LiteBatch: orodje za vodenje šaržnih procesov na PLK platformi

Giovanni Godena[1], Janez Tancek[2], Igor Steiner[2], David Čuk[2], Lovro Šubelj[3], Jože Grilj[4], Peter Kosin[2]
[1]Institut "Jožef Stefan", Jamova 30, 1000 Ljubljana
[2]Inea d.o.o., Stegne 11, 1000 Ljubljana
[3]Kalingerjeva ulica 6, 1000 Ljubljana
[4]Katarija 6a, 1251 Moravče
giovanni.godena@ijs.si, janez.tancek@inea.si, igor.steiner@inea.si, david.cuk@inea.si, lovro.subelj@gmail.com, joze.grilj@guest.arnes.si, peter.kosin@inea.si

LiteBatch: a Tool for Batch Process Control on the PLC Platform

A new tool was developed for recipe-based control of batch processes according to the S88.01 standard, executed on the PLC platform, without PCs coupled within the real-time control loop. The tool features simultaneous execution of more recipes with automatic allocation of units, support of parallel as well as selective branches and the execution of the state-transition algorithm of individual phases, extended by the notion of superstates to improve the abstraction level. The main benefits of the tool are in the relocation of the execution of recipes from the inherently unreliable PC platform to the considerably more reliable PLC platform and in the simplification of recipe-based batch process control systems, without essentially reducing the expressive power and the abstraction level.

1 Uvod

Standard ANSI/ISA S88.01-1995 [1] (IEC 61512-3) definira referenčne modele, terminologijo in koncepte za opis šaržne proizvodnje in vodenja šaržnih procesov na osnovi koncepta tkim. modularne šaržne avtomatizacije (Modular Batch Automation). S88.01 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 skozi koncept receptov omogoča tudi nestrokovnjakom za programsko opremo spreminjanje obnašanja sistemov glede na proizvodne zahteve različnih izdelkov.

Standard S88.01 je omogočil razvoj večjega števila računalniških orodij, ki omogočajo tako razvoj sistemov šaržnega vodenja, kot tudi samo izvajanje tega vodenja, večinoma na platformi osebnih računalnikov [2].

V zvezi z uporabo standarda S88.01 in na njem temelječih Batch orodjih pa obstajajo tudi nekateri problemi in sicer:

1. Previsoka cena teh orodij za večino manjših in srednjih projektov.
2. Nezadovoljivo časovno obnašanje PC okolja (počasnost, časovna nedeterminističnost).
3. Prenizka zanesljivost in varnost PC okolja (kar je rešljivo, toda ob dodatnih stroških).
4. Velika kompleksnost orodij, pri čemer vse funkcionalnosti večinoma ne potrebujemo.
5. Prenizek nivo abstrakcije in s tem premajhna izrazna moč na področju modela obnašanja faz kot končnih avtomatov.

2 Alternativne rešitve šaržnega vodenja

Zaradi nekaterih od zgoraj naštetih problemov (predvsem previsoke cene Batch 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 krmilnikov, ki pa so večinoma preveč poenostavljene in zaradi tega ob večino izrazne moči, ki jo ponujajo na S88.01 temelječa orodja.

Najpogostejši primer je izvedba recepta kot sekvence korakov. Ker se v vsakem koraku izvaja le po en postopek, so ti postopki neelementarni, 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 glomaznih 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.

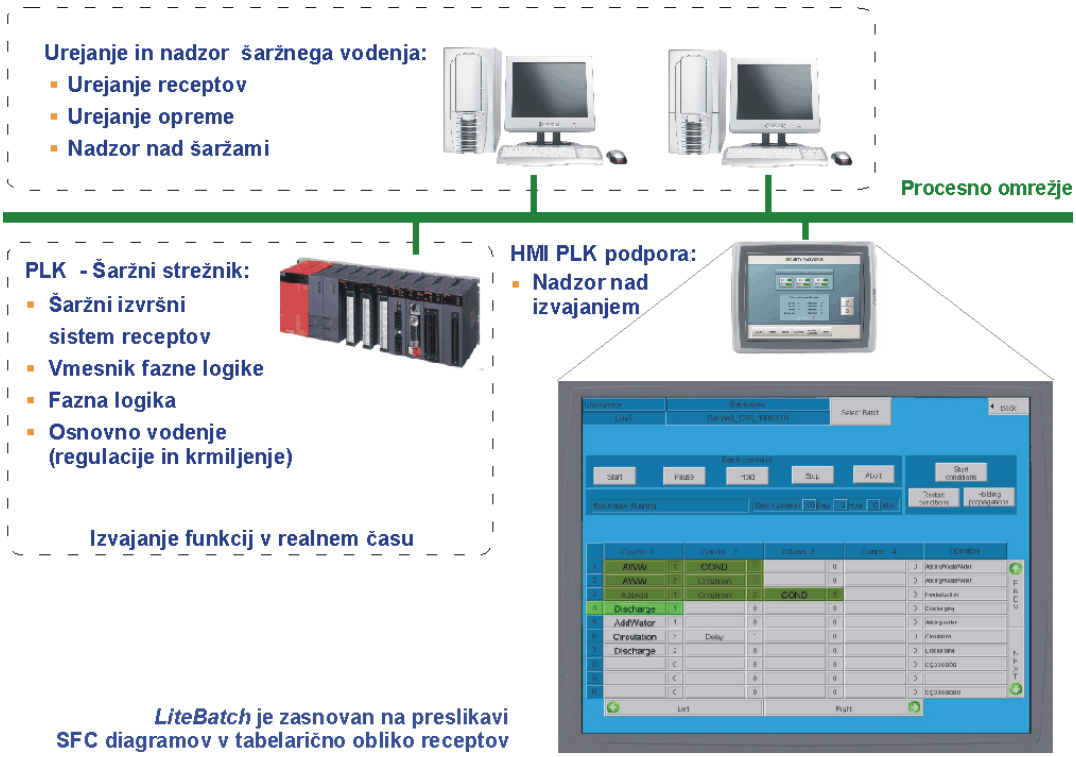
3 Orodje LiteBatch

3.1 Zasnova rešitve

Nova rešitev temelji na dveh konceptih: tabelarični predstavitvi SFC diagrama, ki omogoča (ob minimalni redukciji izrazne moči) izvajanje receptov s sočasnimi fazami na krmilniški platformi, in predstavitvi posameznih elementarnih postopkov kot kompleksnih končnih avtomatov (z gnezdenimi stanji – nadstanji in podstanji), fino granulacijo akcij posameznih stanj na entry, loop in exit akcije ter definicijo kompleksnih prehodov stanj iz enega v drugo mirujoče stanje skozi množico sekvenc (entry, loop ali exit sekvenc posameznih stanj oziroma akcij prehodov), kar predstavlja zelo enostavno implementacijo (navadne sekvence!), dokaj kompleksnega modela obnašanja.

3.2 Arhitektura sistema vodenja LiteBatch

Sistem vodenja, na katerem teče orodje LiteBatch je sestavljen iz krmilnika, osebnega računalnika in operaterskega panela. Delitev funkcij med temi komponentami je naslednja:



Slika 1 Arhitektura sistema LiteBatch

1. Na PC računalniku se izvajajo naslednje funkcije, oziroma moduli:
   - Urejevalnik opreme
   - Urejevalnik/upravljalec receptov
   - Upravljalec šarž
   - Operaterski vmesnik za upravljanje in nadzor vodenja (ta funkcija se lahko izvaja tudi na operaterskem panelu)
2. Na krmilniku se izvajajo naslednje funkcije:
   - Izvajanje receptov (Interpreter receptov)
   - Izvajanje avtomata stanj posameznih faz (proženje sekvenc fazne logike)
   - Posamezne sekvence fazne logike
3. Na panelu se izvaja operaterski vmesnik za upravljanje in nadzor vodenja

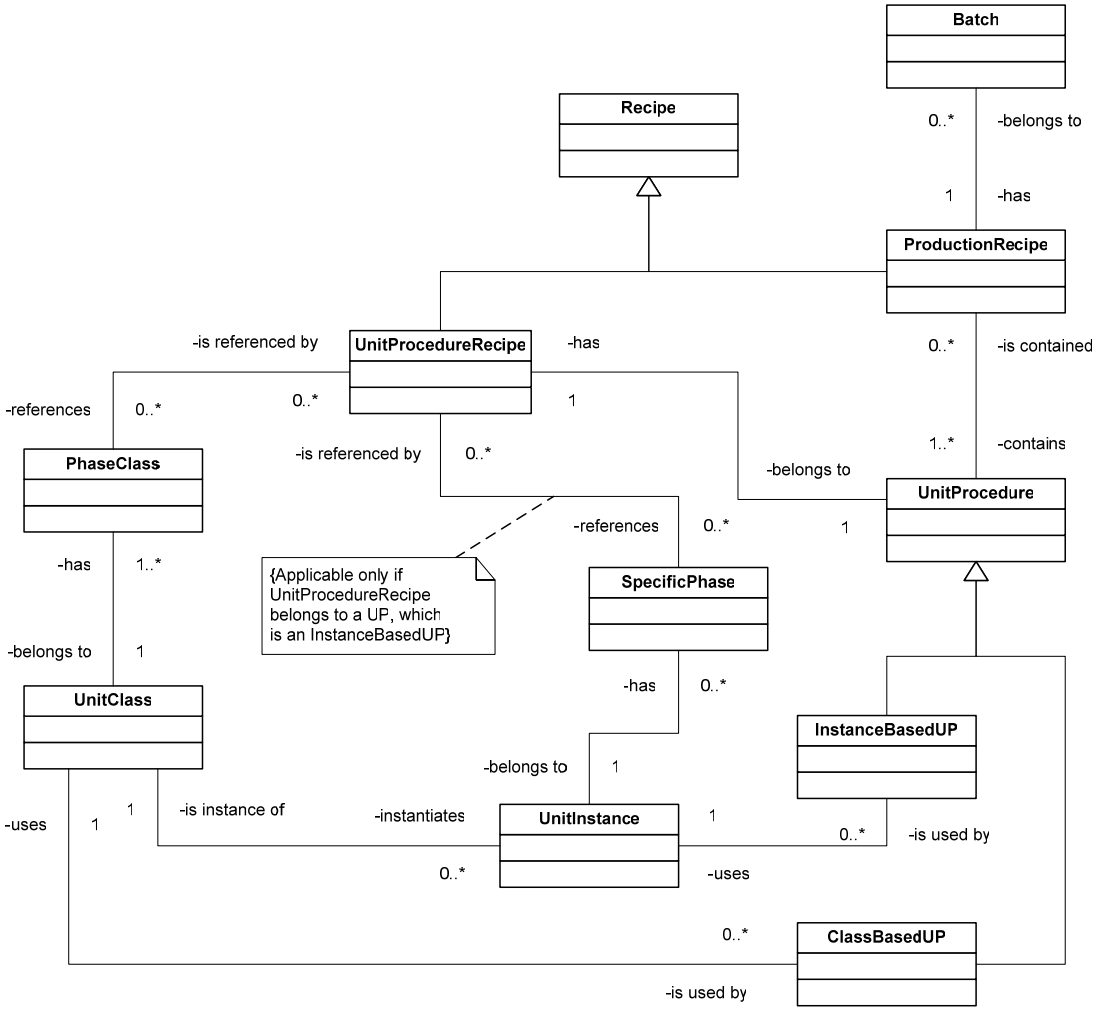
Arhitekturo sistema šaržnega vodenja LiteBatch prikazuje Slika 1.

4 Model entitet recepturnega sistema

Osnovna entiteta izvajanja šaržnega vodenja je šarža. Šarži pripada recept (torej formula), ki je zgrajen iz elementov postopkovnega modela, uporablja pa elemente fizičnega modela. Entitete šarž, fizičnega modela in postopkovnega modela ter njihove medsebojne relacije prikazuje diagram entitet na Sliki 2.

4.1 Model opreme (fizični model)

Model opreme vsebuje dva nivoja in sicer nivo enot in nivo modulov opreme (ki so jim pridružene faze). Enake enote so združeno predstavljene z razredi enot. Vsak razred enote vsebuje razrede faz, to je vse tiste faze, ki so



Slika 2 Model entitet recepturnega sistema

skupne vsem enotam tega razreda. Posamezne instance razreda enote, torej konkretne enote, poleg faz, ki so definirane v sklopu razreda, lahko vsebujejo tudi specifične faze.

4.2 Postopkovni model

Tehnolog ustvarja proizvodne recepte, ki vsebujejo postopke enot. Vsak postopek enote pripada določeni enoti (oziroma uporablja določeno enoto) in ima svoj recept, ki je po (tabelarični) strukturi enak proizvodnemu receptu, vsebuje pa faze te enote. Vsak postopek enote in njegov recept je lahko zgrajen kot *instance based* (ko je zgrajen za določeno konkretno enoto) ali kot *class based* (ko ni zgrajen za konkretno enoto, temveč za eno izmed enot določenega razreda. Če je recept zgrajen kot *class based*, lahko vsebuje le faze, ki so definirane pri razredu enote, če pa je *instance based*, lahko poleg teh faz vsebuje tudi specifične faze konkretne enote (če jih ta enota ima). Sistem v trenutni verziji ne podpira dinamične alokacije enot (torej avtomatske izbire enote med izvajanjem na krmilniku), temveč se v primeru *class based* postopka enote konkretni postopek izbere tik pred nalaganjem šarže in njenega recepta na krmilnik.

5 Možne strukture receptov

Struktura recepta je dvonivojska: zgornji nivo vsebuje tabelo postopkov enot, od katerih vsak vsebuje svoj recept, to je svojo tabelo spodnjega nivoja, v kateri nastopajo faze te enote.

Tabela recepta je v bistvu tabelarična predstavitev SFC diagrama recepta z delno omejeno strukturo. Omejitev strukture SFC diagrama je pri sočasnih razvejitvah, in sicer v tem, da je le v eni izmed trenutno aktivnih sočasnih vej več kot en korak (pri čemer je korak lahko faza ali pa nova sočasna razvejitev). Vzrok te omejitve je v zagotavljanju deterministočnosti pogojev za prehod v nov korak (vrstico) recepta. Opisana omejitev strukture v večini primerov ne vpliva na lastnosti sistema, ko pa vpliva, se to manifestira le kot nekoliko nižja stopnja sočasnosti recepta od največje teoretično možne.

Tabela v trenutni verziji ima omejeno število stolpcev (torej omejeno število hkrati aktivnih faz) in omejeno število vrstic (torej omejeno število vseh korakov recepta). Omejitev je na 7 stolpcev in 10 vrstic (enaka omejitev velja tako za tabelo zgornjega, kot tudi spodnjega nivoja). V naslednji verziji orodja bo ta omejitev odpravljena.

6 Zgradba in izvajanje fazne logike

V tem poglavju bomo okvirno opisali strukturo faz, ki jih lahko vgrajujemo v recepte, model obnašanja faz ter način izvedbe aplikativne programske opreme fazne logike.

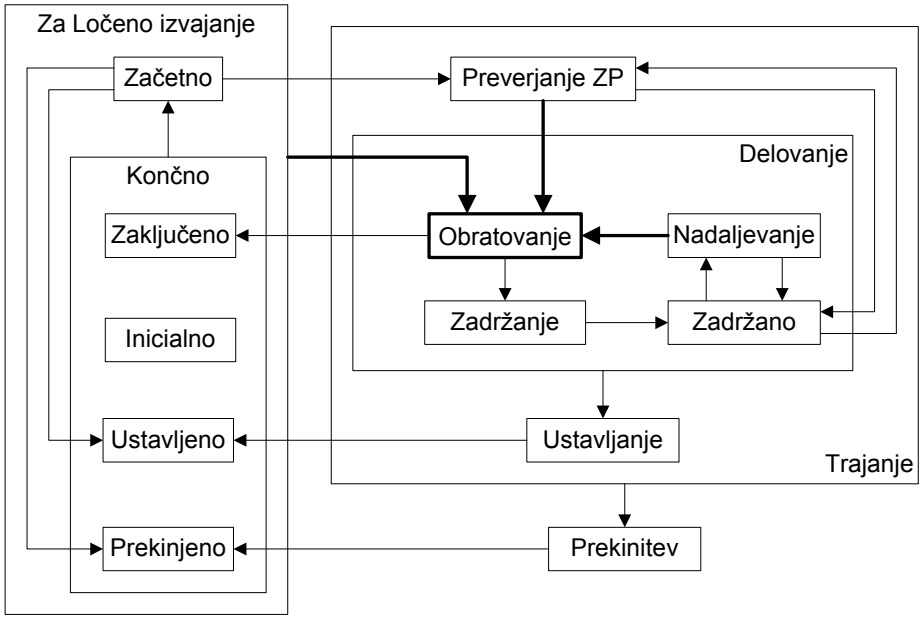
6.1 Stanja faz

Stanja faz izhajajo iz modela obnašanja faz, kot ga definira standard S88.01. V določenih elementih je model razširjen z elementi notacije ProcGraph [3], zaradi izboljšanja atributov enostavnosti, transparentnosti, razumljivosti in zanesljivosti aplikativne programske opreme. Diagram prehajanja stanj faze prikazuje Slika 3.

6.2 Izvajanje faz

Model izvajanja faz je bil zgrajen s ciljem maksimalne poenostavitve aplikacijske kode (fazne logike) ter fleksibilnosti modela obnašanja tako, da si aplikativni programer (oziroma inženirska organizacija) lahko izbere podmnožico definiranega modela ter si tako dejansko zgradi svoj lastni model obnašanja faz, katerega nivo abstrakcije je prilagojen tako problemski domeni, kot tudi miselnemu modelu in sposobnostim izvajalcev sistemov vodenja.

Da bi posamezne sekvence fazne logike bile kar se da enostavne, je nujno zgraditi model, pri katerem bodo vse sekvence res le enostavne sekvence, brez notranjih stanj ali razvejitev in brez potrebe po pomnjenju točke vstopa v sekvenco (npr. ali smo v sekvenco *Obratovanje ENTRY* prišli iz sekvence stanja *Začetno* ali pa iz sekvence stanja *Nadaljevanje*, saj se akcije v teh dveh primerih razlikujejo). Opisani problemi



Slika 3 Diagram prehajanja stanj faze

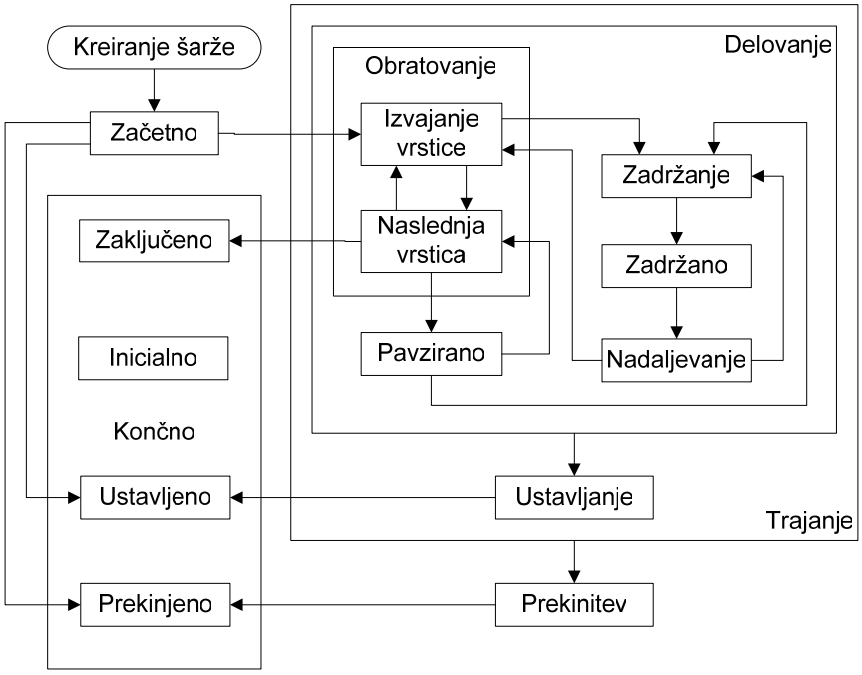
namreč zelo poslabšajo razumljivost aplikacijske programske opreme ter so velik vir napak.

Da bi se izognili zgoraj naštetim težavam, smo definirali model obnašanja faz, ki je razširjen model končnega avtomata. Razširitev je v naslednjih lastnostih:

1. Uvedba gnezdenih stanj (osnovnih stanj in večjega števila nadstanj).
2. Uvedba zelo drobno granulirane strukture akcij in sicer:
   a) akcije prehodov in
   b) akcije stanj, in to za vsako stanje tri različne sekvence:
      - vhodne (ENTRY), ki se izvedejo le enkrat ob prehodu v določeno stanje,
      - v zanki (LOOP), ki se izvajajo ciklično ves čas dokler je faza v določenem stanju,
      - izhodne (EXIT), ki se izvedejo le enkrat ob izhodu iz določenega stanja.

Med elementarnimi stanji (stanja na najnižjem nivoju gnezdenja) razlikujemo stabilna stanja in prehodna stanja. Stabilna stanja so tista, v katerih faza ostane do zunanjega proženja prehoda, kot je ukaz operaterja ali avtozadržanje (primeri teh stanj so *Obratovanje*, *Zadržano* ali *Ustavljeno*), prehodna stanja pa so tista stanja, ki so na poti prehoda med enim in drugim stabilnim stanjem in se zaključijo potem ko se njihove sekvence izvedejo (primeri teh stanj so *Zadržanje*, *Ustavljanje* ali *Nadaljevanje*).

Niz stanj med enim in drugim stabilnim stanjem smo poimenovali *kompleksni prehod*. Izvajanje vodenja poteka tako, da interpreter v vsakem stabilnem stanju preverja potrebo po sprožitvi enega izmed za to stabilno stanje možnih kompleksnih prehodov. Ko so pogoji (oziroma vzroki) nastopa kompleksnega prehoda izpolnjeni, interpreter izvede kompleksni prehod tako da zaporedno (eno za drugo) sproži vse sekvence tega kompleksnega prehoda. Sinhronizacija med interpreterjem in sekvencami poteka tako, da interpreter sekvenci dvigne prožilno zastavico, sekvenca pa ob zaključku izvajanja zastavico zopet spusti in s tem interpreterju sporoči konec izvajanja sekvence. Opisani protokol izvajanja avtomata stanj faz omogoča, da je aplikacijsko programiranje zreducirano le na enostavno polnjenje izbrane podmnožice sekvenc.



Slika 4 Diagram prehajanja stanj recepta

Teoretično število sekvenc za vsako fazo je 65, vendar pa se izkaže, da je veliko sekvenc nepotrebnih, tako da jih dejansko imamo le 24. V primeru, da je tudi teh 24 sekvenc določenemu programerju (ali inženirski organizaciji) preveč, ker si je zgradil enostavnejši model obnašanja faz (z manjšim številom sekvenc), mora v sekvencah, ki jih ne želi uporabljati, zgolj spustiti zastavico. Teoretično lahko tudi programira aplikacije z uporabo le ene sekvence, čeprav bi to v večini realnih (netrivialnih) primerov bilo bolj podobno "štrikanju" kot programiranju).

7 Stanja recepta

Tudi recept se, podobno kot faze, izvaja kot končni avtomat v skladu s standardom S88.01. Razlika je le v tem, da je pri receptu stanje *Obratovanje* sestavljeno iz dveh podstanj – *Izvajanje vrstice* in *Naslednja vrstica*. V stanju *Izvajanje vrstice* interpreter najprej spravi v stanje *Začetno* (z ukazom *Reset*) vse faze tekoče vrstice, zatem jih požene (z ukazom *Start*) in potem čaka da so izpolnjeni pogoji za prehod v novo vrstico tabele recepta. Ko se to zgodi, pride do prehoda interpreterja v stanje *Naslednja vrstica*, v katerem se "počisti" tekoča vrstica (ustavijo vse odvisne faze, ki se kot ista instanca ne ponavljajo v naslednji vrstici).

Interpreter med posameznimi svojimi stanji vpliva na faze tekoče vrstice tako da jim posreduje ukaze. Tako npr. v stanju recepta *Izvajanje vrstice* interpreter fazam posreduje ukaza *Reset* in *Start* (kot je že bilo omenjeno), med stanjem recepta *Naslednja vrstica* ukaz *Stop*, med stanjem recepta *Zadržanje* ukaz *Zadrži*, med stanjem recepta *Ustavljanje* ukaz *Stop* in med stanjem recepta *Prekinitev* ukaz *Prekini*.

Diagram prehajanja stanj recepta prikazuje Slika 4.

8 Literatura

[1] *ANSI/ISA-S88.01-1995, Batch Control, Part 1: Models and Terminology, 1995*.

[2] *G. Kandare, G. Godena, Vodenje šaržnih procesov po standardu ANSI/ISA S88.01*, Avtomatika (1999). [Tiskana izd.], 2000, let. 2, št. 8, str. 30-33.

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