UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Puh

NADGRADNJA PROGRAMSKEGA ORODJA ZA VODENJE PROCESOV

DIPLOMSKO DELO VISOKOŠOLSKEGA STROKOVNEGA ŠTUDIJA

MENTOR: dr. Igor Rožanc

LJUBLJANA, 2008

Kazalo

Povzetek

Diplomska naloga obravnava nadgradnjo orodja za vodenje procesov, ki rešujejo probleme, ki so se pokazali med razvojem in pri uporabi. Pred začetkom razvoja se je bilo potrebno odločiti, katera razvojna orodja in tehnologije se bo uporabilo. Uporabljena orodja in tehnologije so predstavljena v prvem delu naloge.

V nadaljevanju sledi opis orodja za vodenje procesov. Najprej je predstavljen proces, ki se ga z orodjem vodi, nato sledi kratek opis in namen razvoja orodja. Za lažjo predstavo je opisan postopek sestavljanja recepta, nato pa sledi opis funkcij in vsebine aplikacije. Naslednji del opisuje razvoj aplikacije, kjer predstavimo pristop oziroma metodologijo razvoja in načrt aplikacije.

Podrobnejša predstavitev problemov in rešitev sledi v naslednjem delu. Predstavljeni problemi so naslednji: problem urejanja podatkov, izogibanje podvajanju podatkov in odvečnemu delu ter vpeljava linij, ki zagotavljajo pravilnost recepta. Pri vseh naštetih problemih smo opisali problem, podali primer in predstavili rešitev. Na koncu so opisane rešitve še predstavljene orodju za vodenje procesov.

Z rešitvijo problema urejanja podatkov smo uporabniku omogočili, da so mu vedno vidne posledice urejanja podatkov. Z vpeljavo linij je vedno zagotovljena pravilnost receptov. Opisane rešitve pa tudi zmanjšujejo količino podvojenih podatkov in uporabniku omogočajo hitrejše in bolj učinkovito delo z orodjem. Opisane rešitve je seveda možno tudi izboljšati in nadgraditi.

1 Uvod

Razvoj aplikacij je zahtevna naloga, saj je težko predvideti, če bodo vgrajene rešitve ustrezale naročnikovim željam. Prav zaradi tega je tesno sodelovanje z naročniki v času razvoja aplikacije zelo dobrodošlo. Naročniki si navadno želijo da bo aplikacija enostavna za uporabo in čim bolj učinkovita. Velikokrat se zgodi, da naročniki, izrazijo potrebo po funkcionalnosti, na katero sami drugače niti pomislili ne bi.

Diplomska naloga predstavlja nadgradnjo obstoječega orodja z rešitvijo problemov, ki so se pokazali v sodelovanju z naročniki, kot tudi tiste, do katerih smo prišli sami. Ti problemi so posledica uporabnikovih potreb in želja. Uporabniki si želijo, da bi bilo delo z orodjem čim bolj enostavno, hitro in brez nepotrebnega dodatnega dela. Problemi in rešitve, ki so predstavljeni v diplomski nalogi, so povezani z urejevanjem podatkov, izogibanju dvojnemu delu oziroma podvajanju informacij in na koncu je predstavljena še vpeljava liniji, ki zagotavljajo pravilnost recepta.

V prvem delu diplomske naloge so predstavljena uporabljena orodja in tehnologije. V drugem delu sledi predstavitev in opis orodja za vodenje šaržnih procesov s krmilniki. Predstavljen je namen aplikacije in prikazan primer uporabe, opisan pa je tudi šaržni proces.

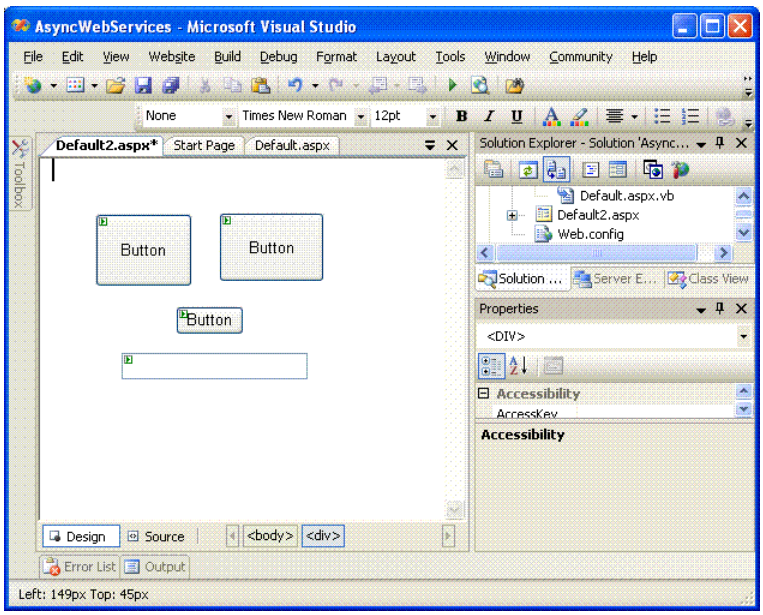
V nadaljevanju sledi opis razvoja orodja, najprej je predstavljena izbrana metodologija nato pa opisan razredni diagram aplikacije. Četrti del podrobneje predstavlja izbrane probleme in rešitev. Primeri teh rešitev so prikazani na orodju za vodenje šaržnih procesov s krmilniki v zadnjem delu

2 Opis uporabljenih orodij in tehnologij

Aplikacija je napisana v razvojnem okolju Microsoft Visual Studio, in sicer v programskem jeziku Visual basic. Pri uporabniškem vmesniku so uporabljene komponente Developer Express-a, ki nam omogočajo enostavno izgradnjo vmesnika. Za shranjevanje podatkov se uporablja označevalni jezik XML. Komunikacija in prenos podatkov z industrijsko opremo (krmilniki) poteka preko OPC strežnika, ki je danes standard pri komunikaciji z industrijsko opremo.

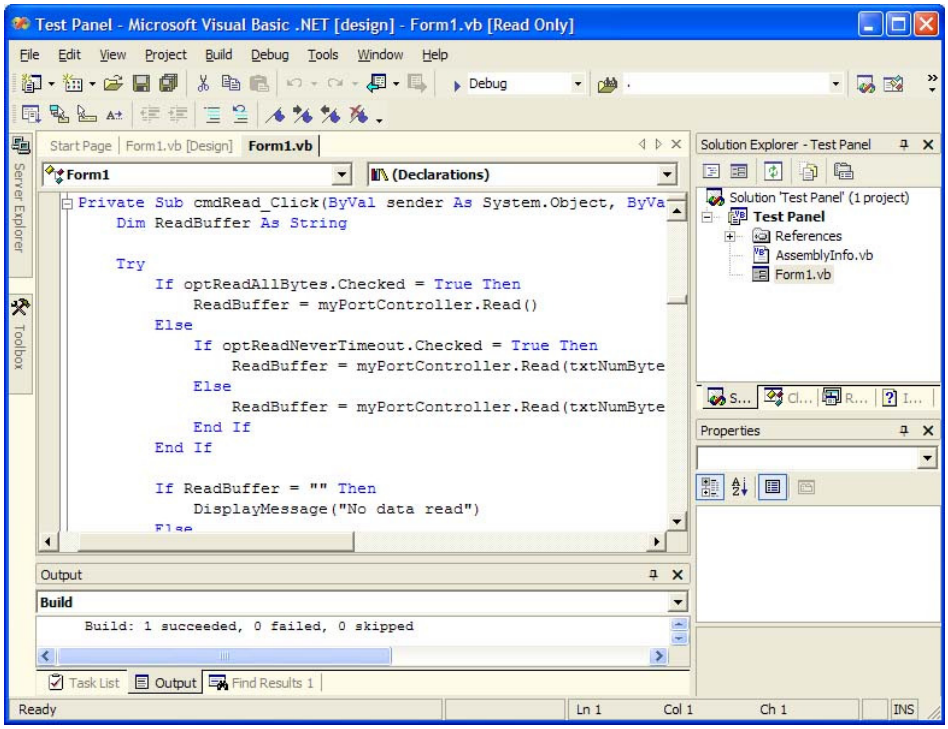
2.1. Microsoft Visual Studio in Visual basic

Microsoft Visual Studio [1] je glavno Microsoftovo integrirano razvojno okolje (ang. Integrated Development Environment - IDE). Z njim lahko razvijamo aplikacije z grafičnimi ali tekstovnim vmesnikom, spletne strani in spletne aplikacije na vseh platformah, ki jih podpira Microsoft. Vsebuje urejevalnik kode, orodje za oblikovanje grafičnih vmesnikov (Slika 1) in še veliko drugih orodij. Urejevalnik kode in razhroščevalnik (ang. debuger) podpirata več programskih jezikov. Vgrajeni jeziki v Visual Studiu so C/C++, VB.NET, in C#, podpira pa tudi druge jezike. To razvojno okolje je bilo izbrano zaradi tega, ker omogoča hiter razvoj aplikacij in je zelo pogosto uporabljeno.



Slika 1 Oblikovanje vmesnika v Visual Studiu

Pri razvoju aplikacije je uporabljena nadgradnja Microsoftova programskega jezika Visual Basic (v nadaljevanju VB) v okolju .NET, ki je vgrajena v razvojno okolje Visual Studio (Slika 2). VB je preprost objektno usmerjen programski jezik , ki omogoča hiter razvoj aplikacij. Primeren je tako za razvijalce, ki so že seznanjeni z VB, kot tudi za tiste, ki se z njim šele spoznavajo.



Slika 2 Primer kode VB

2.2. Označevalni Jezik XML

XML (E**X**tensible **M**arkup **L**anguage) [2] je označevalni jezik. Z razliko od HTML-ja je bil ustvarjen za prenos in hrambo podatkov in ne za prikaz podatkov. Čeprav je XML jezik pravzaprav ne izvaja nobenih aktivnosti ampak samo strukturira in hrani podatke. Etikete (ang. TAG) v XML niso v naprej definirane ampak jih uporabnik definira sam.

Primer:

```
<kontakt>
        <ime>Janez</ime>
        <priimek>Novak</priimek>
        <telefon>1234567</telefon>
        <email>janez.novak@email.si</email>
</kontakt>
```

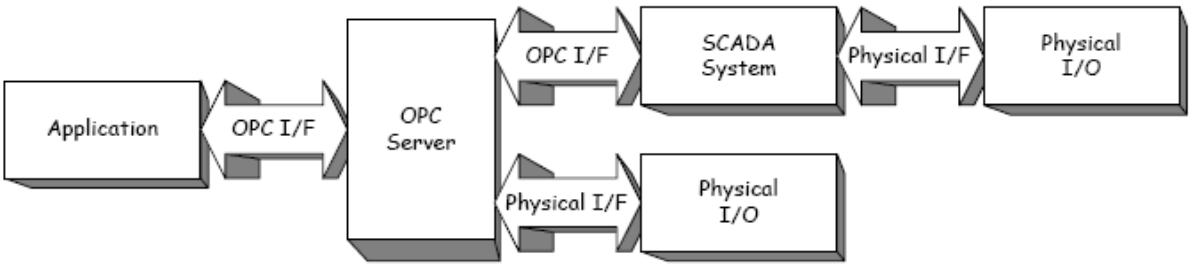
Prednost XML-a je to da poenostavi prenašanje podatkov, ker so podatki shranjeni v navadnih tekstovnih datotekah. To omogoča programsko in strojno neodvisno hranjenje podatkov, ki si jih različne aplikacije, sistemi in platforme enostavno izmenjujejo.
XML dokumenti imajo obliko drevesne strukture. Vsebovati morajo korenski (ang. root) element, ki je oče (ang. parent) vsem ostalim elementom. Vsak element ima lahko tudi podelemente (ang. child element), kot v spodnjem primeru.

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Pri razvoju PLC Batch-a se XML uporablja za shranjevanje vseh podatkov. Podatke se shranjuje s pomočjo pretvarjanja objektov v tok zlogov (ang. serialization) v okolju .net, ki omogoča pretvorbo XML dokumentov v objekte in obratno. Spreminjanje XML v običajne objekte omogoča da se XML dokumente enostavno obdeluje z običajnimi programskimi jeziki.

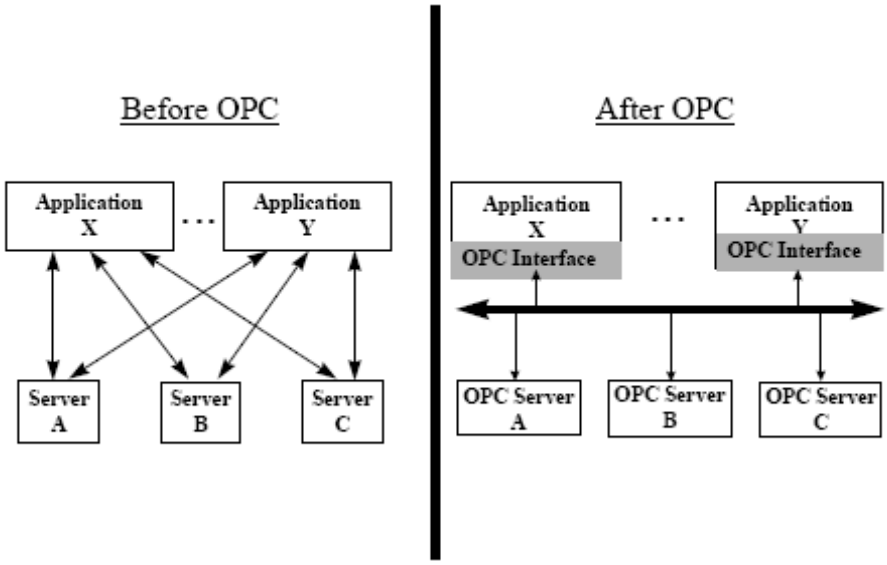
2.3. OPC strežnik

OLE for Process Control [3, 4] (v nadaljevanju OPC) je industrijski standard, ki omogoča medsebojno povezovanje aplikacij in procesne opreme različnih proizvajalcev (Slika 3). Prvi standard je bil rezultat sodelovanja med vodilnimi podjetji na področju avtomatizacije in Microsoftom. Specifikacija, ki je temeljila na Microsoftovih OLE COM (ang. Component Object Model) in DCOM (ang. Distributed Component Object Model) tehnologijah, je definirala standardno množico objektov, vmesnikov in metod za uporabo v krmiljenju procesov in razvoju aplikacij za avtomatizacijo z namenom večje uporabnosti. Tehnologiji COM/DCOM sta nudili ogrodje za razvoj programskih produktov.



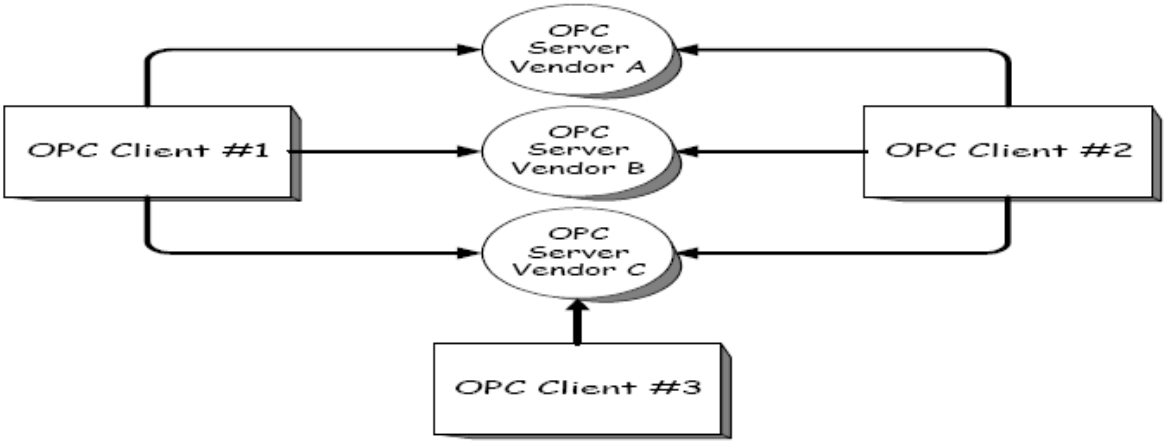
Slika 3 Relacije med aplikacijo OPC strežnikom in opremo

Potrebo po izvirni specifikaciji za dostop do podatkov lahko predstavimo z gonilniki za tiskalnike v okoljih DOS in kasneje Windows. Razvijalci so morali za vsako svojo aplikacijo napisati tudi gonilnik za vsak tiskalnik. Napisati so morali ločene gonilnike za vsak tiskalnik, ki so ga hoteli podpirati. Podobno se je dogajalo tudi v svetu industrijske avtomatizacije, kjer je vsak proizvajalec za vmesnik z opremo moral napisati svoj gonilnik za vsako industrijsko napravo različnih proizvajalcev (Slika 4).



Slika 4 Stanje pred in po OPC

Microsoft je problem s tiskalniki rešil tako, da je podporo za tiskalnike vgradil v operacijski sistem. Tako je en gonilnik podpiral vse aplikacije. Gonilnike so napisali proizvajalci tiskalnikov ne pa razvijalci aplikacij. Microsoft je priskrbel tudi rešitev za gonilnike industrijskih naprav. S tem, ko je dodal specifikacije k OLE tehnologiji, je omogočil standardizacijo. Zdaj so proizvajalci opreme lahko napisali OPC strežnik in programska oprema je postala OPC odjemalec (Slika 5).



Slika 5 Relacije med OPC strežniki in odjemalci

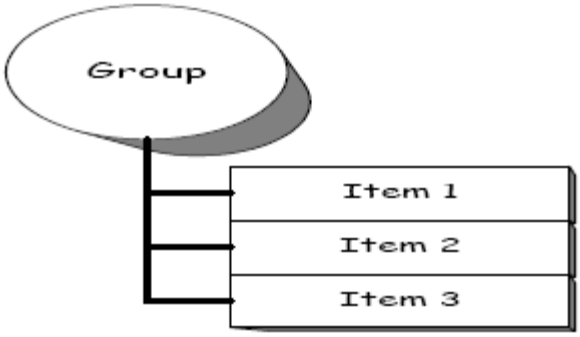
Ta rešitev je omogočila razvijalcem programske opreme, da se posvetijo bistvenim funkcijam aplikacije in se manj časa ukvarjajo s povezljivostjo, uporabnikom pa je rešitev prinesla večjo fleksibilnost. Programsko opremo so lahko izbirali na osnovi funkcionalnosti, namesto da bi pazili na to, če vsebuje gonilnik za njihovo napravo.

Originalne specifikacije so standardizirale samo pridobitev procesnih podatkov, kmalu pa se je izkazalo, da bi tudi komunikaciji z drugimi podatki koristila standardizacija.Tako so nastali standardi za :

1. dostop do podatkov (ang. data access),
2. alarme in dogodke (ang. Alarms & Events),
3. beleženje podatkov (ang.Historical Data) in
4. prenos podatkov pri šaržnih procesih (ang. Batch data).

Na kratko si bomo ogledali dostop do podatkov. OPC strežnik sestavlja več objektov: strežnik (ang. server), skupina (ang. group) in predmet (ang. item). OPC strežnik hrani informacije o strežniku in strežnikih, ki vsebujejo objekte OPC skupine. OPC skupina vsebuje informacije o sebi in zagotavlja mehanizme za vsebovanje in logično organiziranje OPC predmetov.

Obstajata dve vrsti skupin javne in lokalne oziroma privatne skupine. Javne so namenjene izmenjavi med več odjemalci, medtem ko so lokalne vidne samo enemu odjemalcu. Znotraj ene skupine lahko odjemalec definira več OPC predmetov (Slika 6).



Slika 6 Relacije med OPC skupinami/predmeti

OPC predmeti predstavljajo povezave do izvorov podatkov znotraj strežnika. OPC predmet ni dosegljiv kot objekt OPC odjemalcu. Vsi dostopi do OPC predmeta se opravljajo preko OPC skupine, ki ta predmet vsebuje oziroma kjer je definiran. Potrebno je poudariti, da OPC predmeti niso vir podatkov, ampak so samo povezave do njih. V bistvu preprosto podajajo naslov podatka in ne dejanski fizični izvor podatkov, ki ga naslov navaja.

2.4. Developer Express

Developer Express [5] je zbirka gradnikov komponent in orodij, ki nam olajšajo izgradnjo uporabniškega vmesnika in mu dajejo lepši izgled. Skoraj vsi elementi grafičnega vmesnika aplikacije so iz te zbirke. Predvsem so uporabljeni gradniki za prikaz tabel in seznamov, uporablja pa se tudi orodje za grajenje poročil s pomočjo katerega se opravlja izvoz podatkov.

3 Programsko orodje za vodenje šaržnih procesov s krmilniki

V nadaljevanju sledi predstavitev in opis programskega orodja za vodenje šaržnih procesov s krmilniki (v nadaljevanju PLCbatch). Pred tem bo tudi na kratko opisan šaržni proces.

3.1. Šaržni proces

Preden nadaljujemo z opisom aplikacije si na kratko oglejmo kaj je šaržni proces. Proces je zaporedje kemičnih, fizičnih ali bioloških aktivnosti, ki nastopajo pri pretvorbi (preoblikovanju), transportu, in/ali skladiščenju materiala, energije in/ali informacije. Industrijske procese razvrščamo na več načinov. Procese delimo glede na: tip elementa, ki ga transformirajo, tip spremembe stanja, način predelave snovi in procesnih spremenljivk. Glede na prevladujoči tip spremenljivk delimo procese na:

1. zvezne procese, kjer se snov, energija ali informacija zvezno pretakajo skozi proces, npr. predelava naftnih derivatov
2. sekvenčne procese, pri katerih nastopajo zaporedja različnih med seboj ločljivih stanj, npr. vožnja dvigala
3. diskretne procese, pri katerih nastopajo posamezni objekti, ki jih preoblikujemo, sestavljamo in skladiščimo, pri čemer objekti ohranjajo svojo identiteto, npr. sestavljanje radijskih aparatov iz sestavnih delov
4. 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 obdelujemo z urejenim zaporedjem procesnih aktivnosti preko omejenega časovnega in pri tem uporabimo enega ali več kosov opreme. V šaržnem procesu proizveden produkt se imenuje šarža (ang. batch), podatki o postopku pa so zbrani v receptu. Šaržni proces je torej sestavljen iz zaporedja 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, zaradi česar 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 prehrambene industrije. Šaržni procesi so primerni pri pogostih spremembah specifikacij proizvoda in pri proizvodnji manjših količin proizvodov, ki imajo tipično veliko dodano vrednost.

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. Zaradi te kompleksnosti se je pojavila potreba po izdelavi konsistentnih modelov za vodenje šaržnih procesov in standardizacijo le-teh. Rezultat tega je ameriški standard ANSI/ISA S88.01-1995 [6], ki uvaja referenčne modele, terminologijo in koncepte za opis šaržne proizvodnje in vodenja šaržnih procesov. Standard S88.01 je leta 1997 povzela tudi mednarodna organizacija IEC in ga izdala kot mednarodni standard za področje šaržnega vodenja lEC 61512-3.

3.2. Kratek opis in namen razvoja

Slabost sedanjih orodij je njihova zahtevnost, visoka cena pri nakupu in uvajanju v proizvodnjo [7]. To zmanjšuje njihovo uporabnost na procesih, kjer je ekonomsko neupravičeno uporabiti takšne kompleksne rešitve.

Velika pomanjkljivost je tudi izvajanje šaržnega vodenja na namenskem PC računalniku - šaržnem strežniku. PC računalnik je na splošno poznan kot šibek člen v sistemu vodenja, kajti odpoved PC računalnika, v tem primeru šaržnega strežnika, pomeni tudi odpoved postopkovnega vodenja procesa in zaustavitev proizvodnega postopka.

Podjetje INEA iz Ljubljane je zato skupaj z Institutom Jožef Stefan razvilo lastno rešitev PLCBatch, ki sledi zahtevam mednarodnega standarda S88.01, po drugi strani pa je enostavno za uporabo in uporabno tudi na procesih, kjer draga in zahtevna orodja niso primerna. Vodilo pri razvoju rešitve PLCBatch je bilo poleg enostavnosti tudi to, da se pri sprotnem izvajanju vodenja šaržnega procesa izloči nezanesljiv člen - PC računalnik z njegovo osnovno funkcijo šaržnega strežnika. Njegove funkcije so prenesene na krmilnik PLK, ki je danes najbolj standarden in zanesljiv element avtomatizacije v industriji. S prenosom krmilnih funkcij iz računalnika na industrijski krmilnik se je znatno povečala zanesljivost celotnega sistema

vodenja. PC računalnik je v rešitvi PLCBatch uporabljen le v nekritičnih delih vodenja, to je pri načrtovanju in urejanju receptov, urejanju opreme in pri nadzoru nad izvajanjem vodenja. Vse to so nekritični in posredni nivoji vodenja, ki ob odpovedi opreme ne povzročijo izpad proizvodnega postopka.

Bistvene lastnosti PLCBatch sistema lahko strnemo v naslednjih točkah:

1. Simultano izvajanje množice receptov,
2. Dinamično kreiranje receptov in upravljanje z recepti,
3. Kreiranje šarž na podlagi razpisanih delovnih nalogov in kontrola nad izvajanjem šarž,
4. Avtomatsko izvrševanje receptov na PLK platformi brez računalnika v kontrolni zanki
5. Preverjanje konsistentnosti med receptom in fizičnim modelom procesa,
6. Pregled nad napakami in zagonskimi pogoji,
7. Enostavno razumljiva tabelarična prestavitev receptov,
8. Podpora za paralelno in selektivno razvejitve v receptih,
9. Nadzor šaržnega vodenja je mogoč tudi preko industrijskega operaterskega panela.

3.3. Postopek sestavljanja recepta in zgradba recepta

Entitete postopkovnega vodenja so faze (osnovne entitete) in postopki enot (večje entitete). Entitete postopkovnega vodenja zlagamo v sekvenčno-sočasne strukture, ki jih imenujemo recepti in ki predstavljajo zapis tehnološkega postopka. Šarža je tehnološki postopek za izdelavo določene količine določenega izdelka po določenem tehnološkem postopku, ki je zapisan v ustreznem receptu.

Da bi lahko pisali recepte, najprej moramo določiti možne gradnike receptov, to je nabor faz. Zaradi boljšega obvladovanja kompleksnosti in izogibanja podvajanju informacije pri gradnji modela opreme uporabljamo princip agregacije (faze združujemo okoli enot, enote združujemo v linije) in princip klasifikacije (faze uvrščamo v razrede faz, enote v razrede enot in linije v razrede linij).

Pisanje recepta se začne z določanjem opreme, npr. reaktor. Na tej opremi določimo faze, ki se na njej izvajajo. Faza je najnižji nivo recepta. Obstajata dve vrsti faz: sistemske in tehnološke. Sistemske faze imajo že vnaprej določene parametre, ki se jih ni možno brisati ali dodajati. Med te faze sodijo: pogoj (ang. condition), zakasnitev (ang. delay), vnosno polje

(ang. input field), skok (ang. jump) in sporočilo (ang. message). Tehnološke faze pa določi uporabnik sam glede na opremo. Primer takih faz so: polnjenje, mešanje, segrevanje, praznjenje ipd.

Ko imamo določene faze, ki se izvajajo na enoti, lahko sestavimo postopek enote. Postopek enote je sestavljen iz urejene množice faz znotraj ene enote. Hkrati se lahko izvaja več postopkov enot, vendar na različnih enotah. Primer postopka enote je hlajenje.

V zadnjem koraku sestavimo recept, ki predstavlja končni izdelek. Sestavlja ga urejeno zaporedje enega ali več postopkov enot. Na koncu se iz recepta kreira še šarža, ki jo lahko naložimo na krmilnik.

3.4. Opis funkcij in vsebine aplikacije

Glavna funkcija aplikacije je kreiranje in urejanje receptov, ki jih kasneje naložimo na krmilnik. Urejevanje je sestavljeno iz štirih večjih delov:

1. urejanje opreme,
2. upravljanje receptov,
3. upravljanje šarž in podatkov šarže ter
4. spreminjanje nastavitev

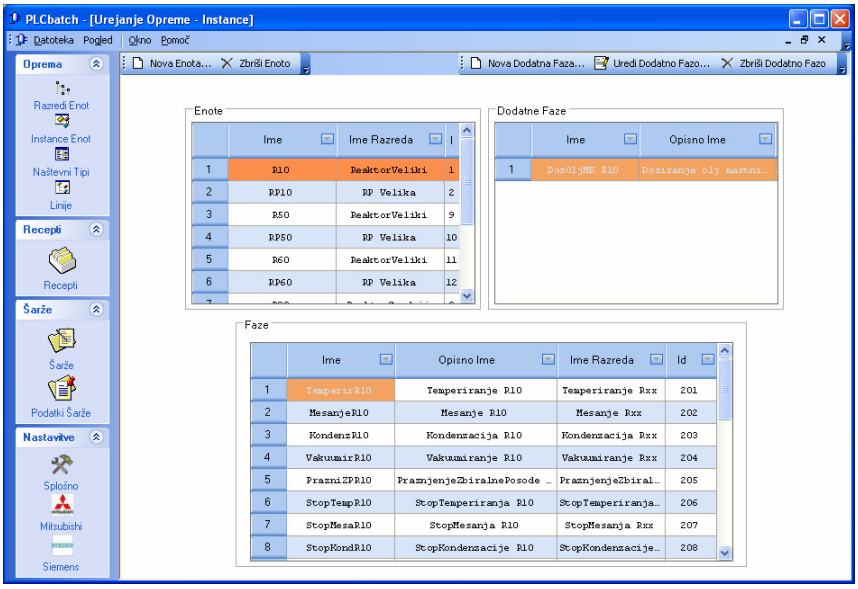
3.4.1. Urejanje opreme

Urejevanje opreme (Slika 7) je namenjeno določanju opreme na kateri se bo izvajal recept. Določamo lahko razrede enot kot tudi pojavitve (ang. instance) enot. Poleg tega imamo možnost dodajanja lastnih naštevnih tipov in dodajanja enot na posamezne linije.

Pri urejanju razredov enot dodajamo razred enote in razrede faz, ki se lahko izvajajo na tem razredu enote. Ko kreiramo razred enote, mu določimo ime in vanj dodajamo razrede faz. Ob dodajanju razreda faze izberemo če bo ta faza sistemska ali tehnološka. Fazi nato določimo ime in atribute in (če je faza tehnološka) dodamo še parametre. Parametrom določimo:

1. ime,
2. tip parametra tipi parametrov so: bool, intiger, real, short time, long time in naštevni tipi, ki jih določamo sami

3. enoto,
4. skalabilnost,
5. minimalno vrednost,
6. maksimalno vrednost in
7. privzeto vrednost.



Slika 7 Urejanje opreme

Recepti se seveda ne morejo izvajati na razredih enot, ki ne predstavljajo konkretne enote, zato moramo določiti vsaj eno pojavitev enote tega razreda, če želimo na njej izvajati recept. Ko dodajamo novo enoto, izberemo njen razred in ji določimo ime. Ker ima enota enake faze kot razred enote, se ti samodejno dodajo in jih ni mogoče urejati. Spremenimo jim lahko samo ime in določimo opisno ime. Enota pa ima lahko tudi kakšno dodatno fazo, ki je razred faze ne vsebuje, zato lahko enoti določimo dodatne faze. Dodatne faze se določi na enak način kot razrede faz.

Pod urejanje opreme sodi tudi določanje naštevnih tipov. Prvih deset je sistemskih, ostale pa lahko uporabnik definira sam. Sistemski so tisti naštevni tipi, ki se uporabljajo v sistemskih fazah. Naštevnemu tipu določimo ime in vrednosti. Vsakemu naštevnemu tipu lahko določimo do 64 vrednosti.

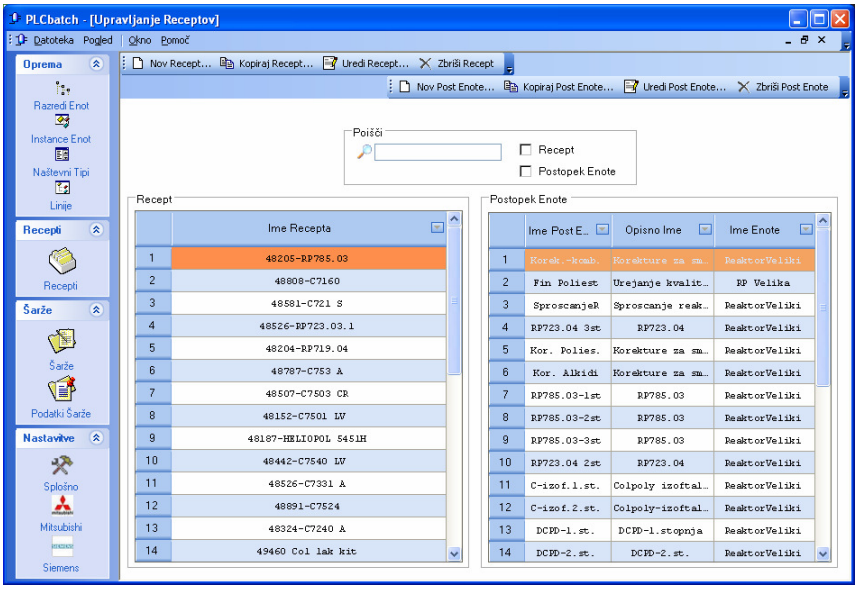
Zadnja stvar, ki sodi med urejanje opreme je še dodajanje enot na linije. Na linijo dodamo lahko cel razred enote ali pa samo posamezne pojavitve.

Opremo je potrebno definirati tudi na krmilniku, saj se podatki o fazah (razen vrednosti parametrov) ne prenašajo na PLK. Urejanje opreme je zaradi tega končnim uporabnikom v proizvodnji (tehnologom in operaterjem) onemogočeno.

3.4.2. Upravljanje receptov

Prvi del upravljanja receptov (Slika 8) predstavlja urejanje postopkov enot. Postopek enote je lahko kreiran na osnovi razreda enote ali pojavitve enote. V primeru, da je kreiran na osnovi razreda se mu instanco določi ob prenosu recepta na krmilnik. Postopku enote določimo ime, opisno ime, dominantnost in sproščanje enote ob koncu. Postopku enote lahko prav tako kot fazam določimo tudi parametre.

Postopek enote vsebuje tabelo velikosti 20 vrstic in 8 stolpcev, v katero dodajamo faze. Faza se lahko razteza čez več vrstic, vendar v eni vrstici ne smeta biti dve isti fazi. V vsaki vrstici mora obstajati vsaj ena dominantna faza. Število faz, ki jih lahko dodamo v postopek enote, je omejeno s številom parametrov faz. Ko presežemo število parametrov vseh faz skupaj, v postopek enote ne moremo več dodajati faz. Ob dodajanju faze najprej določimo način vnosa in vrednosti parametrom nato pa še atribute faze. Tabelo postopka enote in podatke o fazah lahko tudi natisnemo ali izvozimo v želenem formatu.

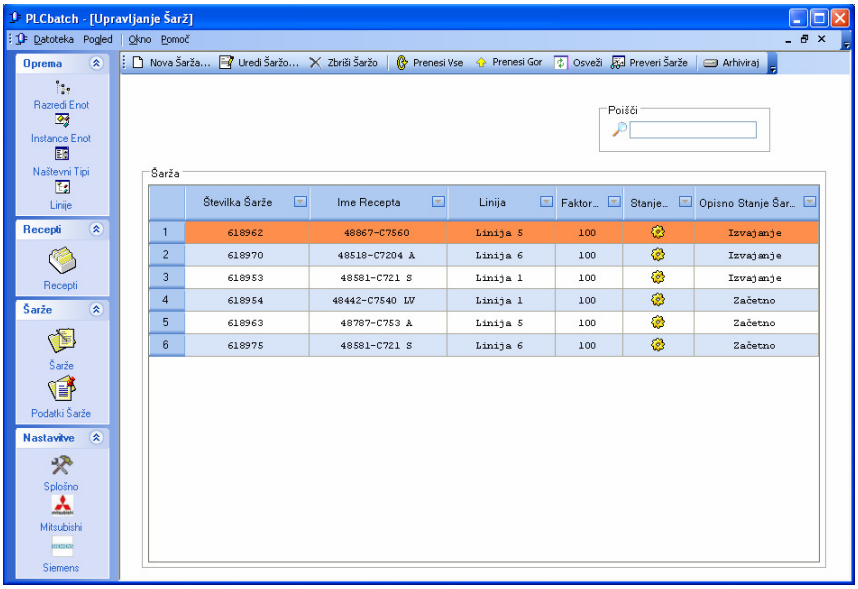


Slika 8 Upravljanje receptov

Recept prav tako vsebuje tabelo recepta, kamor dodajamo postopke enot. Tudi postopek enote lahko razteza čez več vrstic in število postopkov enot, v tabeli je prav tako omejeno s številom parametrov postopkov enot. V vsaki vrstici mora obstajati vsaj en dominanten postopek enote, če pa je v vrsti več postopkov enot na različnih enotah, mora za vsako enoto obstajati vsaj en dominanten postopek enote. Pri urejanju recepta je tudi ves čas vidno, na katerih linijah se lahko izvaja. Tabelo recepta in podatke o postopkih enot lahko tudi natisnemo ali izvozimo v želenem formatu.

3.4.3. Upravljanje šarž in podatkov šarže

Upravljanje šarž (Slika 9) je namenjeno urejanju šarž in prenosu šarž na krmilnik. Ob dodajanju šarže izberemo recept, ki ga želimo izvajati, izberemo faktor skaliranja (vpliva na vse parametre, ki so skalabilni) in šarži določimo ime. Nato lahko izberemo linijo, na kateri se bo recept izvajal, in določimo instance razredom enot, če ti v receptu obstajajo. Te odločitve lahko sprejmemo tudi kasneje, ob nalaganju šarže.

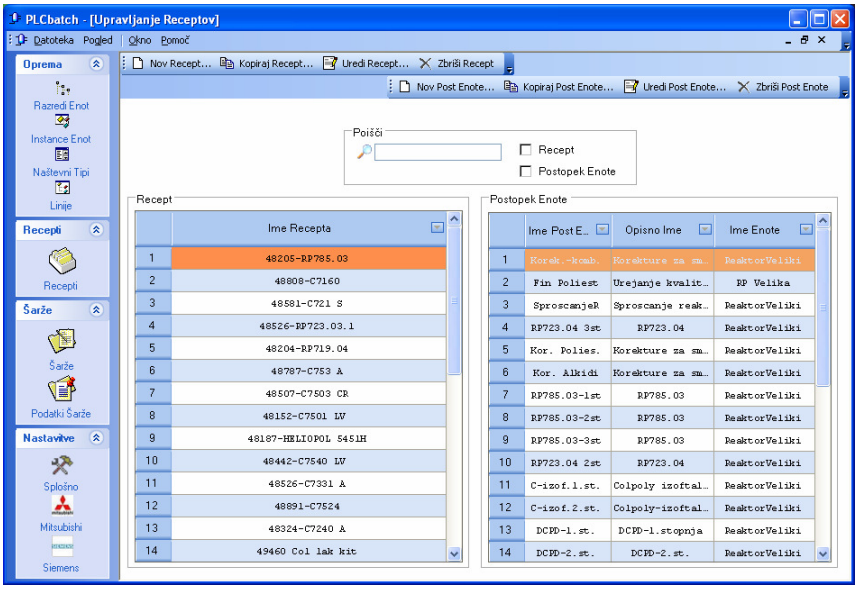


Slika 9 upravljanje šarž

Iz seznama šarž je razvidno, katere šarže so naložene na krmilniku in kakšno je njihovo stanje. Šarže ki še niso naložene se lahko naložijo, zaključene šarže pa so pripravljene za prenos iz krmilnika in jih kasneje lahko arhiviramo. Šaržam lahko tudi osvežimo stanje ali pa preverimo, če se ujemajo s tistimi na krmilniku.

3.4.4. Spreminjanje nastavitev

Spreminjamo lahko splošne nastavitve (Slika 10) kot so tip krmilnika, OPC strežnik, nastavitve opreme in pot podatkov. V splošnih nastavitvah lahko tudi pobrišemo vse podatke s krmilnika ali pa kar vse podatke na PC-ju. Drugi del nastavitev prestavljajo nastavitve začetnih registrov, kamor se prenesejo podatki, za obe vrsti krmilnikov.



Slika 10 Spreminjanje splošnih nastavitev

4 Razvoj orodja PLCbatch

V nadaljevanju bo predstavljen razvoj aplikacije. Najprej bo predstavljena bo tudi izbrana metodologija razvoja programske opreme in razredni diagram aplikacije.

Pri razvoju orodja sem začel sodelovati, ko je orodje doseglo osnovno stopnjo uporabe, vendar se ši ni uporabljalo v nobenem podjetju. Moja naloga je bila nadaljevati razvoj orodja predvsem v smislu nadgradnje.

Orodje trenutno uporablja eno podjetje. S pomočjo orodja vodijo proces sinteze smole. Recepte pišeta oziroma sestavljata dva tehnologa, prenose šarž pa opravljajo operaterji.

4.1. Metodologija

Razvoj orodja poteka na osnovi sproti ugotovljenih potreb v sodelovanju z naročniki, zato stalno prihaja do sprememb in vključevanja novih funkcionalnosti v aplikacijo. Vse to so značilnosti agilnih metodologij. Agilne metode sledijo dvanajstim osnovnim principov[8]:

1. Najvišja prioriteta je zadovoljiti naročnike z hitrim in neprestanim dostavljanjem programske opreme.
2. Spremembe zahtev so dobrodošle tudi proti koncu razvoja. Spremembe se izkoriščajo za naročnikovo konkurenčno prednost.
3. Pogosto dostavljanje delujoče programske opreme, vsakih nekaj tednov ali mesecev oziroma čim bolj pogosto.
4. Naročniki in razvijalci sodelujejo skozi celoten projekt.
5. Projekti so zgrajeni okrog motiviranih posameznikov. Nuditi jim je potrebno okolje in podporo, ki jo potrebujejo, in jim zaupati, da bodo delo uspešno opravili.
6. Najbolj učinkovita in uspešna metoda prenašanja informacij k in znotraj razvojne ekipe je osebni pogovor.
7. Delujoča programska oprema je glavno merilo napredka.
8. Agilne metode spodbujajo nenehen razvoj. Sponzorji, razvijalci in naročniki morajo biti sposobni vzdrževati stalen tempo skozi celoten razvoj.
9. Nenehno strmenje k tehnični odličnosti in dobra zasnova povečuje agilnost

10. Preprostost, maksimiziranje še neopravljenega dela, je bistvena.
11. Najboljše arhitektura, zahteve in načrti se ustvarijo v samostojno organiziranih ekipah
12. V rednih časovnih intervalih mora ekipa razmisliti, kako bi postala bolj učinkovita, in nato ustrezno ukrepati.

Razvoj sicer ni potekal po nobeni formalni metodologiji, vendar pa je potekal v skladu z principi agilnih metod. Ko je aplikacija dosegla določeno stopnjo uporabnosti so jo začeli naročniki uporabljati in razvoj se je nadaljeval v sodelovanju z njimi.

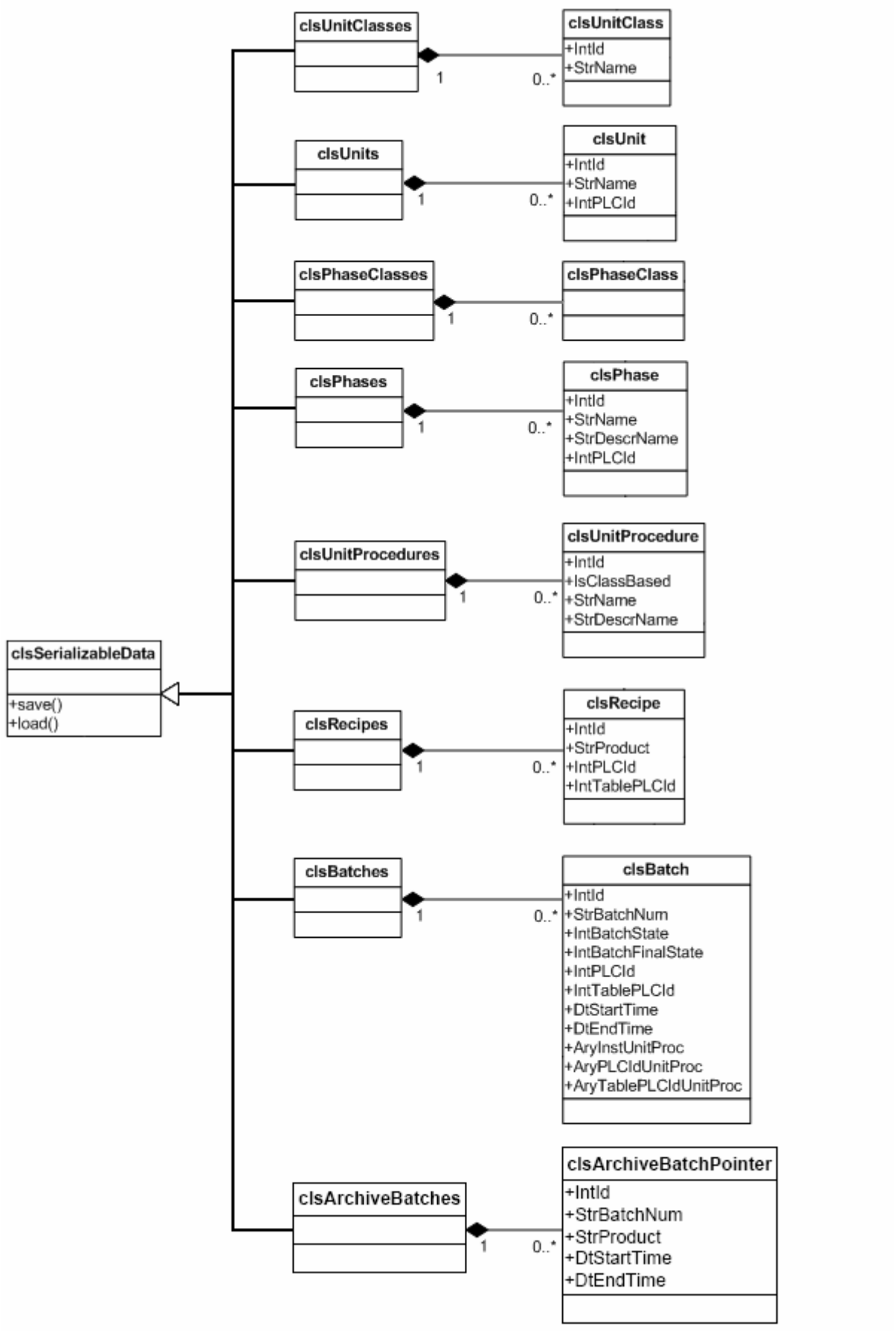
4.2. Načrt aplikacije

Aplikacijo sestavljajo razredi, moduli in okna – forme. Razredi predstavljajo vse elemente, ki sestavljajo recept. Moduli vsebujejo globalne spremenljivke ter funkcije in metode, ki so povezane z nalaganjem in shranjevanjem podatkov, komunikacijo z OPC strežnikom in pomožne metode in funkcije. Okna prestavljajo uporabniške vmesnike.

V nadaljevanju bomo predstavili razredni diagram aplikacije. Na Slika 11 vidimo razrede, ki predstavljajo osnovne entitete recepta (faza, postopek enote, recept…) in nekatere pomožne razrede. Vsi ti razredi so vsebovani v pripadajočem razširjenem razredu razreda `clsSerializableData`, ki vsebuje metodi `load` in `save`. Metodi s pomočjo pretvarjanja stanja objektov v tok zlogov (ang. serialization) shranita oziroma naložita vrednosti spremenljivk v XML datoteko. Torej so v eni XML datoteki shranjene faze, v drugi postopki in tako naprej.

Razredni diagram je razdeljen na

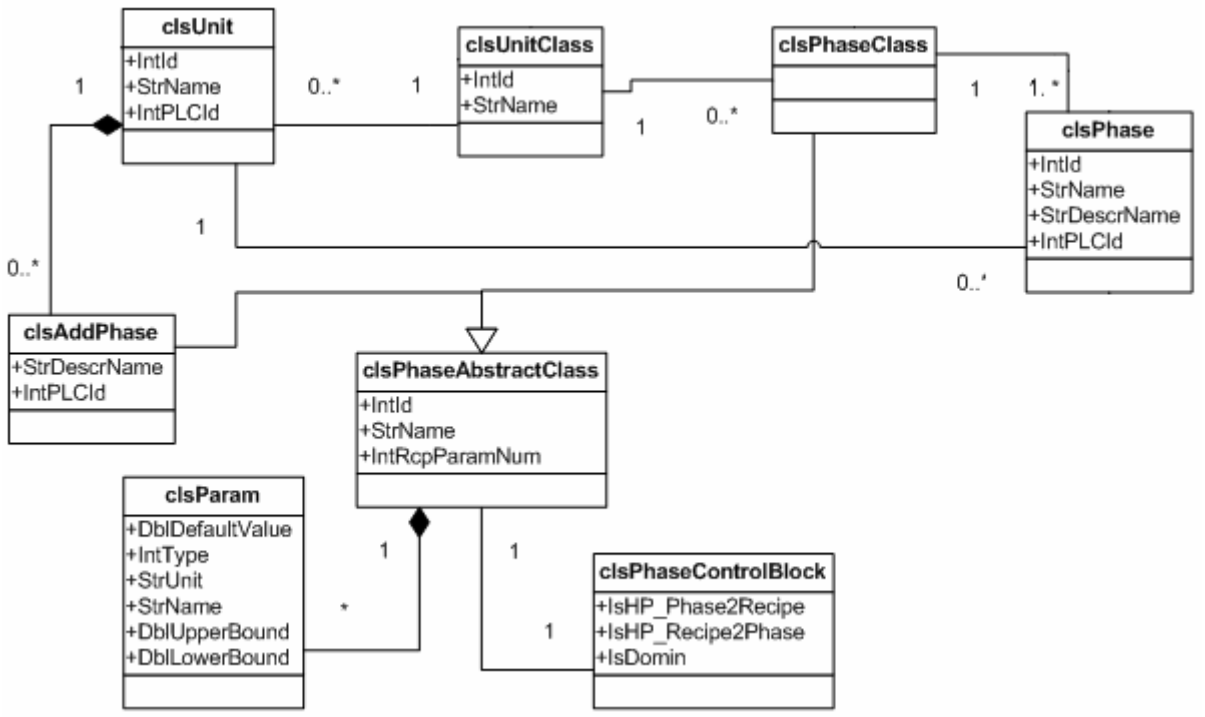
1. opremo,
2. postopke enot,
3. šarže in
4. arhiv.



Slika 11 Diagram osnovnih razredov

4.2.1. Oprema

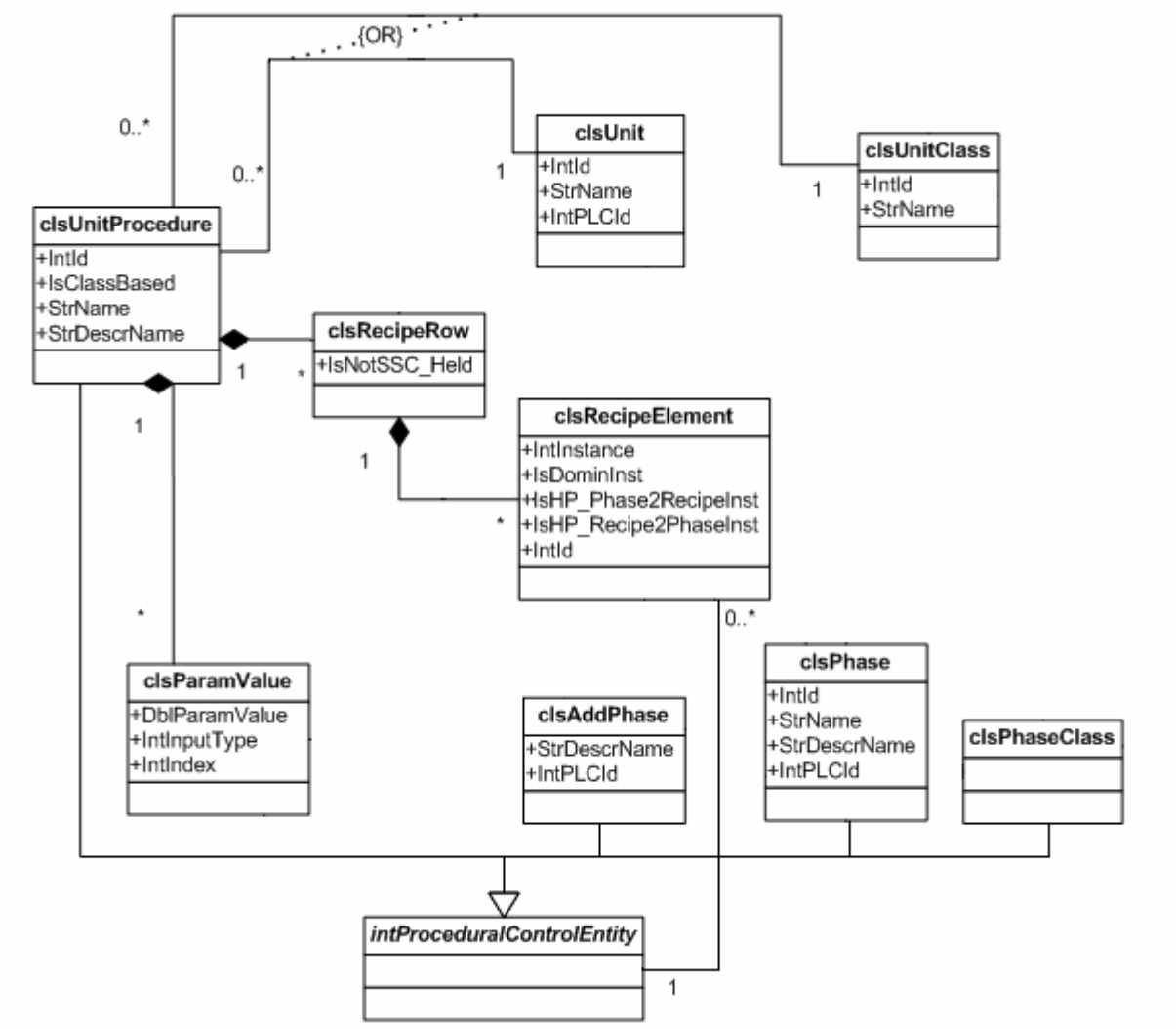
Opremo sestavljajo razredi enot in enote, ki jim pripadajo razredi faz oziroma faze, te pa vsebujejo parametre. Iz Slika 12 so razvidne relacije med razredi. Vse faze so razširitev abstraktnega razreda, ki vsebuje kontrolni blok faze in parametre. Posebnost so dodatne faze, ki niso vsebovane v svojem razredu ampak v enoti.



Slika 12 Diagram opreme

4.2.2. Postopki enot

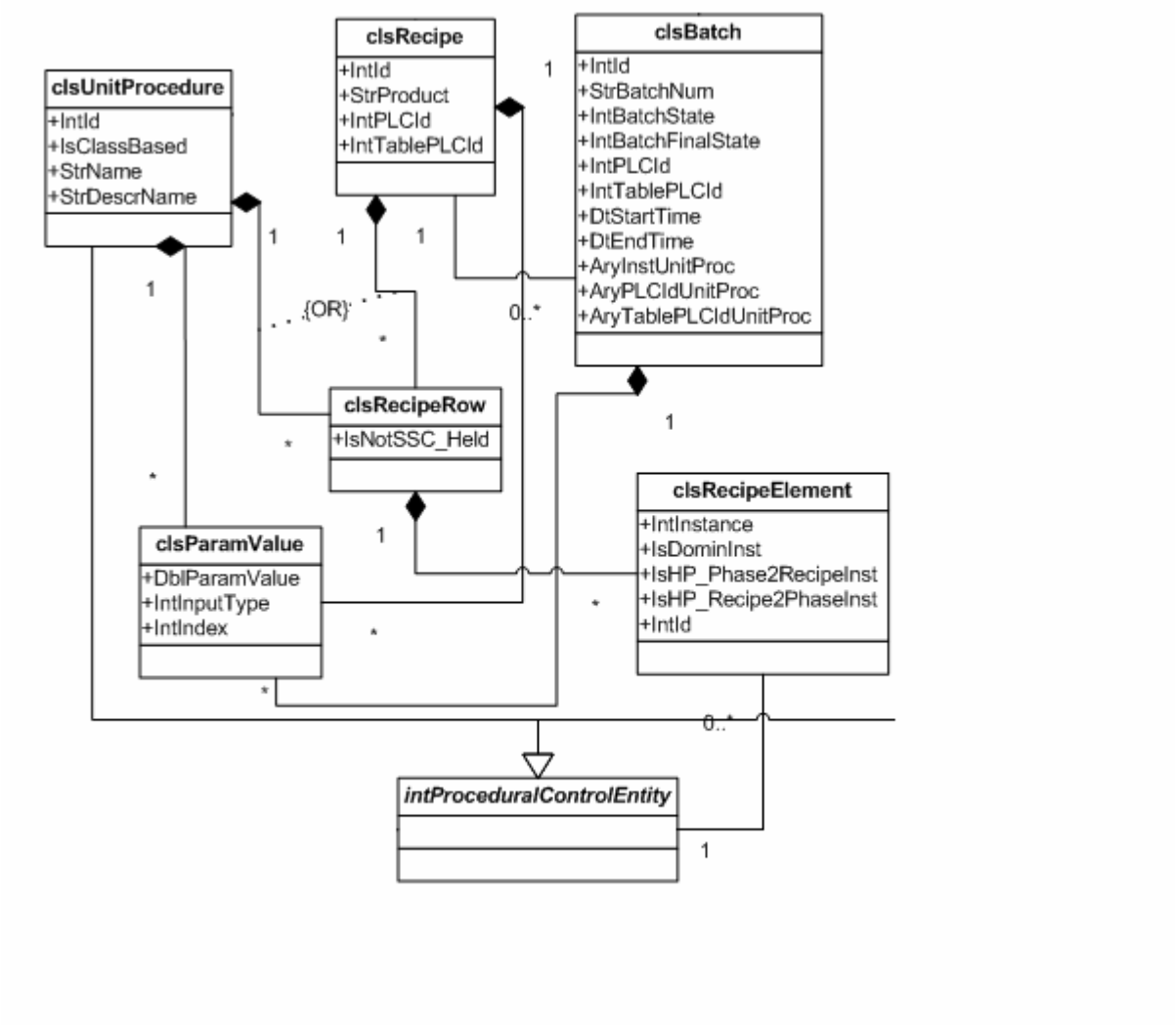
Slika 13 prikazuje relacije med postopki enot, enotami in fazami. Postopek enote vsebuje parametre, vrednosti parametrov faz (vrednosti parametrov posamezne entitete so vedno vsebovane v entiteti višjega nivoja) in tabelo postopka enote, ki jo predstavljata razreda `clsRecipeRow` (vrstica) in `clsRecipeElement` (element vrstice). Faze so s postopkom enote povezane prek tabele postopka enote. V posamezno polje tabele se faze povezujejo s pomočjo vmesnika `intProceduralControlEntity`.



Slika 13 Diagram postopkov enot

4.2.3. Recepti

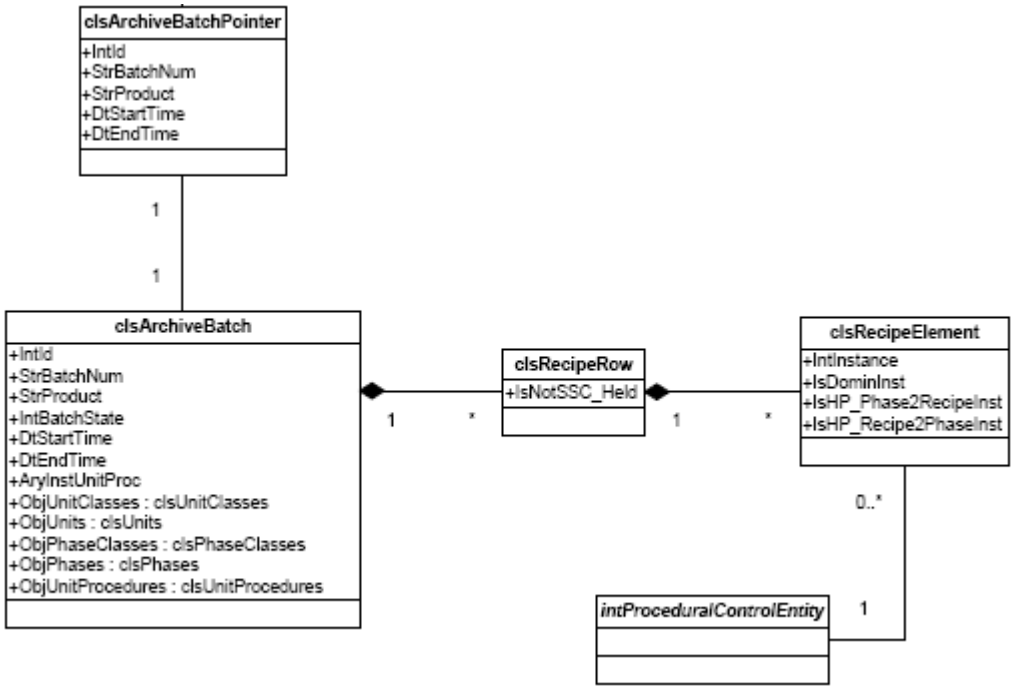
Slika 14 prikazuje relacije med postopki enot, recepti in šaržami. Vidimo. da so relacije med postopki enot in recepti enake kot med postopki enot in fazami. Šarže ne vsebujejo tabele, ker jim pripada samo en recept, vsebujejo pa vrednosti parametrov recepta.



Slika 14 Diagram receptov

4.2.4. Arhiv

Arhiv (Slika 15) predstavlja arhivirane šarže, ki so bile prenesene iz krmilnika. Na vsako tako šaržo kaže kazalec, saj je šarža shranjena v svojo XML datoteko.



Slika 15 Diagram arhiva

5 Opis problemov in rešitev

Pri razvoju se v sodelovanju z uporabniki (poleg tistih na katere naletimo sami) pojavljajo nove zahteve in želje uporabnikov. Te zahteve so pogosto povezane s čim preprostejšo uporabo aplikacije in čim bolj učinkovitim uporabniškim vmesnikom.

V nadaljevanju sledi opis problemov in zahtev, ki prišle s strani uporabnikov ali pa smo na njih naleteli sami in opis mojih rešitev.

Predstavil bom:

- Urejanje podatkov,
- Izogibanje podvajanju podatkov in dvojnemu delu ter
- Vpeljavo linij.

5.1. Urejanje podatkov

Problem urejanja posameznih entitet je vpliv na entitete višjega nivoja, ki vsebujejo to entiteto. Poglejmo si primer brisanja faze, ki jo uporablja nek postopek enote: Pri izbrisu faze ni poskrbljeno da bi se je izbrisala referenca na to fazo iz tabele postopka enote in parametri iz tabele parametrov.

Sprva smo tako situacijo rešili na najbolj preprost način: preprečili smo urejanje entitet, ki jih uporablja entiteta višjega nivoja. Ta način je sicer zelo varen, a kot se je izkazalo tudi zelo omejujoč, saj je potrebno entiteto, ki jo hočemo urediti, najprej izbrisati iz vseh entitet višjega nivoja, ki jo vsebujejo, nato pa jo spet dodati.

Zaradi tega je bilo treba izbrati način, ki je varen in tudi bolj uporabniku prijazen. Izbrali smo način, ki uporabniku omogoča poljubno urejanje entitet, pri čemer ga aplikacija opozori, če kakšna entiteta zaradi urejanja postane neveljavna. Entiteta ali parameter, ki je postal neveljaven s tem povzroči, da postanejo neveljavne tudi vse entitete višjih nivojev, ki to entiteto uporabljajo.

Pravila, ki veljajo pri urejanju entitet so enaka za vse, zato bom podrobneje predstavil samo urejanje razreda faze. Pod urejanje razreda faz štejemo spreminjanje imena, dodajanje in brisanje parametrov, spreminjanje vrednosti parametrom, brisanje in dodajanje razreda faze.

Spreminjanje imena faze ni problematično, saj postopek enote nima podatka o imenu faze.
Pri dodajanju in brisanju parametrov faz pa je potrebno parameter dodati oz. izbrisati iz tabele parametrov faze v postopku enote.

Za dodajanje parametrov skrbita funkciji `addPhaseParameterToUnitProcedure`, ki se jo kliče ko dodajamo parametre v razred faze ali fazo in `addAddPhaseParameter ToUnitProcedure`, ki se jo kliče, ko dodajamo parametre v dodatno fazo. Za brisanje parametrov pa skrbita funkciji `deletePhaseParameterFromUnitProcedure` in `delete AddPhaseParameterFromUnitProcedure`. Funkcije za vsak postopek enote preverijo, če vsebuje fazo, ki smo ji dodali/izbrisali parameter, in v tabeli parametrov ustrezno premakneta parametre in tako dodata/izbrišeta parameter.

Poseben primer pri urejanju parametrov faze nastane, ko uredimo naštevni tip, ki se pojavi kot eden izmed parametrov faze.

Če naštevnemu tipu izbrišemo vrednost ali dodamo nove, moramo osvežiti vse parametre tega tipa, saj je pomembno da poznamo zgornjo in spodnjo mejo, ki se je spremenila. Da se parametri ustrezno osvežijo skrbi funkcija `refreshEnumerationParamters`. Ta funkcija za vse entitete preveri, če vsebujejo parameter tega naštevnega tipa in mu ponovno nastavi spodnjo in zgornjo mejo in na novo privzeto vrednost.

V primeru, če smo izbrisali vrednost naštevnega tipa, moramo vse vrednosti parametrov faze tega naštevnega tipa, ki imajo za vrednost izbrisano vrednost naštevnega tipa, označiti kot neveljavne. S tem smo uporabniku sporočili, da vrednosti niso več veljavne, in da jih mora kasneje ustrezno popraviti. Preverjanje veljavnosti vrednosti parametrov entitet, če so naštevnega tipa, opravlja funkcija `validateParameterValues`.

Pri največjem posegu, brisanju celotnega naštevnega tipa pa je potrebno kot neveljaven označiti vsak parameter tega naštevenga tipa, ki se pojavi v entiteti. V tem primeru pa uporabniku sporočimo, da je celoten parameter ne samo njegova vrednost postal neveljaven

in da ga je potrebno izbrisati. To nalogo opravlja funkcija `validateEnumaration Parameters`.

Pri brisanju celotnega razreda faze iz razreda enote postanejo neveljavni vsi postopki enot, ki uporabljajo izbrisan razred faze. Funkcija `removePhase` v tabeli vsakega postopka enote preveri, če vsebuje izbrisan razred faze ali fazo (ki je izpeljana iz tega razreda faze) in polje (v katerem se nahaja) označi kot neveljavno. Uporabnik tako vidi kateri postopki enote so postali neveljavni in mora neveljavno fazo izbrisati, pri tem se izbrišejo tudi vrednosti parametrov, iz tabele postopka enote. V primeru brisanja razreda faze je seveda potrebno izbrisati tudi vse faze iz vseh instanc enot, ki so izpeljane iz razreda enote iz katerega smo izbrisali razred faze.

Dodajanje razreda faze pa ne predstavlja večje težave, saj še ni uporabljen v nobenem postopku enote. Potrebno pa ga je dodati vsem pojavitvam enote, ki so izpeljane iz razreda faze, ki smo mu dodali razred faze.

Prednost takega načina urejanja entitet je ta da so uporabniku ves čas vidne posledice urejanja. V primeru, da uporabnik izbriše razred faze, so takoj označeni vsi neveljavni postopki enot, ki so uporabljale ta razred faze. Uporabnik tako ve, da mora popraviti vse neveljavne postopke enot in potrebi dodati novo fazo.

5.2. Izogibanje podvajanju podatkov in dvojnemu delu

V naslednjem poglavju bom predstavil problem podvajanja podatkov, ki je pogosto povezan tudi s podvajanjem dela oziroma z odvečnim ali nepotrebnim delom. Ti problemi so predvsem neugodni za uporabnika, ki sestavlja recept.

Podatkovni model aplikacije je sestavljen tako, da se podvajanje podatkov ne pojavlja. Vendar pa se v nekaterih primerih podvajanju zaradi različnih vzrokov ne moremo izogniti. Primer tega so vrednosti parametrov faz in postopkov enot, ki se ponovno pojavijo v šarži. Spreminjanje strukture receptov in postopkov enot, ki so že uporabljeni v razpisani nezaključeni šarži ni dovoljeno, spreminjamo pa lahko vrednosti parametrov. Navadno pa ne želimo, da te spremembe vplivajo na že razpisane šarže, zato vrednosti parametrov ob razpisu pripnemo šarži.

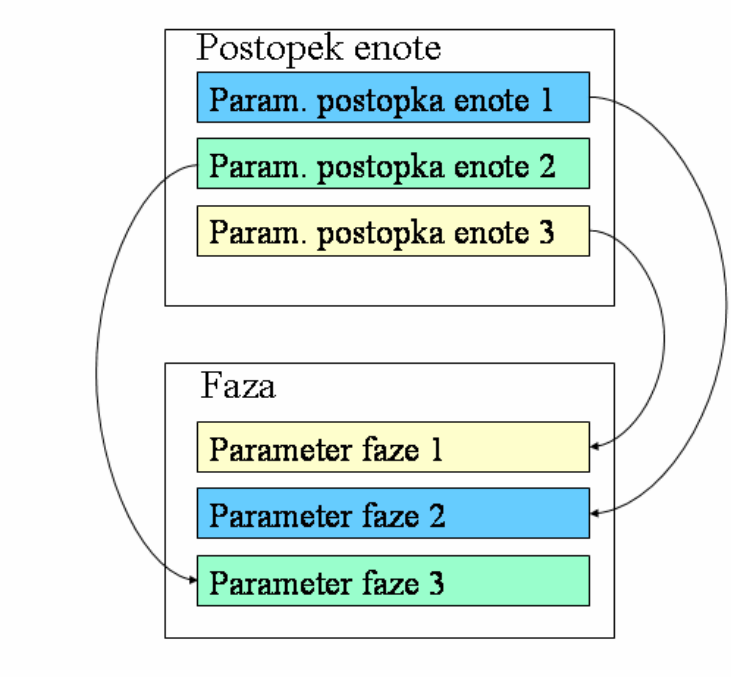
Razdelitev opreme na razrede in pojavitve nam pogosto prihrani delo in čas pri sestavljanju receptov. Kadar imamo več enakih enot opreme, ki opravljajo enake naloge, je smiselno, da jih predstavimo skupaj kot razred enote, posamezne enote pa kot pojavitve enote, ker informacije o njih podaja razred enote. Razredi enot vsebujejo ime, identifikacijsko številko (ID) in seznam razredov faz, ki jim pripadajo. Pojavitve enot pa se razlikujejo od razredov enot v tem, da nimajo seznama, faz ampak ID razreda enote, kateremu pripadajo. Vendar pa ima lahko instanca enote poljubno število dodatnih faz, zato lahko v takem primeru, instanca enote vsebuje celo več informacij. Instanca enote torej po vsebini ni nujno enaka razredu enote, medtem, ko je faza enaka razredu faze zato o fazi hranimo precej manj informacij. Razredi faz poleg imena in kontrolnega bloka vsebujejo še parametre z njihovimi atributi. Tako nam je potrebno o fazi vedeti le to kateremu razredu in kateri instanci enote pripada.

Tako nam ni potrebno za vsakega posebej pisati postopka enote. Postopku enote določimo, da je sestavljen na osnovi razreda, in ga sestavimo. Takemu postopku enote pojavitev enote določimo ob razpisu oziroma ob nalaganju šarže.

Vpeljava odlaganja vrednosti parametrov prav tako prispeva k zmanjšanju dela in podvajanja podatkov. Odlaganje vrednosti parametrov je namenjeno odlaganju določanja vrednosti na kasnejši čas. V ta namen se vpelje nov atribut vrednosti parametra, ki določa način prirejanja vrednosti. Ima lahko naslednje vrednosti:

1. ob kreiranju recepta
2. ob razpisu šarže
3. ob nalaganju šarže
4. od nadrejene entitete ali
5. Iz polja podatkov šarže.

Vrednost tega atributa ni fiksni atribut parametra, temveč se ga določi pri urejanju parametra faze v postopku enote oziroma pri urejanju vrednosti parametra postopka enote v receptu. Z odlaganjem parametrov lahko precej zmanjšamo število postopkov enot. Postopkov enot, ki imajo enako strukturo, vendar pa imajo različne nekatere vrednosti parametrov faz, ni potrebno več podvajati, ampak kreiramo samo enega. Parametrom, katerih vrednosti bi bile različne, pa določimo način prirejanja na vrednost od nadrejene entitete in izberemo parameter te nadrejene entitete. To pomeni, da bo tak parameter dobil vrednost od parametra nadrejene entitete, ki smo ga izbrali (Slika 16 barva param. faze predstavlja param. postopka enote od katerega dobi vrednost). Tako te vrednosti določimo šele, tedaj ko dodamo postopek enote v recept, saj takrat določimo vrednosti parametrov postopka.



Slika 16 Parameter faze dobi vrednost od parametra postopka enote

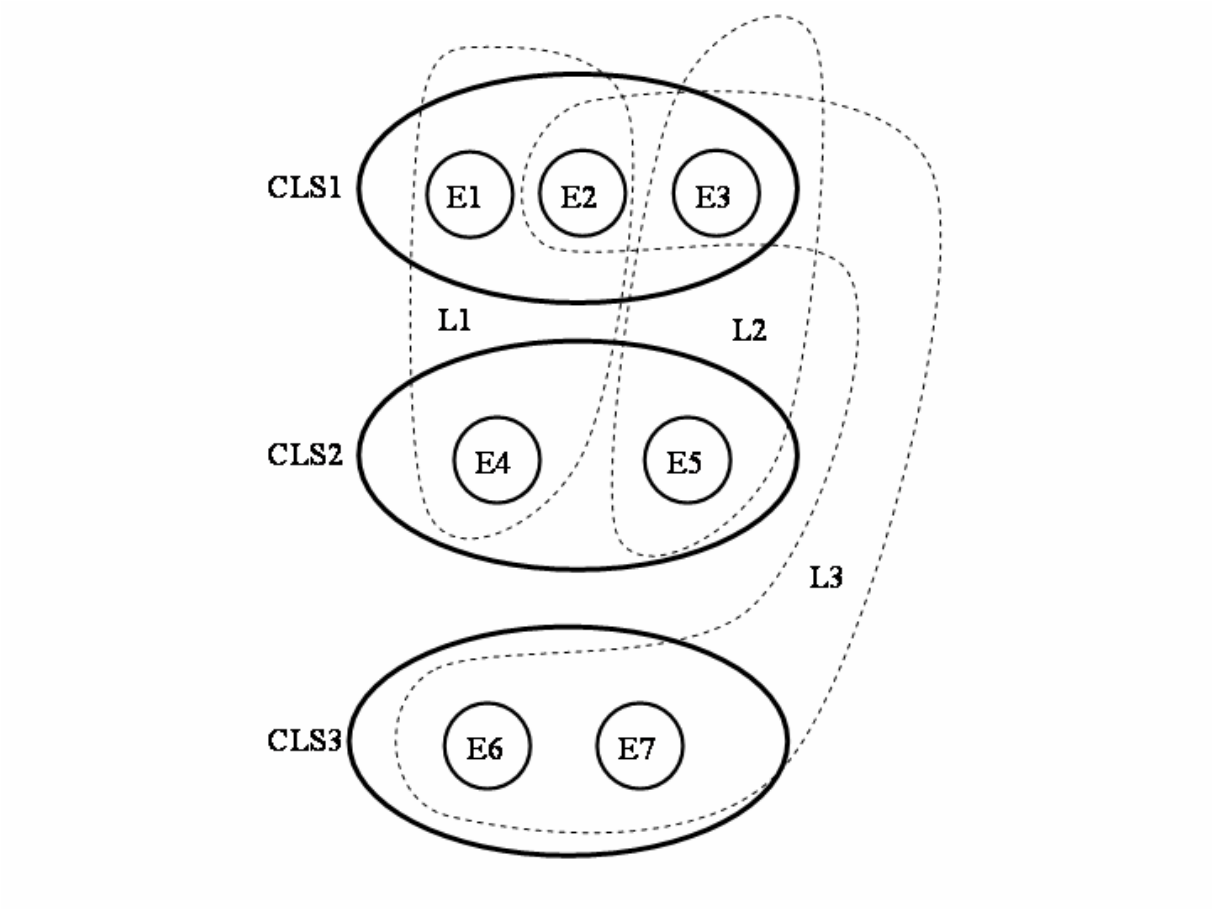
5.3. Vpeljava linij - zagotavljanje pravilnosti recepta

Vpeljava linij je pomembna pri zagotavljanju pravilnosti/veljavnosti recepta. Pod besedo linija razumemo sistem naprav, strojev, ki omogoča popolnoma avtomatiziran delovni postopek Smiselno je torej da se recept izvaja samo na eni liniji, torej v receptu ne smejo biti postopki enot na različnih linijah.

Linija določa množico enot, ki se jih lahko uporabi v enem receptu. Določena enota lahko nastopa v večjem številu linij. Pri tem veljajo naslednje relacije:

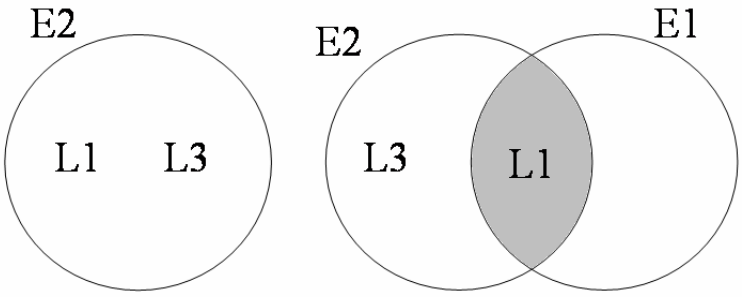
1. Oprema je razvrščena v razrede enot in združena v linijah. V linijo se lahko združuje celotne razrede enot ali pa samo posamezne pojavitve,
2. Pri izvajanju recepta morajo biti vsi njegovi postopki enot na isti liniji.
3. Za vsak recept se lahko ugotovi možen nabor linij, na katerih se lahko izvaja. To je presek množic linij posameznega postopka enote tega recepta.
4. Med kreiranjem recepta mora biti možen nabor linij v vsakem trenutku neprazna množica. Preprečiti se mora dodajanje novega postopka enote, če bi po njegovem dodajanju možen nabor linij recepta postal prazna množica.
5. Med povezovanjem recepta z opremo (preslikavo razreda v enote) je za vsak razred enote možno izbrati tiste pojavitve, pri katerih možen nabor linij ne bi postal prazna množica. To se poenostavi tako, da se najprej izbere želena linija šele, nato pa se izbere ustrezne pojavitve
6. Na koncu povezovanja recepta z opremo je možen nabor linij neprazna množica. Navadno je to le ena linija lahko pa tudi več kot ena.

Oglejmo si relacije na naslednjem primeru (Slika 17):
Imamo tri razrede enot(CLS1, CLS2, CLS3), sedem instanc(E1, E2, E3, E4, E5, E6 in E7) enote in tri linije(L1, L2, L3). Prve tri instance (E1-E2) sodijo v razred CLS1, enoti eE4 in E5 v razred CLS2 in zadnji dve v razred CLS 3. V liniji L1 so združene enote E1, E2 in E4, v liniji L2 enoti E3 in E5 in v liniji L3 enote E2, E3, E6 in E7.



Slika 17 Relacije med razredi enot, enot in linijami

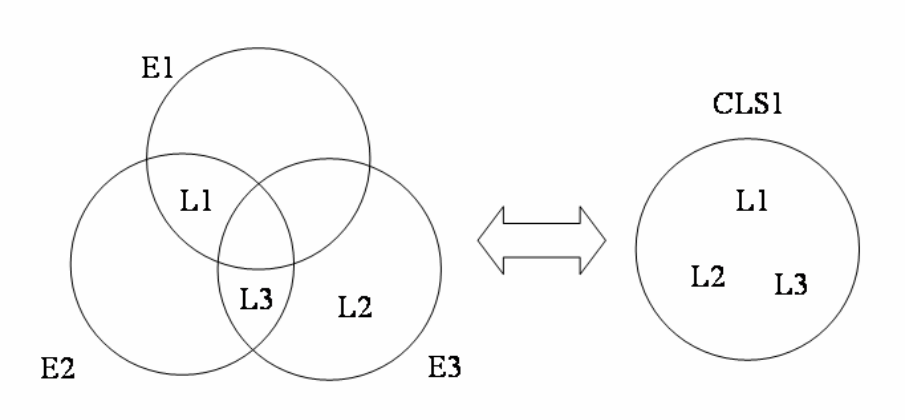
Ko imamo tako definirane relacije med enotami in linijami lahko začnemo dodajati enote v recept. Če v recept dodamo enoto E2 (v resnici dodajamo postopek enote) je možen nabor linij, na katerih se recept lahko izvaja, množica linij L1 in L3. Torej lahko dodamo le enote, ki so združene v linijah L1 in L3, če ne želimo, da postane nabor linij prazna množica. Dodajmo v recept še enoto E1, ki je v liniji L1; sedaj pa se recept lahko izvaja samo na liniji L1, saj je presek linij enote E1 in E2 ravno linija L1(Slika 18), enako pa velja tudi za enoto E4.



Slika 18 Nabor linij pred in po dodajanju enote E1 v recept

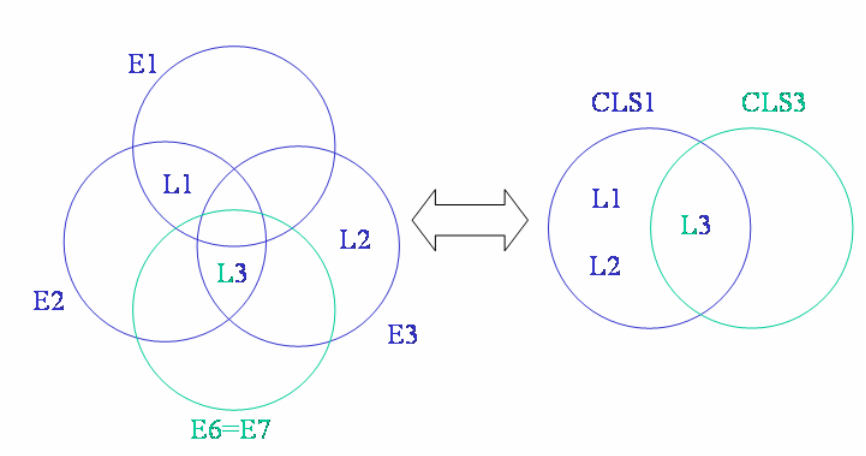
V primeru, da bi namesto enote E1 ali E4 dodali enoto E6 ali E7, bi se recept lahko izvajal samo na liniji L3. Vidimo torej, da se recept lahko vedno izvaja samo na eni liniji, razen v primer da bi bilo ali več linij enakih, kar pomeni, da bi bile v teh linijah združene enake enote.

V prejšnjih primerih smo v recept dodajali samo pojavitve enot, zato si poglejmo, kaj se zgodi, če v recept dodajamo razrede enot. Dodani razred enote se lahko ob povezovanju recepta z opremo preslika v katerikoli pojavitev tega razreda, ki je v ustrezni liniji. Dodajmo v recept razred enote CLS1 recept se lahko izvaja na linijah L1, L2, L3 to pa so linije, v katerih so instance tega razreda. Če bi v receptu ostala samo ta enota, bi se razred enote lahko preslikal v katerokoli instanco, kar pa pomeni, da se recept izvaja na tisti liniji, v kateri je izbrana instanca enote(Slika 19).



Slika 19 Relacija med enotami in razredom enote

V recept poleg razreda enote cls1 dodajmo še razred enote CLS3; sedaj pa se recept lahko izvaja samo na liniji L3. Razred enote CLS3 se lahko izvaja samo na liniji L3, ker sta njegovi instanci E6 in E7 v liniji L3, zato je presek linij razredov enot cls1 in CLS3 linija L3 (Slika 20). Pri preslikavi razredov enot v enote se torej razred enote cls1 lahko preslika v enoto E2 ali E3, razred enote cls3 pa v enote E6 ali E7. Vidimo torej, da moramo ob dodajanju razreda enote v recept upoštevati, v katerih linijah so instance enot tega postopka enote.



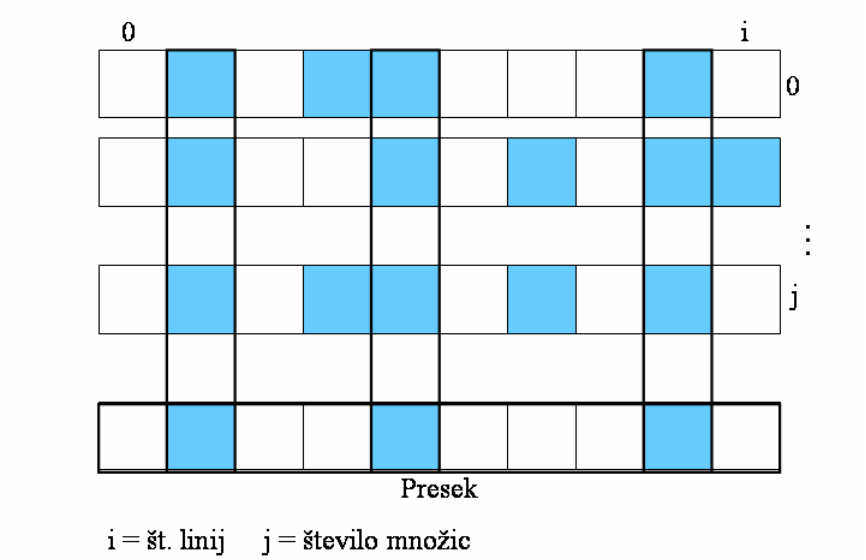
Slika 20 Nabor linij po dodajanju razreda enote CLS3

Opraviti imamo z operacijami nad množicami, predvsem s presekom. Množice so zaradi enostavnejšega izvajanja operacij nad njimi predstavljene s poljem boolovih spremenljivk. Linijo (`clsLine`) predstavlja polje dolžine največjega števila dovoljenih enot. Kjer vrednost true pomeni, da je enota v liniji, vrednost false pa pomeni, da enota ni v liniji. Množica linij posamezne enote/postopka enote (`clsSetsOfLines`) pa je predstavljena s poljem dolžine števila linij kjer vrednost true, na prvem mestu pomeni, da je enota v prvi liniji.

Preverjanje, na katerih linijah se lahko recept izvaja, poteka v dveh korakih. V prvem koraku se iz recepta dobi množico množic linij posamezne enote (`clsSetsOfLines`) to množico vrne funkcija `getLinesFromRecipe`. Funkcija za vsak postopek enote v tabeli recepta izračuna množico linij, v katerih je enota tega postopka enote. V primeru, da je postopek enote izpeljan iz pojavitve enote se za vsako n-to linijo preveri, če je na i-tem mestu (i predstavlja ID enote, iz katere je izpeljan postopek enote) v polju linije vrednost true. Če je na i-tem vrednost true pomeni, da je ta enota v n-ti liniji, potem se tudi v polju množice linij posamezne enote na n-tem mestu vrednost nastavi na true. Ko pa je postopek enote izpeljan iz razreda enote se postopek ponovi za vsako enoto, ki je izpeljana iz tega razreda enote. Na ta način dobimo toliko množic linij, kolikor je postopkov enot v receptu.

V drugem koraku pa je potrebno iz dobljene množice množic linij posamezne enote izvesti še operacijo preseka. Funkcija `intersection` opravlja nalogo operacije preseka in vrne množico presek, ki je istega tipa kot množica linij posamezne enote (`clsSetsOfLines`) . Presek se izvaja tako, da se za vsako linijo preveri, če je v vseh množicah linij posamezne enote. Preverja se tako, da se za vsako i-to linijo preveri, če je v vsaki j-ti množici na i-tem

mestu v polju vrednost true. Če to velja za vsako množico, se tudi v množici presek na i-tem mestu nastavi vrednost true (Slika 21).



Slika 21 Prikaz operacije preseka

Na zgoraj opisani način torej dobimo množico linij, na katerih se lahko recept izvaja, ki pa ni nikoli prazna. To dosežemo tako, da je že v začetku onemogočeno dodajane postopkov enot, ki bi povzročili, da bi bila množica linij prazna. Preprečevanje dodajanja takih postopkov enot poteka tako, da za vsakega ugotovimo, če bi postala množica linij prazna v primeru ,če bi ga dodali. To pa poteka tako kot preverjanje na katerih linijah se lahko recept izvaja z dvema razlikama. Po prvem koraku se množici množic linij posamezne enote doda še množica linij enote katere postopek enote bi lahko dodali v recept. Potem, ko se izvede še drugi korak (operacija presek), se preveri, če je množica prazna. V primeru, da množica ni prazna, je postopek enote možno dodati v recept, v nasprotnem primeru pa ne. Preverjanje če je množica prazna opravlja funkcija `isSetEmpty`, ki preveri, če imajo vsa polja v množici vrednost false.

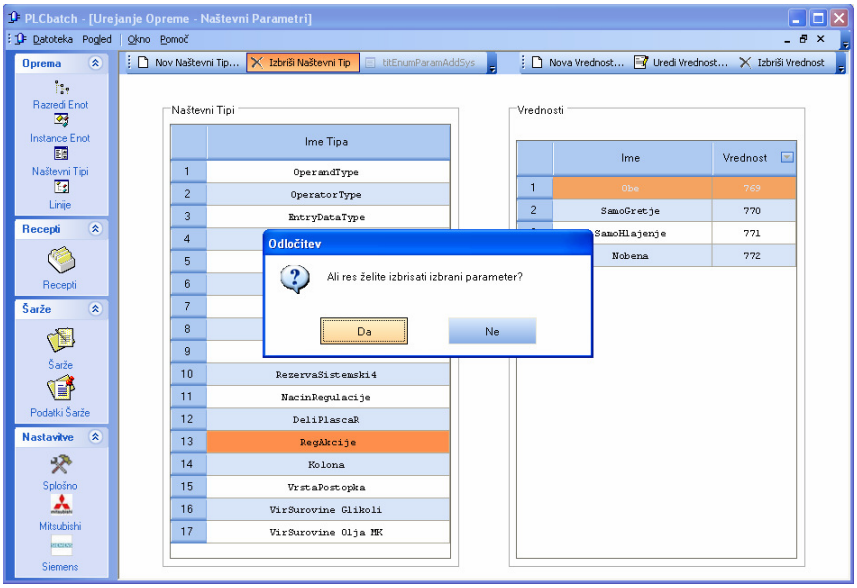
Način predstavitev množic, z poljem boolovih spremenljivk omogoča enostavno izvajanje operacije preseka. Ko preverjamo, če je linija v vseh množicah, v vsakem trenutku vemo, na katerem mestu se linija nahaja v posamezni množici in nam je zato ni potrebno iskati. Podobno velja tudi takrat, ko ugotavljamo množico linij posameznega postopka enote.

6 Prikaz rešitev na aplikaciji

V nadaljevanju je na kratkih primerih prikazana uporaba rešitev opisanih v prejšnjem poglavju.

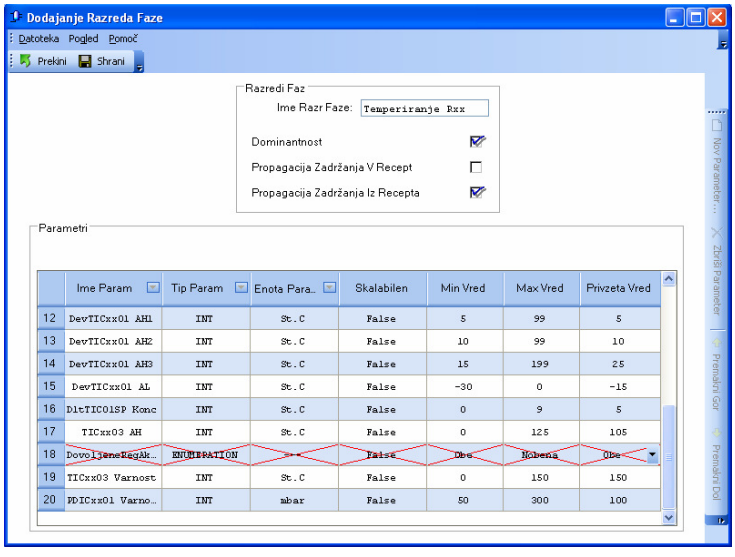
6.1. Urejanje podatkov

Pogledali si bomo dva primera urejanja podatkov. V prvem je prikazano brisanje naštevnega tipa, v drugem pa brisanje razreda faze. Predstavljene bodo tudi posledice teh dveh posegov. Slika 22 prikazuje brisanje naštevnega parametra.



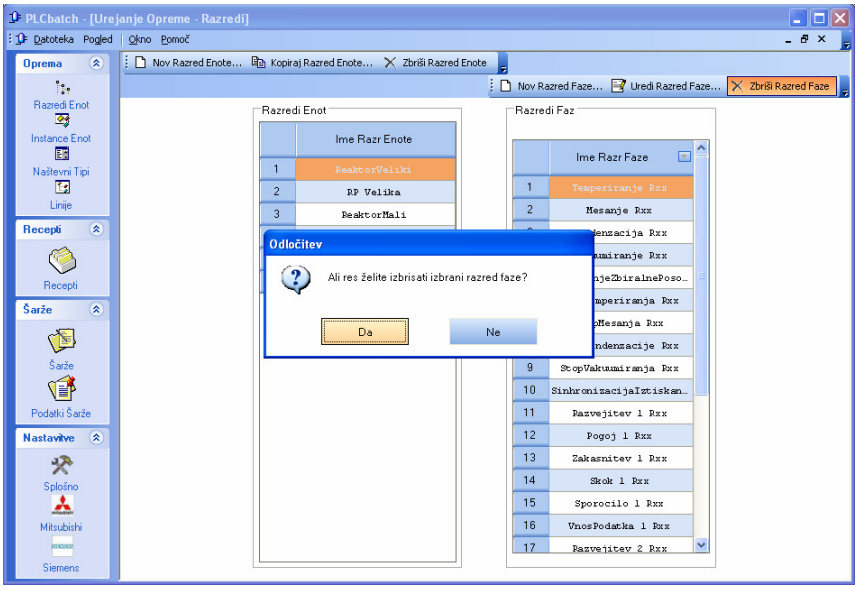
Slika 22 Brisanje naštevnega tipa

Vsi parametri, ki so izbrisanega naštevnega tipa, postanejo neveljavni. To prikazuje Slika 23.



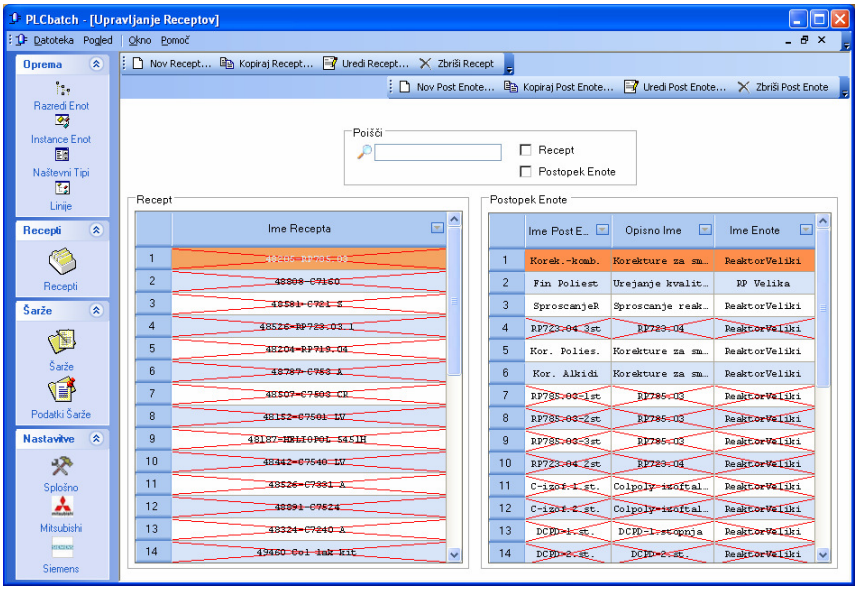
Slika 23 Neveljavni paramter

Naslednji primer predstavlja brisanje razreda faze, kar je prikazuje Slika 24.



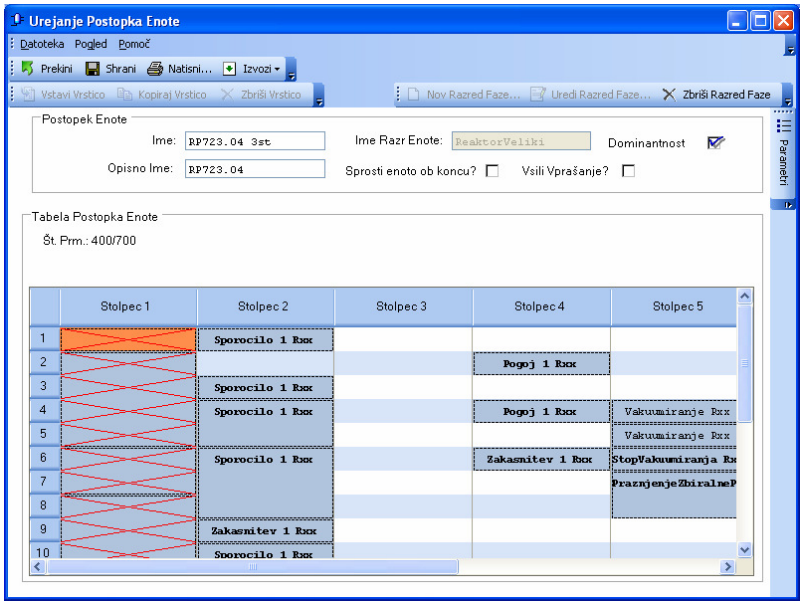
Slika 24 Brisanje razreda faze

S tem, ko smo izbrisali razred faze, so postali neveljavni vsi postopki enot, ki uporabljajo izbrisano fazo. Prav tako pa postanejo neveljavni tudi vsi recepti, ki uporabljajo te razrede faz. Vse to prikazuje Slika 25.



Slika 25 Neveljavni postopki enot in recepti

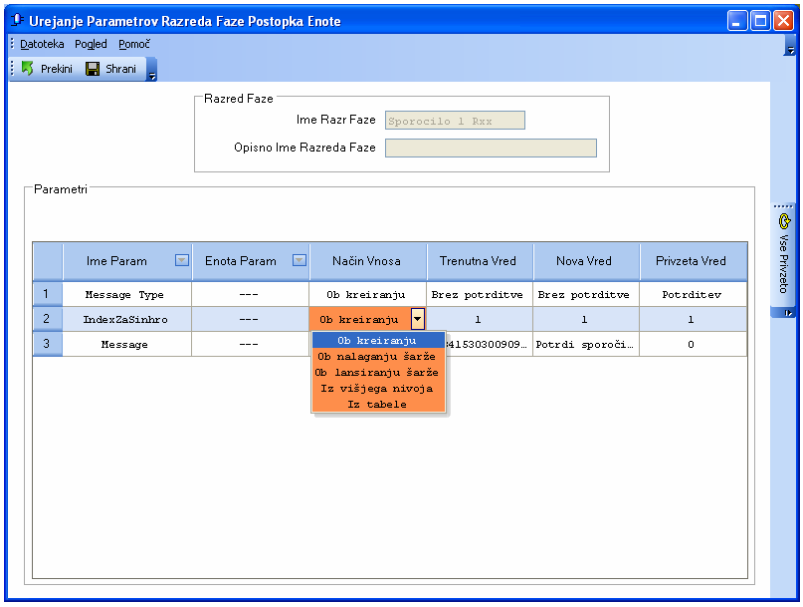
Izbrisan razred faze je označen tudi znotraj postopka enote v njegovi tabeli, kar prikazuje Slika 26.



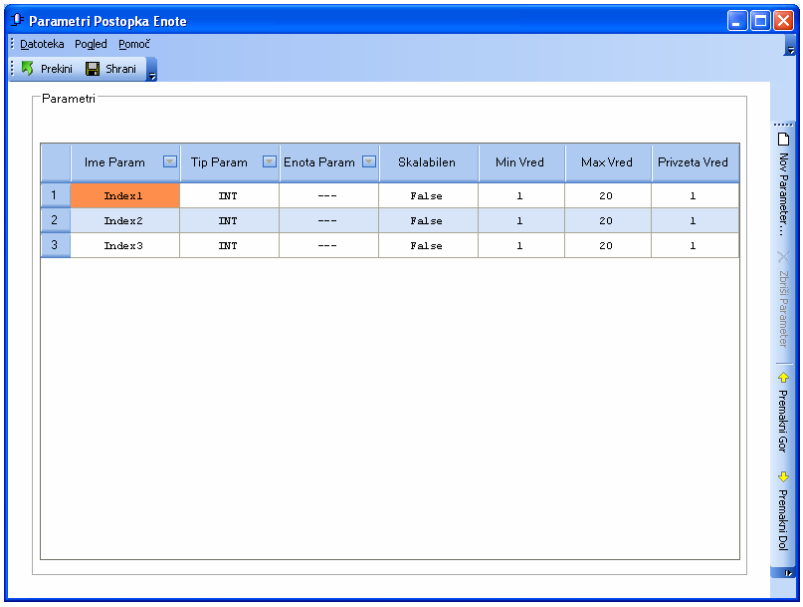
Slika 26 Neveljavni razredi faz

6.2. Izogibanje podvajanju podatkov in dvojnemu delu

Naslednji primer prikazuje odlaganje parametrov. Izbranemu parametru faze bomo določili način prirejanja vrednosti na iz višjega nivoja (Slika 27 Odlaganje parametrov), kar pomeni da bo dobil vrednost od parametra nadrejene entitete, v tem primeru postopka enote (parametri postopka enote prikazuje Slika 28).

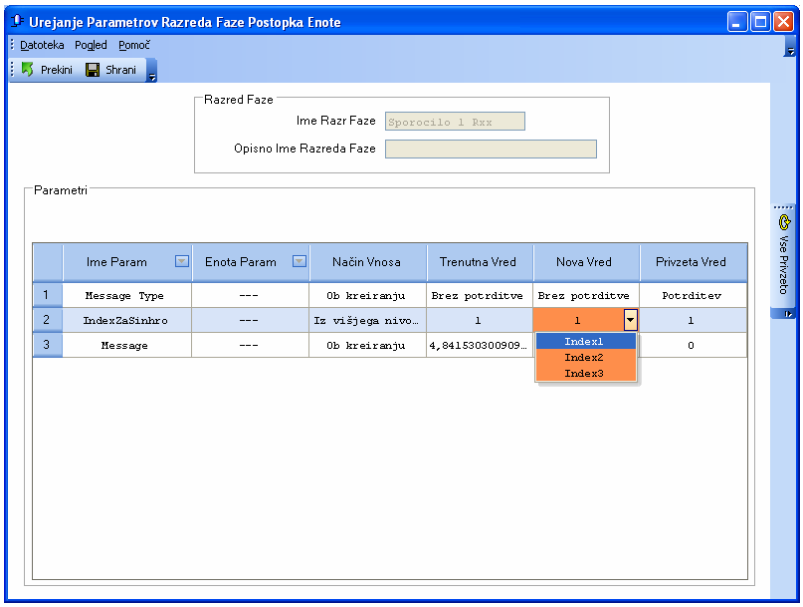


Slika 27 Odlaganje parametrov



Slika 28 Parametri postopka enote

Na koncu še izberemo parameter postopka enote od katerega bo dobil vrednost(Slika 29)

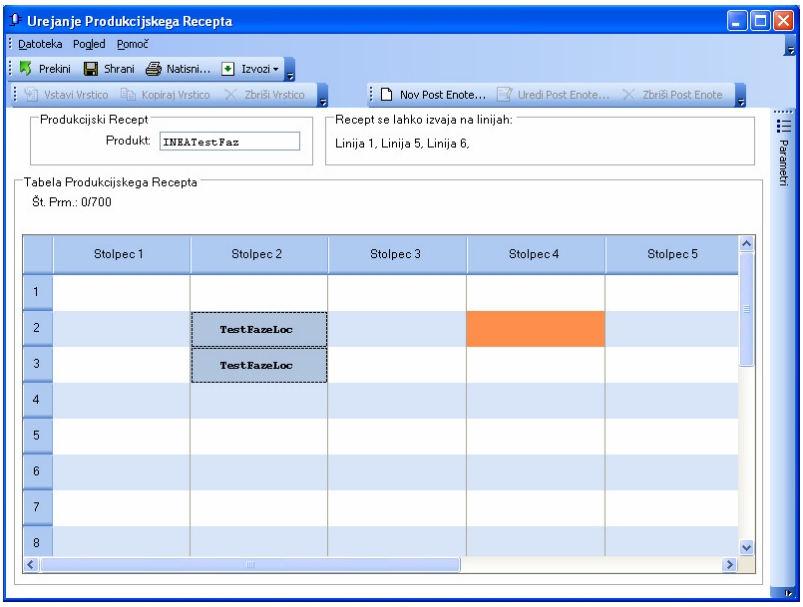


Slika 29 Izbiranje parametra

.

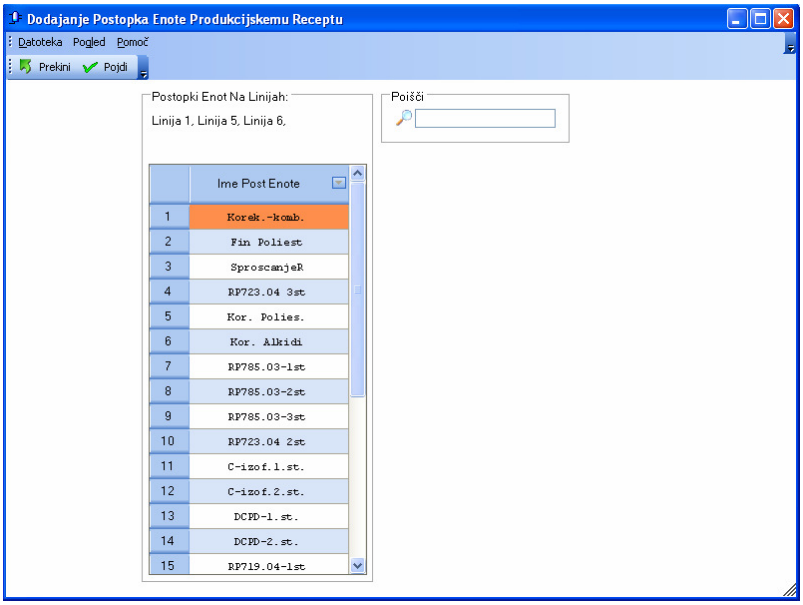
6.3. Vpeljava linij - zagotavljanje pravilnosti recepta

Slika 30 prikazuje recept, iz katerega je razvidno na kateri liniji se lahko izvaja.



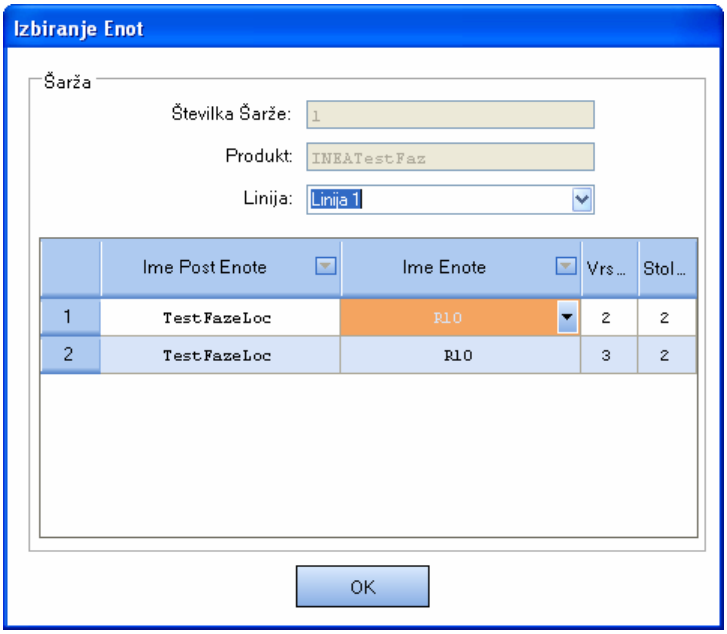
Slika 30 Recept

Ob dodajanju postopka enote v recept lahko dodamo samo tiste postopke enot, ki se lahko izvajajo na istih linijah kot recept. Dodajanje postopka enote je prikazano na Slika 31.



Slika 31 Dodajanje postopka enote

Ob kreiranju šarže ali pa ob nalaganju šarže izberemo, na kateri enoti se bo recept izvajal. Izbor je prikazan na Slika 32.



Slika 32 Izbiranje linije in enot

7 Sklep

V diplomski nalogi smo predstavil nadgradnjo orodja za vodenje procesov z rešitvijo problemov, ki so se pokazali v sodelovanju z naročniki, kot tudi tiste, do katerih smo prišli sami. Lastne rešitve teh problemov smo prikazali tudi na kratkih primerih.

Probleme in njihove lastne rešitve smo skušali opisati čim bolj jasno s podajanjem primerov, da bi bralec le te čimbolj razumel. Rešitve zanesljivo delujejo, saj je orodje že v aktivni uporabi.

Prednosti opisanih rešitev so naslednje:

- Uporabljena rešitev problema urejanja podatkov z označevanjem neveljavnih entitet omogoča uporabniku, da ima ves čas pregled nad urejanjem. S tem se uporabnik zaveda posledic urejanja in ve, katere entitete mora popraviti.
- Različne rešitve povezane z izogibanjem odvečnemu delu in podvajanju uporabniku omogočajo hitro in enostavno delo z urejevalnikom receptov. Z vpeljavo odlaganja parametrov se je močno zmanjšalo število postopkov enot.
- Vpeljava linij zagotavlja pravilnost recepta. Opisana rešitev uporabniku niti ne dovoli, da bi v recept dodajal postopke enot, ki bi se izvajali na različnih linijah.

Opisane rešitve je seveda možno tudi izboljšati in nadgraditi. V primeru izbire linije na kateri naj se izvaja recept, bi lahko linijo določili že ob kreiranju recepta. Tako bi v recept lahko dodajali samo tiste postopke enot, ki se izvajajo na tisti liniji, ki smo jo določili ob kreiranju recepta. Tudi glede podvajanja podatkov in zmanjšanja odvečnega dela je možno še nekaj rešitev. Ker pa razvoj aplikacije še vedno poteka v aktivnem sodelovanju z naročniki so bo aplikacija še naprej dopolnjevala, tako da novih izzivov ne bo zmanjkalo.

Viri

[1] Visual Studio Developer Center. Dostopno na:
http://msdn.microsoft.com/en-us/vstudio/default.aspx

[2] XML Tutorial. Dostopno na:
http://www.w3schools.com/xml/default.asp

[3] OPC Overview. Dostopno na:
www.opcfoundation.org/DownloadFile.aspx/General/OPC%20Overview%201.00.pdf

[4] What is OPC? Dostopno na:
http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp

[5] DevExpress. Dostopno na:
http://www.devexpress.com/

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

[7] Dokumentacija podjetja INEA d.o.o.

[8] Twelve Principles of Agile Software. Dostopno na:
http://www.agilemanifesto.org/principles.html

Izjava o samostojnosti dela

S svojim podpisom zagotavljam, da sem diplomsko delo izdelal samostojno pod vodstvom dr. Igorja Rožanca.

Gregor Puh