanje*:
   deaktiviraj preverjanje izjem v modulih osnovnega vodenja za V1 in P1

5. Sekvenca *Vhod* stanja *Obratovanje*:
   notranje stanje sekvence *Zanka* stanja *Obratovanje* := *NePrazni*

6. Sekvenca *Zanka* stanja *Obratovanje*:

   IF notranje stanje sekvence *Zanka* stanja *Obratovanje* = *NePrazni* THEN
   IF primerek faze je dominanten
   AND nivo v zbiralni posodi je blizu ciljne spodnje meje
   THEN zaključi izvajanje faze
   ELSIF nivo v posodi ≥ ciljna zgornja meja OR primerek faze je dominanten THEN
   IF odpoved napajanja osnovnega vodenja THEN odpoved faze
   ELSE
   sproži zahtevo in čakaj na dodelitev zbiralne posode
   izvedi sekvenco osnovnega vodenja za prehod iz *NePrazni* v *Prazni*
   notranje stanje sekvence *Zanka* stanja *Obratovanje* := *Prazni*
   ENDIF
   ENDIF
   ELSE (* Prazni *)
   izračunaj trenutno transportirano količino
   IF odpoved napajanja osnovnega vodenja v trajanju nad dovoljenim THEN
   odpoved faze
   ELSIF nivo v zbiralni posodi je manjši ali enak kot ciljna spodnja meja THEN
   izvedi sekvenco osnovnega vodenja za prehod iz *Prazni* v *NePrazni*
   notranje stanje sekvence *Zanka* stanja *Obratovanje* := *NePrazni*
   sprosti zbiralno posodo
   IF primerek faze je dominanten THEN zaključi izvajanje faze ENDIF
   ENDIF
   ENDIF

7. Sekvenca *Vedno* stanja *Obratovanje*:

   Izračunaj vrednosti logičnih izrazov za odpovedi (vzroke zadržanja) faze:

   - »FirstScan«
   - nerazpoložljivost P1 ali V1 (ustrezen izraz je »NOT remote-auto mode OR failure), medtem ko velja: notranje stanje sekvence *Zanka* stanja *Obratovanje* = *Prazni*
   - stanje faze *Vakuumiranje* ni enako *Neaktivno*

8. Sekvenca *Prehodno* stanja *Zadržanje*:

   IF notranje stanje sekvence *Zanka* stanja *Obratovanje* = *Prazni*

THEN izvedi sekvenco osnovnega vodenja za prehod iz *Prazni* v *NePrazni* ENDIF

9. Sekvenca *Prehodno* stanja *Ustavljanje*:
   IF notranje stanje sekvence *Zanka* stanja *Obratovanje* = *Prazni*
   THEN izvedi sekvenco osnovnega vodenja za prehod iz *Prazni* v *NePrazni* ENDIF

10. Sekvenca *Prehodno* stanja *Prekinjanje*:
    IF notranje stanje sekvence *Zanka* stanja *Obratovanje* = *Prazni*
    THEN izvedi sekvenco osnovnega vodenja za prehod iz *Prazni* v *NePrazni* ENDIF

4.2.7 Zaključki

V tem poglavju smo predstavili nov model obnašanja faz, za katerega se je med preizkušanjem v praksi (glej poglavje 1.6 Okvir za preizkušanje rezultatov disertacije) pokazalo, da ima velik potencial za dvig kakovosti in izboljšanje obvladovanja kompleksnosti med življenjskim ciklom programske opreme za vodenje šaržnih procesov. Nov model obnašanja faz se lahko komu na prvi pogled zdi prekompleksen (glede na strukturo izobrazbe njegovih potencialnih uporabnikov), vendar pa gre pri konceptu zgolj za delni premik inherentne kompleksnosti programske opreme za vodenje šaržnih procesov iz posameznega aplikativnega programa (ki se ga razvija vedno znova ob vsakem projektu) v sistemski program (vmesnik fazne logike, PLI). To aplikativnemu programerju nudi možnost za doseganje boljše kakovosti aplikativne programske opreme oziroma približevanje danes pogosto omenjanim ciljem procesa razvoja programske opreme – »right first time, every time« in »correct by construction« (namesto »construct by correction«).

5 Objektni model opreme in postopkovnega vodenja

5.1 Analiza aktualnega objektnega modela opreme in postopkovnega vodenja

5.1.1 Uvod

Med glavne težave, ki jih po naših izkušnjah srečujemo pri vodenju šaržnih procesov v skladu s standardom ISA 88.00.01 in ob uporabi na standardu temelječih orodij, sodita visoka stopnja podvajanja informacij v receptih in nizka stopnja njihove ponovne uporabe. Posledici teh težav sta večji obseg dela pri vzdrževanju receptov in večja verjetnost napak v receptih. Izvor omenjenih težav je po našem mnenju v pomanjkljivostih objektnega modela šaržnega vodenja, kot ga določa standard ISA 88.00.01, katerih vzrok je v fiksni strukturi razredov enot in v preveč enostavni razredni hierarhiji.

Zgoraj omenjeni problem se presenetljivo ne omenja v strokovni ali znanstveni literaturi. Med obsežnim pregledom literature v večjem številu podatkovnih baz, kot npr. INSPEC, Science Direct ali Google Scholar, nismo našli niti enega besedila, ki bi obravnavalo ta problem. Iskanje smo izvajali na osnovi različnih kombinacij naslednjih ključnih besed: »batch«, »process«, »control«, »recipe«, »reuse«, »repetition of information«, »reusability«.

Ustrezna rešitev za zgoraj omenjeni problem je po našem mnenju v uvedbi bolj sofisticiranega objektnega modela opreme in postopkovnega vodenja. Korak v smeri bolj sofisticiranega objektnega modela je sicer že bil narejen v sklopu drugega dela standarda za vodenje šaržnih procesov ISA 88.00.02 [2], ki je vpeljal bolj kompleksno razredno hierarhijo v obliki večkratnega dedovanja v objektnem modelu enot na način, da posamezna entiteta opreme lahko pripada večjemu številu razredov opreme. Vendar pa pri tem v bistvu gre za prirejanje različnih vlog določeni enoti, tako da lahko ima npr. določena posoda vlogo reaktorja in shranjevalnega tanka, kar pa nima nobenega vpliva na podvajanje informacij v receptih in na njihovo ponovno uporabo. Rešitev vidimo v novem objektnem modelu, ki bo zasnovan na osnovi dinamično definiranih in potencialno prekrivajočih se razredih enot oziroma na razmerju večkratnega dedovanja med enotami in razredi enot, to je možnosti, da posamezna enota pripada več kot enemu razredu enot.

Preden predstavimo nov objektni model opreme in postopkovnega vodenja, bomo v naslednjih treh podpoglavjih te analize aktualnega objektnega modela opreme in postopkovnega vodenja obravnavali tri tematike. Le-te so pomembne po eni strani za

razumevanje širšega okvirja v kontekstu modelov šaržnega vodenja in v kontekstu objektne tehnologije, po drugi strani pa za razumevanje motiva prehoda od standardnega na nov objektni model opreme in postopkovnega vodenja. Najprej bomo (v podpoglavju 5.1.2 Modeli vodenja šaržnih procesov) predstavili osnovni koncept objektnega modeliranja opreme in postopkovnega vodenja ter receptov, kot ga uporablja standard ISA 88.00.01 in ki ga bomo uporabljali tudi mi pri zasnovi novega objektnega modela. To je potrebno za razumevanje strukture tako standardnega kot tudi novega objektnega modela. V nadaljevanju bomo (v podpoglavju 5.1.3 Objektna tehnologija pri vodenju šaržnih procesov) predstavili objektni model, ki ga uporablja standard ISA 88.00.01 in na njem temelječa orodja, s poudarkom na sistemu klasifikacije enot in razredov enot, temelječem na konceptu dedovanja. V nadaljevanju bomo opisali nekatere slabosti tega modela in ugotovili, da bi se jim lahko izognili z uporabo bolj kompleksnega sistema klasifikacije, temelječega na konceptu večkratnega dedovanja, ki ga v preostanku podpoglavja na kratko predstavimo. Na koncu bomo (v podpoglavju 5.1.4 Standardni objektni model opreme in postopkovnega vodenja) bolj podrobno predstavili strukturo standardnega objektnega modela opreme in postopkovnega vodenja. Opisali bomo njegove pomanjkljivosti in argumentirali trditev, da le-te izvirajo iz preveč enostavnega modela opreme in postopkovnega vodenja, temelječega na konceptu enostavnega dedovanja.

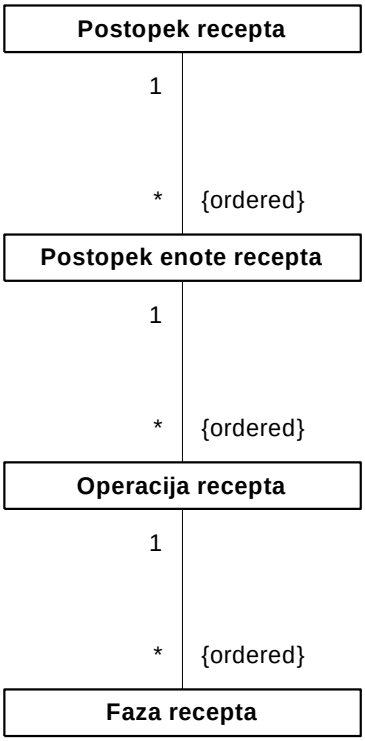
5.1.2 Modeli vodenja šaržnih procesov

V tem podpoglavju bomo zaradi celovitosti na kratko povzeli nekaj pomembnih pojmov in konceptov standarda ISA 88.00.01, ki so bili bolj podrobno predstavljeni v poglavju 2.4.2 . Standard opisuje šaržno vodenje na osnovi treh modelov, in sicer procesnega modela, fizičnega modela in modela postopkovnega vodenja. Procesni model opisuje hierarhično dekompozicijo šaržnega procesa z vidika entitet, relevantnih z vidika procesnega inženirja, kot so procesi, procesne stopnje, procesne operacije in procesne akcije. Fizični model opisuje hierarhijo fizičnih dobrin, kot so podjetja, podružnice, obrati, procesne celice, enote, moduli opreme in moduli vodenja. Model postopkovnega vodenja je sestavljen iz hierarhične strukture postopkovnih elementov, ki vodijo in usmerjajo izvajanje procesno usmerjenih akcij v urejenem vrstnem redu, s ciljem izvedbe proizvodnega postopka, kot ga določa procesni model. Hierarhija je sestavljena iz nivoja postopka (ki je sestavljen iz sekvenčno-sočasne kombinacije postopkov enot), nivoja postopka enote (ki je sestavljen iz sekvenčno-sočasne kombinacije operacij) in nivoja operacije (ki je sestavljen iz sekvenčno-sočasne kombinacije faz). Entitete postopkovnega vodenja v bistvu predstavljajo zmožnost procesiranja entitet fizičnega modela, pri čemer je faza opreme najmanjša entiteta procesiranja. Entitete treh modelov so v medsebojni relaciji, ki je hierarhično organizirana po načelu, da postopek, realiziran na določeni opremi, rezultira v proces. Tako, na primer, na najnižjem nivoju hierarhije izvajanje faze opreme, ki je element postopkovnega modela, na ustreznem modulu opreme daje procesno funkcionalnost za izvedbo procesne akcije, ki

pa je entiteta procesnega modela. Pri tem elementi postopkovnega modela pošiljajo ukaze osnovnemu vodenju, ki je vsebovano v entitetah fizičnega modela. Relacije med tremi modeli prikazuje Slika 1 na strani 13.

Zgoraj opisani koncepti so bolj podrobno opisani v poglavjih 5.1 in 5.2 v [1].

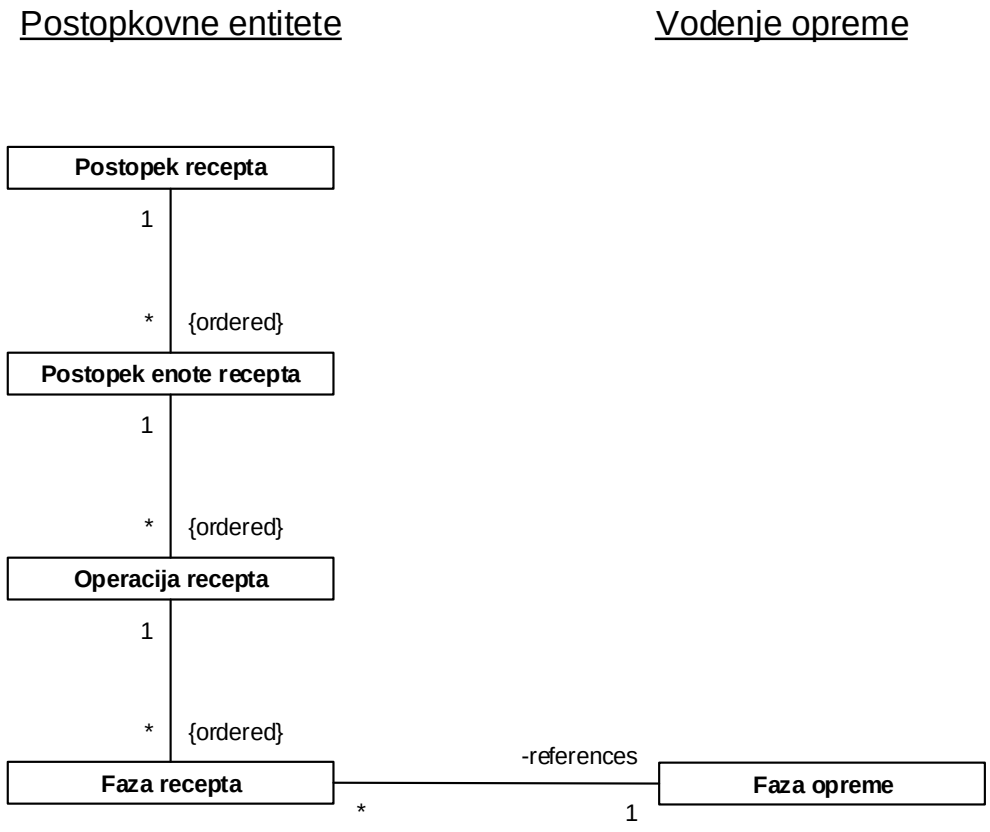
Poleg treh zgoraj opisanih modelov standard določa še recepte, ki vsebujejo vse informacije, ki enolično določajo proizvodne zahteve za določen produkt. Kot smo že omenili v poglavju 2.4.3 obstaja več vrst receptov (glavni recept, podružnični recept, temeljni recept in recept vodenja), ki se razlikujejo glede na splošnost, oziroma specifičnost informacije, ki jo vsebujejo. V tem delu se bomo ukvarjali le z dvema nižjenivojskima receptoma, temeljnim receptom in receptom vodenja. V poglavju 2.4.3 smo omenili tudi več delov receptov, od katerih je v kontekstu tega dela najpomembnejši postopek recepta (ang. *recipe procedure*), ki podaja hierarhično sekvenčno-sočasno dekompozicijo procesiranja, potrebnega za izdelavo določenega produkta. Dekompozicijo postopka recepta prikazuje Slika 11, kjer pravokotniki predstavljajo entitete procesiranja na posameznem nivoju hierarhije, povezave med nivoji pa beremo »višjenivojska entiteta je sestavljena iz urejene množice nižjenivojskih entitet«.



Slika 11: Postopek recepta vodenja.

Na prvem nivoju dekompozicije nastopa Postopek enote recepta (ang. *Recipe unit procedure*), na drugem nivoju Operacija recepta (ang. *Recipe operation*) in na tretjem, zadnjem, nivoju Faza recepta (ang. *Recipe phase*). Postopek recepta vodenja je v bistvu

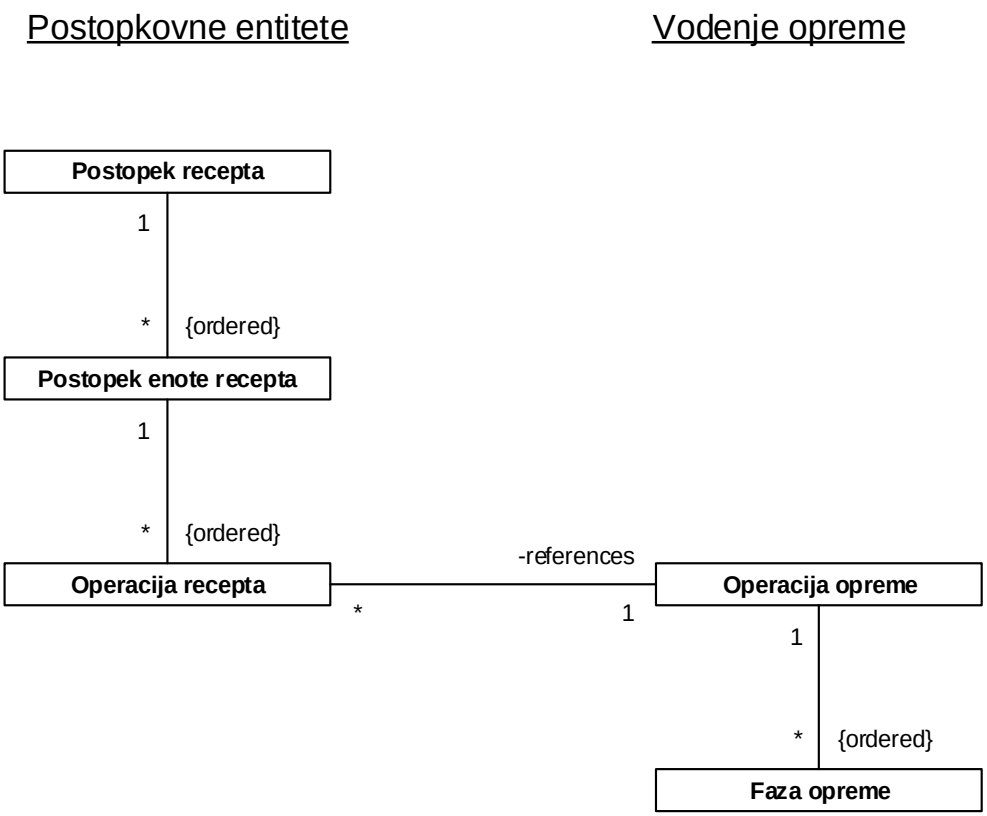
sestavljen iz hierarhije abstraktnih entitet, ki vsebujejo informacijo, potrebno za povezavo s konkretnimi entitetami vodenja opreme. Ta informacija vključuje referenco na entiteto vodenja opreme (za izbiro ustrezne entitete oziroma za doseganje fleksibilnosti na nivoju strukture procesiranja) in podatke (recepturne parametre) za njeno delovanje v konkretnem primeru (za določanje načina delovanja entitete v konkretnem primeru oziroma za doseganje fleksibilnosti na nivoju načina procesiranja). Tako je, na primer, faza recepta abstraktna notacija, ki vsebuje referenco (povezavo) na ustrezno fazo opreme. Na določeno fazo opreme lahko kaže več faz recepta, saj se procesiranje določene faze opreme lahko uporablja v večjem številu postopkov recepta. Povezovanje postopka recepta s fazo opreme prikazuje Slika 12.



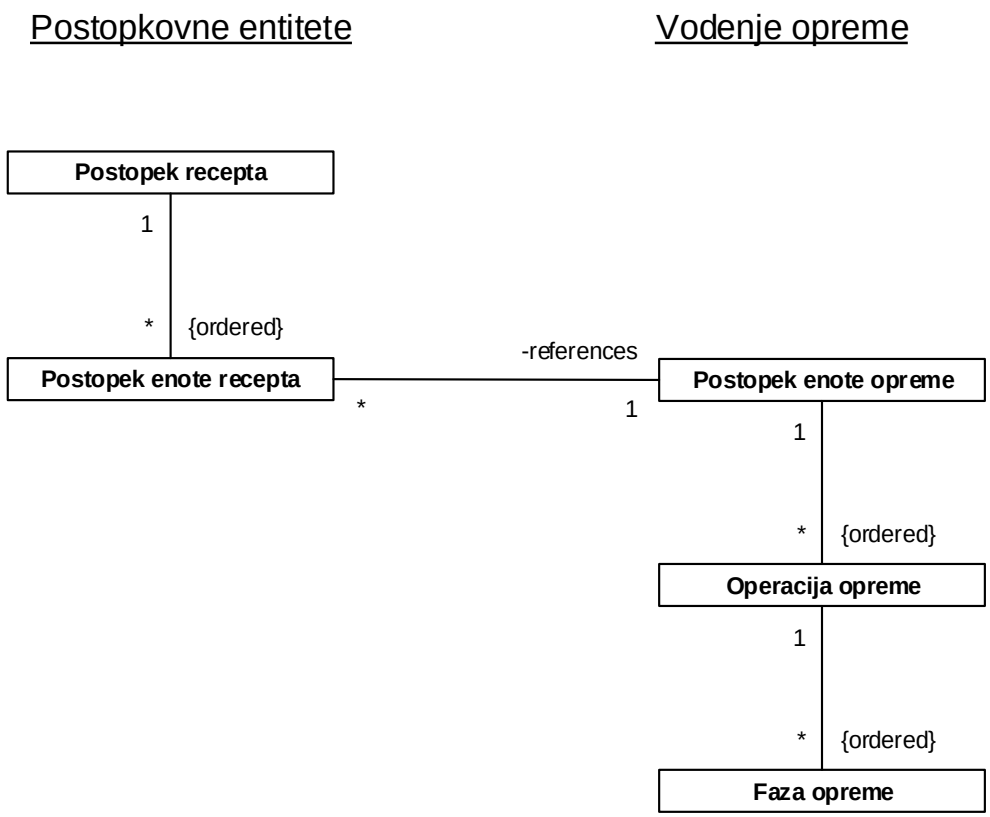
Slika 12: Povezovanje recepta in vodenja opreme na nivoju faz.

Povezovanje med postopkom recepta in vodenjem opreme lahko nastopa ne le na nivoju faz, kot prikazuje Slika 12, temveč tudi na kakšnem od treh višjih nivojev postopkovnega vodenja, torej na nivoju operacij (Slika 13), na nivoju postopkov enot (Slika 14) ali kar na nivoju postopka (Slika 15). Povezovanje na nižjem nivoju pomeni večjo fleksibilnost recepturnega vodenja, saj so tukaj gradniki vodenja opreme najmanjši, število nivojev dekompozicije postopka recepta pa največje. S premikom povezovanja proti višjim nivojem postopkovnega vodenja prihaja do tega, da so gradniki vodenja opreme, ki se jih

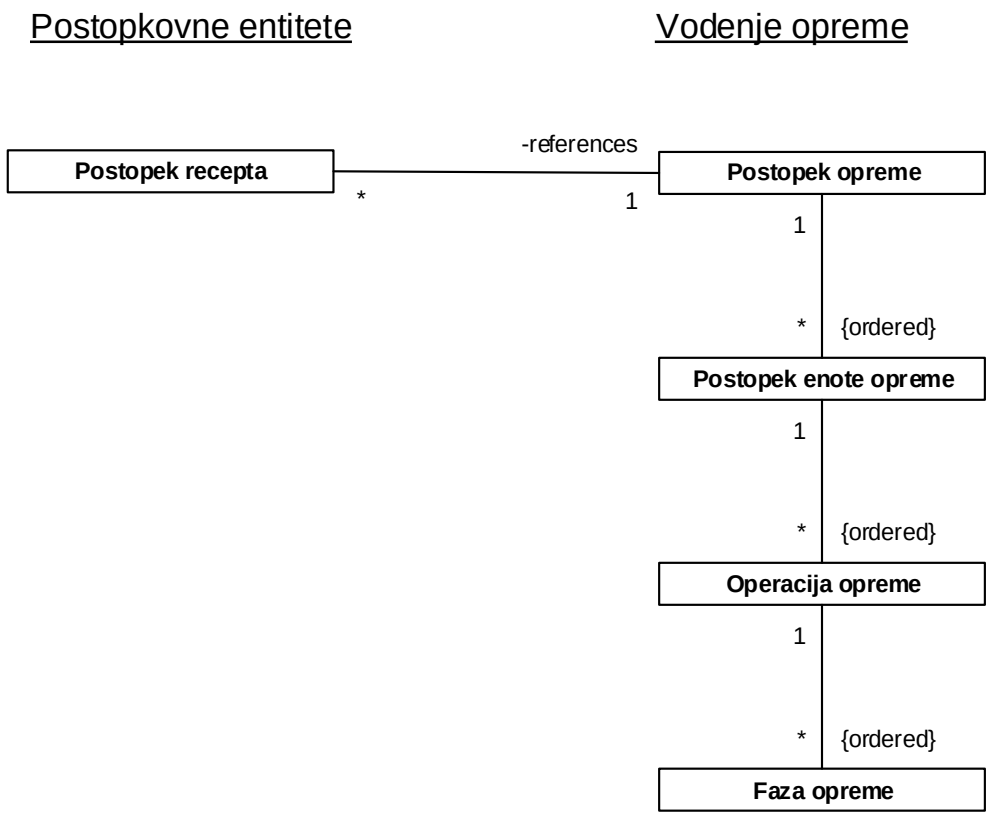
povezuje s postopkom recepta večji, kar posledično seveda pomeni manjšo fleksibilnost (pa tudi manjšo kompleksnost) recepturnega vodenja.



Slika 13: Povezovanje recepta in vodenja opreme na nivoju operacij.



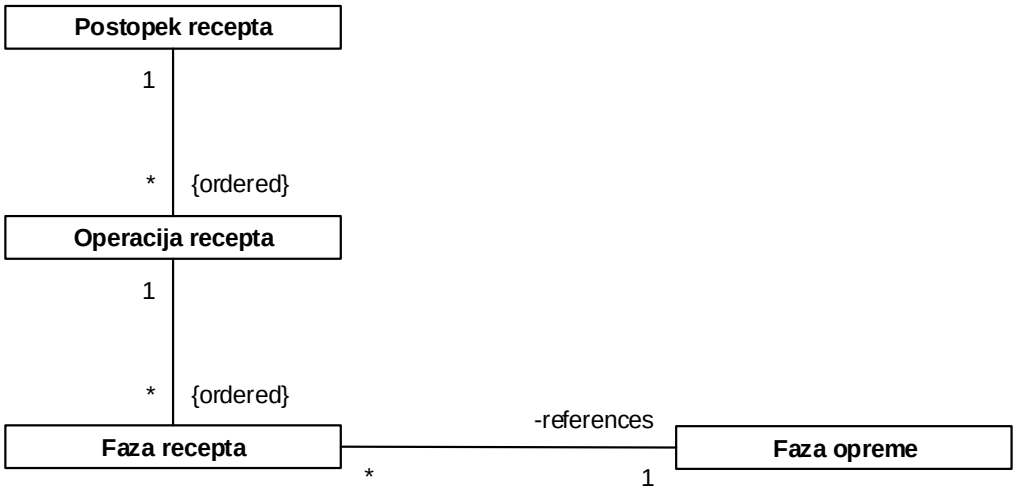
Slika 14: Povezovanje recepta in vodenja opreme na nivoju postopkov enot.



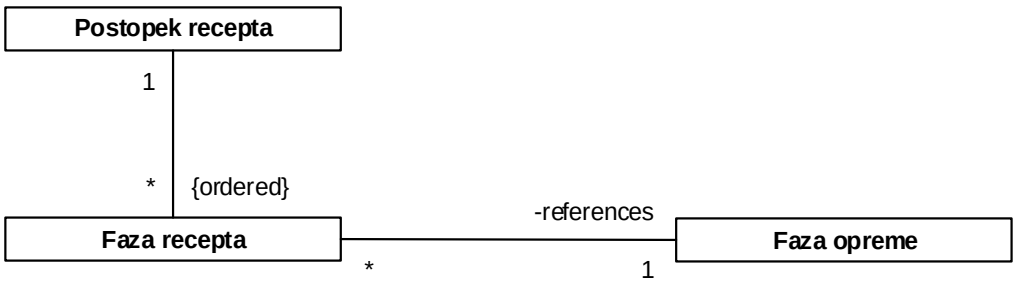
Slika 15: Povezovanje recepta in vodenja opreme na nivoju postopkov.

Nadaljnja variabilnost modela postopkovnega vodenja, ki jo predvideva standard ISA 88.00.01, je v tem, da je število nivojev hierarhije lahko tudi manjše od 4 – temu rečemo možnost sesedanja postopkovnega vodenja (ang. *procedural control collapsibility*). Naslednje slike prikazujejo primere, pri katerih je izpuščen en ali več nivojev, povezovanje postopka recepta z vodenjem opreme pa je izvedeno na različnih nivojih. V primeru, ki ga prikazuje Slika 16, je izpuščen nivo postopkov enot, povezovanje pa nastopa na nivoju faz; v primeru, ki ga prikazuje Slika 17, sta izpuščena nivoja postopkov enot in operacij, povezovanje pa nastopa na nivoju faz; v primeru, ki ga prikazuje Slika 18, sta prav tako izpuščena nivoja postopkov enot in operacij, povezovanje pa tukaj nastopa na nivoju postopka; v primeru, ki ga prikazuje Slika 19, je izpuščen nivo faz, povezovanje pa nastopa na nivoju operacij.

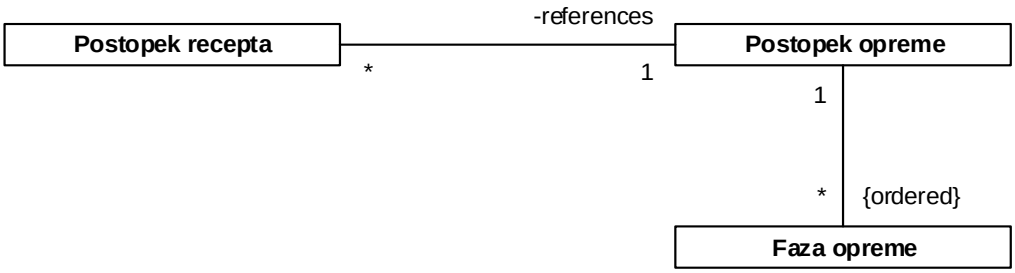
Zgoraj opisani koncepti so bolj podrobno opisani v poglavju 5.3 v [1]. Podroben opis povezovanja postopka recepta in postopkovnega vodenja naprav je v poglavju 5.3.3.3 v [1].



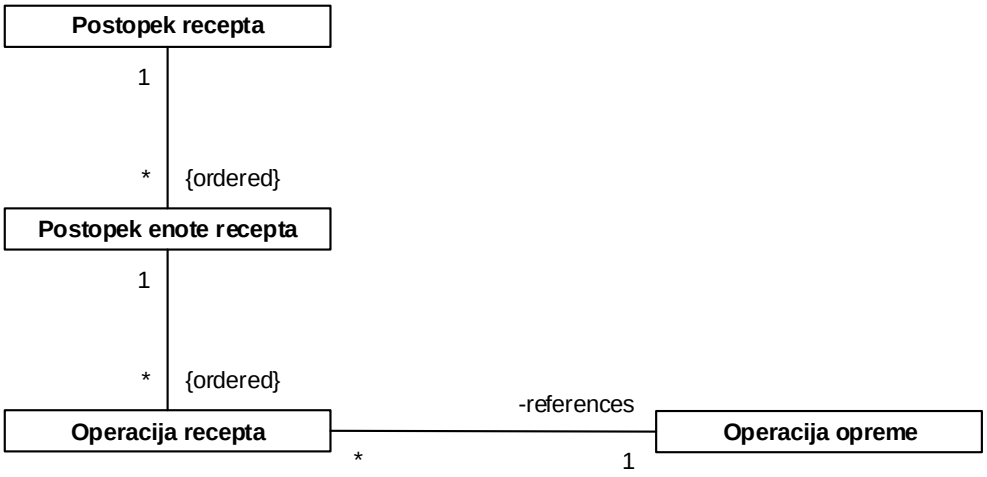
Slika 16: Redukcija nivojev modela, primer 1.



Slika 17: Redukcija nivojev modela, primer 2.



Slika 18: Redukcija nivojev modela, primer 3.



Slika 19: Redukcija nivojev modela, primer 4.

5.1.3 Objektna tehnologija pri vodenju šaržnih procesov

Objektna tehnologija je omogočila pomemben pozitiven premik pri razvoju programske opreme nasploh. Med najpomembnejšimi učinki uvedbe objektne tehnologije sta povišanje stopnje ponovne uporabe [27, 28] in znižanje obsega podvajanja informacij [29]. Seveda pričakujemo, da to velja tudi za orodja za vodenje šaržnih procesov v skladu s standardom ISA 88.00.01 in da naj bi uvedba objektnih modelov opreme in postopkovnega vodenja preprečila podvajanje informacij v receptih in omogočila visoko stopnjo njihove ponovne uporabe.

Zaradi zadoščanja zahtevam po ponovni uporabi in izogibanju podvajanja informacij so modeli opreme in postopkovnega vodenja v raznih komercialnih orodjih za vodenje šaržnih procesov nadgrajeni z objektnim sistemom razredov in primerkov opreme ter s postopki enot opreme, ki so lahko napisani bodisi za razred bodisi za primerek enote opreme. Osnova za klasifikacijo enot je njihova zmožnost (ang. *capability*), to je nabor faz določene enote. Razred enote predstavlja več enot enakih zmožnosti oziroma z enakim naborom faz. Postopek enote razreda enot je napisan le enkrat, za razred enot, izvaja pa se lahko na kateri koli enoti tega razreda. Razred enot lahko predstavlja tudi enote, ki niso popolnoma enakih

zmožnosti oziroma nimajo popolnoma enakega nabora faz. V tem primeru razred enot vsebuje le tiste razrede faz, katerih primerki so vsebovani v vseh enotah tega razreda enot. Ostalih faz posameznih enot ni v razredu enot, temveč nastopajo kot specifične faze določene enote. Postopek enote, ki je napisan za razred enote, lahko vsebuje le skupne faze, če pa želimo uporabiti kakšno specifično fazo, mora biti postopek enote napisan za konkretno enoto.

Zgoraj opisani model deluje dobro v enostavnih primerih, še posebej tistih, kjer so primerki enot vsakega posameznega razreda enot medsebojno identični, brez specifičnih faz. Po drugi strani, v bolj kompleksnih primerih, ko razredi enot niso popolnoma homogeni oziroma ko imajo posamezni primerki enot specifične faze, kar je po naših izkušnjah v industriji dokaj pogost primer, se opisani model izkaže kot neuspešen pri izogibanju podvajanju informacij v receptih in pri maksimiranju njihove ponovne uporabe. Naša hipoteza je, da bi bil v teh primerih veliko bolj uspešen model, temelječ na večkratnem (in ne enkratnem) dedovanju.

Dedovanje je eden od najpomembnejših konceptov objektne tehnologije in predstavlja ključni mehanizem za povečanje ponovne uporabe [30, 31] in zniževanje redundance oziroma podvajanja informacij [29]. Dedovanje se uporablja za modeliranje generalizacije in specializacije [32, 33], pa tudi za modeliranje vlog, transakcij ali naprav tipa »eden od« (ang. *kind-of*) [34].

Vendar pa je dedovanje treba uporabljati previdno, saj je njegova nepravilna uporaba lahko nevarna [35]. V bistvu ga je treba jemati zgolj kot mehanizem za razširitev funkcionalnosti aplikacije s ponovno uporabo nadrejenih razredov [33]. Razlikujemo dva tipa dedovanja, in sicer enkratno dedovanje, pri katerem podrejeni razred deduje od le enega nadrejenega razreda, in večkratno dedovanje, pri katerem podrejeni razred deduje od dveh ali več nadrejenih razredov. Večkratno dedovanje je močnejše in ima večjo izraznost kot enkratno dedovanje [36]. Omogoča večjo ponovno uporabo in bolj naravno definiranje relacij med razredi [37]. Pravilna uporaba večkratnega dedovanja ni trivialna [38], saj vsebuje nekaj potencialnih problemov, kot npr. ponavljajoče dedovanje (ang. *repeated inheritance*) [39]. Zaradi omenjenega se večkratno dedovanje lahko uporablja le v nekaterih izmed današnjih objektno usmerjenih programskih jezikov (npr. C++ in Eiffel). Ali večkratno dedovanje lahko poenostavi ali pa le zapleta objektno usmerjeno tehnologijo, ostaja danes predmet debat [36]. Kakor koli že, koristi večkratnega dedovanja presegajo njegove pomanjkljivosti posebej v nekaterih specifičnih domenah, kot na primer pri objektno usmerjenem modeliranju. Tako je, na primer, uporaba večkratnega dedovanja pri modeliranju geografskih podatkov bistvena [40, 41].

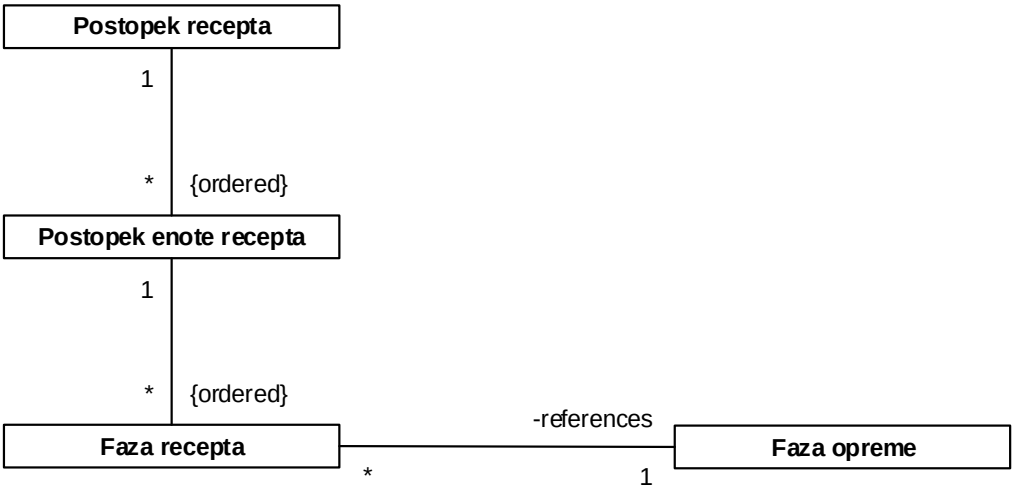
Kljub omenjenim dvomom verjamemo, da večkratno dedovanje v domeni vodenja šaržnih procesov predstavlja ustrezen mehanizem za reševanje problema visoke stopnje podvajanja informacij v receptih in njihove nizke ponovne uporabe.

5.1.4 Standardni objektni model opreme in postopkovnega vodenja

V tem delu bomo predstavili model postopkovnega vodenja, reduciran na tri nivoje, brez nivoja operacij. Tako smo se odločili zaradi tega, ker smo tak model uporabili kot osnovo za razvoj orodja za vodenje šaržnih procesov na platformi PLK. Razvito orodje se imenuje PLCbatch, uporabili pa ga bomo pri raziskavi učinkov, ki jih uporaba uvedenih novih konceptov prinese v praksi. Omenjena redukcija ne bo imela nobenega vpliva na potek in sklepe raziskave, saj bi bilo enostavno vse rezultate in sklepe raziskave prevesti na štirinivojski model, kot ga določa standard [1].

5.1.4.1 Struktura standardnega objektnega modela

Reduciran model postopkovnega vodenja prikazuje Slika 20. Na sliki je vidno, da je povezava med postopkom recepta in vodenjem opreme na najnižjem nivoju postopkovnega vodenja, to je na nivoju faz. To pomeni, da je vodenje opreme sestavljeno zgolj iz faz, kar zagotavlja najvišjo možno fleksibilnost receptov.



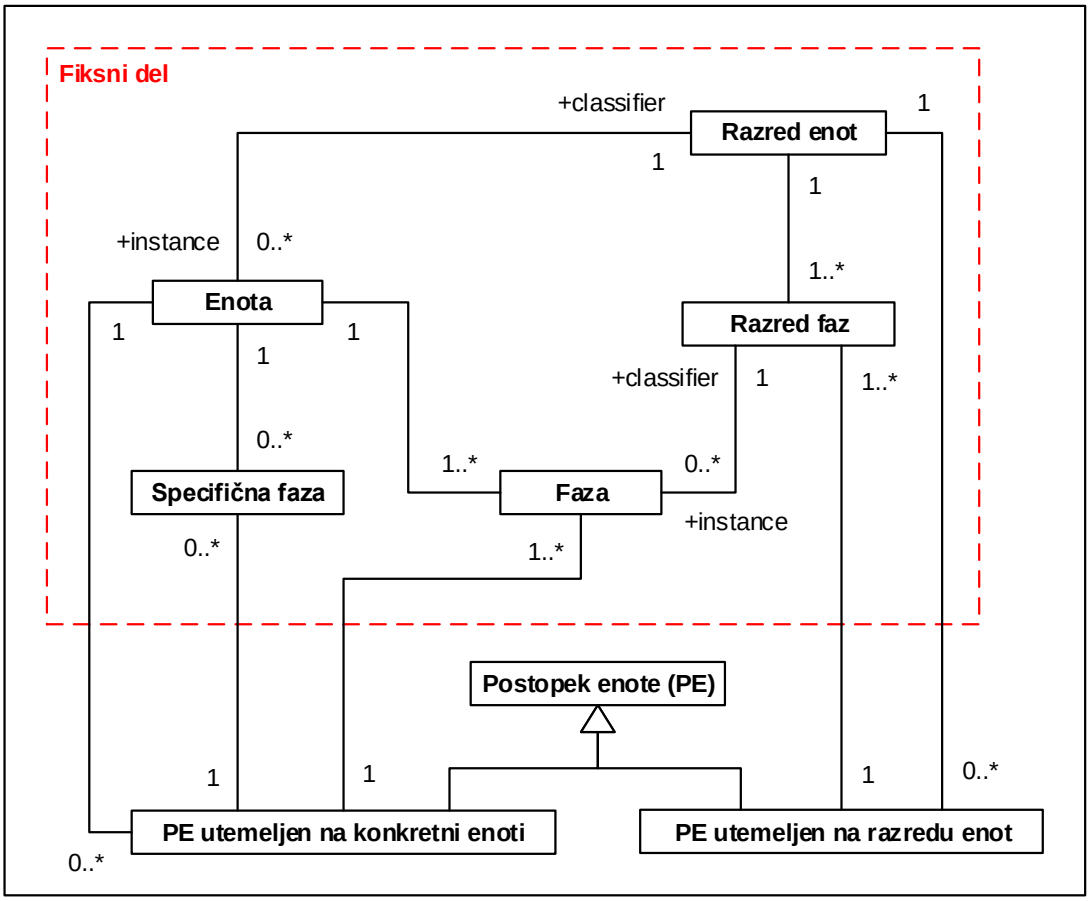
Slika 20: Reduciran model postopkovnega vodenja.

Slika 21 prikazuje objektni model opreme in postopkovnega vodenja, kot je določen v [1] in [2], in sicer (omejeno navzgor) do nivoja postopka enote.

Na najvišjem nivoju v modelu nastopa generična oprema skozi razrede enot in njihove razrede faz. Na nižjem nivoju so konkretne enote, ki so primerki prej omenjenih razredov enot. Vsaka enota ima po en primerek faze za vsak razred faz razreda enot, ki mu ta enota pripada, poleg teh faz pa lahko ima tudi dodatne, specifične faze. Postopki enot so lahko utemeljeni na razredu enot ali na konkretni enoti. V prvem primeru vsebujejo le faze, ki so primerki razredov faz, pripadajočih temu razredu enote, v drugem pa lahko vsebujejo katero koli fazo konkretne enote, vključno s specifičnimi fazami. Del modela znotraj rdečega (črtkanega) okvirja predstavlja fiksni del modela, ki vsebuje tiste elemente sistema vodenja,

ki se jih definira med razvojem sistema in ki jih potem, med fazo uporabe sistema vodenja, uporabniki ne morejo več spreminjati. Del modela izven rdečega okvirja pa predstavlja fleksibilni del, sestavljen iz receptov, katerih razvoj in spreminjanje sta v domeni uporabnikov.

Med razvojnim procesom ob uporabi tradicionalnih orodij na osnovi standardnega objektnega modela se model opreme gradi na način od zgoraj navzdol tako po agregacijski kot tudi po klasifikacijski hierarhiji. Gradnja njegovega fiksnega dela se začne z razredi enot in njihovimi razredi faz, čemur sledijo enote in njihove morebitne specifične faze. Fleksibilni del modela pa sestoji iz receptov na treh (ali manj, v primeru redukcije) nivojih: nivoju postopka, nivoju postopka enote in nivoju operacije.



Slika 21: Standardni model opreme in postopkovnega vodenja.

5.1.4.2 Pomanjkljivosti standardnega objektnega modela

V kakšni meri opisani standardni objektni model lahko prispeva k izogibanju podvajanja informacij v receptih in k njihovi ponovni uporabi, je odvisno od podobnosti in razlik med posameznimi enotami. Če so razredi enot homogeni (združujejo popolnoma identične enote), enote različnih razredov enot pa so medsebojno popolnoma različne, se standardni

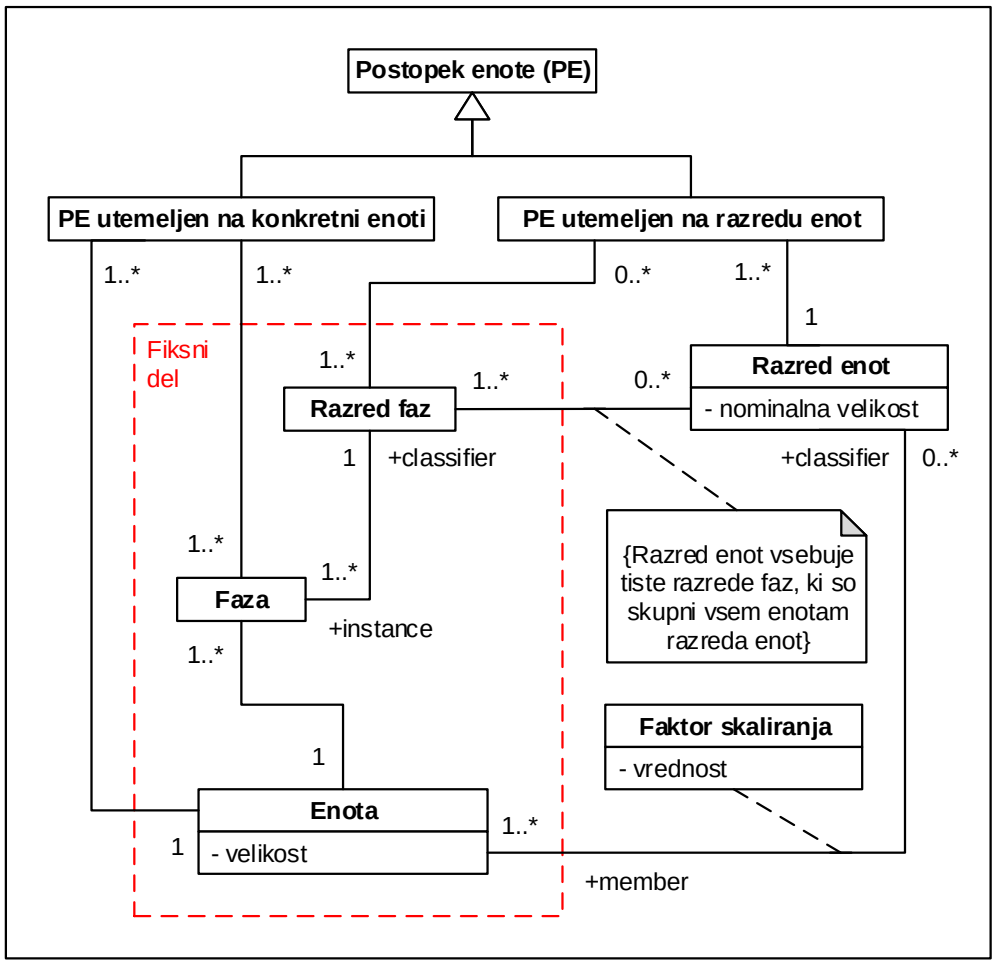
objektni model izkaže za primernega. V primerih, ko prva predpostavka ne drži (če posamezni razredi enot niso popolnoma homogeni), morajo biti postopki enot, ki vsebujejo kakšno specifično fazo, napisani več kot enkrat, kar je, seveda, zelo nezaželeno. Druga težava, ki nastopi v povezavi s podobnimi enotami, ki pripadajo različnim razredom enot, ni tako očitna. Težava izhaja iz nezmožnosti optimiranja postopkov enot na način, da bi podobne dele postopkov enot, pripadajočih različnim razredom enot, pisali le enkrat. Predpostavimo, da imamo tri razrede enot, pri čemer ima vsaka svoje lastne zmožnosti (faze), obstajajo pa tudi neprazni preseki zmožnosti (skupne faze) za vsak par razredov enot (torej trije preseki) in še presek zmožnosti vseh treh razredov enot. Vsi postopki enot, ki uporabljajo zgolj zmožnosti določenega preseka, bi lahko bili skupni za vse razrede enot, ki tvorijo ta presek. Pa vendar v sistemu, ki temelji na trenutno uporabljenem objektnem modelu, to ni možno in posledično se za različne razrede enot pojavljajo postopki enot z identičnimi deli.

Očitno je glavni vir zgoraj opisanih težav standardnega objektnega modela opreme in postopkovnega vodenja v fiksni strukturi razredov enot. Pri tem s fiksno strukturo menimo strukturo, določeno vnaprej, med razvojem sistema vodenja in brez možnosti njenega spreminjanja s strani uporabnikov (procesnih inženirjev) med uporabo sistema vodenja. Rešitev opisanih težav vidimo v uvajanju izboljšanega objektnega modela opreme in postopkovnega vodenja s fleksibilno določenimi (dinamičnimi) razredi enot in z možnostjo prekrivanja teh razredov, to je možnostjo obstoja nepraznih presekov zmožnosti (naborov faz) določenih skupin razredov enot.

5.2 Nov objektni model opreme in postopkovnega vodenja

Nov objektni model opreme in postopkovnega vodenja je bil predstavljen v [79]. Model do nivoja postopkov enot prikazuje Slika 22. Predstavitev podajamo le do nivoja postopkov enot zaradi lažje primerjave s standardnim objektnim modelom (Slika 21), saj standard ISA-88 le do tega nivoja definira objektni model opreme in postopkovnega vodenja na dovolj podroben in formalen način, da je primerjava dveh modelov mogoča. Nov model vsebuje enote, ki lahko pripadajo večjemu številu razredov enot oziroma dedujejo od večjega števila razredov enot. Lahko rečemo tudi, da v modelu lahko nastopajo prekrivajoči se razredi enot. To je v modelu označeno s kardinalnostjo »0..*« na povezavi med »Enota« in »Razred enot«, kar pomeni, da enota lahko pripada večjemu številu razredov enot, kar v bistvu predstavlja določeno obliko večkratnega dedovanja. Najpomembnejša lastnost novega modela je, da razredov enot ni potrebno definirati med razvojem sistema vodenja kot v primeru standardnega modela in na njem temelječih komercialnih orodjih. Pri novem modelu razrede enot lahko kreira procesni inženir (tehnolog) kadar koli med življenjskim ciklom sistema vodenja. Ta lastnost je še posebej pomembna v primerih, ko med življenjskim ciklom sistema vodenja prihaja do razvoja novih produktov, pri katerih se recepti za njihovo izdelavo strukturno razlikujejo od receptov prejšnjih produktov. Glavna

razlika med dvema modeloma je v tem, da pri novem modelu obstajajo le trije vnaprej definirani elementi, in sicer razredi faz, enote in faze, kar je na sliki (Slika 22) ponazorjeno z rdečim črtkanim okvirjem, ki obkroža omenjene tri vnaprej definirane elemente. Po drugi strani pa so pri standardnem modelu vnaprej definirani isti trije elementi in dodatno še dva – razredi enot in specifične faze, kar je na sliki (Slika 21) ponazorjeno z rdečim črtkanim okvirjem, ki tudi tukaj obkroža omenjenih pet vnaprej definiranih elementov.



Slika 22: Nov model opreme in postopkovnega vodenja do nivoja PE.

V razvojnem procesu, temelječem na novem objektnem modelu, se gradnja sistema vodenja začne z definiranjem razredov faz, ki mu sledi definiranje enot skupaj z njihovimi fazami, ki se jih izbere iz nabora predhodno definiranih razredov faz. Ta del razvoja opravijo strokovnjaki za vodenje sistemov. V tej točki se razvoj sistema vodenja zaključi, nadaljnje delo do dokončanja funkcionalnega recepturnega sistema vodenja pa opravijo procesni inženirji, to je uporabniki sistema vodenja.

Delo procesnih inženirjev pri gradnji recepturnega sistema vodenja se začne z definiranjem razredov enot, pri čemer se vsakemu novemu razredu enot dodeli ustrezna podmnožica enot, in to tako, da je posamezna enota lahko dodeljena večjemu številu

razredov enot. Razred enot vsebuje množico razredov faz, ki je presek množic faz posameznih enot, ki so pridružene temu razredu enot. Določeno enoto je možno dodati razredu enote le v primeru, če dodajanje te enote ne bi povzročilo, da množica razredov faz razreda enote postane prazna. Če bi po dodajanju enote množica razredov faz razreda enot postala prazna, mora urejevalnik opreme dodajanje te enote preprečiti.

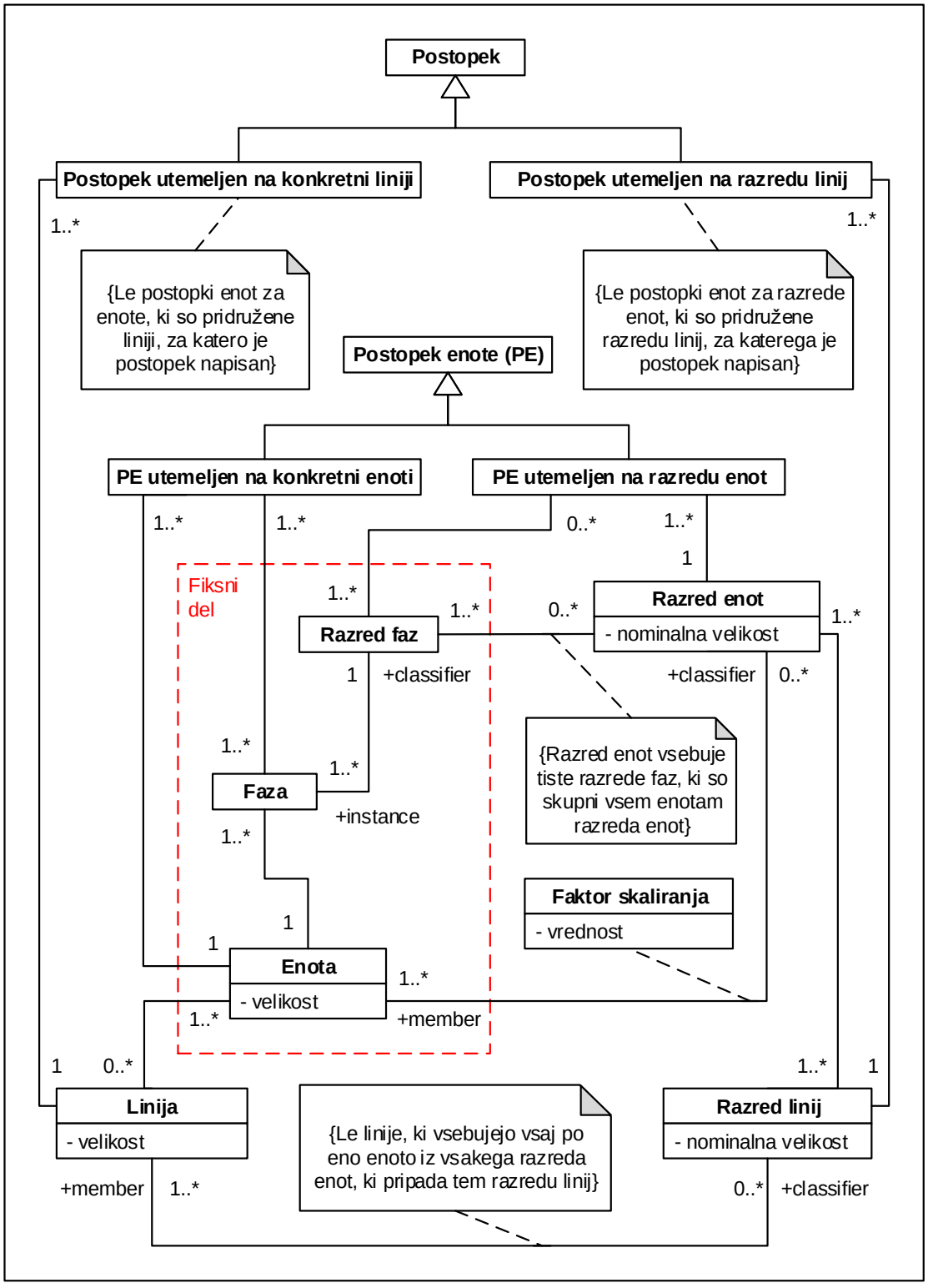
V tej točki naj vpeljemo pojem velikosti enote in razreda enote. Vsaka enota ima atribut velikosti (prostornine), vsak razred enot pa ima atribut nominalne velikosti (nominalne prostornine), ki je enaka velikosti njegove največje enote. Poleg tega ima vsaka enota osnovni faktor skaliranja glede na razred enot, ki mu pripada. Očitno je torej, da lahko ima določena enota različne osnovne faktorje skaliranja glede na različne razrede enot, ki jim pripada. Torej, osnovni faktor skaliranja enote lahko smatramo za asociacijski razred med enoto in enim od razredov enot, ki jim ta enota pripada.

Gradnja sistema se nadaljuje z definiranjem postopkov enot (ker smo izpustili nivo operacij), ki so lahko napisani za razred enot ali za konkretno enoto. Če je postopek enote napisan za razred enot, lahko vsebuje le razrede faz tega razreda enot, če pa je napisan za konkretno enoto, lahko vsebuje le faze te enote.

Naslednji korak je definiranje linij, ki so sestavljene iz enot, in razredov linij, ki so sestavljene iz razredov enot in posplošujejo linije. Tukaj se spet srečamo z nekaj pravili in omejitvami. Tako, na primer, za vsak razred enot, ki pripada razredu linij, obstaja zahteva, da mora vsaj ena od njegovih enot pripadati vsakem razredu enot razreda linij. Z drugimi besedami, razredu linij lahko pridružimo le tiste linije, ki imajo vsaj po eno enoto v vsakem razredu enot tega razreda linij. Nadalje, vsaka linija ima atribut velikosti (prostornine) in vsak razred linij ima atribut nominalne velikosti (nominalne prostornine). Velikost določene linije je enaka velikosti njene dominantne enote, ki jo je določil procesni inženir (tehnolog). Ob tem naj omenimo, da dominantna enota linije ni nujno njena največja enota. In nazadnje, nominalna velikost razreda linij je enaka velikosti največje linije tega razreda linij.

Zadnji korak pri gradnji recepturnega sistema vodenja je definiranje temeljnih receptov (ang. *master recipes*) in njihovih postopkov recepta, ki so lahko napisani za razred linij ali za konkretno linijo. Če je postopek recepta napisan za razred linij, lahko vsebuje le postopke enot, napisane za kateri koli razred enot, vsebovan v tem razredu linij. Če pa je postopek recepta napisan za linijo, lahko vsebuje le postopke enot, napisane za katero koli enoto, vsebovano v tej liniji.

Celoten nov model opreme in postopkovnega vodenja, omejen navzgor do nivoja linij in postopkov recepta, prikazuje Slika 23.



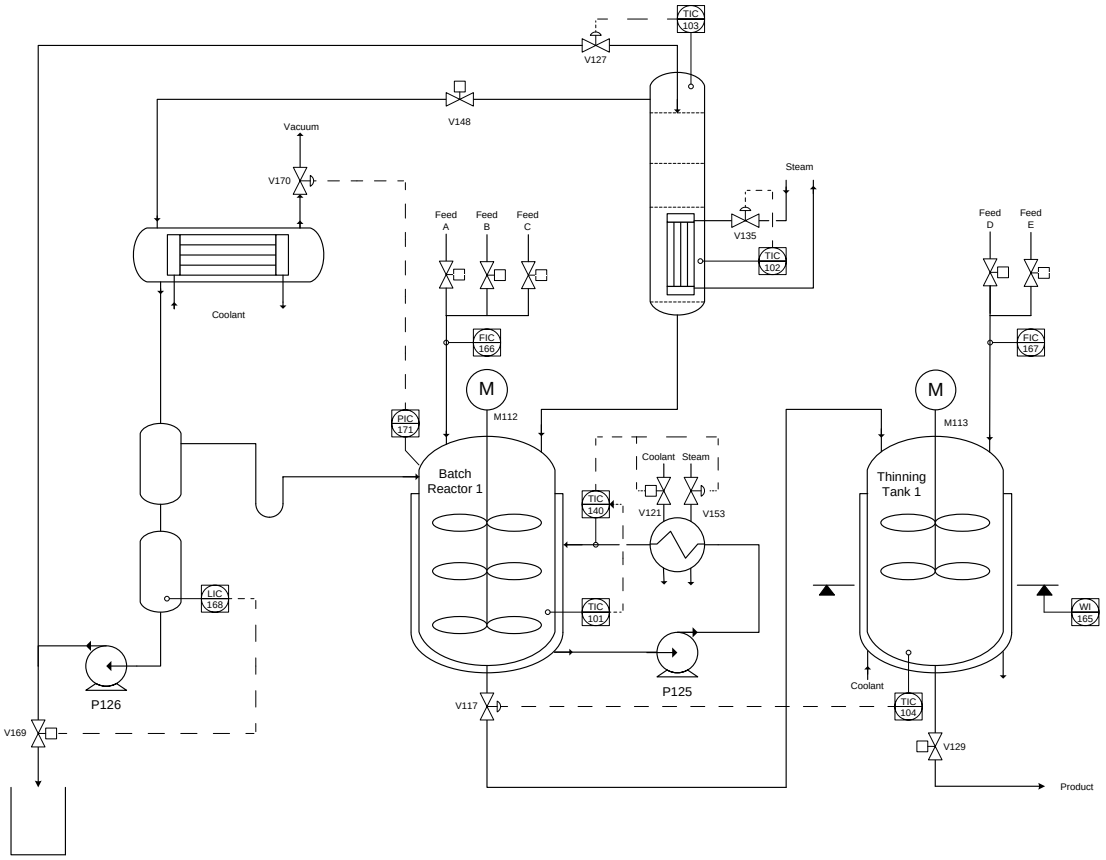
Slika 23: Nov model opreme in postopkovnega vodenja do nivoja linij in postopkov recepta.

5.3 Uporaba novega koncepta

Vrednotenje novega koncepta smo opravili z uporabo orodja PLCbatch, v katerem je koncept implementiran. Vrednotenje je potekalo v okviru realnega industrijskega projekta izvedbe sistema vodenja šaržnega procesa sinteze smol [68]. Pri tem smo uporabili temeljna načela pristopa »študija primera« (ang. *case study*), ki predstavlja običajno metodo za vrednotenje v programskem inženirstvu [80]. Študija primera je opisana v naslednjem podpoglavju.

5.3.1 Opis študije primera novega koncepta

Šaržni proces, ki smo ga uporabili v študiji primera, je proces sinteze smol v tovarni Helios Količevo (takrat Color d.d. Medvode). Proces sestoji iz ene procesne celice, ki je razdeljena na šest linij, od katerih vsaka vsebuje po dve enoti – šaržni reaktor in razredčevalno posodo. Reaktor ima možnost regulacije temperature z uporabo grelno/hladilnega sistema v plašču reaktorja. Slika 24 prikazuje eno od šestih linij procesa sinteze smol.



Slika 24: Linija procesa sinteze smol.

Proizvodni proces se začne z doziranjem potrebnih surovin v šaržni reaktor. Po

končanem doziranju se vsebina reaktorja segreje do temperature, ki je potrebna za začetek reakcije. Segrevanje se izvaja z regulacijo temperature po premici s predpisanim naklonom. Pri gretju vsebine reaktorja se sprošča para, ki gre skozi kondenzator, kjer pride do ločevanja destilacijske vode kot stranskega izdelka. Hkrati s segrevanjem se izvaja tudi vakuumiranje, ki pospeši proces sinteze. Po dokončanem procesu sinteze se izdelana smola prečrpa v razredčevalno posodo, ki mora biti predhodno napolnjena z zahtevano količino topil. Vsebino razredčevalne posode je med redčenjem smole potrebno hladiti, temperaturo v posodi pa se uravnava s spreminjanjem hitrosti iztiskanja smole iz reaktorja v razredčevalno posodo. Med redčenjem je vsebino razredčevalne posode potrebno nenehno mešati. Na koncu se izvede preverjanje kakovosti, nakar je vsebina razredčevalne posode pripravljena za prečrpavanje v shranjevalne posode.

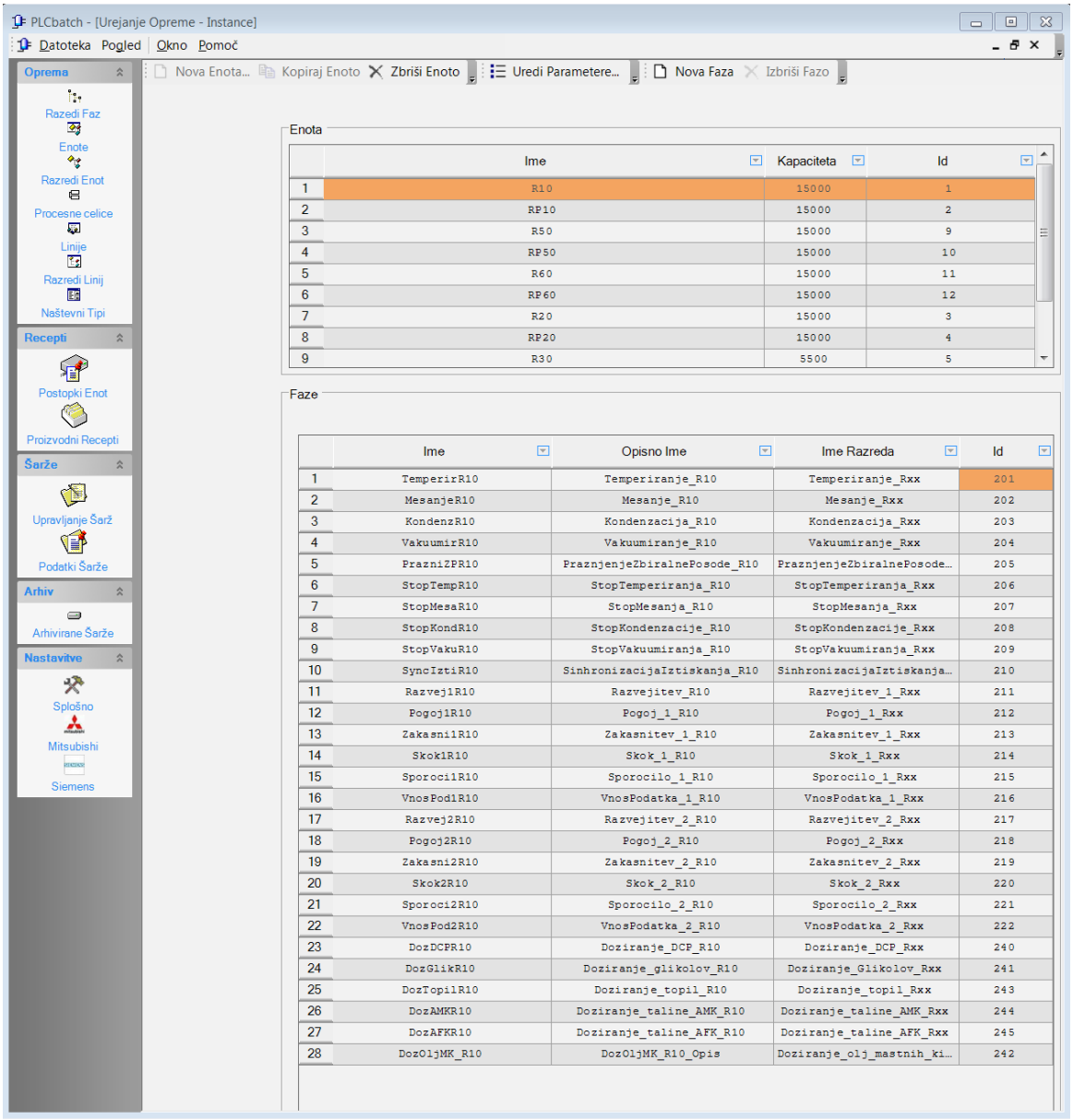
Šest reaktorskih enot se medsebojno razlikuje glede na zmožnost (nabor faz posameznega reaktorja) in na velikost (prostornino). Reaktorji R1, R5 in R6 so veliki, reaktor R2 je srednje velik, reaktorja R3 in R4 pa sta mala. Po zmožnosti se reaktorji razlikujejo predvsem glede na možnost doziranja posameznih surovin, kar pomeni, da ni možno dozirati vseh surovin v vse reaktorje. Edina razlika v zmožnosti reaktorjev, ki se ne nanaša na doziranje surovine, je faza za sinhronizacijo konca procesa z razredčevalno posodo, ki jo imajo reaktorji R1, R2, R3, R5 in R6, reaktor R4 pa te faze nima, ker je pri tem reaktorju sinhronizacijo konca z razredčevalno posodo treba izvesti ročno, s potrditvijo operaterja. Surovine so razvrščene v sedem skupin: diciklopentadien, glikoli, olja in mastne kisline, topila, raztopina AMK, raztopina AFK in stiren. Slednjega ni možno dozirati v reaktorje, temveč le v razredčevalne posode, torej imamo šest skupin surovin, ki se jih lahko dozira v določene reaktorje. V reaktorja R1 in R2 je možno dozirati surovine vseh šestih omenjenih skupin; reaktorji R4, R5 in R6 podpirajo doziranje surovin petih skupin (brez skupine olj in mastnih kislin); reaktor R3 pa podpira doziranje surovin štirih skupin (brez skupin diciklopentadiena in raztopine AMK).

Recepturni sistem vodenja je bil razvit z uporabo orodja PLCbatch, v katerem je implementiran zgoraj opisan nov objektni model opreme in postopkovnega vodenja, kar je omogočilo visoko stopnjo ponovne uporabe receptov in bistveno zmanjšanje obsega podvajanja informacij v receptih. Kot je že omenjeno v poglavju 5.2 se z razvojem in konfiguriranjem sistema recepturnega vodenja v skladu z novim modelom prvi začnejo ukvarjati strokovnjaki za vodenje sistemov. Pri tem je prvi korak določanje vseh razredov faz, ki lahko nastopajo na katerikoli enoti sistema. Gre torej za vse elementarne entitete postopkovnega vodenja v sistemu, ali z drugimi besedami, za vse elementarne funkcije v procesu. Seveda je predhodno same faze potrebno sprogramirati na PLK-ju. Slika 25 prikazuje okno za upravljanje razredov faz v orodju PLCbatch. Na sliki lahko vidimo, da je vseh razredov faz 47.



Slika 25: Vsi razredi faz sistema vodenja procesa sinteze smol.

Naslednji korak pri konfiguriranju sistema vodenja je definiranje enot in za vsako enoto nabor njenih faz, ki jih razvijalec izbere iz nabora vseh v predhodnem koraku definiranih razredov faz. Faze posamezne enote so torej instance prej definiranih razredov faz. Kot je že omenjeno zgoraj, je vseh enot dvanajst, in sicer šest reaktorskih enot in šest razredčevalnih posod. Slika 26 prikazuje prikazuje okno za upravljanje enot v orodju PLCbatch. V tabeli v zgornjem delu okna vidimo seznam enot, v tabeli v spodnjem delu okna pa vidimo vse faze trenutno izbrane enote (reaktorja R10). Vidimo lahko, da ima ta reaktor 28 faz.



Slika 26: Okno za upravljanje enot v orodju PLCbatch.

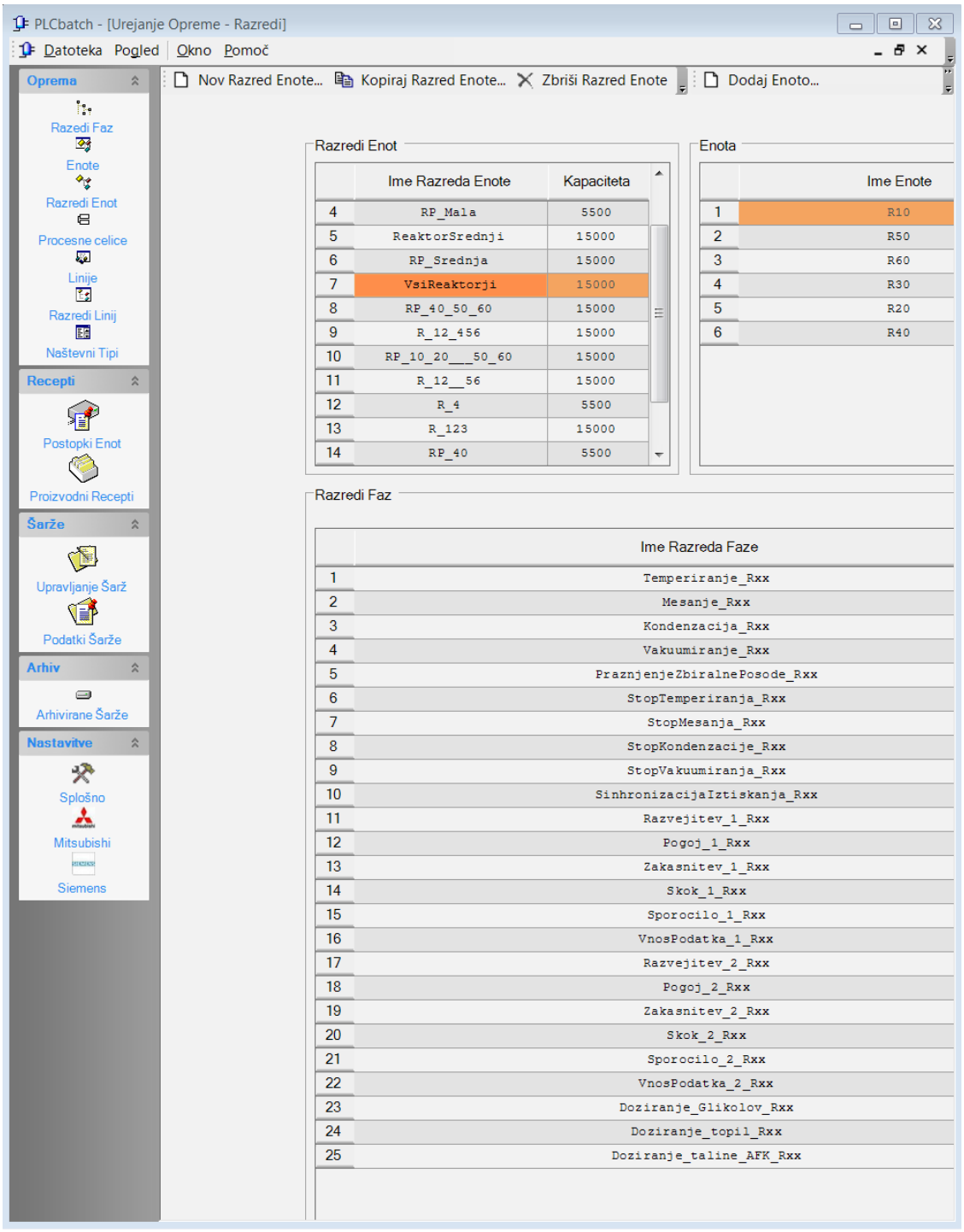
Tukaj se zaključi delo razvijalcev sistema vodenja, saj so v zgoraj opisanih korakih obdelane vse tri skupine entitet, ki jih v novem modelu definirajo razvijalci. V nadaljevanju

delo prevzamejo procesni inženirji, ki najprej definirajo razrede enot. Za primer si oglejmo razrede reaktorskih enot, ki so definirane v trenutni konfiguraciji sistema vodenja procesa sinteze smol (kot je bilo omenjeno zgoraj, lahko procesni inženirji kadarkoli med življenjskim ciklom sistema vodenja poljubno spreminjajo nabor razredov enot). Procesni inženirji so definirali naslednje razrede (reaktorskih) enot (pri čemer je iz imena posameznega razreda enot razvidno, kateri reaktorji so vključeni v ta razred): R_12_56, R_12_456, R_123, R_4, VsiReaktorji. Kot je bilo opisano zgoraj, vsak razred enot vsebuje množico razredov faz, ki je presek množic razredov faz prirejenih posameznim enotam tega razreda enot. Tako ima, na primer, razred enot R_123 iste razrede faz (26 razredov) kot enota R3, ker je nabor razredov faz te enote podmnožica razredov faz enot R1 in R2 (ki imata enak nabor razredov faz). Drug primer je razred enot VsiReaktorji, ki ima od razredov faz za doziranje surovin le tiste za doziranje surovin iz skupin glikolov, topil in raztopine AFK, saj so te tri skupine surovin edine, ki jih lahko doziramo v vse reaktorje. Slika 27 prikazuje okno za upravljanje razredov enot, v katerem je v tabeli zgoraj levo izbran razred enot VsiReaktorji. V tabeli zgoraj desno vidimo, da ta razred enot vsebuje vseh šest reaktorskih enot, v spodnji tabeli pa vse razrede faz tega razreda enot, ki jih je 25. Pomen tega je enostaven. Če želimo določene recepte izvajati na vseh reaktorjih, je jasno, da lahko ti recepti vsebujejo le faze, ki so skupne vsem reaktorjem, teh faz pa je 25. Omenjene recepte torej enostavno pišemo za razred enot VsiReaktorji, pri čemer nam bo urejevalnik recptov omogočal uporabo natanko teh 25 omenjenih faz.

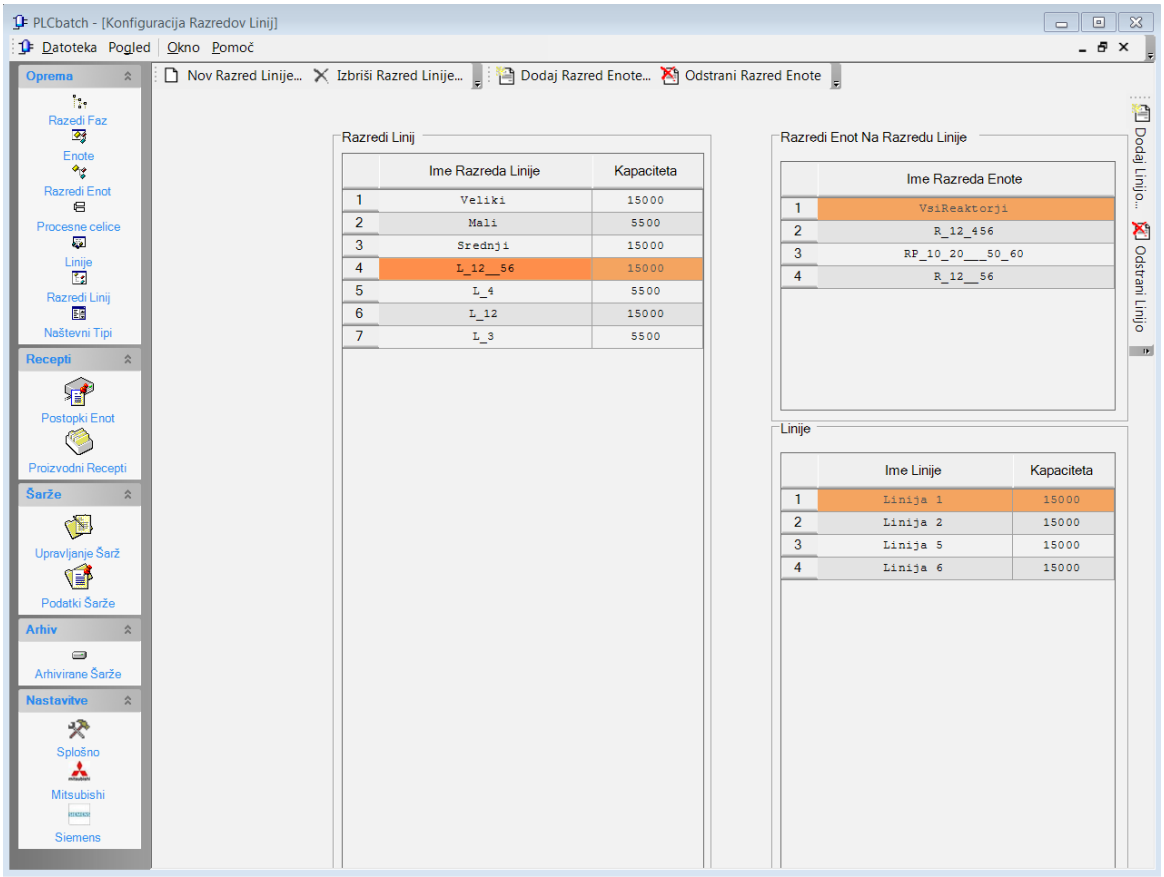
V nadaljevanju so bili definirani naslednji štirje razredi linij:

1. razred linij *L_12*, ki vsebuje reaktorska razreda enot R_123 in VsiReaktorji,
2. razred linij *L_12_56*, ki vsebuje reaktorske razrede enot R_12_56, R_12_456 in VsiReaktorji,
3. razred linij *L_3*, ki vsebuje reaktorska razreda enot R_123 in VsiReaktorji, in
4. razred linij *L_4*, ki vsebuje reaktorske razrede enot R_4, R_12_456 in VsiReaktorji.

Slika 28 prikazuje okno za upravljanje razredov linij. V tabeli zgoraj levo vidimo vse štiri zgoraj omenjene razrede linij, poleg tega pa še tri razrede linij, ki so v sistemu definirani zaradi preizkušanja različnih struktur procesa, vendar se trenutno ne uporabljajo. Na sliki vidimo, da je izbran razred linij *L_12_56*. V tabeli zgoraj desno, vidimo razrede enot pripadajoče temu razredu linij – poleg treh zgoraj omenjenih reaktorskih razredov enot še razred enot razredčevalnih posod RP_10_20_50_60. V tabeli desno spodaj vidimo linije, ki so v tem razredu linij.

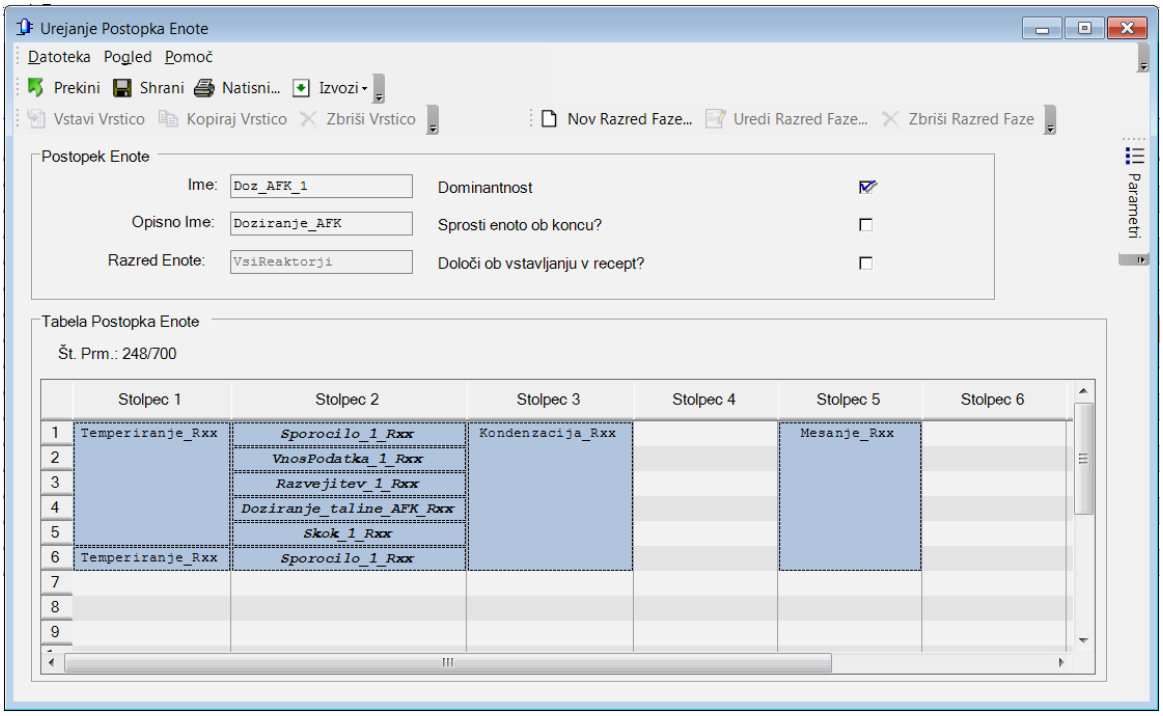


Slika 27: Razred enot VsiReaktorji, njegove enote in njegovi razredi faz.



Slika 28: Okno za upravljanje razredov linij v orodju PLCbatch.

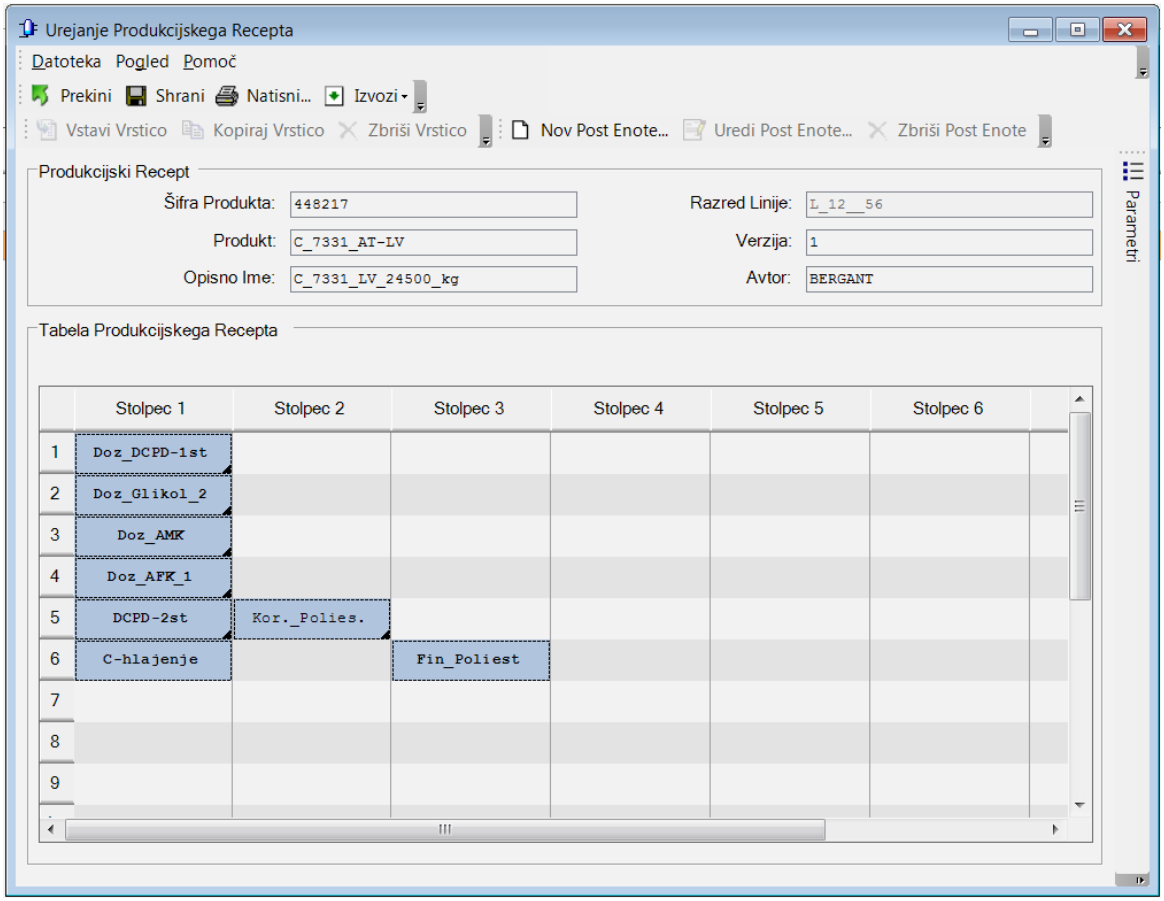
Za konkretno ilustracijo koristi, ki jih je prinesel nov model, si podrobno oglejmo del zgoraj opisanega projekta iz študije primera. Postopki enot (napisani za razred enote) z največjim potencialom za ponovno uporabo so tisti, napisani za razred enot VsiReaktorji, saj te postopke enot lahko uporabimo v glavnem receptu (oziroma v njegovem postopku recepta) za katerega koli od štirih zgoraj naštetih razredov linij (*L_12*, *L_12_56*, *L_3* in *L_4*). To pomeni, da bodo taki postopki enot napisani le enkrat namesto štirikrat. Kot primer takega postopka enote (za razred enote) si oglejmo postopek enote Doz_AFK_1, ki ga prikazuje Slika 29.



Slika 29: Postopek enote za doziranje AFK.

Postopek enote Doz_AFK_1 vsebuje, poleg faz, ki izvajajo temperiranje, kondenzacijo in mešanje (stolpci 1, 3 in 5), tudi kombinacijo sistemskih in tehnoloških faz (stolpec 2), ki služijo za izbiro načina doziranja AFK (avtomatsko v obliki raztopine ali ročno v obliki lusk) in nato izvedbo doziranja na izbran način. Jedro omenjene kombinacije faz predstavljata fazi ***Razvejitev*** in ***Skok***, ki skupaj tvorita ustrezno »OR« strukturo recepta. Kombinacija faz se začne v drugi vrstici s fazo VnosPodatka_1_Rxx, ki zahteva vnos operaterjeve izbire načina doziranja. Če je izbran ročni način, faza Razvejitev v tretji vrstici izvede skok v šesto vrstico, kjer faza Sporočilo_1_Rxx posreduje sporočilo operaterju z zahtevo po izvedbi ročnega doziranja ter čaka na operaterjevo potrditev, da je doziranje opravljeno. Po prejemu te potrditve se izvajanje faze Sporočilo_1_Rxx zaključi in recept preide v sedmi korak, kjer se izvajanje tega postopka enote zaključi. Če pa je operater pri izbiri v drugi vrstici izbral avtomatski način doziranja, recept po zaključku faze Razvejitev v tretji vrstici z izvajanjem nadaljuje v naslednji, četrti vrstici, kjer se nahaja faza Doz_AFK_1, ki izvede avtomatsko doziranje raztopine AFK. Po zaključku faze avtomatskega doziranja recept preide v peto vrstico, kjer faza Skok izvede skok v sedmo vrstico recepta, kjer se izvajanje postopka enote zaključi.

Kot smo že omenili, zgoraj opisani postopek enote lahko najdemo v večjem številu glavnih receptov za vse štiri razrede linij, zaradi česar je zelo pomembna možnost njegove ponovne uporabe. Slika 30 prikazuje postopek recepta za razred linij *L_12_56*, v katerem je opisani postopek enote uporabljen (četrta vrstica prvega stolpca).



Slika 30: Uporaba postopka enote za doziranje AFK v postopku recepta.

5.3.2 Ugotovitve preizkušanja novega koncepta

Izkušnje z uporabo novega orodja so pokazale, da ima nov objektni model za posledico nekoliko višji nivo abstrakcije recepturnega sistema in posledično tudi kompleksnosti njegove uporabe, po drugi strani pa omogoča popolno eliminacijo podvajanja informacij v receptih in bistveno povečanje njihove ponovne uporabe. Omeniti je treba še, da orodje omogoča tudi gradnjo objektnih modelov opreme in postopkovnega vodenja na klasičen način, kot se to izvaja v drugih orodjih za šaržno vodenje, in da se uporabnik sam odloča, ali bo uporabil nov ali klasični način.

5.4 Vrednotenje novega koncepta

Uspešnost povečanja stopnje ponovne uporabe receptov in zmanjšanja obsega podvajanja informacij v receptih lahko ugotovimo na podlagi zmanjšanja števila postopkov temeljnih receptov in postopkov enot, ki so bili kreirani z uporabo novega objektnega modela. Le-to primerjamo s številom postopkovnih elementov, ki bi jih za doseganje enake procesne funkcionalnosti potrebovali z uporabo standardnega objektnega modela. V nadaljevanju

tega poglavja bomo predstavili rezultate analize študije primera s ciljem izvedbe omenjenega vrednotenja.

Število temeljnih receptov je 262, medtem ko bi jih z uporabo standardnega modela potrebovali 340. Število kreiranih postopkov enot je 43, le-teh pa bi z uporabo standardnega modela potrebovali kar 108, kar pomeni 60% zmanjšanje (ali zmanjšanje za faktor 0,4). Vseh 65 postopkov enot, ki bi jih dodatno potrebovali ob uporabi standardnega modela, bi v bistvu predstavljalo obseg podvajanja informacij v receptih. Pri tem je treba omeniti, da je zmanjšanje števila postopkov enot (60%, kot je omenjeno zgoraj) veliko bolj pomembno kot zmanjšanje temeljnih receptov, ker so postopki enot veliko bolj kompleksni in njihov razvoj in konfiguriranje zahtevata veliko več časa in napora, kot pa to velja pri temeljnih receptih. Tabela 4 prikazuje najpomembnejše podatke iz naše študije primera, to je število postopkov enot, ki so bili napisani za vsak razred enot, v primerjavi s številom postopkov enot, ki bi jih za enako funkcionalnost potrebovali, če bi uporabili standardni objektni model opreme in postopkovnega vodenja.

V nekaterih primerih bi se lahko zgodilo, da bi število postopkov enot, ki so bili definirani ob uporabi novega modela, bilo večje, kot če bi uporabili standardni objektni model. Do tega bi lahko prišlo v primeru definiranja gradnikov receptov zelo drobne granularnosti (vsebinsko gledano pod nivojem postopka enote), v katere bi združevali segmente enakega procesiranja na različnih razredih enot. V tem primeru bi zmanjšanje števila postopkov enot zaradi uporabe načela »vsako procesiranje zapiši le enkrat« bilo manjše od povečanja števila postopkov enot zaradi delitve procesiranja in ločenega zapisovanja delov procesiranja, ki so enaki za večje število postopkov enot (in ki jih je lahko veliko število in so zelo majhnega obsega procesiranja). Vendar pa bi tudi v tem primeru obstala bistvena korist uporabe novega objektnega modela, ki je v izogibanju podvajanja informacij. Tukaj naj omenimo, da so v predstavljeni študiji primera vsi postopki enot bili »pravi« postopki enot, in ne taki, ki bi nastali zaradi zgoraj opisanega združevanja manjših segmentov enakega procesiranja na različnih razredih enot. Zaradi tega je tudi bilo tako enostavno izračunati zgoraj omenjeno zmanjšanje števila postopkov enot; število postopkov enot, ki bi jih za enako funkcionalnost potrebovali ob uporabi standardnega objektnega modela, smo izračunali tako, da smo vsak postopek enote našega primera pomnožili s številom različnih razredov enot, za katere je ta postopek enote bil napisan.

Tabela 4: Zmanjšanje števila postopkov enot.

Razred enot	Število postopkov enot ob uporabi standardnega modela	Število postopkov enot ob uporabi novega modela	Faktor izboljšanja
R_12_56	2	1	2.0
R_12_456	51	17	3.0
R_123	36	18	2.0
R4	3	3	1.0
VsiReaktorji	16	4	4.0
Skupaj	108	43	2.5

Na koncu naj omenimo še, da bi izboljšanje v ponovni uporabi receptov bilo še večje, če ne bi obstajala omejitev števila recepturnih parametrov postopka enote (ki je 20 v trenutni verziji orodja PLCbatch, v sklopu katerega je bil nov objektni model opreme in postopkovnega vodenja implementiran in preizkušen).

6 Zaključki

Disertacija obravnava modele programske opreme za vodenje šaržnih procesov.

Vodenje šaržnih procesov, ki je kombinacija zveznega in sekvenčnega vodenja, je bolj kompleksno kot vodenje čistih zveznih ali čistih sekvenčnih procesov. Še posebej se ta povečana kompleksnost pozna pri programski opremi. S ciljem boljšega obvladovanja te kompleksnosti je nastal standard ISA-88, ki je uvedel referenčne modele, terminologijo in koncepte za opis šaržne proizvodnje in vodenje šaržnih procesov na osnovi koncepta modularne šaržne avtomatizacije. Uvajanje standarda ISA-88 je omogočilo velik napredek na področju obvladovanja kompleksnosti programske opreme za vodenje šaržnih procesov, žal pa ni rešilo vseh problemov.

V disertaciji smo se ukvarjali s tremi področji, na katerih smo opravili analizo trenutnega stanja stroke in opredelili nekatere odprte probleme.

Prvo je področje abstrakcije receptov, kjer smo ugotovili potrebo po uvedbi bolj enostavne notacije ob čim manjši izgubi izrazne moči.

Drugo je področje modela obnašanja faz, kjer smo ugotovili, da je model obnašanja faz, kot ga določa standard ISA-88 in je implementiran v večini orodij za vodenje šaržnih procesov, preveč poenostavljen. To ima za posledico preveč kompleksno uporabniško kodo, kar slabo vpliva na njeno kakovost in na obvladljivost procesa njenega razvoja.

Tretje je področje objektne predstavitve opreme in postopkovnega vodenja. Ugotovili smo, da objektni model žal v svoji trenutni obliki ni dovolj uspešen pri odpravljanju podvajanja informacij v receptih in v povečanju stopnje njihove ponovne uporabe.

V disertaciji smo vse tri zgoraj naštete probleme na izviren način rešili.

Nova abstrakcija receptov se je izkazala kot izredno kompaktna, enostavna in razumljiva operaterjem, ki jo brez izjeme sprejemajo zelo pozitivno.

Za nov model obnašanja faz se je med preizkušanjem v praksi pokazalo, da ima velik potencial za dvig kakovosti in izboljšanje obvladovanja kompleksnosti med življenjskim ciklom programske opreme za vodenje šaržnih procesov.

Nov objektni model opreme in postopkovnega vodenja omogoča popolno eliminacijo podvajanja informacij v receptih in bistveno povečanje njihove ponovne uporabe, uporabniku pa pušča možnost izbire, do kakšne stopnje bo uporabljal nov model ter na ta način sebi prilagodil razmerje med učinkovitostjo in kompleksnostjo uporabljenega

modela.

Z omenjenimi rešitvami smo potrdili hipotezo podano v uvodu.

Originalni prispevki disertacije so naslednji:

1. Zasnova, razvoj in preizkušanje nove abstrakcije receptov za izvajanje na platformi programirljivih logičnih krmilnikov, ki ima po eni strani izrazno moč podobno izrazni moči jezika SFC, po drugi strani pa je bolj enostavna tako za razumevanje in uporabo s strani operaterjev kot tudi za implementacijo na omenjeni platformi.
2. Zasnova, razvoj in preizkušanje novega modela obnašanja faz, ki temelji na razširjeni abstrakciji končnih avtomatov in ima večjo izrazno moč kot aktualen model ter omogoča boljše obvladovanje procesa razvoja postopkovnega dela vodenja šaržnih procesov.
3. Definiranje, razvoj, preizkušanje in validacija novega objektnega modela opreme in postopkovnega vodenja, ki vsebuje najsodobnejše koncepte objektne tehnologije, vključno s konceptom večkratnega dedovanja, in ki omogoča doseganje fleksibilne strukture razredov opreme in z njo nižanje obsega podvajanja informacij v receptih in njihovo večjo ponovno uporabo.

Z naštetimi rezultati smo potrdili hipotezo, podano v uvodu te disertacije, da je mogoče zasnovati nove modele programske opreme za šaržno vodenje, ki bodo poenostavili implementacijo na krmilnikih, olajšali uporabo operaterjem, izboljšali obvladovanje razvojnega procesa in omogočili bistveno zmanjšanje podvajanja informacij v receptih in njihovo večjo ponovno uporabo.

7 Zahvale

V prvi vrsti se iskreno zahvaljujem mentorju prof. dr. Stanku Strmčniku za dolgoletno vzpodbujanje in podporo.

Posebej se zahvaljujem sodelavcem podjetja Inea d.o.o. Janezu Tancku, dipl. inž., Gregorju Puhu, dipl. inž., mag. Igorju Steinerju in Petru Kosinu, univ. dipl. inž., s katerimi sem sodeloval pri razvoju orodij in sistemov vodenja šaržnih procesov, ob čemer so rezultati, predstavljeni v tej disertaciji, bodisi nastajali bodisi bili uvajani in preizkušani v praksi.

Iskreno se zahvaljujem sodelavcem podjetij Color d.d. Medvode (sedaj Helios Količevo d.o.o.) in Cinkarna Celje d.d., ki so mi zaupali ter mi omogočili uvajanje in preizkušanje novih konceptov v sisteme vodenja njihovih šaržnih procesov; tukaj bi še posebej izpostavil Franca Berganta, Romana Pestotnika in Nikolajo Podgoršek Selič, univ. dipl. inž.

Zahvaljujem se tudi vsem sedanjim in bivšim ožjim sodelavcem z Odseka za sisteme in vodenje IJS, s katerimi sem si tekom let izmenjeval raziskovalne, razvojne in aplikativne izkušnje pri vodenju šaržnih procesov, predvsem pa dr. Tomažu Lukmanu, ki je sodeloval pri delu raziskav, ki so predmet te disertacije, in dr. Gregorju Dolancu, ki je sodeloval pri industrijskem projektu, v katerem so rezultati teh raziskav bili preneseni v prakso.

Zahvaljujem se tudi ministrstvom in javnim agencijam Republike Slovenije (MIZŠ, ARRS, JAPTI), ki so sofinancirale večje število raziskovalnih in razvojnih projektov in programov, v sklopu katerih so nastajala znanja in izkušnje, nepogrešljive pri raziskavah, ki so predmet te disertacije.

Na koncu se zahvaljujem tudi Vesni in Emmi, ki sta skozi leta bili ob meni in me potrpežljivo in z ljubeznijo podpirali.

8 Literatura in viri

[1] ISA-88.00.01 – 1995, Batch Control Part 1: Models and Terminology.

[2] ISA-88.00.02 – 2001, Batch Control Part 2: Data Structures and Guidelines for Languages.

[3] ISA-88.00.03 – 2003, Batch Control Part 3: General and Site Recipe Models and Representation.

[4] ISA-88.00.04 – 2006, Batch Control Part 4: Batch Production Records.

[5] E. Muñoz, A. Espuña, L. Puigjaner, Towards an ontological infrastructure for chemical batch process management, Computers and Chemical Engineering 34 (2010) 668–682.

[6] E. Muñoz, E. Capón-García, A. Espuña, L. Puigjaner, Ontological framework for enterprise-wide integrated decision-making at operational level, Computers and Chemical Engineering 42 (2012) 217–234.

[7] IEC-61512-1:1997, Batch Control – Part 1: Models and terminology.

[8] IEC-61512-2:2001, Batch Control – Part 2: Data structures and guidelines for languages.

[9] IEC-61512-3:2008, Batch Control – Part 3: General and site recipe models and representation.

[10] IEC-61512-4:2009, Batch Control – Part 4: Batch production records.

[11] M. Colla, T. Leidi, M. Semo, Design and implementation of industrial automation control systems: A survey. Zbornik IEEE International Conference on Industrial Informatics (INDIN'09), Cardiff, Velika Britanija.

[12] F. Vinther, Batch 10+ years after S88, in: WBF European Conference, Mechelen, Belgium, 2006.

[13] M.L., Crane, J. Dingel, UML vs. Classical vs. Rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6 (4), 2007, 415-435.

[14] G, Booch, I. Jacobson, J. Rumbaugh (1999). The Unified Modelling Language User Guide, Addison Wesley.

[15] D. Howe (editor), The free on-line dictionary of computing. http://www.foldoc.org/.

[16] C. Huizing, W.P., de Roever, Introduction to design choices in the semantics of statecharts. *Information Processing Letters* 37 (4), 1991, 205-213.

[17] D. Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8 (1987), 231-274, North Holland.

[18] D. Harel, A. Naamad, The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 5 (4), 1996, 293-333.

[19] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, On the formal semantics of statecharts. In: Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, 1987,pp. 54-64. Computer Society Press of the IEEE.

[20] A. Pnueli, M. Shalev, What is in a step: On the semantics of statecharts. In: Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Lecture Notes in Computer Science, vol. 526, pp. 244-264. Springer, Heidelberg, 1991.

[21] M. Von der Beeck, A comparison of statecharts variants. In: Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94), Lecture Notes in Computer Science, vol. 863, pp. 128-148. Springer, Heidelberg, 1994.

[22] D. Harel, M. Politi, Modeling Reactive Systems with Statecharts: The STATEMATE Approach. McGraw-Hill, 1998.

[23] OMG, Unified modelling language: Superstructure version 2.4.1. Document formal/2011-08-06, Object Management Group, 2011.

[24] D. Harel, H. Kugler, The RHAPSODY semantics of statecharts (or, on the executable core of the UML) (preliminary version). In: SoftSpez Final Report, Lecture Notes in Computer Science, vol. 3147, pp. 325-354. Springer, Heidelberg, 2004.

[25] K. Thramboulidis, IEC 61499 in factory automation. In: Proceedings of IEEE International Conference on Industrial Electronics, Technology and Automation (IETA'05), Bridgeport, CT, USA, 2005. Springer, pp 115-124.

[26] Y. Chia-han, V. Vyatkin, Model transformation between MATLAB simulink and Function Blocks. In: Proceedings of IEEE International Conference on Industrial Informatics (INDIN '10), Osaka, Japan, 2010,. IEEE, pp 1130-1135.

[27] J.A. Lewis, S.M. Henry, D.G. Kafura, R.S. Schulman, An Empirical Study of the Object-oriented Paradigm and Software Reuse, in: ACM Conference Proceedings on Object-oriented Programming Systems, Languages, and Applications OOPSLA '91, New York, 1991, pp. 184–196.

[28] M. Stark, Impacts of Object-oriented Technologies: Seven Years of Software Engineering, Journal of Systems and Software 23(2) (1993) 163–169.

[29] J.M. Armstrong, R.J. Mitchell, Uses and Abuses of Inheritance, Software Engineering Journal 9(1) (1994) 19–26.

[30] L. Bettini, M. Loreti, B. Venneri, On Multiple Inheritance in Java, in: T. D' Hondt (Ed.) Technology of Object-Oriented Languages, Systems and Architectures, Springer Science+Business Media, New York, 2009, pp. 1–15.

[31] B.S. Heck, L.M. Wills, G.J. Vachtsevanos, Software Technology for Implementing Reusable, Distributed Control Systems, IEEE Control Systems Magazine 23(1) (2003)

21–35.

[32] L.F. Capretz, P.A. Lee, Object-oriented Design: Guidelines and Techniques, Information and Software Technology 35(4) (1993) 195–206.

[33] I.S. Deligiannis, M. Shepperd, S. Webster, M. Roumeliotis, A Review of Experimental Investigations into Object-oriented Technology, Empirical Software Engineering 7(3) (2002) 193–231.

[34] P. Coad, M. Mayfield, Java-Inspired Design: Use Composition, Rather Than Inheritance, American Programmer 10 (1997) 22–31.

[35] A.J. Riel, Object-oriented Design Heuristics, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[36] B. Alpern, A. Cocchi, S. Fink, D. Grove, Efficient Implementation of Java Interfaces: Invokeinterface Considered Harmless, In: Proceedings of the 16th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications OOPSLA ’01, New York, 2001, pp. 108–124.

[37] G.B. Singh, Single Versus Multiple Inheritance in Object Oriented Programming, SIGPLAN OOPS Mess 5(1) (1994) 34–43.

[38] E. Tempero, R. Biddle, Simulating Multiple Inheritance in Java. Journal of Systems and Software 55(1) (2000) 87–100.

[39] K. Thirunarayan, G. Kniesel, H. Hampapuram, Simulating Multiple Inheritance and Generics in Java, Computer Languages 25(4) (1999) 189–210.

[40] M.J. Egenhofer, A. Frank, Object-oriented Modeling for GIS, Journal of the Urban and Regional Information Systems Association 4(2) (1992) 3–19.

[41] F.T. Fonseca, M.J. Egenhofer, C.A. Davis Jr, K.A.V. Borges, Ontologies and Knowledge Sharing in Urban GIS, Computers, Environment and Urban Systems 24(3) (2000) 251–272.

[42] B. Vogel-Heuser, D. Witsch, U. Katzke, Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. Zbornik IEEE International Conference on Control and Automation (ICCA’05), Budimpešta, Madžarska.

[43] M. Maurmaier, Leveraging model-driven development for automation systems development. Zbornik IEEE International Conference on Emerging Technologies and Factory Automation (ETFA’08), Hamburg, Nemčija.

[44] D. Friedrich, B. Vogel-Heuser, Benefit of system modeling in automation and control education. Zbornik American Control Conference (ACC’07), New York, ZDA.

[45] S. Braun, M. Obermaier, B. Vogel-Heuser, Usability evaluation of modeling notations for software engineering in machine and plant automation, Zbornik IEEE International Conference on Emerging Technologies and Factory Automation (ETFA’10), Bilbao, Španija.

[46] D. Hästabacka, T. Vepsäläinen, S. Kuikka, Model-driven development of industrial

process control applications. Journal of Systems and Software, 84(7) (2011), 1100-1113.

[47] K. Thramboulidis, D. Perdikis, S. Kantas, Model driven development of distributed control applications. The International Journal of Advanced Manufacturing Technology, 33(3) (2007), 233-242.

[48] E Estevez, M. Marcos, I. Sarachaga, D. Orive, A Methodology for Multidisciplinary Modeling of Industrial Control Systems using UML. Zbornik IEEE International Conference on Industrial Informatics (INDIN'07), Dunaj, Avstrija.

[49] K. Thranboulidis, Using UML in control and automation: a model driven approach. Zbornik IEEE International Conference on Industrial Informatics (INDIN'04), Berlin, Nemčija.

[50] U. Katzke, B. Vogel-Heuser, UML-PA as an engineering model for distributed process automation. Zbornik 16th IFAC World Congress, Praga, Češka (2005).

[51] C. Tranoris, K. Thramboulidis, A tool supported engineering process for developing control applications. Computers in Industry, 57(5) (2006), 462-472.

[52] S. Panjaitan, G. Frey, Combination of UML Modeling and the IEC 61499 Function Block Concept fort he Development of Distributed Automation Systems. Zbornik IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Praga, Češka.

[53] E. Estevez, M. Marcos, D. Orive, Automatic generation of PLC automation projects from component-based models. The International Journal of Advanced Manufacturing Technology, 35(5) (2007), 527-540.

[54] T. Strasser, M. Rooker, G. Ebenhofer, I. Hegny, M. Wenger, C. Sunder, A. Martel, A. Valentini, Multi-domain model-driven design of industrial automation and control systems. Zbornik IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), Hamburg, Nemčija.

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

[56] G. Kandare, G. Godena, S. Strmčnik, A new approach to PLC software design. ISA Transactions, 42(2) (2003), 279-288.

[57] G. Kandare, S. Strmčnik, G. Godena, Domain specific model-based development of software for programmable logic controllers. Computers in Industry, 61(5) (2010), 419-431.

[58] T. Lukman, G. Godena, J. Gray, M. Heričko, S. Strmčnik, Model-driven engineering of process control software – beyond device-centric abstractions. Control Engineering Practice, 2013, Vol. 21 No. 8, 1078-1096.

[59] D.C. Schmidt, Model-Driven Engineering. IEEE Computer, 39(2) (2006), 25-31.

[60] E. Seidewitz, What models mean. IEEE Software, 20(5) (2003), 26-32.

[61] S. Kelly, J.-P. Tolvanen, Domain-Specific Modeling. John Wiley & Sons (2007).

[62] G. Godena, J. Tancek, I. Steiner, PLCbatch: features and application, in: WBF – 2010 North American Conference, Austin, TX, USA, 2010.

[63] G. Godena, I. Steiner, J. Tancek, M. Svetina, Batch process automation executed on the PLC platform, in: WBF – 2008 North American Conference, Philadelphia, PA, USA, 2008.

[64] G. Godena, I. Steiner, J. Tancek, M. Svetina, Design of a batch process control tool on the programmable logic controller platform. V: Hawkins, William M. (ur.), Brandl, Dennis (ur.), Boyes, Walt (ur.). *ISA-88 implementation experiences*, (The WBF book series, vol. 1). New York: Momentum Press, 2010, str. 157-173.

[65] M. Arh, B. Tavčar, I. Steiner, J. Demšar, J. Tancek (2013). User Manual (Confidential – for internal use only), Document code: INEA-DP-42/2013.

[66] G. Dolenc, I. Steiner (2009). Avtomatizacija avtoklava – osnovno in postopkovno vodenje – Uporabniški priročnik, oznaka dokumenta: 1C305.

[67] J. Tancek, I. Steiner (2014). Modernizacija mešalnice v obratu Kones Konex d.o.o. – Uporabniška navodila, oznaka dokumenta: INEA-DP-14/2014.

[68] G. Godena, G. Dolanc, I. Steiner, G. Dolenc (2012). Rekonstrukcija vodenja procesa sinteze smol v podjetju Color d.d. Medvode, Projekt izvedenih del (Funkcijske specifikacije), Verzija 1.2, IJS DP-10090.

[69] S. Strmčnik (ur.), Celostni pristop k računalniškemu vodenju procesov, Fakulteta za elektrotehniko, Ljubljana, 1998.

[70] GE Fanuc (2009). Proficy™ Batch Execution, Application Guide

[71] Rockwell Automation, Inc. (2015). FactoryTalk™ Batch, User Guide

[72] Schneider Electric Software, LLC (2015). Wonderware InBatch™ User's Guide

[73] Siemens (2006). SIMATIC, Process Control System PCS 7, SIMATIC BATCH, Manual

[74] IEC (1988). IEC 60848: Preparation of function charts for control systems. Technical report. International Electrotechnical Commission.

[75] IEC (1993). IEC 61131-3 International standard, Programmable controllers - Part 3: Programming languages.

[76] ISA-88.00.01 – 2010, Batch Control Part 1: Models and Terminology.

[77] Rockwell Automation, Inc. (2015). FactoryTalk™ Batch, Technical Reference Guide, Volume 1

[78] G. Godena, A new proposal for the behaviour model of batch phases, ISA Transactions 48 (2009) 3–9.

[79] G. Godena, T. Lukman, I. Steiner, F. Bergant, S. Strmčnik, A new object model of batch equipment and procedural control for better recipe reuse. *Computers in industry*, ISSN 0166-3615. [Print ed.], jun. 2015, vol. 70, 46-55.

[80] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14(2) (2009) 131–164.

Kazalo slik

Slika 1: Relacije med modeli ISA 88.00.01. 13
Slika 2: Razvojni proces šaržnega vodenja. 15
Slika 3: Primer SFC diagrama iz razvojnega okolja Mitsubishi GX IEC Developer. 20
Slika 4: SFC recept. 23
Slika 5: Označen SFC recept. 27
Slika 6: Primer recepta iz sistema vodenja procesa sinteze smol. 31
Slika 7: Diagram prehajanja stanj faz po ISA 88.00.01. 38
Slika 8: Strukturiran diagram prehajanja stanj faz po ISA 88.00.01. 40
Slika 9: Poenostavljen diagram prehajanja stanj faz komercialnega orodja. 41
Slika 10: Nov model obnašanja faz. 45
Slika 11: Postopek recepta vodenja. 57
Slika 12: Povezovanje recepta in vodenja opreme na nivoju faz. 58
Slika 13: Povezovanje recepta in vodenja opreme na nivoju operacij. 59
Slika 14: Povezovanje recepta in vodenja opreme na nivoju postopkov enot. 60
Slika 15: Povezovanje recepta in vodenja opreme na nivoju postopkov 60
Slika 16: Redukcija nivojev modela, primer 1. 61
Slika 17: Redukcija nivojev modela, primer 2. 61
Slika 18: Redukcija nivojev modela, primer 3. 62
Slika 19: Redukcija nivojev modela, primer 4. 62
Slika 20: Reduciran model postopkovnega vodenja. 64
Slika 21: Standardni model opreme in postopkovnega vodenja. 65
Slika 22: Nov model opreme in postopkovnega vodenja do nivoja PE. 67
Slika 23: Nov model opreme in postopkovnega vodenja do nivoja linij in postopkov recepta. 69
Slika 24: Linija procesa sinteze smol. 70
Slika 25: Vsi razredi faz sistema vodenja procesa sinteze smol. 72
Slika 26: Okno za upravljanje enot v orodju PLCbatch. 73

Slika 27: Razred enot VsiReaktorji, njegove enote in njegovi razredi faz. 75
Slika 28: Okno za upravljanje razredov linij v orodju PLCbatch. 76
Slika 29: Postopek enote za doziranje AFK. 77
Slika 30: Uporaba postopka enote za doziranje AFK v postopku recepta. 78

Kazalo tabel

Tabela 1: Tabelarni recept. 24
Tabela 2: Označena tabela recepta. 26
Tabela 3: Matrika prehajanja stanj faz po ISA 88.00.01. 37
Tabela 4: Zmanjšanje števila postopkov enot. 80

Priloga A: Seznam lastnih publikacij

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

[56] G. Kandare, G. Godena, S. Strmčnik, A new approach to PLC software design. ISA Transactions, 42(2) (2003), 279-288.

[57] G. Kandare, S. Strmčnik, G. Godena, Domain specific model-based development of software for programmable logic controllers. Computers in Industry, 61(5) (2010), 419-431.

[58] T. Lukman, G. Godena, J. Gray, M. Heričko, S. Strmčnik, Model-driven engineering of process control software – beyond device-centric abstractions. Control Engineering Practice, 2013, Vol. 21 No. 8, 1078-1096.

[62] G. Godena, J. Tancek, I. Steiner, PLCbatch: features and application, in: WBF – 2010 North American Conference, Austin, TX, USA, 2010.

[63] G. Godena, I. Steiner, J. Tancek, M. Svetina, Batch process automation executed on the PLC platform, in: WBF – 2008 North American Conference, Philadelphia, PA, USA, 2008.

[64] G. Godena, I. Steiner, J. Tancek, M. Svetina, Design of a batch process control tool on the programmable logic controller platform. V: Hawkins, William M. (ur.), Brandl, Dennis (ur.), Boyes, Walt (ur.). *ISA-88 implementation experiences*, (The WBF book series, vol. 1). New York: Momentum Press, 2010, str. 157-173.

[68] G. Godena, G. Dolanc, I. Steiner, G. Dolenc (2012). Rekonstrukcija vodenja procesa sinteze smol v podjetju Color d.d. Medvode, Projekt izvedenih del (Funkcijske specifikacije), Verzija 1.2, IJS DP-10090.

[78] G. Godena, A new proposal for the behaviour model of batch phases, ISA Transactions 48 (2009) 3–9.

[79] G. Godena, T. Lukman, I. Steiner, F. Bergant, S. Strmčnik, A new object model of batch equipment and procedural control for better recipe reuse. *Computers in industry*, ISSN 0166-3615. [Print ed.], jun. 2015, vol. 70, 46-55.