SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Boris Milašinović

GENERIČKI OKVIR ZA UPRAVLJANJE PROTOKOM POSLOVA TEMELJEM PARCIJALNO DEFINIRANIH MODELA

DOKTORSKA DISERTACIJA

Zagreb, 2010.

Doktorska disertacija je izrađena u **Zavodu za primijenjeno računarstvo Fakulteta elektrotehnike i računarstva u Zagrebu**.

Mentor: **Prof.dr.sc. Krešimir Fertalj**

Disertacija ima 117 stranica.

DISERTACIJA BR.:

Povjerenstvo za ocjenu doktorske disertacije:

1. Dr.sc. Damir Kalpić, redoviti profesor
   Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

2. Dr.sc. Krešimir Fertalj, izvanredni profesor
   Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

3. Dr.sc. Nataša Hoić-Božić, izvanredna profesorica
   Sveučilište u Rijeci – Odjel za informatiku

Povjerenstvo za obranu doktorske disertacije:

1. Dr.sc. Damir Kalpić, redoviti profesor
   Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

2. Dr.sc. Krešimir Fertalj, izvanredni profesor
   Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

3. Dr.sc. Nataša Hoić-Božić, izvanredna profesorica
   Sveučilište u Rijeci – Odjel za informatiku

4. Dr.sc. Vesna Bosilj Vukšić, redovita profesorica
   Sveučilište u Zagrebu Ekonomski fakultet

5. Dr.sc. Mario-Osvin Pavčević, izvanredni profesor
   Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva

Datum obrane disertacije: 27. svibnja 2010. godine

Zahvaljujem mentoru prof.dr.sc. Krešimiru Fertalju na povjerenju, poticaju i pomoći prilikom izrade ove disertacije, mr.sc. Ružici Piskač na pomoći prilikom nabave literature i svim kolegama sa Zavoda za primijenjeno računarstvo na podršci i prijateljskom radnom okruženju.

Sadržaj

1. Uvod

Projektiranjem programske potpore u obliku višeslojnih (najčešće troslojnih) aplikacija omogućeno je odvajanje prezentacijskih dijelova aplikacije, poslovnog sloja te sloja za pristup bazi podataka ili nekom drugom spremištu podataka. Navedenim pristupom, moguće je lakše izdvojiti pojedini sloj te ga lakše održavati i po potrebi zamijeniti nekom drugom realizacijom. Primjerice, manjom promjenom baze podataka, poslovni i prezentacijski elementi neće nužno doživjeti promjene, nego će samo doći do promjene u sloju za pristup bazi podataka. Iako najčešće istican kao primjer zamjene u višeslojnim aplikacijama, u dobro strukturiranoj bazi podataka, sloj za pristup podacima manje promjene ne doživljava toliko često, koliko su takve promjene vjerojatne u poslovnoj logici. Promjene u poslovnoj logici nastaju uslijed promjena načina poslovanja ili radi poboljšanja postojećih poslovnih procesa.

Pravilno prepoznati poslovni proces i uspješno ga programski podržati jedna je od temeljnih postavki uspješne informatizacije poslovanja. Ukoliko organizacija sama ne može kvalitetno opisati što koji pojedinac, odnosno grupa pojedinaca radi u nekom trenutku, ni najbolji softver joj neće pomoći. Slično, ukoliko je softverska podrška loša te se organizacija više brine o tome kako softver radi, a ne što radi, onda dolazi do gubitka vremena na nebitne stvari. Upravo je prepoznavanje poslovnog procesa, kao i upravljanje protokom poslova postalo jedna od disciplina kojoj se proteklih nekoliko godina pridaje sve veći značaj. Kroz model protoka poslova takvo poslovanje moguće je automatizirati, a uz dobru formalnu definiciju takvog modela moguća je i analiza i verifikacija modela protoka poslova.

Međutim, implementacija modela procesa je samo jedan kotačić unutar mozaika unaprjeđenja vlastitog poslovanja. Jednom ugrađen, sustav se stalno mijenja. Poslovni postupci se s vremenom mijenjaju, ljudski i materijalni resursi se mijenjaju, prelaze iz jedne kategorije u drugu, uočavaju se propusti napravljeni pri inicijalnoj implementaciji i nužna je mogućnost popravka, dorade i nadgradnje. Ukoliko sustav nije dovoljno elastičan za nužne male *ad hoc* promjene, odnosno ako nije moguće lako izvršiti prilagodbe postojećih postupaka i parametara sustava, onda se takav sustav ne može nazvati kvalitetnim.

Iako tradicionalno vezano uz poslovnu domenu, oblikovanje poslovnih procesa i modela protoka poslova, može se primijeniti i na fakultetsko okruženje modeliranjem procesa vezanih za računalom podržano učenje, ali i modela za oblikovanje studijskih programa. Također, naglasak treba staviti na jednostavnost izrade modela protoka poslova te omogućiti izradu manjih, parcijalnih modela protoka poslova. Primjerice, razumno je za pretpostaviti da u sustavu izrade studijskog programa sudjeluje više sudionika i da će svaki nositelj kolegija (odnosno autor nastavne cjeline u sustavu za udaljeno učenje) lako opisati svoj kolegij i navesti njegove preduvjete te stoga treba omogućiti jednostavnu integraciju tako parcijalno definiranih modela u jedan integralni model uz provjeru ispravnosti modela, kao što su otkrivanje potencijalnih ciklusa među preduvjetima, neostvarivih uvjeta i slično. Na taj način, promjene se obavljaju samo u manjim, parcijalnim modelima, dok se integralni model ponovno izgrađuje prema za to predviđenim algoritmima iznesenim u poglavljima 3, 4 i 5.

Cilj ovog rada je bio omogućiti izradu parcijalnih modela, njihovu verifikaciju i objedinjavanje u integralni model pretvoren u jedan od konkretnih modela za upravljanje protokom poslova. Rad je podijeljen u osam poglavlja. Nakon uvoda u drugom poglavlju dan je kratak povijesni pregled razvoja sustava za upravljanje protokom poslova. Nakon povijesnog pregleda slijedi kratki pregled mogućnosti upotrebe tehnologije Windows Workflow Foundation (u daljnjem tekstu WF) namijenjene za upravljanje protokom poslova, odabrane za praktičnu demonstraciju ugradnje parcijalnih modela zbog svoje jednostavnosti upotrebe i interoperabilnosti. Poglavlje završava opisom prethodnih pokušaja povezivanja sustava za udaljeno učenje i sustava za upravljanje protokom poslova i identificiranjem ključnih nedostataka prikazanih pokušaja. Treće poglavlje uvodi pojam parcijalno definiranih modela protoka poslova te definira osnovne elemente takvih modela i algoritme kojima se reguliraju njihovi međusobni odnosi. Za složenije slučajeve potrebno je uvesti vremensku komponentu, što za sobom povlači i probleme raspoređivanja čime se bavi četvrto poglavlje. Iznesen je prijedlog kako uvesti vremensku komponentu u pojedine elemente te proizvesti sučelje koje će kroz konkretne implementacije pomoću genetskih algoritama omogućiti rješenje problema raspoređivanja. Peto poglavlje uvodno ukazuje na problematiku pretvorbe u konkretni WF model nakon čega je predloženo rješenje nastalog problema u vidu kloniranja pojedinih elemenata. Definiraju se algoritmi za stvaranje WF modela, a poglavlje završava opisom omotača protoka poslova koji bi omogućio ispravan rad s klonovima pojedinih elemenata. Šesto poglavlje ukratko opisuje programsko i grafičko sučelje vlastitog rješenja koje omogućuje izradu tako definiranih parcijalnih modela protoka poslova. Praktična primjenjivost parcijalnih modela opisana je u sedmom poglavlju na primjeru dvije analize slučaja na kojima se mogu primijeniti parcijalni modeli: izradi studijskog programa po programu FER-2 Fakulteta elektrotehnike i računarstva te izradi redoslijeda cjelina unutar modula za učenje u sustavu Adaptive Hypermedia Courseware (AHyCo), koji se koristi za provjere znanja i za e-učenje na Fakultetu elektrotehnike i računarstva u Zagrebu i Odsjeku za informatiku Filozofskog fakulteta u Rijeci. Rad završava zaključkom i pregledom citirane literature.

2. Upravljanje protokom poslova

2.1 Povijesni razvoj

Preoblikovanje (reinženjerstvo) poslovnih procesa (eng. *Business process re-engineering*, u daljnjem tekstu BPR) i upravljanje protokom poslova (eng. *Workflow Management*, u daljnjem tekstu WfM) su dvije popularne discipline u području razvoja informacijskih sustava. Iako su i BPR i WfM orijentirani na procese i koriste slične modele, potrebno je istaknuti razliku. BPR se može definirati kao postupak temeljne reorganizacije poslovnog procesa, koji fundamentalno mijenja poslovni proces i donosi dramatične promjene u pogledu troškova, usluga i kvalitete, dok se upravljanje protokom poslova bavi automatizacijom poslovnog procesa.

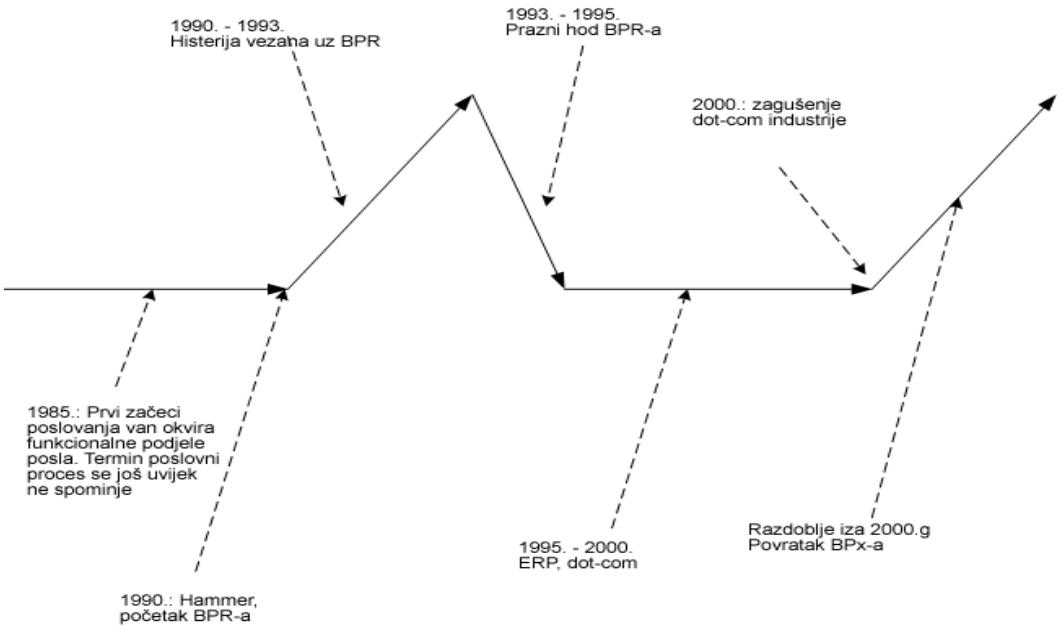
Workflow Management Coallition (WfMC) [WfMC], kao globalna organizacija koja okuplja dionike upravljanja protokom poslova i preoblikovanjem poslovnih procesa definira protok poslova kao *automatizaciju poslovnog procesa, djelomično ili u cjelini, prilikom čega se dokumenti, informacije i zadaci za pojedinu akciju prosljeđuju od jednog sudionika procesa prema drugom prema definiranom skupu proceduralnih pravila* [WfMC-TC-1011].

Može se reći da se BPR i WfM međusobno nadopunjavaju, jer samo automatizirati postojeći poslovni proces može kratkoročno donijeti određeni dobitak, no drastična poboljšanja nastaju tek zamjenom postojećih poslovnih procesa efikasnijim procesima. S druge strane, temeljem uspješne implementacije protoka poslova moguće je uvidjeti koje sve pomake treba napraviti u preoblikovanju određenog procesa. Primjena programske podrške s naglaskom na model procesa i prateće podatkovne strukture nije imala uspjeha sve do pojave sustava za upravljanje protokom poslova sredinom prošlog desetljeća koji su omogućili automatsko koordiniranje aktivnosti i resursa na osnovu formalno definiranog modela poslovnih procesa.

Povijest izgradnje poslovnih procesa, a samim time i oblikovanja protoka poslova kao zasebne discipline, može se promatrati s aspekta izgradnje informacijskih sustava te kroz aspekt promjena metoda upravljanja poslovanjem određene organizacije. Ova dva pogleda su međusobno isprepletena, jer međusobno utječu jedan na drugog. Gledano kroz napredak izgradnje informacijskih sustava, može se reći da se prvi začeci modeliranja protoka poslova vežu uz sedamdesete godine prošlog stoljeća uz rad Skipa Ellisa i Michaela Zismana na projektu „*Office Automation Systems*“ unutar Xeroxa [BPMResearch]. Usprkos tako ranim začecima, razvoj sustava za upravljanje poslovnim procesima nije intenzivnije napredovao do prošlog desetljeća. Nekoliko je razloga za to, počevši od nepostojanja računalnih mreža kao osnovnog tehničkog preduvjeta za upravljanje protokom poslova, do krutosti ranih programskih proizvoda koji su uplašili potencijalne korisnike, u to vrijeme nesvjesne koristi izdvajanja poslovnih procesa iz aplikacija (što upravo ukazuje na međusobnu povezanost ranije navedena dva aspekta). Tijekom godina, usporedno s razvojem tehnologije i uočavanjem nedostataka postojećih rješenja, nastojalo se iz programa izdvojiti što je moguće više generičkih elemenata za podršku poslovnim procesima.

Ukoliko bi se promatrala arhitektura generičkih rješenja kroz godine dobila bi se sljedeća dekompozicija [Aalst2004] i [Muehlen2004]:

- 1965 – 1975: Rastavljanje aplikacija: Tijekom ovog razdoblja informacijski sustavi sadržavali su odvojene aplikacije koje su imale odvojena spremišta podataka i različite strukture podataka. Razmjena podataka nije bila direktno omogućena, niti je bilo moguće dobiti sumarne vrijednosti koristeći različite izvore podataka. Korisnici su mogli imati različita imena i uloge u svakoj od aplikacija. Aplikacije nisu imale korisničko sučelje ili je ono bilo zasebno razvijano za svaku aplikaciju.
- 1975 – 1985: Upravljanje bazama podataka: Ovo razdoblje obilježava nastanak sustava za upravljanje bazama podataka. Ideja prisutna u ovom razdoblju je izdvajanje upravljanja podacima iz aplikacija pa ga se može nazvati podatkovno orijentirani pristup razvoju sustava. Postojanje jedinstvene baze podataka omogućava jednostavno dijeljenje podataka između aplikacija.
- 1985 – 1995: Upravljanje korisničkim sučeljem: Dotadašnja organizacija izrade sučelja pokazala se prespora i kao takva usko grlo u razvoju softvera, ali i neprikladna zbog potrebe za standardizacijom sučelja. Pojavljuju se prvi sustavi za upravljanje korisničkim sučeljem što omogućava brzi, standardizirani razvoj sučelja.
- 1995 – sadašnjost: Upravljanje protokom poslova: Nakon što su upravljanje bazama podataka i upravljanje korisničkim sučeljem međusobno odvojeni, nova razina unaprjeđenja razvoja sustava je bilo izdvajanje poslovne logike. Izoliranjem poslovnih postupaka od same aplikacije razvoj sustava je ubrzan, a ujedno je olakšano i njegovo održavanje.



Slika 1. Razvoj BPR-a [Sharp2008]

Gledano sa stajališta načina upravljanja poslovanjem, orijentacija na poslovne procese započela je u osamdesetim godinama kada se sve više uviđa da se u pretjeranim specijalizacijama gubi pregled nad cjelinom poslovnog procesa. Slika 1 grafički prikazuje razvoj BPR-a u protekla tri desetljeća. Prema [Sharp2008] napredak BPR-a počinje 1990. s [Hammer1990] no nakon početnog entuzijazma dolazi do određenog praznog hoda i razočaranja u BPR. Umjesto za preoblikovanje i poboljšanje procesa, BPR postaje popularna fraza korištena za razne besmislene primjene[1]. Istovremeno primat u trendu dobivaju ERP sustavi i razni oblici elektroničkog poslovanja. No, nakon načela „stvori više s više“ (novaca, resursa…) i zagušenja takozvane dot-com industrije, ponovo se stavlja naglasak na proces kako bi se „napravilo više s manje“ (novaca, resursa), što predstavlja novi uzlet BPR-a i upravljanja poslovnim procesima (eng. *Business Process Management*, BPM), objedinjenih zajedničkom skraćenicom BPx.

Napredak BPx-a i pojava procesno orijentiranog modeliranja sustava uzrokovali su i napredak sustava za upravljanje protokom poslova te njihovu brzu ekspanziju. U [Aalst2004] navodi se da je već 2000. godine postojalo više od 30 takvih sustava (iako mnogi od njih nisu bili u upotrebi) a s rastom sustava otvorenog koda ta lista postaje još veća[2]. Usprkos nastojanjima WfMC-a da se standardi ujednače i da se osigura međusobna povezanost sustava, različiti pristupi izradi protoka poslova i različite poslovne politike uzrokovale su postojanje nekoliko različitih standarda. Ideja da se definiraju uobičajeni obrasci protoka poslova dana je u [Kiepuszewski2003] te je također provedena analiza podržanosti navedenih obrazaca za određeni broj standarda i sustava za upravljanje protokom poslova [WfPat-Eval]. Načinjena analiza je pokazala da većina promatranih sustava i standarda podržava osnovni skup obrazaca, što još više potiče razvoj novih sustava za upravljanje protokom poslova [Milašinović2006].

Odabir sustava za upravljanje protokom poslova nije jednostavan zadatak i prema [Aalst2004] odabir bi trebao započeti popisom zahtjeva koje je jednostavno provjeriti, posebno onih vezanih uz integraciju uz postojeću programsku podršku. Razvojni okvir *Windows Workflow Foundation* (WF) odabran za praktičnu ugradnju modela u ovom radu sadrži gotovu funkcionalnost za jednostavnu izradu i izvršavanje protoka poslova. Prednost WF-a manifestira se posebno u situacijama kad se protok poslova veže uz određenu aplikaciju te kada je potrebna interakcija s korisnikom. Iako je sličan način upravljanja protokom poslova već bio ugrađivan u sustave za koordiniranje poslovnih procesa (primjerice [BizTalk]), ugradnja sustava za upravljanje protokom poslova u aplikacijski okvir *.NET Framework* 3.0 [WikiFramework], omogućila je automatizaciju jednostavnijih poslovnih procesa bez potrebe za zasebnim sustavima[3].

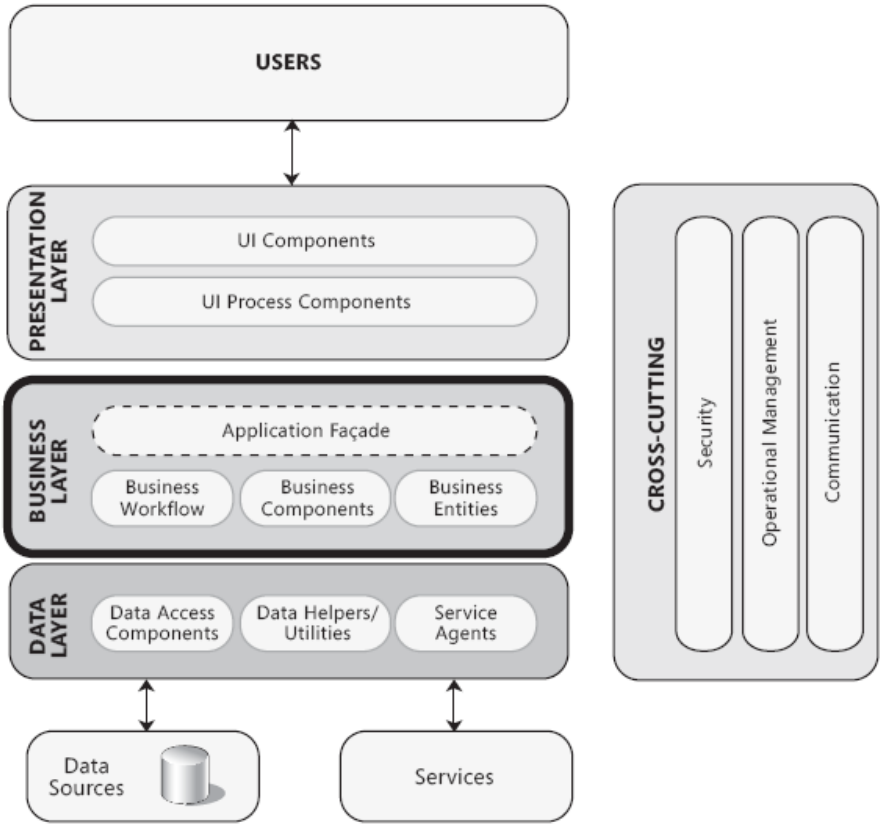
[1] U [Sharp2008] autori iznose nekoliko krivih upotreba termina preoblikovanje (reinženjerstvo), primjerice: „Podrška korisnicima preoblikovana je otpuštanjem 30% osoblja" i sličnim frazama koje nemaju veze s preoblikovanjem procesa.

[2] Primjerice, u [JavaWf] je naveden popis samo onih sustava za upravljanje protokom poslova temeljenih na otvorenom kodu i pisanih u Javi, a cjelokupna lista bi zasigurno bila puno veća.

[3] Primjer prikladnosti korištenja WF-a naspram *Biztalka* (i obrnuto) dan je u [Evdemon2006] i [Carbajal2006].

2.2 Višeslojne arhitekture i upravljanje protokom poslova

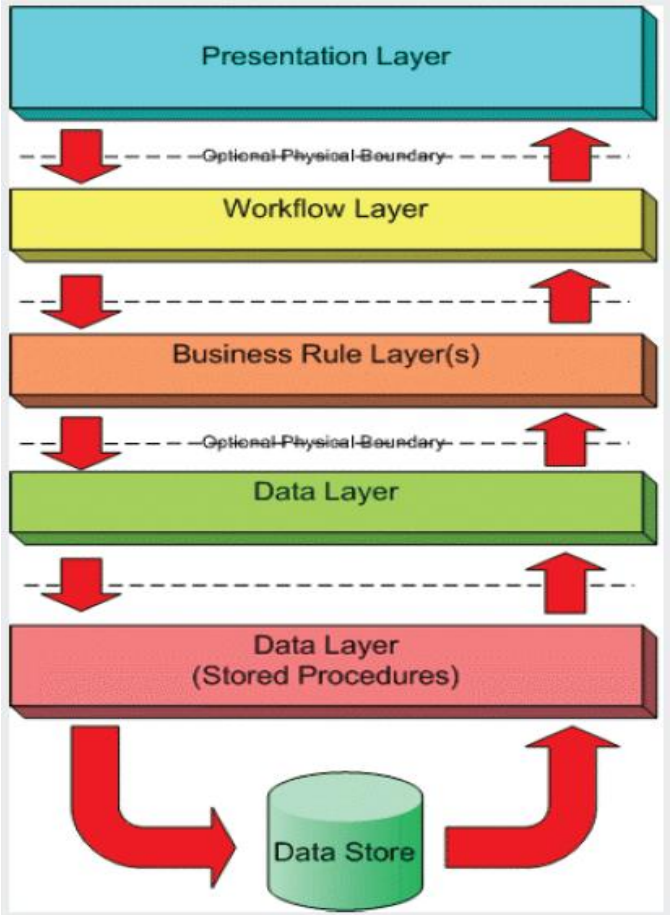
Smještaj upravljanja protoka poslova ovisi o varijanti odabrane višeslojne arhitekture. Prema [MsAppArch] uobičajena je klasična troslojna arhitektura s prezentacijskim, poslovnim i podatkovnim slojem u kojoj se protok poslova nalazi unutar poslovnog sloja (slika 2). U pojedinim izvorima naziv sloja protoka poslova nosi drugačiji naziv, primjerice *Interface Control* u [Lhotka2008].



Slika 2. Podjela poslovnog sloja na podkomponente unutar logičke arhitekture višeslojnog sustava [MsAppArch7]

Nakon što korisničko sučelje prikupi podatke od korisnika, aplikacija koristi prikupljene podatke u određenom poslovnom procesu koji može sadržavati više koraka izvedenih u određenom redoslijedu i po određenim pravilima.

Zastupljenost protoka poslova u poslovnom sloju ovisi o složenosti poslovnih procesa koji se modeliraju, iako postoji mogućnost da se cijeli poslovni sloj izvede pomoću modela protoka poslova [AlZabir2007] čime se postiže preglednost i smanjuje potreba za dokumentacijom uslijed jasnog grafičkog prikaza.

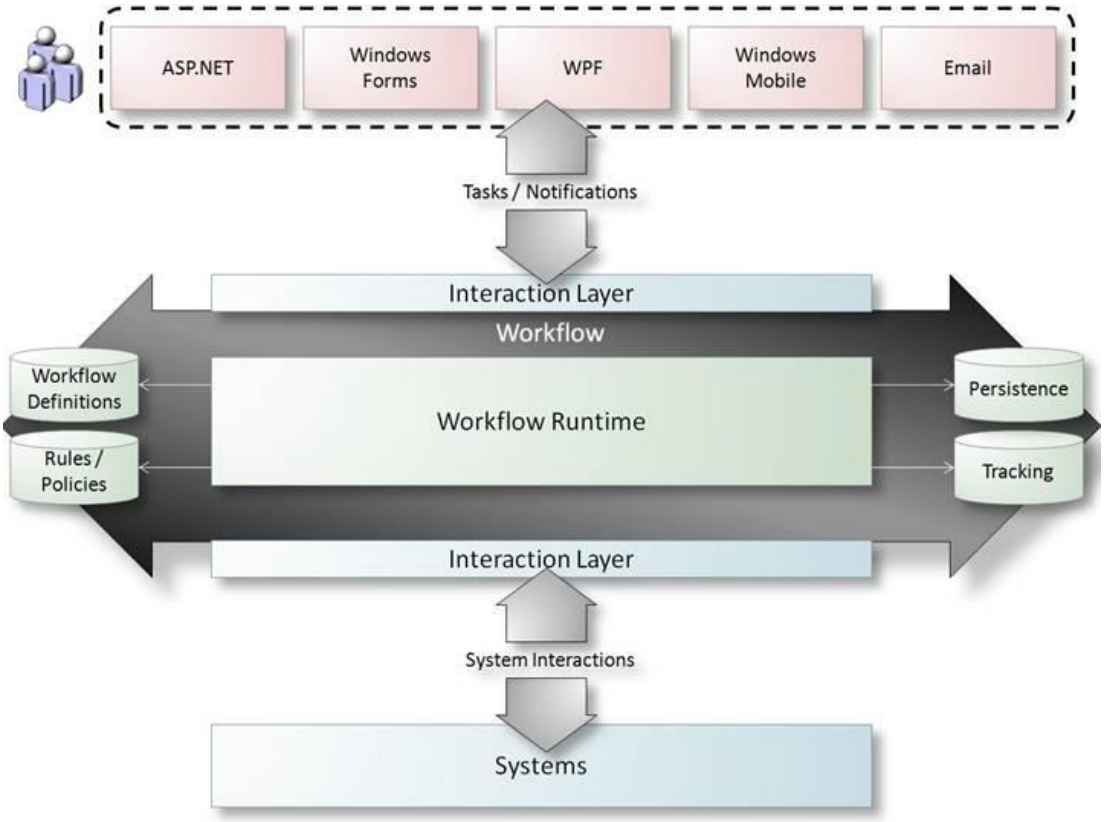


Slika 3. Peteroslojna logička arhitektura [Hyatt]

Za razliku od modela troslojne arhitekture, u peteroslojnoj arhitekturi, upravljanje protokom poslova nije samo dio poslovnog sloja, već je izdvojeno u zasebni sloj između prezentacijskog i poslovnog sloja [Fertalj2009], [Hyatt], što je prikazano na slici 3. Iako je i u troslojnoj arhitekturi unutar poslovnog sloja napravljena distinkcija između protoka poslova i poslovnih entiteta i ostalih komponenti, odvajanjem u zasebni sloj ta podjela je još jasnije istaknuta. Svako miješanje među slojevima može otežati održavanje i ponovnu iskoristivost pojedinog sloja [Lhotka2008].

2.3 WF kao primjer iskoristivosti modela protoka poslova u različitim vrstama aplikacija

Bez obzira na varijantu uslojavanja, upravljanje protokom poslova sastoji se od skupa kojeg čine okruženje (eng. *Workflow Runtime*), u kojem se izvršavaju pojedine instance protoka poslova, i skup pripadajućih servisa za interakciju sa susjednim slojevima (slika 4). Osim jednostavnosti izrade modela protoka poslova kroz WF i lakoće ugradnje u aplikacije, upravo komunikacijske mogućnosti i ugrađeni servisi, poput servisa za praćenje trenutnih instanci i servisa za postojanost podataka (eng. *persistence*) daju WF-u dodatni značaj.



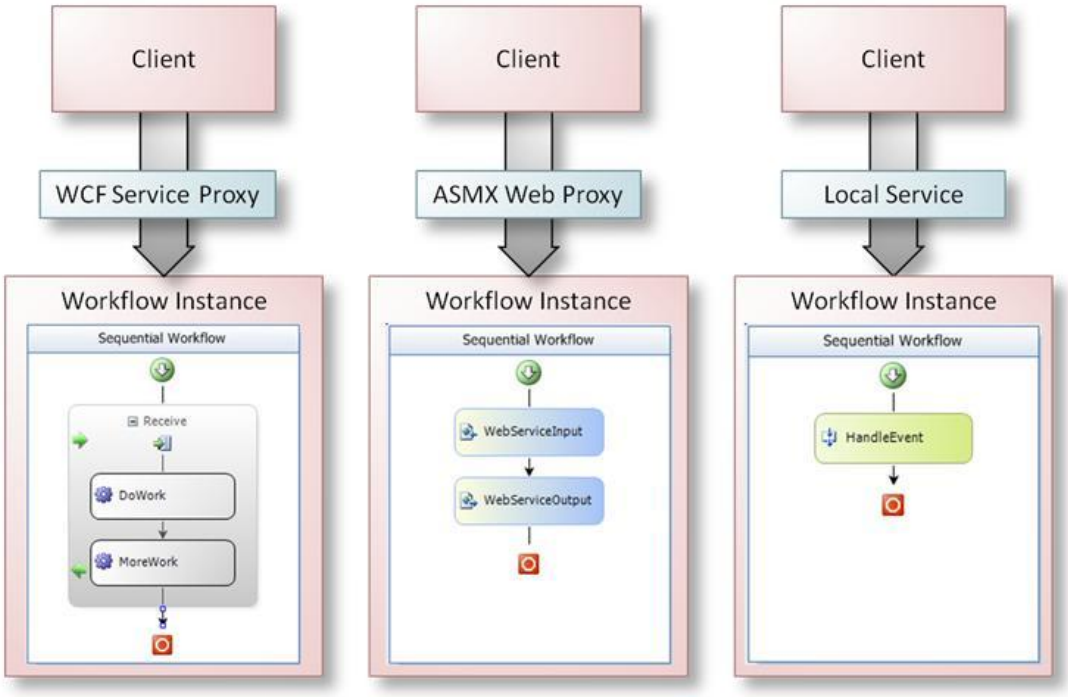
Slika 4. Windows Workflow Foundation kao srednji sloj u troslojnoj aplikaciji

Servis za postojanost omogućava da instanca protoka poslova koja čeka na neki događaj bude pospremljena na medij trajne pohrane (u datoteku ili bazu podataka) zajedno sa svojim stanjem, kako bi po pojavi događaja mogla nastaviti sa svojim izvršavanjem. Ova funkcionalnost je bitna jer pojedina instanca protoka poslova može dulje trajati (satima, danima…). Na ovaj način dolazi do očuvanja memorijskih resursa i mogućnosti da životni vijek pojedine instance protoka poslova bude dulji od perioda zaustavljanja i ponovnog pokretanja poslužitelja. Štoviše, moguće je jednom pohranjenu instancu protoka poslova nastaviti koristiti iz neke druge aplikacije.

2.3.1 Ostvarivanje komunikacija sa slojem protoka poslova

Osim u jednostavnim slučajevima protoka poslova, u kojima je eventualno potrebno samo poslati početne argumente za kreiranje nove instance protoka poslova i dobiti povratnu vrijednost nakon završetka konkretne instance protoka poslova, važna stavka u izradi modela protoka poslova je modeliranje međusobne komunikacije između sloja za upravljanje protokom poslova i njegovog domaćina (eng. *workflow host*), odnosno između sloja za upravljanje protokom poslova i vanjskih sustava.

Slika 4 ukazuje na dvosmjernu komunikaciju koju WF ostvaruje prema susjednim slojevima, a načini komunikacije prema instanci protoka poslova, odnosno iz instance protoka poslova prema susjednim slojevima ili vanjskim sustavima prikazani su na slici 5, odnosno na slici 6.



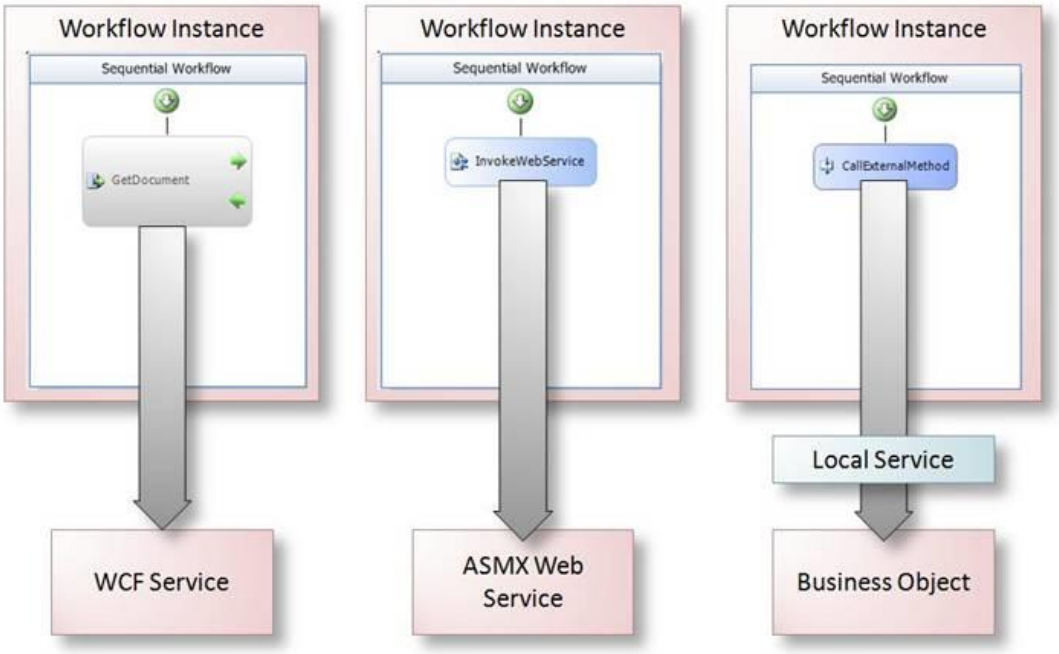
Slika 5. Komunikacija klijenta s instancom protoka poslova [WFScenario]

Iako je moguće ostvariti komunikaciju s vanjskim sustavima korištenjem lokalnih servisa i poslovnih objekata vezanih uz lokalne servise takva komunikacija primarno je namijenjena jednostavnoj komunikaciji između instance protoka poslova i njenog domaćina. Ograničenje ovog načina komunikacije sastoji se u tome da kreator nekog događaja i komponenta koja obrađuje taj događaj moraju istovremeno biti dio iste aplikacije, čime se otežava razdvajanje sloja za upravljanje protokom poslova. Komunikacija putem web servisa omogućuje izdvajanje protoka poslova iz domaćina, pa čak i smještaj na drugom računalu, što predstavlja proširenje mogućnosti u odnosu na komunikaciju putem lokalnih servisa.

Razvojni okvir *Windows Communication Foundation* (WCF) kao programski model predstavlja nadgradnju u odnosu na komunikaciju putem web servisa[4], te omogućuje lakšu i jasniju izradu komunikacijskih modela [WCF]. WCF servis sastoji se od tri dijela[5]: adrese servisa, načina povezivanja i ugovora (eng. *contract*) kojeg čine skup razreda i sučelja s definiranim postupcima koji se koriste u komunikaciji. Adresa i povezivanje definiraju se u konfiguracijskim postavkama pojedine strane u komunikaciji, dok se unutar samog koda koristi prethodno definirani ugovor.

[4] Model komunikacije pomoću WCF servisa pojavio se s verzijom 3.5 te nije postajo u verziji WF3.0.
[5] Izrada WCF servisa se često spominje kao ABC princip, gdje je ABC skraćenica nastala od početnih slova engleskih naziva triju dijelova WCF servisa: address, binding, contract

Ugovor definira skup razreda i postupaka koji se koriste u komunikaciji te ga moraju imati obje strane u komunikaciji. Promjena adrese na kojoj se nalazi WCF servis i promjena načina povezivanja nemaju utjecaja na ugovor niti predstavljaju ograničenje za njegovo definiranje.

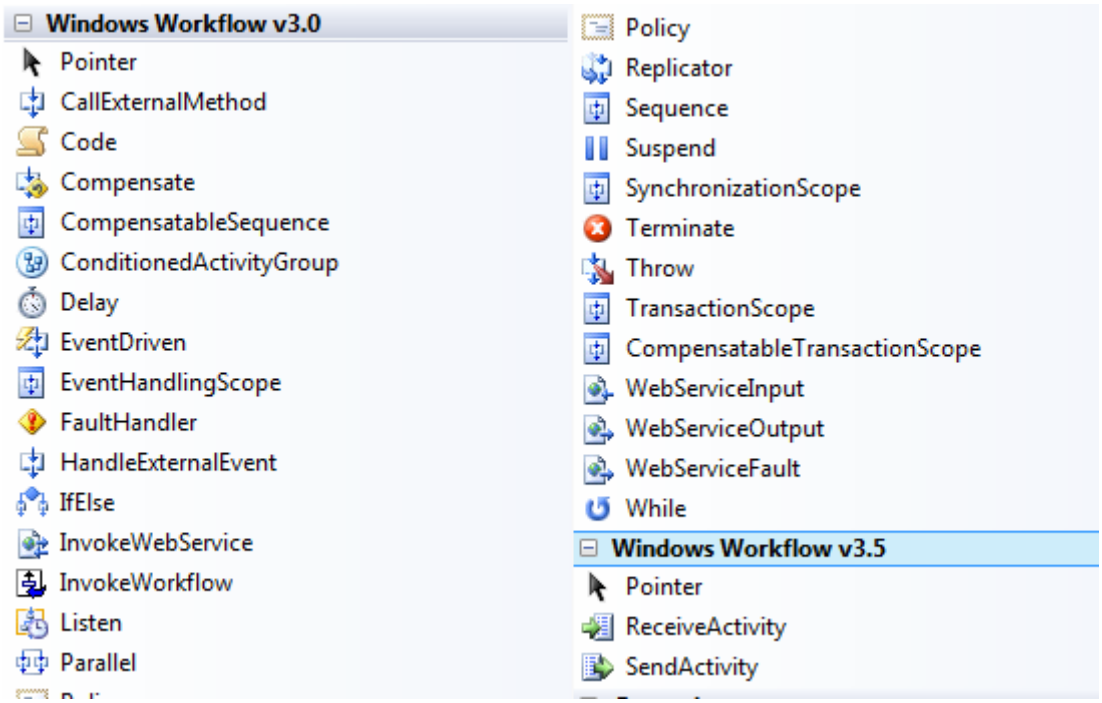


Slika 6. Komunikacija prema vanjskim sustavima ili prema susjednim slojevima inicirana od instance protoka poslova [WFScenario]

Za rad s WCF servisima unutar WF-a zadužene su aktivnosti *SendActivity* i *ReceiveActivity*. *SendActivity* poziva određeni postupak definiran ugovorom, čime se poziva određeni WCF servis na prethodno konfiguriranoj adresi. Komunikaciju u drugom smjeru moguće je ostvariti ako je protok poslova izložen kao WCF servis. U tom slučaju aktivnost *ReceiveActivity* predstavlja implementaciju određenog postupka definiranog ugovorom i čeka na poziv izvana. Svaka aktivnost *ReceiveActivity* vezana je uz određeni postupak iz ugovora. U slučaju da je više takvih aktivnosti vezano za isti postupak, međusobno razlikovanje se ostvaruje korištenjem oznake konteksta (eng. *ContextToken*).

Posebnost WCF servisa u komunikaciji s instancama protoka poslova je mogućnost ponovnog kreiranja konteksta za instancu protoka poslova čije je zatečeno stanje pomoću servisa za postojanost u nekom trenutku bilo pohranjeno. Navedena funkcionalnost omogućava, po pozivu WCF servisa, nastavak instance protoka poslova iz zatečenog stanja i poziv unaprijed određene aktivnosti na osnovu poznavanja identifikatora pokrenute instance (parametar *instanceId*) i identifikatora oznake konteksta pojedine aktivnosti *ReceiveActivity* (parametar *conversationId*) što je iskorišteno pri izgradnji omotača protoka poslova u poglavlju 5.4.

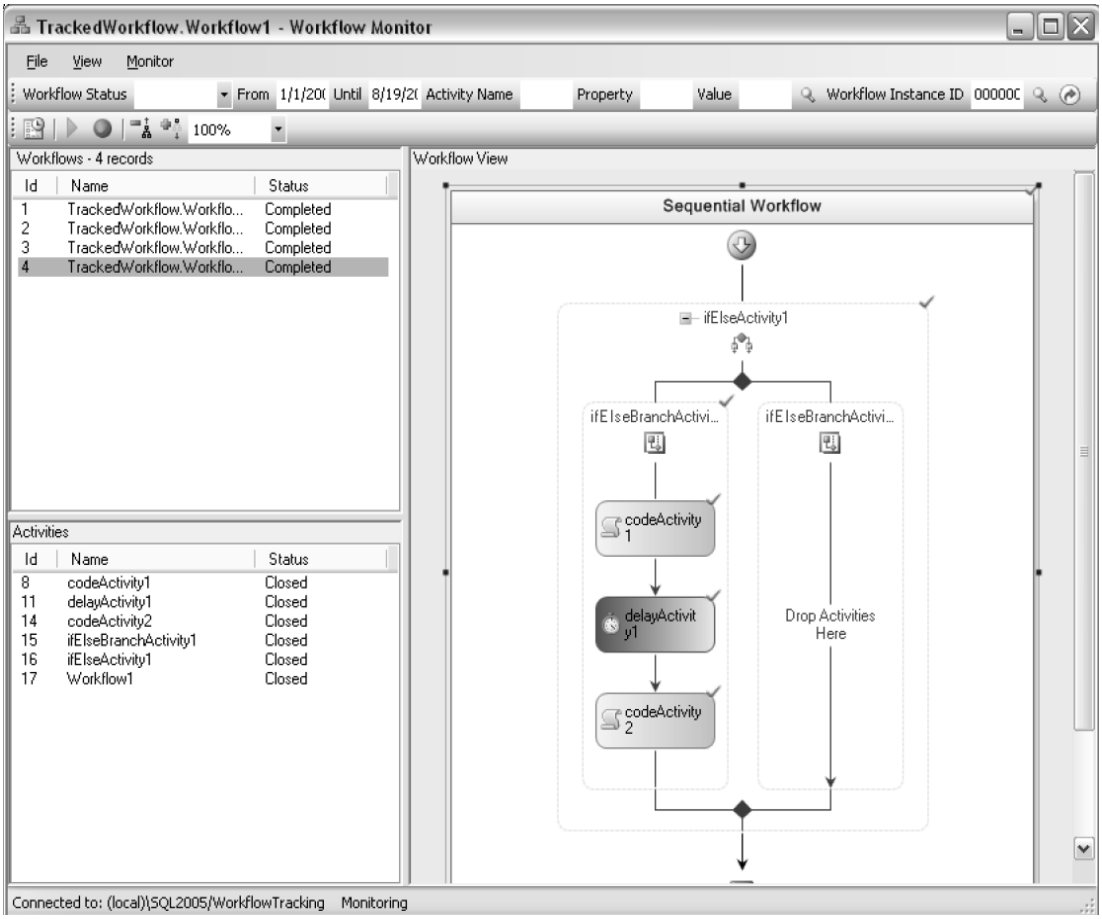
Osim navedenih aktivnosti *SendActivity* i *ReceiveActivity*, osnovni skup kontrola za izradu WF modela sastoji se od dvadesetak kontrola (slika 7) te se može proširiti vlastitim kontrolama. U terminima WF-a kontrole se nazivaju aktivnostima. Detaljni opis svih aktivnosti može se pronaći na [MsdnWFAct], a u poglavlju 5 dan je kratki opis za svaku od korištenih aktivnosti.



Slika 7. Prikaz osnovnih kontrola (aktivnosti) unutar WF-a u .NET Frameworku 3.5

2.3.2 Praćenje aktivnosti i dinamičke promjene aktivnih modela

WF među ugrađenim servisima sadrži servis za praćenje rada instanci protoka poslova. Sam postupak za aktiviranje ovog servisa svodi se na jednokratnu instalaciju baze podataka za praćenje aktivnosti i na uključivanje servisa u konfiguracijskim parametrima za pojedinu instancu protoka poslova. U odnosu na neke vlastite metode praćenja rada, prednost ovog servisa je mogućnost grafičkog prikaza stanja neke instance protoka poslova uz grafički i tekstualni prikaz stanja pojedine aktivnosti u instanci. Slika 8 prikazuje primjer aplikacije izgrađene nad servisom za praćenje rada instanci protoka poslova. Implementacijom postupaka za dohvat podataka o trenutnom stanju neke instance protoka poslova [Scribner2007], grafički prikaz stanja instance protoka poslova, kao i odgovarajući niz detaljnijih tekstualnih podataka bit će dostupan vanjskom sustavu na za to predviđenim postupcima iz ugovora definiranog u poglavlju 6.



Slika 8. Primjer aplikacije za praćenje rada instance protoka poslova [Scribner2007]

Osim mogućnosti grafičkog i tekstualnog prikaza potrebno je omogućiti dinamičke promjene aktivne instance protoka poslova. Prilikom promjene aktivne instance protoka poslova moguće je mijenjati ili brisati aktivnosti koje nisu započele te dodavati nove aktivnosti unutar aktivnih sekvenci i onih sekvenci koje još nisu započele. Također, moguće je dodavati nove grane u paralelne aktivnosti. Glavno ograničenje svodi se na nemogućnost promjene aktivnosti koja se trenutno izvršava, odnosno aktivnosti koja je u statusu čekanja na neki događaj. Postupak promjena aktivne instance opisan je u [Bukovics2007].

Potreba za promjenom aktivne instance javit će se u poglavlju 5.3 prilikom pretvorbe skupa mogućih prethodnika u konačni model i otkazivanja i brisanja elemenata kada su uvjeti iz nekog skupa mogućih prethodnika zadovoljeni ili prilikom nadogradnje modela uslijed neispunjenja nekog od pretpostavljenih uvjeta uslijed vremenskog isteka roka ili nekog drugog razloga.

2.4 Upravljanje protokom poslova i sustavi za udaljeno učenje

Tradicionalno, protok poslova je najčešće vezan za poslovne sustave u kojima je priroda poslovanja procesno orijentirana, primjerice izrada putnog naloga, zapošljavanje djelatnika, zahtjev za kredit i slično, no može se primijeniti i na sustave za udaljeno učenje[6].

Prema [Sadiq2002] tipičan sustav za udaljeno učenje sastoji se od tri glavne komponente: materijala za učenje, alata za kolaboraciju te alata za koordinaciju i kontrolu procesa učenja. Velika pažnja se pridaje pravilnom oblikovanju nastavnih materijala, kako bi se, dobrim grafičkim dizajnom i interaktivnošću, takvi materijali što više približili korisnicima. Međutim, problem objedinjavanja i razmjene nastavnih materijala pohranjenih u različite repozitorije objekata učenja (eng. *learning object repository*) je i dalje prisutan [Fertalj2010].

Kontrola procesa učenja sastoji se od raznovrsnih načina provjere znanja, kako bi se potaknuo aktivni pristup učenju [Botički2008], ali i usmjeravanja postupka učenja. Za razliku od tradicionalnog učenja, koje se izvodi pod nadzorom nastavnika, sustavi za udaljeno učenje omogućavaju veću fleksibilnost procesa učenja. Prema [Sadiq2002], često pogrešna interpretacija fleksibilnosti je izostavljanje određenog oblika koordinacije i kontrole procesa učenja. U [Lin2001] među glavnim nedostacima sustava za udaljeno učenje identificirani su nemogućnost praćenja aktivnosti napretka pojedine osobe i orijentacija na pojedine zadatke, a ne cjelokupni proces učenja. Važnost sagledavanja cjelokupnog konteksta učenja posebno dolazi do izražaja kada se proces učenja treba individualizirati, to jest kada treba omogućiti prilagodbu procesa učenja pojedinoj osobi sukladno predznanju i pokazanom napretku pojedinca [Vantroys2001]. U tom slučaju izuzetno je važno odrediti kada, kome i kojim redom prezentirati određene materijale. Takav sustav sazrijevat će s vremenom, dodavanjem novih funkcionalnosti i nastavnih materijala te je prirodno za očekivati da se i redoslijed prikaza materijala konstantno mijenja. Kako bi se povećala njegova kvaliteta i mogućnosti održavanja, nužno je odvojiti prezentacijsku i poslovnu logiku (što i kako prezentirati, kako dohvatiti podatke, izvedbu provjera znanja) od protoka poslova kojim se opisuje kada i kojim redoslijedom se pojedini materijali prikazuju.

Koncept kako bi se proces učenja trebao opisati kao poslovni proces dan je u [Avgeriou2003], ali ne ulazi u područje izgradnje protoka poslova. U [Fong2003] navedeni su principi kojih bi se trebalo pridržavati pri izgradnji modela protoka poslova. Kao što se sustavi za udaljeno učenje (ili njegovi dijelovi za dohvat nastavnih materijala) mogu oblikovati kao skup web servisa [Vossen2003], [Fertalj2010], tako je i sustav za upravljanje protokom poslova poželjno izložiti kroz web servise. Takav pristup omogućava odvajanje sloja za upravljanje protoka poslova i neovisnost o prezentacijskom sloju čime se proširuje skup podržanih klijentskih platformi [Vantroys2002].

Primjer ugradnje modela protoka poslova opisan je u [Lin2001] i [Lin2002] na način da su nastavnici imali mogućnost modeliranja redoslijeda prikaza pojedinih nastavnih materijala. Modeliranje protoka poslova vršilo se na razini pojedinog predmeta, pri čemu se za pojedini

[6] Pod terminom udaljeno učenje, podrazumijeva se bilo koje neizravno, računalom podržano učenje.

predmet moglo definirati više predložaka modela protoka poslova čime bi studenti mogli odabrati njima poželjan redoslijed. Nedostatci sustava očituju se u potrebi da se za različite redoslijede predlošci moraju unaprijed izraditi od strane nastavnika te da su modeli definirani na razini pojedinog predmeta uglavnom podržavali jednostavne sljedove cjelina. Sustav opisan u [Vantroys2003] rješava problem potrebe da se personalizirani model protoka poslova izrađuju unaprijed od strane nastavnika, ali kao nedostatak se ističe nepostojanje provjere tako personaliziranih modela u odnosu na postavljene međusobne relacije među cjelinama.

Uz kritiku [Lin2001], sustav opisan u [Cesarini2004] nudi mogućnost modeliranja protoka poslova koristeći, osim uobičajenog slijeda niza cjelina, petlje i mogućnost odabira u interakciji s korisnikom. Nedostatak sustava je potreba da se model gradi odjednom bez personaliziranih modela protoka poslova te problemi s postojanosti instance protoka poslova i problem složenosti izgradnje interakcije s korisnikom.

Primjer korištenja modela protoka poslova i udaljenog učenja dan u [Raposo2004] podržava sljedove cjelina uz mogućnost postojanja petlje. Sustav se bazira na hijerarhijskoj razradi cjelina na manje dijelove čime je nastavnicima omogućeno naknadno mijenjanje redoslijeda aktivnosti u pojedinoj cjelini. Upravljanje protokom poslova bazirano je na vlastitom sustavu upravljanja protokom poslova uz operatore kojim se definiraju odnosi među cjelinama pri čemu neka cjelina može omogućiti, uzrokovati, blokirati ili odblokirati neku drugu cjelinu.

Veza između sustava za udaljeno učenje i sustava za upravljanje protokom poslova u [Yong2005] je iskorištena kao način za modeliranje međusobnih veza u postupku izrade nastavnih materijala i administracije sustava, a u [Xi2007] za opis postupka integracije nastavnih materijala iz različitih izvora, no ne ulaze u modeliranje redoslijeda među cjelinama. Prijedlog da se protok poslova poveže s nekim od postojećih sustava za udaljeno učenje dan je u [Rodriguez2009] za sustav Moodle [Moodle], bez opisa kako bi se protok poslova trebao vezati s pojedinim cjelinama.

U svim navedenim rješenjima pretpostavlja se da se cjelokupni model protoka poslova gradi odjednom i centralizirano. Nasuprot ovim praktičnim primjerima, [Scorm] kao jedan od standarda za izradu nastavnih materijala [Carnet], predlaže *SCORM Navigation and Sequencing* [ScormSeqNav] kao shemu za definiranje uobičajenih obrazaca protoka poslova unutar sustava za udaljeno učenje, međutim implementacijske probleme prepušta samim projektantima sustava za udaljeno učenje, što opet vodi ka izvedbi vlastitog sustava za upravljanje protokom poslova.

Nedostatci gore navedenih rješenja ukazuju na potrebu za izradom modela protoka poslova temeljenih na manjim, parcijalnim modelima, čije bi održavanje bilo jednostavno. Uz parcijalne modele, potrebno je izraditi i algoritme za verifikaciju i integraciju modela. Tako definiran sustav za upravljanje protokom poslova trebao bi biti dostupan putem web servisa, kako bi omogućio jednostavnu integraciju s aplikacijama koje imaju potrebu za takvim sustavom.

Za razliku od klasičnog učenja, sustav udaljenog učenja pogodniji je za automatizaciju, jer ne zahtijeva manipulaciju dodatnim ljudskim i materijalnim resursima, kao što su organizacija i koordinacija nastavnika i studenata, koordiniranje zauzetosti prostorija i slično, no s teorijskog stajališta ta dva sustava dijele zajednički proces učenja koji se može procesno oblikovati i automatizirati.

Općenito , korištenje parcijalnih modela ne mora nužno biti vezano samo za proces učenja, već za bilo koji proces u kojem je, uslijed složenosti sustava, pogodno modele izrađivati parcijalno te ih potom spojiti u integralni model. Primjer takve upotrebe je objedinjavanje UML dijagrama temeljem međusobne povezanosti pojedinih komponenti dijagrama [Hlaoui2009].

U kontekstu integracija parcijalnih modela mogu se i svrstati sljedeći radovi [Ye1997], [Mendling2006] i [Sun2006]. [Ye1997] se bavi minimizacijom preklapanja i objedinjavanjem istih dijelova grafa, ali je primjenjiv samo na digitalne krugove koji sadrže operatore AND i OR. [Mendling2006] reducira modele protoka poslova identificirajući semantički jednake elemente, ali ne rješava problem neuparenih paralelnih grananja i sinkronizacijskih točki [Milašinović2007]. [Sun2006] korištenjem Petrijevih mreža objedinjava modele protoka poslova tražeći međusobna preklapanja, što ga zbog ograničenja Petrijevih mreža [Aalst2002] čini nepogodnim za neke konstrukcije u modelu protoka poslova poput skupa mogućih prethodnika iz poglavlja 3.3.

3. Parcijalno definirani modeli protoka poslova

Model protoka poslova može se kreirati za više namjena, primjerice za studijski program u kojem su osnovne stavke kolegiji u tom programu, za sustav za udaljeno učenje gdje se vrši raspoređivanje modula znanja, za definiranje rasporeda cjelina za učenje unutar nekog modula, za definiranje odnosa među poslovnim procesima i slično. Spektar primjena je širok, ali mogu se identificirati određene zajedničke karakteristike na osnovu kojih se može definirati skup parcijalno definiranih modela protoka poslova koji integracijom tvore cjeloviti sustav za kojeg je potrebno provjeriti da li je ispravan.

3.1 Element

Definicija: Element je bilo koji sastavni dio modela protoka poslova, jednoznačno određen svojom šifrom unutar modela.

Element ne mora nužno biti atomarni dio modela protoka poslova, to jest takav da se ne može dalje rastavljati u sastavne dijelove, već može predstavljati i neki drugi model protoka poslova. Ukoliko se radi model protoka poslova za studijski program, tada element može biti pojedini kolegij. U sustavu za udaljeno učenje element može biti određeni modul znanja ili neka cjelina unutar modula. Neki konkretni primjeri značenja elementa dani su u poglavljima 7.1 i 7.2. Sa stanovišta izrade modela protoka poslova na osnovu parcijalnih modela, bitan je samo odnos među definiranim elementima, ne i njihova konkretna značenja.

3.2 Preduvjet

Definicija: Preduvjetom nekog elementa Y smatra se element X, ako za početak izvršavanja elementa Y element X mora biti završen i to prema prethodno definiranim uvjetima.

Definicijom se želi naglasiti da je uspješan dovršetak elementa X nužan uvjet, jer je element mogao biti prisilno dovršen istekom dopuštenog vremena i slično. U ovisnosti o pojedinoj situaciji, neispunjavanje uvjeta može voditi do ponavljanja elementa X ili do situacije u kojoj element Y nikad neće moći biti izvršen. Koja od situacija će se dogoditi ovisi o konkretnoj implementaciji elementa.[7] Prilikom definiranja modela za pretpostavku se uzima da će preduvjet biti ostvaren, a prilikom stvarne izvedbe u slučaju neostvarenja uvjeta dolazi do preslagivanja modela (poglavlje 2.3.2) pri čemu su u novi model već uključeni dotad završeni elementi.

[7] Za navedenu situaciju je moguće naći međusobno suprotne primjere ponašanja sustava u slučaju neostvarivanja preduvjeta kao što su:

- prijelaz na sljedeću razinu znanja uvjetovan je ponavljanjem određene cjeline sve dok se cjelina ne položi.
- pad predmeta tijekom studija povlači ili ponovni upis predmeta (pri čemu je broj ponavljanja ograničen) ili odabir nekog drugog predmeta, ako se nije radilo o obveznom predmetu.
- neuspješno rješavanje određenog testa uvjetuje drugačiju putanju učenja

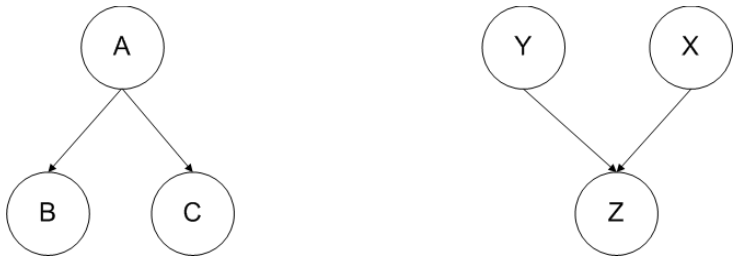
Iz definicije je jasno da je preduvjet tranzitivna relacija. Budući da se u algoritmima koji slijede ponekad treba razlikovati situacija da li se radi o direktnoj relaciji između dva elementa ili onoj ostvarenoj putem tranzitivnosti, uvodi se termin neposrednog preduvjeta.

Definicija: Element X je neposredni preduvjet elementa Z ako je X preduvjet elementa Z te ne postoji niti jedan element Y koji je preduvjet elementa Z takav da je X preduvjet elementa Y.

Model protoka poslova može se prikazati usmjerenim grafom $G(V, A)$, gdje je V skup vrhova, a A skup lukova grafa. Vrhovi usmjerenog grafa predstavljaju pojedine elemente iz modela protoka poslova, a lukovi usmjerenog grafa predstavljaju određenu vezu između pojedinih elemenata. Pojedini parcijalni model tada predstavlja podskup grafa G.

Na taj način, situacija u kojoj je element X preduvjet elementa Z može se prikazati pomoću usmjerenog grafa u kojem su vrhovi grafa X i Z spojeni lukom koji izlazi iz X i ulazi u Z. Formalno, može se reći da je nužan uvjet (ali ne i dovoljan zbog 3.2.3) da je X neposredni preduvjet od Z postojanje luka $a = (X, Z)$ u usmjerenom grafu $G(V, A)$, gdje je V skup vrhova grafa, pri čemu svaki vrh $X \in V$ predstavlja odgovarajući element X i A skup usmjerenih lukova grafa. Posljedica je da je X preduvjet od Z ako i samo ako u grafu postoji put od vrha X do vrha Z.

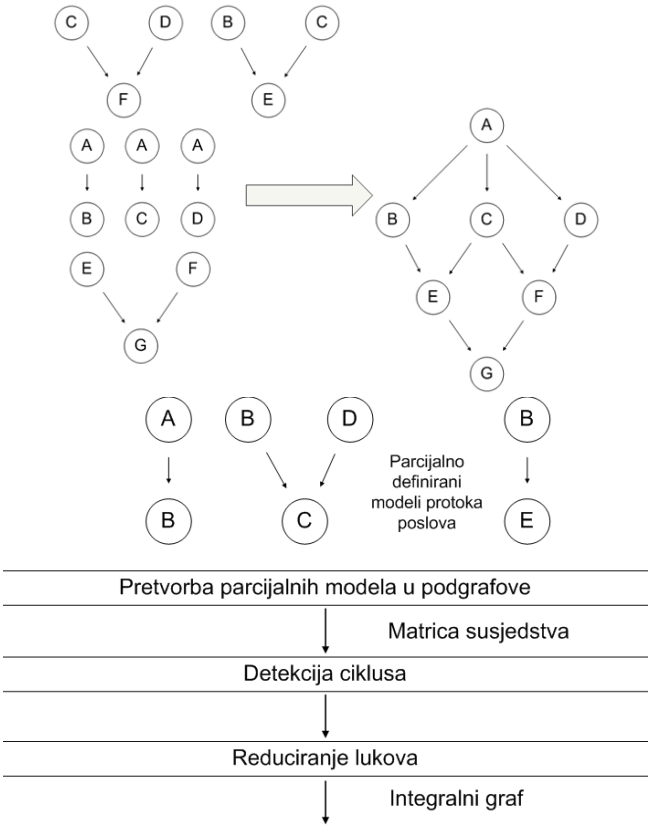
S obzirom da se radi o dodavanju vrhova i lukova u usmjereni graf, efektivno je svejedno da li će svi preduvjeti, to jest odgovarajući lukovi i vrhovi, biti definirani u jednom parcijalnom modelu ili će biti definirati u odvojenim parcijalnim modelima. Također, jednak je efekt da li se definira koji elementi su preduvjeti nekog elementa (slika 9, desna strana) ili kojim elementima je neki element preduvjet (slika 9, lijeva strana), jer će u konačnici u zajedničkom usmjerenom grafu situacija uvijek biti ista.



Slika 9. Element A kao preduvjet elemenata B i C te elementi X i Y kao preduvjeti elementa Z

Razumno je za očekivati da se pri definiciji nekog elementa znaju svi njegovi preduvjeti, jer je time i opisan sam element, pa će desni prikaz sa Slika 9 biti prirodniji način definiranja preduvjeta. Sljedbenike elemenata u trenutku njegovog definiranja nije potrebno znati, štoviše oni u tom trenutku ne moraju ni postojati. Primjerice, uvođenjem novog kolegija, njegovi preduvjeti bit će zavedeni u posebnom parcijalnom modelu, bez promjena svih dotad definiranih modela u kojima se nalaze njegovi preduvjeti.

3.2.1 Objedinjavanje parcijalno definiranih preduvjeta



Slika 10. Spajanje parcijalnih modela za preduvjete u zajednički usmjereni graf

Prikažu li se parcijalni modeli pomoću usmjerenih grafova, tada se objedinjavanje parcijalno definiranih modela, kojima se opisuju preduvjeti pojedinih elemenata, svodi na izradu zajedničkog, integralnog usmjerenog grafa. Slika 10 prikazuje postupak objedinjavanja parcijalno definiranih modela. Vizualno gledano, stvaranje zajedničkog grafa vrši se preklapanjem istih vrhova. Za razliku od [Mendling2006] ili [Sun2006], u ovom slučaju vrhove grafa čine samo elementi, pa je brz i efikasan način da se elementi numeriraju redom od 1 do N (pri čemu se N povećava pojavom dotad nekorištenog elementa), čime se stvara matrica susjedstva. Već u ovom koraku moguće je otkriti logičke pogreške u modelu, poput situacije u kojoj su dva elementa jedan drugome preduvjeti. Nakon spajanja u zajednički (integralni) graf algoritam obavlja provjeru postojanja ciklusa, nakon čega se vrši reduciranje lukova. Iz tako dobivenog integralnog grafa, skup elemenata se može, po željama krajnjeg korisnika, dodatno suziti čime će odabrani elementi, zajedano sa svojim preduvjetima, biti dodani u konkretni WF model.

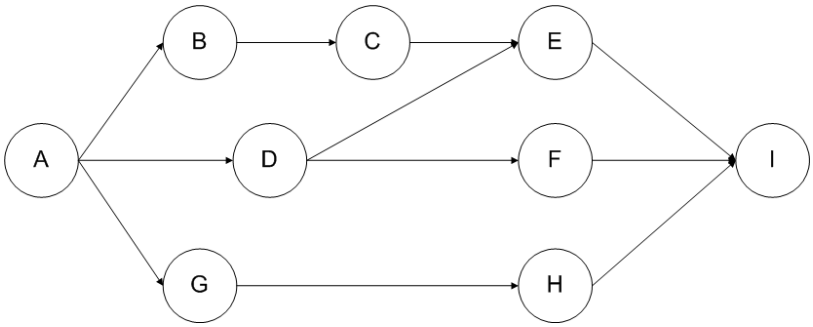
3.2.2 Provjera postojanja ciklusa

Nakon spajanja parcijalno definiranih modela u integralni graf, potrebno je provjeriti da li se u grafu nalazi ciklus. Postojanje ciklusa bi ukazalo da model nije dobro definiran, jer bi zbog tranzitivnosti relacije preduvjeta značilo da je neki element preduvjet sam sebi. Prikazom veza među elementima u obliku matrice susjedstva, gdje se u retku *i* i stupcu *j*

matrice susjedstva nalazi vrijednost 1 ako luk izlazi iz elementa *i* te ulazi u element *j*, a 0 inače, vrlo brzo se može uočiti trivijalna situacija u kojoj je neki element sam sebi preduvjet (vrijednost 1 na dijagonali matrice susjedstva), odnosno situacija u kojoj postoji ciklus duljine 2, to jest situacija u kojoj je istovremeno i na mjestu (*i,j*) i na mjestu (*j,i*) vrijednost 1.

Sljedeća trivijalna situacija je ona u kojoj ne postoji niti jedan vrh s ulaznim stupnjem 0, što bi odgovaralo situaciji u kojoj ne postoji niti jedan stupac u matrici koji sadrži samo nule. Takva situacija povlači da je ili cijeli graf jedan ciklus ili skup međusobno nepovezanih ciklusa.

Sve ostale, netrivijalne situacije, u kojima eventualno postoji ciklus, potrebno je pronaći nekom od metoda za traženje ciklusa. Algoritmi za traženje ciklusa u grafu mogu se razlikovati u slučaju da je graf povezan ili se mogu primijeniti i na grafove koji nisu povezani, no, općenito, može se reći da se ciklus može pronaći sa složenošću O(n+m), gdje je n broj vrhova, a m broj lukova korištenjem neke od sljedećih dviju metoda: postupnim brisanjem vrhova ulaznog stupnja nula [Knuth1973] čime se ili brišu svi vrhovi ili preostaju oni koji tvore ciklus ili korištenjem pretrage u dubinu (eng. *depth-first search*) i označavanjem posjećenih vrhova [Tarjan1972]. Traženje svih ciklusa složeniji je problem [Johnson1975]. Očekuju li se kratki ciklusi, pogodna metoda iznesena je u [Nishizawa1996] gdje se iz početne matrice susjedstva stvaraju nove matrice koje predstavljaju povezanost vrhova u određenom broju koraka (matrica susjedstva predstavlja povezanost u prvom koraku).



Slika 11. Neki od mogućih topoloških poredaka za graf na slici: ABDGCEFHI, ABCGHDEFI, ADFGHBCEI, ...

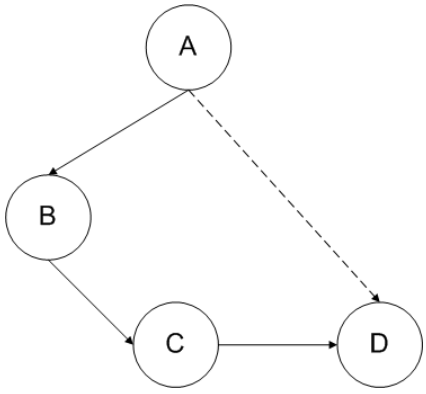
Traženje ciklusa u grafu povezano je i s pojmom topološkog sortiranja, odnosno topološkog poretka vrhova usmjerenog grafa. Prema [Weisstein] topološki poredak je permutacija vrhova grafa na način da postojanje luka (*i,j)* uzrokuje da se vrh *i* u poretku pojavljuje prije vrha *j*, što se može zapisati kao *i<j*, pri čemu obrnuta tvrdnja ne mora vrijediti. Topološko sortiranje primjenjuje se samo na usmjerene grafove koji ne sadrže ciklus. Relacija poretka je relacija totalnog uređaja i kao posljedica definicije može se reći da ako je *i<j*, tada ne postoji put od *j* do *i*. Permutacija vrhova nastala topološkim sortiranjem ne mora biti jedinstvena te ovisi o algoritmu traženja topološkog poretka. Slika 11 ilustrira takav jedan primjer u kojem je moguće više različitih topoloških poredaka.

Algoritmi za pronalaske topološkog poretka ili svih mogućih topoloških poredaka [Knuth1974] najčešće su slični algoritmima za traženje ciklusa. Činjenica da topološki poredak postoji u usmjerenom grafu ako i samo ako graf ne sadrži ciklus iskorištena je u [Haeupler2008] kako bi se prilikom dodavanja pojedinog luka u graf istovremeno odvijala provjera postojanja ciklusa i održavanje topološkog poretka. Primjerice, dodavanjem luka (v, w) u slučaju kad je u trenutnom topološkom poretku vrijedilo $v < w$, topološki poredak i nakon dodavanja luka (v, w) ostaje ispravan. U tom slučaju nema potrebe za traženjem postoji li put od w do v, jer postojeći topološki poredak osigurava da takav put ne postoji. U slučaju da je $w < v$ potrebno je provjeriti može li se iz vrha w doći do vrha v. Ako takav put postoji, to ukazuje da je dodavanje novog luka upravo stvorilo ciklus. U protivnom potrebno je obnoviti topološki poredak što prema [Haeupler2008] predstavlja značajno poboljšanje u odnosu na ponovno generiranje topološkog poretka.

Ovakav pristup pogodan je prilikom rada s parcijalnim modelima, želi li se odmah pri dodavanju pojedinog modela prijaviti pogreška ukoliko postoji ciklus. Nakon dodavanja svih parcijalnih modela algoritam daje jedan od mogućih topoloških poredaka. Topološki poredak bit će iskorišten u algoritmima za numeriranje vrhova pri pretvorbi zajedničkog grafa u WF model, pri radu s vremenskom komponentom modela te pri reduciranju skupova mogućih prethodnika.

3.2.3 Reduciranje lukova

Nakon što je zadovoljen uvjet da ne postoji ciklus u grafu, može se prijeći na korak reduciranja lukova. Budući da je relacija preduvjeta tranzitivna, moguće su situacije u grafu u kojem su pojedini lukovi suvišni te je potrebno ostaviti samo one lukove kojima se opisuje da je neki vrh neposredni preduvjet nekog drugog vrha. Slika 12 je primjer takvog slučaja u kojem je element A preduvjet elemenata B i D, B je preduvjet elementa C, a C je preduvjet elementa D, što sugerira da se direktni luk između A i D može ukloniti, jer ne donosi nikakvu dodatnu informaciju. Štoviše, izbacivanjem svih takvih lukova, što je nužan uvjet za algoritam pretvorbe u WF model iz poglavlja 5.2, vrijedit će tvrdnja da je neki element X neposredni preduvjet elementa Y ako i samo ako u zajedničkom grafu postoji luk iz X prema Y.



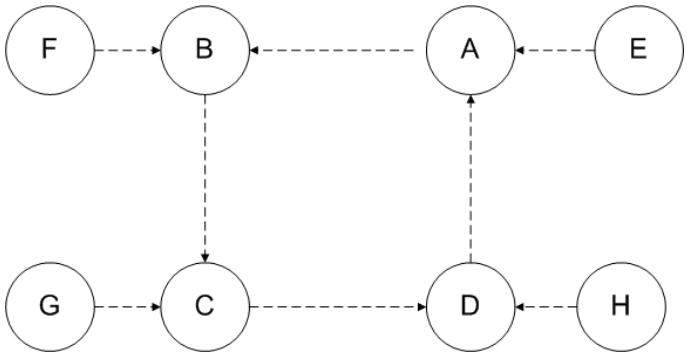
Slika 12. Uklanjanje luka iz elementa A u element D, zbog postojanja tranzitivne veze A→B→C→D

Algoritam reduciranja lukova treba za svaki luk $a = (v_1, v_2)$ provjeriti postoji li neki drugi put između v_1 i v_2 te, ako da, ukloniti navedeni luk iz grafa. Problem je ekvivalentan problemu tranzitivne redukcije grafa [Aho1972]. Primijeni li se princip istovremenog održavanja topološkog poretka i detekcije ciklusa, onda je moguće reduciranje lukova vršiti nakon svakog dodavanja novog parcijalnog modela, kako bi se smanjio broj lukova pri detekciji ciklusa. Uklanjanjem luka između dva vrha uslijed postojanja nekog drugog puta između tih vrhova, topološki poredak se ne narušava.

3.3 Skup mogućih prethodnika

Za razliku od prethodne situacije u kojoj je element X kao preduvjet elementa Z morao biti dovršen da bi započelo izvršavanje elementa Z, nije neuobičajena situacija u kojoj je iz određenog skupa elemenata potrebno odabrati jedan ili više elemenata koji će tada imati funkciju preduvjeta. Odabir potrebnog broja elemenata iz skupa prethodnika može biti načinjen prilikom izrade konačnog modela ili u trenutku izvođenja određene instance modela protoka poslova, gdje korisnik u nekom određenom vremenskom trenutku odabire željene elemente.

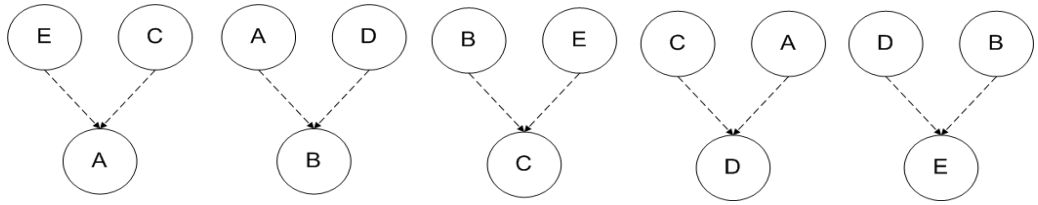
Postojanje izbora uzrokuje da se standardni postupci objedinjavanja parcijalnih modela, provjere i reduciranja lukova primijenjeni kod preduvjeta ne mogu izvesti i u slučaju prethodnika. Neke situacije koje bi kod preduvjeta ukazivale na pogrešku, kod prethodnika mogu predstavljati valjanu situaciju. Zanemari li se činjenica da je takva situacija malo vjerojatna, ali ipak moguća, slika 13 predstavlja jedan takav slučaj.



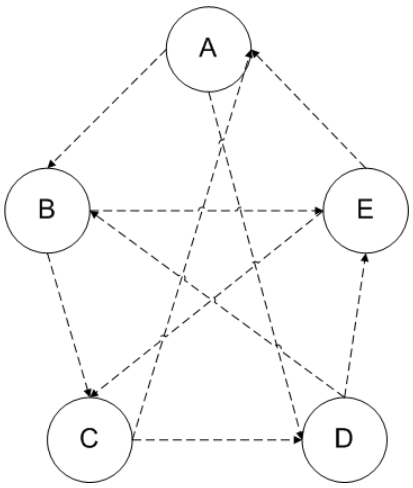
Slika 13. Skup prethodnika, primjer potencijalno valjanog slučaja za ciklus

Iscrtane strelice u ovom slučaju predstavljaju vezu između potencijalnog prethodnika i pojedinog elementa pri čemu je za izvršavanje nekog od elemenata potrebno dovršiti barem jednog od njegovih prethodnika. U ovom primjeru, za izvođenje elementa B potrebno je dovršiti element A ili element F. Za izvođenje elementa C potrebno je dovršiti B ili G. Za izvođenje elementa D potrebno je dovršiti C ili H, a za pokretanje elementa A, potrebno je dovršiti D ili E. Prethodno napravljeno objedinjavanje grafa dovelo bi do situacije u kojoj postoji ciklus koji se sastoji od vrhova A, B, C i D, što bi u terminima preduvjeta bilo nemoguće izvesti, međutim, ovdje je realizacija sasvim moguća. Dovršetkom bilo kojeg od elemenata E, F, G ili H, može se aktivirati neki od elemenata A, B, C ili D i mogu se izvršiti svi ostali elementi.

Postojanje ciklusa može ukazivati na situaciju u kojoj je rješenje nemoguće (model nije izvediv), to jest nije moguće stvoriti model protoka poslova koji bi zadovoljio sve postavljene uvjete. Za ilustraciju takvog slučaja poslužit će parcijalni modeli na slici 14. Slika 15 prikazuje usmjereni graf nastao integracijom parcijalnih modela sa slike 14. U zajedničkom grafu nalazi se nekoliko manjih ciklusa, ali i jedan veliki ciklus kojeg tvore elementi A, B, C, D i E. Za razliku od slučaja sa slike 13, ovdje ne postoji niti jedan luk koji ulazi u neki od elemenata A, B, C, D, E, a da je došao iz nekog elementa van ciklusa, što ukazuje na nemogućnost izvođenja.



Slika 14. Parcijalno definirani prethodnici koji daju nemoguće rješenje



Slika 15. Skup prethodnika, primjer neispravnog slučaja za ciklus nastao objedinjavanje parcijalnih modela sa slike 14

Navedeni slučaj sugerira da mogućnost izvođenja ovisi o postojanju ulaznih lukova u elemente u ciklusu, no prije iskaza same tvrdnje, potrebno je definirati određene pojmove vezane uz skupove mogućih prethodnika. U prethodnim primjerima uzeto je da je za izvođenje elemenata bilo dovoljno dovršiti samo jedan od dva prethodnika. Slijedi formalna definicija za općeniti slučaj.

Definicija: *Skup elemenata M={E_1, E_2, …, E_n} je skup mogućih prethodnika elementa X, ako je za početak izvođenja elemenata X potrebno dovršiti najmanje m elemenata iz skupa M, gdje je $1 \leq m < n$. Vrijednost m se definira se kao rang skupa prethodnika M, označava se s r(M) i zadaje se pri inicijalnoj definiciji skupa.*

Napomena: Alternativno rang skupa mogućih prethodnika može se zadati i u postotku, pri čemu se prije same provjere i modeliranja rang izračunava i izražava kao broj.

Koristeći navedenu definiciju, preduvjeti se mogu definirati kao skup mogućih prethodnika čiji je rang jednak kardinalnom broju skupa. Posljedica gornje definicije i zaključka vezanog uz preduvjete je sljedeća. Ukoliko bi pri izvršavanju algoritma objedinjavanja parcijalnih modela protoka poslova, u nekom trenutku izbacivanjem pojedinih elemenata iz skupa mogućih prethodnika, kardinalni broj skupa mogućih prethodnika postao jednak rangu tog skupa, tada navedeni skup mogućih prethodnika predstavlja skup preduvjeta. Ako u algoritmu objedinjavanja kardinalni broj skupa mogućih prethodnika postane manji od ranga skupa, takav sustav smatra se nerješivim, jer nije moguće ispuniti uvjete postavljene navedenim skupom mogućih prethodnika.

3.3.1 Dijeljeni skupovi mogućih prethodnika

Moguća je situacija da pojedini element ima više skupova mogućih prethodnika te je potrebno utvrditi da li su skupovi samostalni ili se mogu međusobno udružiti. Iako na prvi pogled zvuči logično da bi skupovi trebali biti samostalni, zbog načina definiranja parcijalnih modela protoka poslova, moguće je da je u jednom trenutku definiran samo dio skupa mogućih prethodnika. Sljedeća dva primjera ilustriraju primjer samostalnih skupova mogućih prethodnika i skupova mogućih prethodnika koji se mogu objediniti, gdje se element odnosi na pojedini predmet, odnosno semestar.

Primjer 1: Za upisivanje predmeta P potrebno je položiti A ili B te C ili D. Na ovaj način su definirane dva skupa mogućih prethodnika M_1 ={A,B} te M_2 ={C,D}. Rang oba skupa je 1. U ovom primjeru nije moguće upisati predmet P ako nije položen barem jedan predmet iz prvog skupa i barem jedan iz drugog, to jest, nije dozvoljen upis s položenim A i B.

Primjer 2: Za upisivanje određenog semestra S potrebno je položiti najmanje 50% predmeta koji su dotada bili definirani, pri čemu su u prvom trenutku postojali samo predmeti A i B. U terminima elemenata i skupova mogućih prethodnika ova definicija bi odgovarala postojanju parcijalnog modela koji definira element S sa svojim skupom prethodnika M_1 ={A,B}. Ako bi u nekoj drugom parcijalnom modelu bio definiran skup mogućih prethodnika M_2 ={C,D,E} tada bi, da bi se pravilo očuvalo, skupovi M_1 i M_2 morali biti objedinjeni u jedan mogući skup prethodnika M={A,B,C,D,E} ranga 3 te bi za upis semestra bilo dovoljno položiti, primjerice, samo C, D i E.

Slijedom ovih primjera potrebno je definirati dvije vrste skupova mogućih prethodnika, „obične“ i „dijeljene“, pri čemu treba voditi računa o tome da ne mora postojati samo jedan dijeljeni skup mogućih prethodnika, već može biti više takvih skupova koji se mogu razlikovati po oznaci (tipu) dijeljenog skupa.

Definicija: *Dijeljeni skup mogućih prethodnika M_i tipa T je onaj skup koji se može udružiti s ostalim dijeljenim skupovima mogućih prethodnika M_j ($j \neq i$) tipa T u novi dijeljeni skup mogućih prethodnika M te je tada $M=\bigcup_i M_i$, a rang skupa M je jednak zbroju rangova skupova M_i.*

Nakon udruživanja svih dijeljenih skupova mogućih prethodnika veza između postojanja rješenja te skupa mogućih prethodnika i postojanja ciklusa u zajedničkom grafu može se iskazati na sljedeći način. Problem prethodnika nema rješenje, ako u grafu $G(V, A)$, gdje je V skup svih vrhova koji predstavljaju elemente i A skup svih lukova pri čemu pojedini luk predstavlja vezu između elementa i njegovog prethodnika, postoji ciklus u kojem niti jedan od elemenata u ciklusu ne može biti pokrenut bez elementa koji sudjeluju u ciklusu. To bi značilo da za postojanje rješenja za barem jedan vrh iz navedenog ciklusa mora postojati barem onoliko ulaznih lukova iz vrhova koji predstavljaju elemente iz nekog skupa mogućih prethodnika koliki je rang tog skupa te to mora vrijediti za svaki skup mogućih prethodnika tog elementa.

3.3.2 Veza skupova mogućih prethodnika i preduvjeta

Skupovi mogućih prethodnika moraju se promatrati u kontekstu svih parcijalno definiranih modela, pa je tako potrebno provjeriti postoji li u vezama koje definiraju skup mogućih prethodnika neki par elemenata koji je već vezan definiranim preduvjetima. Primjerice, ako je za neki element C definirano da mora imati dovršen jedan od prethodnika A ili B, a među preduvjetima je već definirano da je element A preduvjet elementa C, tada definicija da je B jedan od mogućih prethodnika elementa C u izgradnju konačnog modelu nema smisla[8], jer se uvjet tog skupa mogućih prethodnika ostvaruje kroz element A.

U sličnoj situaciji, kad bi element C za preduvjet imao skup mogućih prethodnika {A, B, D} ranga 2, postojanje veze u kojoj je A preduvjet C, izbacilo bi element A iz skupa mogućih prethodnika i smanjilo rang tog skupa mogućih prethodnika za 1. Ako bi ista situacija vrijedila i za elemente B i C (B prethodnik od C), tada bi se izbacivanjem elementa B rang skupa mogućih prethodnika smanjio za jedan, pa bi takvim smanjivanjem ranga skupa na 0 taj skup mogućih prethodnika nestao iz konačnog modela. Kao suprotna situacija ovom primjeru, pojava parcijalnog modela u kojem je C preduvjet elementa D, dovela bi do izbacivanja elementa D iz skupa mogućih prethodnika elementa C, ali bez smanjivanja ranga, čime kardinalni broj skupa mogućih prethodnika postaje jednak rangu skupa te se elementi A i B „sele“ u preduvjete elementa C.

3.3.3 Reduciranje skupova mogućih prethodnika

Postupak u kojem bi se rang pojedinog skupa mogućih prethodnika smanjio, čime bi se neke skupove mogućih prethodnika moglo izbaciti iz modela ili bi se moglo pokazati da problem nema rješenje, može se nazvati reduciranjem skupova mogućih prethodnika, a opisan je algoritmom u tablici 1. Algoritam pretpostavlja da su svi dijeljeni skupovi mogućih prethodnika objedinjeni te da su parcijalni modeli kojima su definirani preduvjeti integrirani u zajednički graf G.

[8] Važno je napomenuti da brisanje navedene veze nije trajnog karaktera te se ne mijenja originalni parcijalni model, jer bi se u slučaju brisanja definicije u kojoj je A preduvjet od C trajno izgubio dio informacije te model ne bi odgovarao stvarnom stanju.

U svakom koraku algoritma se za neki element X promatraju njegovi skupovi mogućih prethodnika. Za svaki element E iz pojedinog skupa mogućih prethodnika se provjerava da li se navedeni element E nalazi među vezama kojima se definiraju preduvjeti te, ako da, tada je potrebno provjeriti postoji li veza između E i X. Ako da, onda se E izbacuje iz skupa mogućih prethodnika te se rang skupa smanjuje za 1 ako je E bio preduvjet od X, odnosno ostaje isti, ako je X bio preduvjet od E.

Nakon što su provjereni svi elementi nekog skupa mogućih prethodnika, provjerava se kardinalni broj skupa mogućih prethodnika i uspoređuje s rangom. Ukoliko je kardinalni broj manji od ranga skupa mogućih prethodnika, problem nije rješiv, a ako je rang skupa jednak kardinalnom broju skupa, tada svi elementi iz tog skupa mogućih prethodnika postaju preduvjeti elementa X. Dodavanjem preduvjeta u koraku 4, mijenja se topološki poredak na način da su novi preduvjeti došli ispred elementa X te je poredak potrebno osvježiti, a X će biti provjeren opet nakon što se provjere njegovi novi preduvjeti.

Tablica 1. Algoritam reduciranja prethodnika

1. Neka je G usmjereni graf nastao integracijom parcijalnih modela kojima se opisuju preduvjeti elemenata.
2. Sve elemente za koje postoji definirani skup mogućih prethodnika poredati po topološkom poretku grafa G. Ako među takvim elementima postoje elementi koji nisu u grafu G, dodati ih na kraj tog poretka.
3. Neka je S skup tako poredanih elemenata
   n := |S|
   i := 0
4. Neka je X i-ti element iz skupa S.
5. Za svaki skup prethodnika M elementa X radi
   za svaki element E ∈ M radi
   ako E ∉ S uzmi sljedeći element iz skupa M
   inače
   ako je E < X tada
   ako u grafu G postoji put od E do X tada
   M := M \ {E}
   r(M) := r(M) – 1
   inače
   ako u grafu G postoji put od X do E tada
   M := M \ {E}
   ako je | M | < r(M) tada
   problem nema rješenje (kraj algoritma)
   inače ako je | M | = r(M) tada
   dodaj sve elemente iz M kao preduvjete elementa X i obriši M
   obnovi skup S prema novom topološkom poretku
   n := |S|
6. Ako nije bilo dodavanja preduvjeta ili ako topološki poredak nije promijenjen tada
   i := i + 1
   Ako je i<n ponovi korak 4.

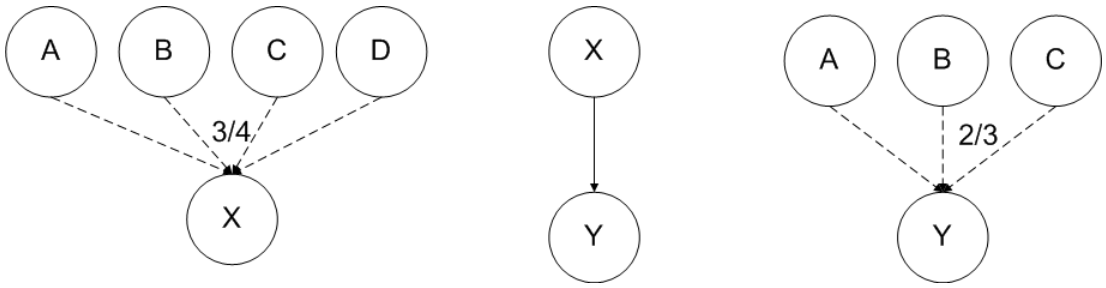
Algoritam radi sa svim elementima koji imaju definirane preduvjete te ih uzima redoslijedom određenim topološkim poretkom nastalim nakon objedinjavanja parcijalnih modela preduvjeta. U slučaju da neki element nije bio u parcijalnim modelima za preduvjete te se ne nalazi u inicijalnom topološkom poretku, potrebno ga je dodati na kraj.

Bitna pretpostavka za ispravnost algoritma je da se prilikom osvježavanja topološkog poretka, poredak prvih i-1 elemenata (koji su dotad obrađeni) ne mijenja. Budući da se elementi uzimaju u topološkom poretku prema redoslijedu definiranom među preduvjetima, sigurno je da će elementi koji su preduvjeti trenutnog elementa biti obrađeni prije trenutnog elementa.

Traženje postojanja puta od E do X (ili analogno od X do E), može se ubrzati na način da se u traženju mogućih putova iz nekog vrha prema vrhu X u obzir uzimaju samo oni lukovi prema vrhovima za koje vrijedi da se u topološkom poretku nalaze ispred X (odnosno ispred E u drugom slučaju).

Dodatno reduciranje skupova mogućih prethodnika može se ostvariti modifikacijom prethodnog algoritma. Korak 5 se može proširiti provjerom veze između trenutnog skupa prethodnika nekog elementa i skupa mogućih prethodnika njegovih preduvjeta. Za objašnjenje ideje može poslužiti sljedeći primjer ilustriran na slici 16.

Primjer: Neka je za izvršenje elementa X potrebno dovršiti najmanje 3 elementa iz skupa mogućih prethodnika {A, B, C, D} te neka je X preduvjet od Y. Ako je za izvršenje elementa Y potrebno izvršiti barem 2 elementa iz skupa mogućih prethodnika {A, B, C} tada se taj uvjet može izbaciti iz konačnog modela, jer je uvjet ostvaren kroz vezu s preduvjetom X. Naime, u bilo kojoj kombinaciji tri elementa za pokretanje elementa X, moraju biti barem dva elementa iz skupa {A, B, C} što je upravo uvjet za izvršenje elementa Y. Kao primjer u kojem to ne bi bilo moguće je situacija kada bi za izvršenje elementa X bilo potrebno dovršiti dva od četiri moguća elementa, pa kombinacija {A, D} ne bi predstavljala dovoljni uvjet za element Y.



Slika 16. Primjer parcijalno definiranih modela u kojima se skup mogućih prethodnika elementa Y može izbaciti iz modela.

Mogućnost potpunog brisanja nekog skupa mogućih prethodnika dana je sljedećim teoremom.

Teorem: Neka je X preduvjet od Y. Ako za neki skup mogućih prethodnika M_Y elementa Y postoji skup mogućih prethodnika M_X elementa X takav da vrijedi r(M_Y) > 0, r(M_X) > 0, r(M_Y) $\leq$ r(M_X) $-$ | $M_X \setminus M_Y$ |, tada se skup prethodnika M_Y može izbaciti iz skupa parcijalno definiranih modela.

Dokaz: Neka je broj elemenata iz M_X koji nisu sadržani u M_Y, to jest kardinalni broj skupa $M_X \setminus M_Y$ jednak k. Neka je r(M_Y) $\leq$ r(M_X) - k, te neka je m = r(M_X) - k. Budući da su rangovi skupova mogućih prethodnika pozitivni brojevi, to je m pozitivan broj. Neka je za izvršenje elementa X iskorišteno svih k elemenata iz skupa $M_X \setminus M_Y$ to povlači da je potrebno izvršiti još najmanje m elemenata iz skupa $M_X \cap M_Y$. Ako je rang skupa M_Y manji ili jednak broju m, onda su uvjeti skupa mogućih prethodnika M_Y ispunjeni te se skup M_Y može izbaciti iz skupa parcijalno definiranih modela.

■

Uklanjanjem pojedinog skupa prethodnika M_Y ne narušava se ispravnost algoritma iz tablice 1. Negativna posljedica je povećanje složenosti provjerama dotad obrađenih skupova mogućih prethodnika i traženjem skupova prethodnika M_X, koje odgovaraju traženim uvjetima.

3.4 Konačni model i ciljevi konačnog modela

Definicija: Konačni model K je podskup skupa svih elemenata E za kojeg vrijede sljedeći uvjeti:

1. $e \in K, p \in E$, p *preduvjet od* $e \implies p \in K, \quad \forall e \in K \; i \; \forall p \in E$
2. $e \in K, M$ *skup mogućih prethodnika od* $e \implies |K \cap M| \geq r(M), \forall e \in K \; i \; \forall M$

Prvi uvjet u gornjoj definiciji uvjetuje da, ako konačni model sadrži neki element, tada sadrži i sve njegove preduvjete. Ako neki element ima definiran skup mogućih prethodnika M, tada se, prema drugom uvjetu, u konačnom modelu mora naći barem onoliko elemenata iz skupa M, koliki je rang skupa M te to mora vrijediti za svaki skup mogućih prethodnika M.

Definicija: Cilj C je uređena četvorka (N, O, P, E) gdje je N naziv cilja, skup O $\subset E$ skup obveznih elemenata koji moraju biti uvršteni u konačni model, skup P $\subset E$ skup preporučenih elemenata za konačni model K i E skup svih elemenata.

Svrha postojanja ciljeva je olakšati odabir elemenata za konačni model, odnosno preporučiti elemente koje bi korisnik mogao odabrati za konačni model. Bitno je naglasiti da O ne mora nužno sadržavati sve preduvjete i skupove mogućih prethodnika nekog elementa. Dodavanjem elementa u konačni model automatski se moraju dodati i svi njegovi preduvjeti te barem onoliko elemenata iz skupova prethodnika kako bi bili zadovoljeni

uvjeti iz definicije konačnog modela. U svakom sustavu mora postojati barem jedan cilj, a trivijalan cilj je C*(N, ∅, ∅, E)* u kojem je odabir elemenata proizvoljan.

Pored navedenih uvjeta, moguće je da konačni model mora zadovoljiti određene dodatna pravila. Primjer dodatnog pravila je ograničenje minimalnog ili maksimalnog broja elemenata u konačnom modelu.

Dodatna pravila definiraju se funkcijom *valjan* specifičnom za pojedinu implementaciju, to jest za pojedina značenja elemenata. Funkcija za svaki konačni model K vraća istinu ili laž, uz razlog zašto konačni model nije valjan.

Definicija. Nužni uvjet ispravnosti konačnog modela K su postojanje barem jednog cilja C(N, O, P, E) takvog da je O ⊂ K i uvjet da je vrijednost funkcije valjan(K) istina.

3.5 Međusobno isključivi parovi elemenata

Definicija. Međusobno isključivi par elemenata je par (p,q), p , q $\in E$ takvih da vrijedi

$$p \in K \Longrightarrow q \notin K,$$ *gdje je K konačni model.*

Međusobno isključive parove elemenata čine oni parovi elemenata koji se ne mogu zajedno pojaviti unutar konačnog modela niti u jednoj kombinaciji.[9] Treba naglasiti da konačni model ne mora nužno biti čitav integralni graf nastao spajanjem preduvjeta, već bilo koji njegov podgraf za kojeg su zadovoljeni uvjeti iz definicije konačnog modela. Kako se parcijalni modeli definiraju za sve elemente, provjera da li je neki skup odabranih elemenata konačni model vrši se nakon što se odaberu svi željeni elementi. Ako su parcijalnim modelima definirani međusobno isključivi parovi, tada je pri dodavanju elementa u skup koji je kandidat za konačni model potrebno provjeriti da li bi dodavanje elementa u taj skup bilo u kontradikciji s uvjetom nekog od međusobno isključivih parova. Ako među parcijalnim modelima postoje takvi modeli čijom integracijom dolazi do uvjeta da je X preduvjet od Y ili Y preduvjet od X i ako postoji međusobno isključiv par (X, Y) tada su parcijalni modeli u međusobnoj koliziji te postoji pogreška u sustavu parcijalnih modela.[10]

Istovremeno postojanje međusobno isključivih parova i skupova mogućih prethodnika predstavlja složenu situaciju, jer može utjecati na rang skupa mogućih prethodnika i mogućnost postojanja konačnog modela. U slučaju da se u skupu mogućih prethodnika elementa Z nalazi element X koji s elementom Y čini međusobno isključivi par elemenata, a Y je ujedno preduvjet od Z, tada je element X potrebno izbaciti iz skupa mogućih prethodnika bez smanjivanja ranga tog skupa. Takvim izbacivanjem rang skupa mogućih prethodnika može postati manji od kardinalnog broja skupa što bi povlačilo da problem nije rješiv. Na sličan način, ako su kardinalni broj i rang skupa mogućih prethodnika jednaki,

[9] Direktna definicija $(p \in K \Longrightarrow q \notin K) \wedge (q \in K \Longrightarrow p \notin K)$ svodi se na uvjet $p \in K \Longrightarrow q \notin K$ ili na uvjet $q \in K \Longrightarrow p \notin K$

[10] Formalno gledano, moguće je stvoriti konačni model bez elemenata X i Y, ali u sustavu postoji dva ili više parcijalnih modela koji su u međusobnoj koliziji te se jedan od elemenata nikad ne može pojaviti u konačnom modelu, jer je međusobno isključiv sa svojim preduvjetom.

tada elementi iz skupa mogućih prethodnika postaju preduvjeti elementa Z. Postupak je nalik algoritmu reduciranja skupa mogućih prethodnika iz tablice 1 i prikazan je u tablici 2.

Tablica 2. Algoritam reduciranja skupova mogućih prethodnika na osnovu međusobno isključivih parova

1. Neka je G usmjereni graf nastao integracijom parcijalnih modela kojima se opisuju preduvjeti elemenata.
2. Sve elemente za koje postoji definirani skup mogućih prethodnika poredati po topološkom poretku grafa G. Ako među takvim elementima postoje elementi koji nisu u grafu G, dodati ih na kraj tog poretka.
3. Neka je S skup tako poredanih elemenata
   n := |S|
   i := 0
4. Neka je X i-ti element iz skupa S.
5. Za svaki skup prethodnika M elementa X radi
   za svaki element E ∈ M radi
   ako ∃ D ∈ S takav da je D preduvjet od E i ∃ međusobno isključiv par (D, E) tada
   M := M \ {E}
   ako je | M | < r(M) tada
   problem nema rješenje (kraj algoritma)
   inače ako je | M | = r(M) tada
   dodaj sve elemente iz M kao preduvjete elementa X i obriši M
   obnovi skup S prema novom topološkom poretku
   n := |S|
6. Ako nije bilo dodavanja preduvjeta ili ako topološki poredak nije promijenjen tada
   i := i + 1
   Ako je i<n ponovi korak 4.

Postojanje međusobno isključivih parova za elemente unutar istog skupa prethodnika može dovesti do toga da uvjet tog skupa mogućih prethodnika nije zadovoljiv, to jest do situacije da nije moguće odabrati dovoljno elementa za konačni model kako bi rang tog skupa mogućih prethodnika bio zadovoljen. Primjerice, za krajnji desni model sa slike 16, i međusobno isključive parove (A, B), (B, C), (A, C), taj skup mogućih prethodnika predstavlja nemoguć uvjet. Nužan uvjet za postojanje rješenja skupa mogućih prethodnika u slučaju kada postoje međusobna isključivanja nekih elemenata koji se nalaze u istom skupu mogućih prethodnika moguće je odrediti izračunom valjanog broja kombinacija. U razmatranje za pojedini skup mogućih prethodnika ulaze samo oni međusobno isključivi parovi gdje su oba elementa sadržana u skupu mogućih prethodnika te se takvi parovi mogu označiti od P_1 do P_q. Ako je kardinalni broj skupa mogućih prethodnika jednak m i rang skupa jednak r, tada bi broj svih mogućih odabira prethodnika, bez uvjeta međusobnog isključivanja bio $\binom{m}{r}$. Uzevši u obzir da se u skupu svih kombinacija ne smije pojaviti nijedan par iz P_1 do P_q, to je od ukupnog broja kombinacija potrebno oduzeti $q * \binom{m-2}{r-2}$, to jest sve one kombinacije u kojima su među r elemenata upravo dva elementa iz nekog od skupova P_i, pa se iz preostalih m-2 elementa, mora odabrati r-2 preostalih. Budući da skupovi P_i međusobno nisu morali biti disjunktni, stoga kardinalni broj unije dvaju

skupova P_i i P_j može biti manji od r, pa se među $q * \binom{m-2}{r-2}$ određeni broj kombinacija pojavljuje više puta te je na broj kombinacija potrebno dodati sve one koji su brojani više puta. Postupak ide dalje kao i kod Sylvesterove formule za vjerojatnost unije događaja [Elezović2010], pa je konačna formula[11] za broj valjanih kombinacija jednaka

$$\binom{m}{r} - q * \binom{m-2}{r-2} + \sum_{i=1}^{q-1} \sum_{j=i+1}^{q} \binom{m - |P_i \cup P_j|}{r - |P_i \cup P_j|} - \sum_{i=1}^{q-2} \sum_{j=i+1}^{q-1} \sum_{k=j+1}^{q} \binom{m - |P_i \cup P_j \cup P_k|}{r - |P_i \cup P_j \cup P_k|} + \sum_{i=1}^{q-3} \sum_{j=i+1}^{q-2} \sum_{k=j+1}^{q-1} \sum_{l=k+1}^{q} \binom{m - |P_i \cup P_j \cup P_k \cup P_l|}{r - |P_i \cup P_j \cup P_k \cup P_l|} - \ldots + \ldots$$

pri čemu se uzastopne sume programski mogu riješiti svođenjem na traženje permutacija x-članih podskupova od m elemenata te korištenjem algoritama za određivanje međusobnog poretka permutacija [Kreher1998].

Primjer: Neka je za izvođenje elementa X potrebno dovršiti barem 4 elementa iz skupa {A, B, C, D, E, F} te neka su definirani međusobno isključivi parovi P_1=(A, D), P_2=(B, E), P_3=(B, F). Tada je broj svih kombinacija u kojima se ne nalaze međusobno isključujući elementi jednak 1 zbog

$$\binom{6}{4} - 3 * \binom{4}{2} + \binom{2}{0} + \binom{2}{0} + \binom{3}{1} - \binom{2}{0} = 1$$

pa se, osim ako nema nekih drugih ograničenja, može ispuniti uvjet tog skupa mogućih prethodnika.

Treba napomenuti da izračun iz prethodnog primjera predstavlja tek nužan uvjet za postojanje rješenje problema, jer je moguće konstruirati primjere u kojem taj uvjet nije dovoljan, odnosno postoje slučajevi u kojima mogućnost stvaranja konačnog modela ovisi o nekim drugim skupovima prethodnika ili o korisnikovom odabiru u nekom trenutku te ga nije moguće deterministički odrediti.

Primjerice, ako je za izvođenje elementa X potrebno dovršiti dva elementa iz skupa {A, B, C}, a za element Y dva elementa iz skupa {D, E, F}, tada, ako su definirani međusobno isključivi parovi (A, D), (B, E), (C, F) bez obzira koji od elemenata A, B ili C bio odabran, nije moguća ni jedna kombinacija od dva elementa iz skupa {D, E, F}, a da ne bude u koliziji s jednim od međusobno isključivih parova.

[11] Situacija se po potrebi ne mora nužno ograničiti na parove, već na proizvoljne n-torke elemenata $(e_1, \ldots, e_n)$ pri čemu se svi članovi n-torke ne mogu istovremeno pojaviti u modelu. U tom slučaju gornja formula se modificira na način da se ne oduzima 2, nego n. Istovremeno je potrebno promijeniti definiciju međusobnog isključivanja u

$(e_1 \in K \Longrightarrow (e_2 \notin K) \wedge (e_3 \notin K) \wedge \ldots \wedge (e_n \notin K)) \wedge$
$(e_2 \in K \Longrightarrow (e_1 \notin K) \wedge (e_3 \notin K) \wedge \ldots \wedge (e_n \notin K)) \wedge \ldots$

Teorijski, problem je rješiv definiranjem skupa elemenata s kojim pojedini element može biti istovremeno izvršen te traženjem ona dva para čiji presjek takvih skupa nije prazan skup, međutim, u slučajevima kad je rang skupa mogućih prethodnika veći ili kada je broj elemenata veći, onda se radi o složenom kombinatornom problemu.

Sličan problem je i sljedeća situacija. Ako je za izvođenje elementa X potrebno dovršiti barem 2 elementa iz skupa {A, B, C}, a za izvođenje elementa Y element D ili E te ako je X preduvjet elementa Y, tada, uz postojanje međusobno isključivih parova (A, D), (B, D), (A, E) mogućnost ostvarenja uvjeta skupa mogućih prethodnika za element Y ovisi da li je odabran element A ili ne, to jest Y je moguće izvršiti samo ako su za element X odabrani B i C, a za element Y element E. Bilo bi neispravno ograničiti mogućnost izbora elementa A za element X, jer je moguće da Y uopće neće sudjelovati u konačnom modelu. Iako bi teoretski moguće bilo pronaći sve valjane kombinacije, zbog složenosti problema, pristup s provjerom svih kombinacija ne bi mogao biti efikasan.

Uslijed svega navedenog, skupove mogućih prethodnika poželjno je što više reducirati, a zatim korisniku omogućiti interaktivnu arbitražu. Ako se i nakon toga u konačnom modelu nalaze međusobno isključivi parovi i skupovi mogućih prethodnika, problem ispravnosti sustava parcijalnih modela se može riješiti traženjem barem jednog rješenja koje zadovoljava uvjete postavljene kroz međusobno isključive parove i skupove mogućih prethodnika. S obzirom na kombinatornu složenost problema, rješenje je pogodno tražiti genetskim algoritmom. Prije početka rada algoritma potrebno je odabrati elemente čiji se skupovi mogućih prethodnika žele ispitati. Rad genetskog algoritma detaljnije je opisan u poglavlju 4.3.1.

3.6 Isključivi odabir

U prethodnom poglavlju definirani su međusobno isključivi parovi, pri čemu se dva elementa koji čine jedan par nikad ne mogu istovremeno naći u konačnom modelu. U određenim slučajevima moguće je da neka dva elementa postanu međusobno isključiva tek nakon nekog određenog trenutka.

Definicija. Isključivi odabir je uređena trojka elemenata (x, y, z), $x, y, z \in E$ *takva da vrijedi*

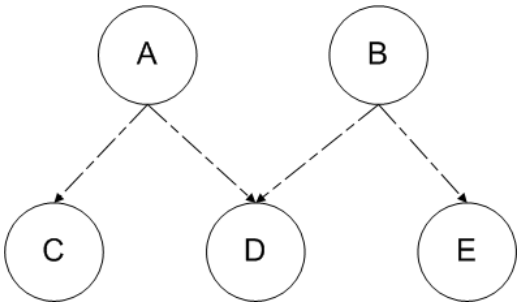
$$x \in K \implies (y \in K \implies z \notin K)$$

gdje je K konačni model.

Primjerice, postojanje isključivog odabira (A, B, C) značilo bi da pojavom elementa A u konačnom modelu K, model može sadržavati ili element B ili element C ili nijednog od njih. U slučaju da se element A ne nalazi u konačnom modelu, tada konačni model može sadržavati i B i C. Ako za neki isključivi odabir (A, B, C) već postoji definirana relacija preduvjeta takva da je A preduvjet od B, A preduvjet od C, B preduvjet od A ili C preduvjet od A tada se takav isključivi odabir briše iz sustava parcijalnih modela te se dodaje međusobno isključivi par (B, C). Situacija u kojoj je B preduvjet C ili C preduvjet B ukazuje na moguću pogrešku u sustavu parcijalnih modela.

Algoritam reduciranja skupova mogućih prethodnika na osnovu isključivog odabira nalik je algoritmu iz tablice 2 uz modifikaciju koraka 5 u kojem se, u ovom slučaju, traže elementi C i D iz skupa preduvjeta takvi da postoji isključivi odabir (C, D, E).

Kao i kod međusobno isključivih elemenata prije dodavanja pojedinih elemenata u skup K', kandidat za konačni model, potrebno je izvršiti provjeru da li bi dodavanje elementa u taj skup bilo u kontradikciji s nekim od isključivih odabira. Ono što je specifično kod modela koji sadrži isključive odabire jest da isključivi odabiri mogu biti međusobno ovisni. Primjerice, odvojeno promatrano, postojanje isključivih odabira (A, C, D) i (B, D, E) znači da se dodavanjem elementa A u skup K', u K' može dodati ili C ili D te da se nakon B u K' može dodati ili D ili E. Kao i kod primjera s preduvjetima i skupovima mogućih prethodnika, isključivi odabir može se prikazati pomoću usmjerenog grafa u kojem lukovi iz nekog vrha predstavljaju međusobno isključive izbore. Usmjereni graf za prethodni primjer bi izgledao kao na sljedećoj slici.



Slika 17. Isključivi odabiri (A, C, D) i (B,D, E)

Iz slike je vidljivo da izbori u vrhovima A i B nisu neovisni. Primjerice, odabere li se nakon vrha A element C, to automatski povlači da se izbor nakon B svodi samo na E (nije moguće odabrati D, jer bi to bilo u suprotnosti s isključivim odabirom načinjenom u vrhu A).

Kako bi se korisniku olakšao izbor elemenata, potrebno je definirati algoritam, koji će omogućiti brz izračun posljedica pojedine odluke. Sve isključive odabire moguće je prikazati u obliku usmjerenog grafa te ih integrirati u jedan, zajednički graf. Tako dobiveni graf $G(V, A)$ sadržavat će vrhove koji predstavljaju elemente koji se nalaze unutar isključivih odabira pri čemu luk iz nekog vrha predstavlja međusobno isključive odabire nakon odabira tog vrha. Graf G ne mora biti povezan. Skup Z sadržavat će popis „zabranjenih“ elemenata, to jest onih elemenata koji se ne mogu naći u konačnom modelu, jer bi time bilo narušeno neko od pravila isključivih odabira. Tablica 3 opisuje algoritam kojim se obnavlja skup elemenata koji su na raspolaganju za dodavanje u konačni model.

Odabirom nekog elementa v koji se nalazi u grafu isključivih odabira, dolazi do provjere svih njegovih roditelja. Ako se pojedini roditelj p nalazi u konačnom modelu, tada se u popis zabranjenih vrhova moraju dodati svi vrhovi čiji je roditelj vrh p. Ako roditelj p nije u konačnom modelu tada je potrebno provjeriti da li se neki od djece vrha p (različit od v) nalazi u konačnom modelu. Ako da, tada je potrebno onemogućiti dodavanje roditelja p u konačni model. Ukoliko je došlo do promjene skupa Z potrebno je provjeriti da li se među

elementima iz skupa zabranjenih vrhova nalazi neki od elemenata koji je član nekog od skupova mogućih prethodnika. Svaki takav element potrebno je izbaciti iz skupa mogućih prethodnika bez smanjivanja ranga skupa. Postane li nakon toga rang skupa mogućih prethodnika nekog elementa jednak kardinalnom broju tog skupa, tada elementi iz skupa mogućih prethodnika postaju preduvjeti navedenog elementa. U slučaju da je rang skupa mogućih elemenata manji od kardinalnog broja tada elementi o čijem se skupu mogućih prethodnika radi postaje član skupa Z ili algoritam prijavljuje pogrešku ako je element već bio odabran za konačni model. Oporavak od pogreške može se izvesti vraćanjem na stanje prije odabira vrha u koraku 3.

Tablica 3. Algoritam održavanja skupa elemenata dostupnih za konačni model

1. Neka je $G(V, A)$ usmjereni graf nastao integracijom isključivih odabira.
2. Neka je Z := Ø skup vrhova koji se ne mogu naći u konačnom modelu
3. Neka je $v \in E \setminus Z$ odabrani vrh za konačni model
4. Ako je $v \in G$ tada
   Za svaki p takav da ∃ $(p, v) \in A$ radi
   K := K ∪ $\{v\}$
   ako $p \in K$ tada
   Z := Z ∪ $\{w \mid \exists\ (p, w) \in A, w \neq v\}$
   inače
   ako je $K \cap \{\ w \mid \exists\ (p, w) \in A, w \neq v\} \neq \emptyset$ tada
   Z := Z ∪ $\{p\}$
5. Ako se skup Z promijenio tada
   za svaki skup mogućih prethodnika M takav da je $M \cap Z \neq \emptyset$
   $M := M \setminus \{Z\}$
   X := element kojem je M skup mogućih prethodnika
   ako je | M | = r(M) tada
   dodaj sve elemente iz M kao preduvjete elementa X i obriši M
   ako je | M | < r(M) tada
   ako je X $\in K$ tada
   problem nema rješenje (kraj algoritma)
   inače
   Z := Z ∪ $\{X\}$
   ponovi korak 5
6. Ponovi korak 3 dok je $E \setminus Z \neq \emptyset$ ili dok korisnik ne završi s odabirom

3.7 Redoslijed algoritama za parcijalno definirane modele protoka poslova

Nakon što su definirani svi parcijalni modeli, prvi korak u provjeri ispravnosti sustava parcijalnih modela je integracija svih uvjeta preduvjeta prema pravilima iz poglavlja 3.2.2. Na integriranom grafu preduvjeta dolazi do provjere postoje li na istom putu od početnog do završnog vrha dva vrha za koje postoji definirano međusobno isključivanje ili isključivi odabir.[12] Provjerom preduvjeta moguće je neke od isključivih odabira pretvoriti u međusobna isključivanja. Sljedeći korak predstavlja integraciju dijeljenih skupova mogućih prethodnika te reduciranje skupova mogućih prethodnika na osnovu preduvjeta, a zatim i na osnovu međusobnih isključivanja i isključivih odabira. Nakon reduciranja prethodnika, genetskim algoritmom se provjerava da li su uvjeti postavljeni na skupove mogućih vrijednosti ostvarivi. Nakon uspješne provjere, korisniku se može prepustiti odabir elemenata za konačni model prema prethodno definiranim ciljevima. Odabirom pojedinog elementa dolazi do dodavanja njegovih preduvjeta i po potrebi smanjivanja mogućih elemenata za daljnji izbor. Dovršetkom odabira elemenata u konačni model se dodaju svi oni elementi potrebni za zadovoljavanje skupova mogućih prethodnika elemenata iz konačnog modela. U slučaju da se mora vršiti raspoređivanje elemenata po vremenskim okvirima (poglavlje 4.3), tada se od korisnika traži eksplicitni odabir prethodnika. Za vrijeme odabira poziva se funkcija *valjan* za konačni model, kako bi se provjerili uvjeti određeni konkretnom implementacijom. Po okončanju stvaranju valjanog konačnog modela dolazi do pretvorbe konačnog modela u WF model (poglavlje 5).

3.8 Definiranje parcijalnih modela logičkim izrazima

Provjera da li definirani elementi kroz sustav preduvjeta, skupove mogućih prethodnika, međusobna isključivanja i isključive odabire mogu ići zajedno, moguća je i korištenjem logičkih izraza te njihovim uvrštavanjem u neki od sustava za provjeru zadovoljivosti logičkih izraza (eng. *Boolean satisfiability problem solver*) [SATWiki]. Kako bi se zapis logičkih izraza pojednostavio, logička varijabla X poprimit će vrijednost istina ako je $X \in K$, odnosno laž ako $X \notin K$, gdje je K konačni model.

Preduvjeti definirani u 3.2 i integrirani u zajednički graf mogu se razložiti u niz ne nužno neposrednih preduvjeta te se pritom činjenica da je element A preduvjet elementa B može iskazati logičkim izrazom B => A. Navedena implikacija je laž jedino u slučaju da je element B u konačnom modelu (B je istina), a A nije (A je laž). U ostalim slučajevima implikacija je istinita, što upravo odgovara opisu preduvjeta. Ako neki element ima više preduvjeta onda će se takva situacija pretvoriti u više implikacija od kojih svaka mora biti zadovoljena.

[12] Provjera međusobnog isključivanja na cijelom grafu, a ne samo na istom putu ne bi bila ispravna, jer ne moraju svi vrhovi iz integriranog grafa biti u konačnom modelu.

Međusobno isključivi parovi (A, B) se mogu definirati izrazom $\neg(A \wedge B)$ [13]. Logički izraz kojim se opisuje međusobno isključivanje n-torke elemenata (E_1, E_2, … E_n) elemenata je

$$(E_1 \wedge \neg E_2 \wedge \neg E_3 \ldots \wedge \neg E_n) \vee (\neg E_1 \wedge E_2 \wedge \neg E_3 \ldots \wedge \neg E_n) \\ \vee (\neg E_1 \wedge \neg E_2 \wedge E_3 \ldots \wedge \neg E_n) \vee \ldots \vee (\neg E_1 \wedge \neg E_2 \wedge \neg E_3 \ldots \wedge E_n)$$

Isključivi odabir (A, B, C) može se prikazati logičkim izrazom $\neg A \wedge \neg B \wedge \neg C$, jer je jedina nedopuštena kombinacija ona u kojoj se sva tri elementa nalaze u konačnom modelu. Poopći li se isključivi odabir na oblik (A, E_1, E_2, … E_n), tada izraz postaje složeniji i nalik onome za međusobno isključivanje n-torke. Ako je A istina, tada za ostale elemente vrijedi međusobno isključivanje, a ako je A laž, tada je međusobni odnos ostalih elemenata nebitan, pa je logički izraz za isključivi odabir (A, E_1, E_2, … E_n).

$$\neg A \vee (E_1 \wedge \neg E_2 \wedge \neg E_3 \ldots \wedge \neg E_n) \vee (\neg E_1 \wedge E_2 \wedge \neg E_3 \ldots \wedge \neg E_n) \\ \vee (\neg E_1 \wedge \neg E_2 \wedge E_3 \ldots \wedge \neg E_n) \vee \ldots \vee (\neg E_1 \wedge \neg E_2 \wedge \neg E_3 \ldots \wedge E_n)$$

Što se tiče skupova međusobnih prethodnika, situacija je prilično slična kao i kod preduvjeta, međutim ovdje je više implikacija vezanih uz jedan element povezano izrazom *ili*, pa je tako, primjerice, za situaciju u kojoj je za element D definiran skup mogućih prethodnika {A, B, C} ranga 1 logički izraz koji opisuje nastalu situaciju

$$(D \Rightarrow A) \vee (D \Rightarrow B) \vee (D \Rightarrow C)$$

Problem nastaje ako je rang skupa veći od 1. Za navedeni primjer i rang skupa 2, ovaj bi se ovaj izraz morao promijeniti u

$$D \Rightarrow (A \wedge B) \quad \vee \quad D \Rightarrow (B \wedge C) \quad \vee \quad D \Rightarrow (A \wedge C)$$

a postupak postaje nepraktičan kada broj mogućih izbora bude veći. Naime, ako za neki element mora biti izvršeno r od m mogućih prethodnika, tada logički izraz ima $\binom{m}{r}$ izraza. Algoritmom iz poglavlja 3.3.3 moguće je reducirati skupove mogućih prethodnika i donekle smanjiti razinu problema.

Tako formirani logički izrazi mogu ukazati na logičku pogrešku u modelu ili potvrditi ispravnost modela, iako zbog složenosti izraza može biti upitna iskoristivost pretvaranja parcijalno definiranih modela protoka poslova u logičke izraze.

3.9 Definiranje složenijih modela

Preduvjeti, skupovi mogućih prethodnika, međusobna isključivanja i isključivi odabir omogućavaju da se temeljem jednostavnih parcijalnih modela izradi integralni model. Ukoliko je u nekom trenutku potrebno napraviti složenije modeliranje protoka poslova, tada se može iskoristiti činjenica da je značenje elementa prepušteno konkretnoj implementaciji. Naime, moguće je neki od elemenata definirati tako da ne predstavlja pojedinu cjelinu ili neki predmet, već da predstavlja pokretanje nekog novog modela

[13] Izraz A XOR B ne bi bio točan, jer bi poprimao vrijednost laž ako je A laž i B laž, što bi odgovaralo situaciji u kojoj niti A niti B ne bi bili sadržani u konačnom modelu, što predstavlja ispravnu situaciju.

protoka poslova koji može biti odvojeno modeliran. Novi model u sebi može sadržavati zasebne elemente, pa će biti uključeni u skup parcijalnih modela bez modifikacija. Alternativno može u sebi sadržavati postojeće elemente, povezane nekim od složenijih uvjeta, kao što su grananja, petlje i slično, koje nije moguće formalno provjeriti.

Tako definiran novi model predstavlja podmodel protoka poslova. U slučaju da podmodel protoka poslova sadrži postojeći element X za kojeg postoje definicije u parcijalnim modelima, tada će se u tom podmodelu svaka pojava tog elementa X zamijeniti sa skupom elemenata koji bi sadržavao ne samo X već i sve njegove preduvjete i skupove mogućih prethodnika. Modeliranje takvog podmodela analogno je modeliranju čitavog integralnog modela. U zajedničkom grafu se mogu promatrati samo oni vrhovi koji su na putu od početnog vrha do elementa X te se za tu situaciju element X može smatrati završnim vrhom.

Ovaj postupak za posljedicu ima umnožavanje objekata modela zbog supstitucije elemenata iz podmodela sa svim elementima s kojima je povezan. Preciznije objedinjavanje nije moguće obaviti, jer uvjeti postavljeni u podmodelu ne moraju biti deterministički. Ipak, na ovaj način čuvaju se odnosi među elementima definirani u parcijalnim modelima protoka poslova, što čini konačni model ispravnim do na uvjete postavljene u podmodelu.

4. Vremenska komponenta parcijalnih modela i problem raspoređivanja

U situacijama u kojima nema vremenskih ograničenja model protoka moguće je izgraditi isključivo na osnovu parcijalnih modela definiranih u poglavlju 3, jer osim ranije iznesenih pravila integracije (postojanje ciklusa, uklanjanja tranzitivnosti, reduciranje lukova, reduciranje skupova mogućih prethodnika…) nema drugih zapreka pri pretvorbi parcijalnih modela u integralni model, a zatim i u konkretni WF model. Korisnik bira između jednog ili više ciljeva čime se u konačni model automatski dodaju obvezni elementi cilja. Po želji, korisnik u konačni model dodaje i ostale elemente poštujući definirana pravila i vodeći računa o rezultatu funkcije za provjeru valjanosti konačnog modela. Po odabiru, u model se dodaju svi preduvjeti odabranih elemenata kao i elementi nužni da bi uvjeti skupova mogućih prethodnika bili zadovoljeni.

U ostatku ovog poglavlja pod zajedničkim imenom prethodnici bit će smatrani svi oni elementi koji svojim izvršavanjem prethode nekom elementu, neovisno da li su bili preduvjeti ili su bili članovi skupa mogućih prethodnika. Analogna tvrdnja vrijedi i za termin sljedbenik. Kada to bude potrebno, eksplicitno će se naglasiti radi li se o preduvjetu ili o članu skupa mogućih prethodnika.

4.1 Vremenska razdoblja

Svaki element modela protoka poslova može biti određen s jednim ili više vremenskih razdoblja u kojima se može izvoditi, pri čemu se razlikuju tri tipa vremenskih razdoblja.

Najjednostavniji tip vremenskog razdoblja je fiksno vremensko razdoblje koje predstavlja razdoblje između dva točno određene datuma, primjerice od 15. rujna 2010. godine do 20. siječnja 2011. Ovakvo vremensko razdoblje može se prikazati uređenim parom (t_s, t_f) gdje su t_s i t_f datum početka, odnosno završetka fiksnog vremenskog razdoblja. Formalni zapis člana t_s i člana t_f ovisi o tome želi li se u prikazu početka i završetka vremenskog razdoblja uključiti datum i vrijeme ili samo datum. Bez smanjenja općenitosti, svaki od njih se može prikazati uređenom trojkom koja će sadržavati samo datum, mjesec i godinu[14] (d, m, g) $\in \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ pri čemu vrijednosti *d* i *m* moraju biti takve da tvore valjani datum[15]. Kako bi ovo vremensko razdoblje bilo valjano mora vrijediti $t_s < t_f$, pri čemu je < prirodno definirana relacija usporedbe datuma.

Drugi tip vremenskog razdoblja sličan je fiksnom vremenskom razdoblju u toj mjeri da su dan i mjesec početka i dovršetka razdoblja fiksni, međutim, vremensko razdoblje predstavlja vrijeme između tih datuma u bilo kojoj godini, primjerice od 15. veljače do 1. lipnja svake godine ili primjerice od 1. prosinca do 1. veljače svake godine. Ovakav tip razdoblja predstavlja ponavljajuće vremensko razdoblje te se također može predstaviti

[14] U slučaju da je potrebno dodati i vrijeme, tada se umjesto uređene trojke (dan, mjesec, godina) može koristiti n-torka (dan, mjesec, godina, sati, minute, sekunde).
[15] Provjera ispravnosti datuma može se provjeriti nekim od regularnih izraza na [RegExLib].

uređenim parom (t_s, t_f) gdje su t_s i t_f (sukladno prethodnoj opasci oko ispuštanja sata, minuta i sekundi) definirani kao uređeni parovi oblika (d, m) $\in \mathbb{N} \times \mathbb{N}$, gdje d i m predstavljaju valjani datum i mjesec u nekoj godini. Za dva datuma t_1 i t_2, koji čine jedno ponavljajuće vremensko razdoblje, relacija usporedbe nema smisla, jer ovisi na koju godinu se odnose te u kakvom su međusobnom poretku unutar uređenog para. Primjerice za t_1=(1,3) i t_2=(1,10) koji bi predstavljali 1. ožujka, odnosno 1. listopada usporedba ovisi da li je vremenski period (t_1 , t_2) ili (t_2, t_1). Prvi predstavlja razdoblje od 1. ožujka do 1. listopada u nekoj godini, dok drugi predstavlja period od 1. listopada do 1. ožujka iduće godine, što zorno pokazuje da takvi elementi moraju biti promatrani u kontekstu dodijeljene im godine, to jest u kontekstu proširenja uređenog para dodavanjem trećeg elementa čime bi se dobili datumi koji bi tvorili jedno fiksno razdoblje.

Označi se li se skup svih uređenih trojki koje predstavljaju jedan član uređenog para u prikazu fiksnih vremenskih razdoblja sa F, a sa R skup svih uređenih parova koje predstavljaju jedan član uređenog para u prikazu ponavljajućih vremenskih razdoblja, tada se za svako vremensko razdoblje p $\in$ R $\times$ R može uvesti proširenje na F $\times$ F, parametrizirano datumom t, na način da dobiveno vremensko razdoblje predstavlja fiksno vremensko razdoblje nakon datuma t. Zapišu li se $t_s \in$ R i $t_f \in$ R kao uređeni parovi (d_s,m_s) i (d_f, m_f), a $t \in$ F kao (d,m,g) gdje su d i m vrijednosti za dan i mjesec, tada se proširenje određeno godinom g koja je godina vremena t može definirati funkcijom $extA(t_s, t_f, t)$: R $\times$ R $\times$ F -> F $\times$ F koja je definirana na sljedeći način:

$$extA(t_s, t_f, t) = \begin{cases} \big((d_s, m_s, g), (d_f, m_f, g)\big) \text{ ako je } (d, m, g) < (d_s, m_s, g)\ i\ (d_s, m_s, g) < (d_f, m_f, g) \\ \big((d_s, m_s, g), (d_f, m_f, g+1)\big) \text{ ako je } (d, m, g) < (d_s, m_s, g)\ i\ (d_f, m_f, g) < (d_s, m_s, g) \\ \big((d_s, m_s, g+1), (d_f, m_f, g+1)\big) \text{ ako je } (d_s, m_s, g) < (d, m, g)\ i\ (d_s, m_s, g) < (d_f, m_f, g) \\ \big((d_s, m_s, g+1), (d_f, m_f, g+2)\big) \text{ako je } (d_s, m_s, g) < (d, m, g)\ i\ (d_f, m_f, g) < (d_s, m_s, g) \end{cases}$$

Na sličan način može se definirati proširenje parametrizirano datumom t tako da dobiveno vremensko razdoblje bude prije trenutka t:

$$extB(t_s, t_f, t) = \begin{cases} \big((d_s, m_s, g), (d_f, m_f, g)\big) \text{ ako je } (d_f, m_f, g) < (d, m, g)\ i\ (d_s, m_s, g) < (d_f, m_f, g) \\ \big((d_s, m_s, g-1), (d_f, m_f, g)\big) \text{ ako je } (d_f, m_f, g) < (d, m, g)\ i\ (d_f, m_f, g) < (d_s, m_s, g) \\ \big((d_s, m_s, g-1), (d_f, m_f, g-1)\big) \text{ ako je } (d, m, g) < (d_f, m_f, g)\ i\ (d_s, m_s, g) < (d_f, m_f, g) \\ \big((d_s, m_s, g-2), (d_f, m_f, g-1)\big) \text{ako je} (d, m, g) < (d_f, m_f, g)\ i\ (d_f, m_f, g) < (d_s, m_s, g) \end{cases}$$

Značenje prvog proširenja je u traženje prvog sljedećeg fiksnog vremenskog razdoblja u kojem se element s ponavljajućim vremenskim razdobljem može izvoditi, a nakon određenog vremenskog trenutka t. Slično, drugo proširenje predstavlja najkasnije fiksno vremensko razdoblje u kojem se element s ponavljajućim vremenskim razdobljem može izvoditi, a prije određenog vremenskog trenutka t.

Primjerice, za t = (1,12,2009) i t_s=(1,10) i t_f=(1,3) , $extA(t_s, t_f, t)$ = ((1,10,2010) , (1,3,2011)), a $extB(t_s, t_f, t)$ = ((1,10,2008) , (1,3,2009)). Dakle, ako određeni element koji ima definirane ponavljajuća vremenske razdoblja od 1. listopada do 1. ožujka ne može početi prije 1. prosinca 2009. godine, tada je njegov najraniji mogući početak 1. listopada

2010. godine sa završetkom 1. ožujka 2011. godine. Ako taj element mora biti dovršen do 1. prosinca 2009. godine, tada je njegovo zadnje vremensko razdoblje u kojem se može izvršiti prije tog datuma ono od 1. listopada 2008. godine do 1. ožujka 2009. godine.

Treći tip vremenskog razdoblja predstavlja varijabilna vremenska razdoblja, to jest razdoblja čije se definirano vrijeme može mijenjati, primjerice ovisno o pojedinoj godini. Primjer takvih razdoblja su ljetni i zimski semestar, koji se ponavljaju svake godine, ali s različitim početcima i završetcima. O kojim se konkretnim datumima radi, to jest koja fiksna vremenska razdoblja odgovaraju kojim varijabilnim vremenskim razdobljima, definira se odvojeno te se podaci međusobno povezuju u trenutku izgradnje konačnog modela.

U pogledu konkretne implementacije, svakom elementu koji ima određenu vremensku komponentu potrebno je dodati proširenje koje bi omogućilo ispunjenje pridružene vremenske komponente. Za element koji bi bio pokrenut prerano, potrebno je omogućiti odgodu početka izvršavanja. Odgoda početka izvršavanja omogućit će da element koji bi inače bio ranije aktiviran, primjerice zbog dovršetka svojih prethodnika, zbilja bude aktivan upravo u onom vremenskom razdoblju koje mu je pridruženo. Za element koji bi inače predugo trajao treba postojati mehanizam za njegov završetak, pri čemu treba razlikovati da li je element završen „nasilno“ zbog isteka vremena ili „normalnim“ putem.

Promatrajući izolirano jedan element, može se pojaviti problem da je trajanje prethodnika bilo predugo te je vrijeme za početak izvršavanja već proteklo. Takva situacija ne bi bila valjana i ne smije se dogoditi u konačnom modelu. Slično kao i u izradi mrežnog plana, ako neki element ima definiranu vremensku komponentu, onda i svi njegovi prethodnici/sljedbenici u modelu moraju implementirati vremensku komponentu da bi krajnji rokovi početaka i završetaka bili takvi da svi elementi mogu početi na vrijeme. Stoga je za svaki od elemenata u konačnom modelu potrebno pronaći najranija i najkasnija moguća vremena početka i završetka. Traženjem vremena početaka i završetaka moći se utvrditi da neki element nije izvodljiv ako bi za izvršavanje njegovih prethodnika bilo potrebno previše vremena. Pri problemu raspoređivanja, određivanjem krajnjih vremena početka i završetka svakog elementa mogu se precizno locirati vremenska razdoblja u kojima se taj element može izvoditi. Na taj način moguće je smanjiti broj vremenskih razdoblja pridruženih elementima, samim time i mogući broj kombinacija, što smanjuje složenost problema raspoređivanja.

4.2 Traženje vremena početaka i završetaka elemenata

Prema pretpostavci iz prethodnog poglavlja, sva varijabilna vremenska razdoblja u trenutku izrade modela bit će povezana s fiksnim vremenskim razdobljima. Stoga, može se reći da je svaki element u konačnom modelu vremenski definiran svojim fiksnim i/ili ponavljajućim vremenskim razdobljima.

Uz notaciju iz prethodnog poglavlja, skup svih vremenskih razdoblja T tako je definiran kao skup uređenih parova, gdje svaki par predstavlja jedno fiksno ili jedno ponavljajuće razdoblje, pa je

$$T = \{ (t_S, t_F) \mid (t_S , t_F \in F \times F \wedge t_s < t_F) \vee (t_S , t_F \in R \times R)\}$$

Prikaže li se konačni model pomoću usmjerenog grafa, tada je svakom vrhu usmjerenog grafa $G(V, A)$ pridruženo nula, jedan ili više vremenskih razdoblja. Označi li se partitivni skup skupa T s $\mathcal{P}$(T) = {X | X ⊂ T} , tada funkcija *tp* svakom element $v \in V$ pridružuje neki element iz partitivnog skupa skupa T, to jest *tp*:V-> $\mathcal{P}$(T).

Nepostojanje vremenskih razdoblja pridruženih nekom elementu, još uvijek ne mora značiti da taj element neće imati ugrađenu vremensku komponentu, jer to ovisi o njegovim prethodnicima odnosno sljedbenicima. U slučaju da neki od njegovih sljedbenika ima definirana fiksna vremenska razdoblja, tada će i promatrani element morati dobiti krajnji rok završetka. Činjenica da neki od prethodnika ima definirana vremenska razdoblja ne mora nužno utjecati na element, osim u obliku smanjivanja broja mogućih vremenskih razdoblja. Promjenu odgode početka nekog elementa na neko kasnije vremensko razdoblje na osnovu izračunatih vremena prethodnika nije nužno obaviti, jer je ionako početak elementa određen završetkom svojim prethodnika.

Tablica 4 pokazuje moguće kombinacije koje utječu na (ne)postojanje vremenske komponente pojedinog elementa. Promatrajući jedan element, tri su mogućnosti: element nema vremenskih razdoblja, element ima samo ponavljajuća vremenska razdoblja, element ima barem jedno fiksno vremensko razdoblje. Promatrajući njegove sljedbenike, također je moguće razlikovati navedena tri slučaja. Tako prvi stupac predstavlja situaciju u kojoj nijedan od sljedbenika nema definirano vremensko razdoblje, drugi stupac situaciju u kojoj među sljedbenicima postoje samo ponavljajuća vremenska razdoblja te treći stupac predstavlja situaciju u kojoj postoji barem jedan sljedbenik koji ima definirano fiksno vremensko razdoblje.

Tablica 4. (Ne)postojanje vremenske komponente u ovisnosti o sljedbenicima elementa

	nema vr.r.	samo pon.vr.r.	postoji f.vr.r.
nema vremenskih razdoblja	-	-	++
samo ponavljajuća vr. razdoblja	+	+	++
postoji fiksno vr. razdoblja	+	+	++

U slučajevima označenim s -, element ne treba dobiti vremensku komponentu, jer nije vremenski uvjetovan. U slučajevima označenim s +, element treba proširiti kako bi zadovoljio uvjete vremenske komponente, ali je izrada i funkcioniranje proširenja za navedeni element neovisna o ostalim elementima u modelu. Za slučajeve označene s ++ potrebno je uskladiti vremensku komponentu tog elementa s vremenskim razdobljima ostalih elemenata u modelu.

Postojanje vremenske komponente nekog elementa određeno je postojanjem i vrstom vremenskih komponenti njegovih sljedbenika. Vremenska razdoblja njegovih prethodnika ne utječu na vrstu njegove vremenske komponente. Primjerice, situacija u kojoj neki element ne može početi prije nekog trenutka (jer njegovi prethodnici zasigurno neće biti dovršeni ranije) i postojanje vremenskih razdoblja za vrijeme prije tog trenutka, samo po sebi nije problematično. Ta vremenska razdoblja će u tom elementu jednostavno biti zanemarena te ta činjenica, osim ako to nisu jedina vremenska razdoblja, nema utjecaj na izvršavanje elementa. Ipak, brisanje takvih razdoblja iz elementa može utjecati na smanjenje broja mogućnosti prilikom eventualnog raspoređivanja, pa će za svaki element biti pronađeni, ako takvi postoje, najraniji i najkasniji mogući početci i završetci.

Budući da se ovisnost može javiti i o prethodnicima i o sljedbenicima, traženje najranijih i najkasnijih vremena početaka i završetaka nije moguće obaviti u jednom prolazu kroz graf. U prvom prolazu računat će se najranije moguće vrijeme početka i završetka, a u drugom najkasnije dozvoljeno vrijeme početka i završetka pojedinog elementa.

Bitno je naglasiti da izračun najranijih i najkasnijih vremena opisan u poglavljima 4.2.1 i 4.2.2 pretpostavlja da su svi elementi u promatranom grafu povezani relacijom preduvjeta. Razlog je detaljnije objašnjen u 4.2.3, nakon pregleda algoritama za izračun vremena početaka i završetaka.

4.2.1 Izračun najranijih mogućih vremena

Algoritam za izračun najranijih mogućih početaka i najranijih mogućih završetaka pojedinih elemenata za pretpostavku uzima da je graf $G(V, A)$ konačnog modela povezan i bez ciklusa. Uvjet povezanosti može se osigurati stvaranjem dvaju novih vrhova u grafu, *Start* i *End*, koji bi bili povezani sa svim vrhovima ulaznog stupnja 0, odnosno svim vrhovima izlaznog stupnja 0.

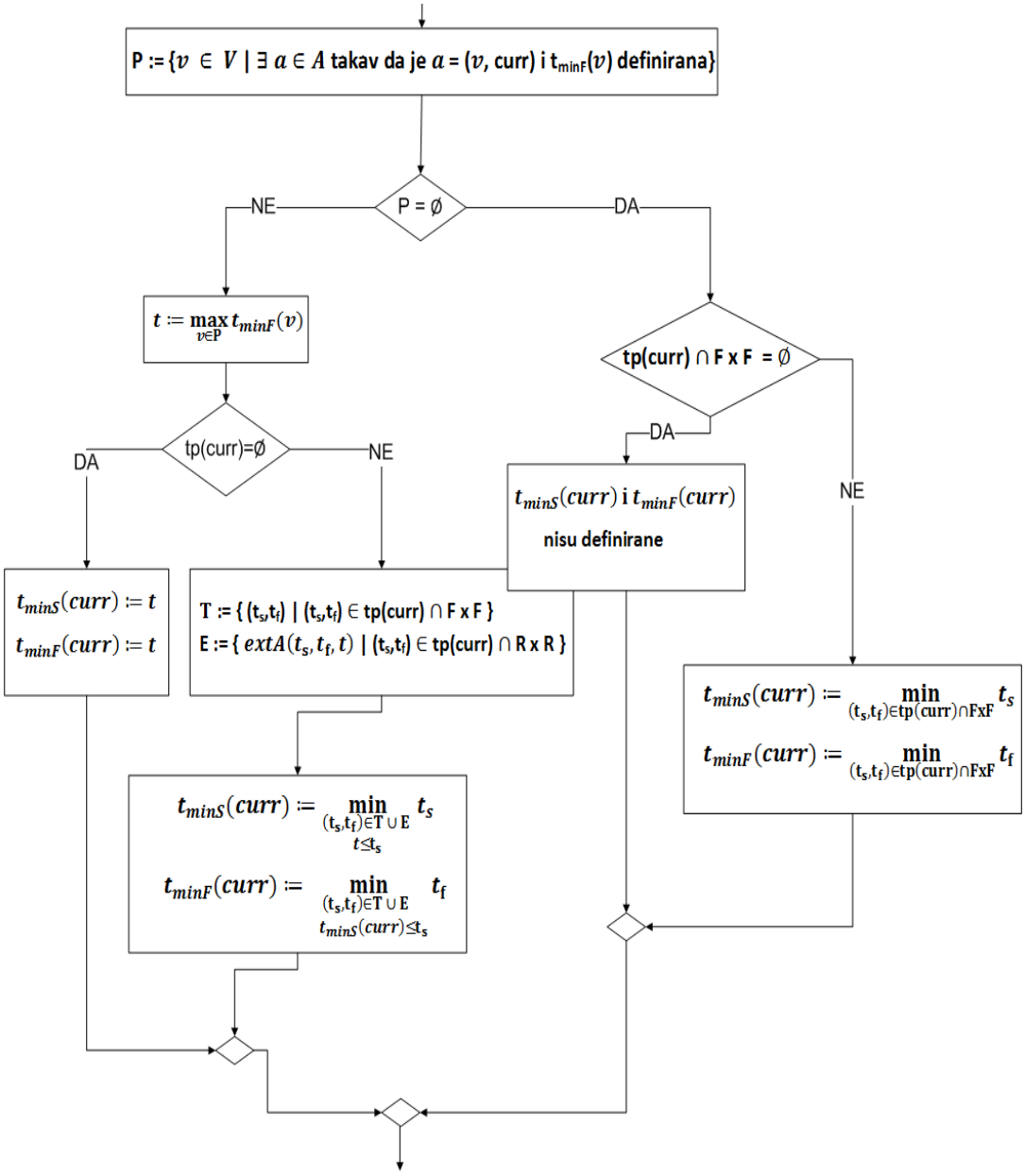
Algoritam počinje od početnog vrha u grafu te se ostali vrhovi uzimaju u topološkom poretku kako bi se osiguralo da su prije obrade nekog vrha obrađeni svi njegovi prethodnici. Za svaki od vrhova mogu se definirati sljedeće funkcije:

$$t_{minS} : V \rightarrow \mathrm{F} \text{ i } t_{minF} : V \rightarrow \mathrm{F}$$

koja pojedinom vrhu pridružuju vrijeme najranijeg mogućeg početka i najranijeg mogućeg završetka pojedinog elementa. Funkcija ne mora biti definirana za svaki vrh u grafu. Primjerice, ako se ne radi raspoređivanje po fiksnim vremenskim razdobljima, tada za početni vrh ove funkcije nisu definirane.

Slika 18 prikazuje korake obrade trenutno promatranog vrha označenog s *curr*. Za trenutni vrh, skup P se definira kao skup svih prethodnika trenutnog vrha, takvih da imaju definirano najranije moguće vrijeme dovršetka.

$$\mathrm{P} := \{v \in V \mid \exists\, a \in A \text{ takav da je } a = (v, \mathrm{curr}) \text{ i } t_{minF}(v) \text{ definirana}\}$$



Slika 18. Izračun najranijeg početka i najranijeg završetka trenutnog elementa

U slučaju da je skup P bio prazan tada postojanje vrijednosti funkcija t_{minS} i t_{minF} ovisi o tome postoji li u skupu tp(curr) par iz skupa F x F. U tom slučaju je

$$t_{minS}\ (curr) := \min_{(t_s,t_f)\in tp\,(curr\ )\cap FxF} t_s$$

Najranije vrijeme dovršetka ne mora biti iz onog vremenskog razdoblja koje odgovara najranijem mogućem početku, već je

$$t_{minF}\ (curr) := \min_{(t_s,t_f)\in tp\,(curr\ )\cap FxF} t_f$$

U slučaju da za trenutni element skup P nije prazan skup (lijeva grana na Slika 18), tada se vrijeme t prije kojeg trenutni element nije mogao početi može izračunati kao najkasnije moguće vrijeme među najranijim vremenima početaka prethodnika iz skupa P.

$$t \coloneqq \max_{v \in \mathrm{P}} t_{minF}\ (v)$$

Za trenutni vrh mora vrijediti da je $t \leq t_{minS}(curr)$. Ako je skup vremenskih razdoblja za trenutni vrh prazan skup tada je $t_{minS}(curr) := t$ i $t_{minF}(curr) := t$, kako bi se ove vrijednosti mogle prenijeti na sljedbenike trenutnog vrha.

Ako je element imao definirana vremenske razdoblja, tada se navedene vrijednosti moraju izračunati koristeći vrijednosti u vremenskim razdobljima koja mogu biti iz skupa F x F ili iz skupa R x R.

Izračun najranijeg mogućeg početka i završetka je u ovom slučaju sličan, ali treba uzeti u obzir da početno vrijeme traženog perioda mora biti nakon vremena t. Neka je T skup svih vremenskih razdoblja trenutnog elementa takvih da su iz F x F, a skup E skup proširenih vremena iz R x R

$$\mathrm{T} := \{ (t_s,t_f) \mid (t_s,t_f) \in tp(curr) \cap \mathrm{F} \times \mathrm{F} \}$$
$$\mathrm{E} := \{ extA(t_s, t_f, t) \mid (t_s,t_f) \in tp(curr) \cap \mathrm{R} \times \mathrm{R} \}$$

Tada se vremena najranijeg početka i najranijeg završetka računaju na sljedeći način:

$$t_{minS}\ (curr) \coloneqq \min_{\substack{(t_s,t_f) \in \mathrm{T} \cup \mathrm{E} \\ t \leq t_s}} t_s$$

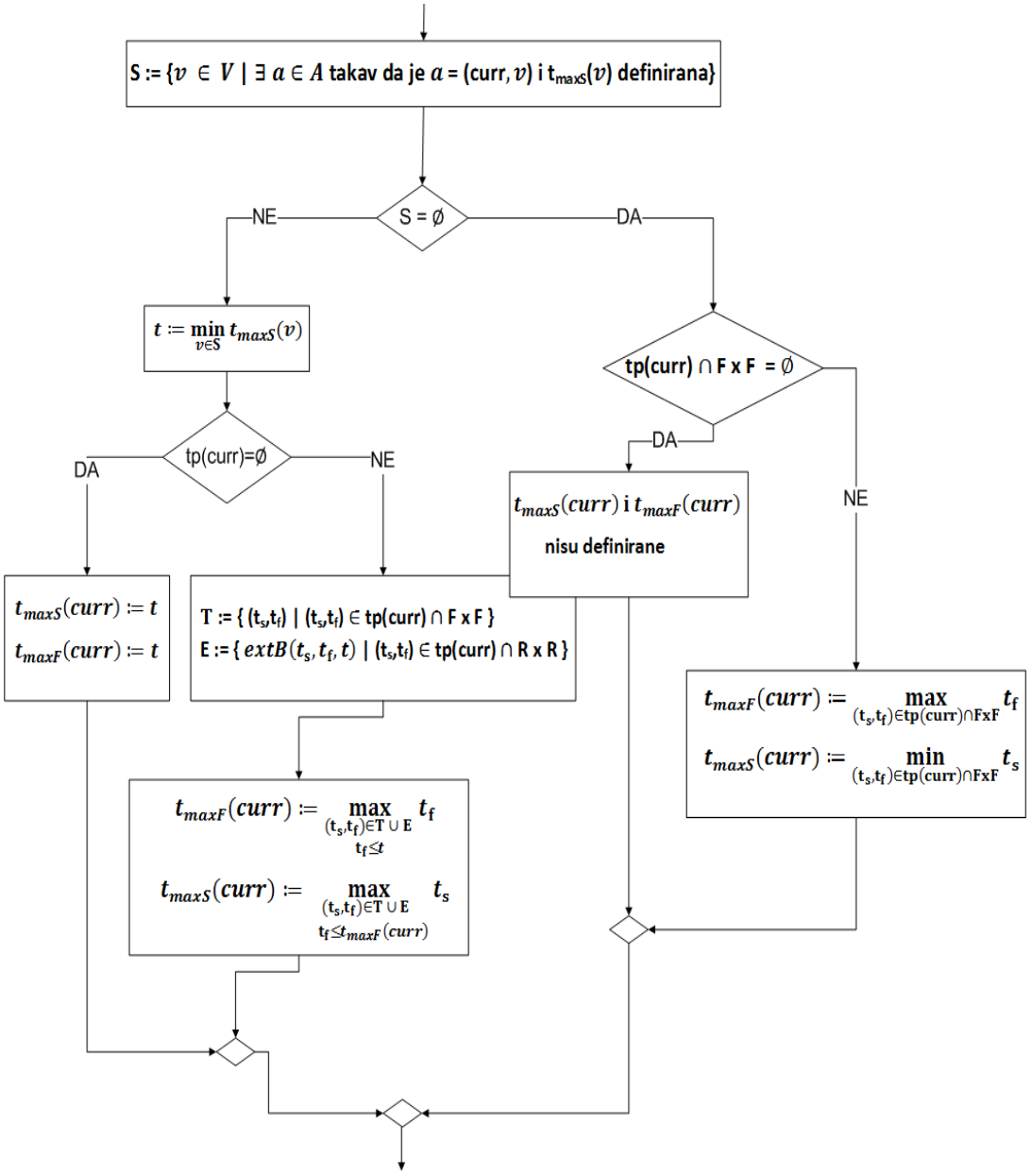
$$t_{minF}\ (curr) \coloneqq \min_{\substack{(t_s,t_f) \in \mathrm{T} \cup \mathrm{E} \\ t_{minS}\ (curr) \leq t_s}} t_f$$

4.2.2 Izračun najkasnijih dozvoljenih vremena

Izračun najkasnijeg dozvoljenog početka i najkasnijeg dozvoljenog završetka pojedinog elementa sličan je izračunu najranijih mogućih vremena. Najčešće se koriste suprotni operatori od onih koji su korišteni za najranija vremena (min umjesto max i obrnuto, početak umjesto završetka), a elementi se uzimaju u obrnutom redoslijedu počevši od završnog vrha grafa za kojeg, kao i kod početnog vrha ne moraju biti definirane vrijednosti najkasnijeg dozvoljenog početka i najkasnijeg mo dozvoljenog gućeg završetka. U ovom koraku definiraju se dvije nove funkcije

$t_{maxS} : V \rightarrow \mathrm{F}$ i $t_{maxF} : V \rightarrow \mathrm{F}$

koje pojedinom vrhu pridružuju vrijeme najkasnijeg dozvoljenog početka, odnosno najkasnijeg dozvoljenog završetka kako bi element mogao biti dovršen prije vremena u kojem njegovi sljedbenici trebaju početi, ako je takav podatak definiran. Slika 19 prikazuje korake obrade pojedinog, trenutno promatranog vrha označenog s *curr*.



Slika 19. Izračun najkasnijeg mogućeg početka i najkasnijeg mogućeg završetka trenutnog elementa

Za trenutni vrh, skup S se definira kao skup svih sljedbenika trenutnog vrha, takvih da imaju definirano najkasnije dozvoljeno vrijeme početka.

S := {$v \in V$ | ∃ $a \in A$ takav da je a = (curr, v) i $t_{maxS}(v)$ definirana}

U slučaju da je skup S bio prazan tada postojanje vrijednosti funkcija t_{maxS} i t_{maxF} ovisi o tome postoji li u skupu tp(curr) par iz skupa F x F. U tom slučaju je najkasnije vrijeme završetka jednako najvećem (to jest najkasnijem) vremenu završetka.

$$t_{maxF}\,(curr) := \max_{(t_s,t_f)\in tp\,(curr\,)\cap FxF} t_f$$

Najranije vrijeme početka ne mora biti iz onog vremenskog perioda koji odgovara najkasnijem dozvoljenom dovršetku, već je

$$t_{maxS}\ (curr) \coloneqq \max_{(t_s,t_f)\in tp\,(curr\ )\cap FxF} t_s$$

Ako za trenutni vrh skup S nije prazan skup (lijeva strana slike), tada se vrijeme t do kojeg trenutni element mora biti završen izračunava promatrajući najkasnije moguće početke njegovih sljedbenika i odabirom najranije među tim vrijednostima.

$$t \coloneqq \min_{v\in S} t_{maxS}\ (v)$$

Za trenutnih vrh mora vrijediti $t_{maxF}(curr) \leq t$. Ako je skup vremenskih razdoblja za trenutni vrh prazan skup tada je $t_{maxS}(curr) := t$ i $t_{maxF}(curr) := t$, kako bi se ove vrijednosti mogle prenijeti na prethodnike trenutnog vrha. Ako je skup imao definirana vremenska razdoblja, tada se navedene vrijednosti moraju izračunati koristeći vrijednosti u vremenskim razdobljima koja mogu biti iz skupa F x F ili iz skupa R x R.

Izračun najkasnijeg dozvoljenog početka i završetka je u ovom slučaju sličan, ali treba uzeti u obzir da vrijeme završetka traženog vremenskog razdoblja mora biti prije vremena t. Neka je T skup svih vremenskih perioda trenutnog elementa takvih da su iz F x F, a skup E skup proširenih vremena iz R x R, gdje je proširenje načinjeno tako da osigura da se prošireno vremensko razdoblje nalazi prije vremena t.

$T := \{ (t_s,t_f) \mid (t_s,t_f) \in tp(curr) \cap F \times F \}$
$E := \{ extB(t_s, t_f, t) \mid (t_s,t_f) \in tp(curr) \cap R \times R \}$

Tada se vremena najkasnijeg početka i najkasnijeg završetka računaju na sljedeći način:

$$t_{maxF}\ (curr) \coloneqq \max_{\substack{(t_s,t_f)\in T\ \cup\ E \\ t_f \leq t}} t_f$$

$$t_{maxS}\ (curr) \coloneqq \max_{\substack{(t_s,t_f)\in T\ \cup\ E \\ t_f \leq t_{maxF}\ (curr\ )}} t_s$$

4.2.3 Opaska oko vremena pridruženih elementima u skupovima mogućih prethodnika

Razlog zašto skupove mogućih prethodnika nije moguće koristiti u algoritmima iz poglavlja 4.2.1 i 4.2.2 bez dodatnih preinaka leži u nedeterminističkom ponašanju takvih skupova. I dok je za vremena najranijeg početka nekog elementa moguće promatrati prvih r vremena u nekom skupu mogućih prethodnika, gdje je r rang tog skupa mogućih prethodnika, a zatim kao najraniji završetak tog skupa odabrati r-to po redu vrijeme, slično nije moguće izvesti u obrnutom smjeru. Krajnje vrijeme početka i završetka nekog elementa u skupu mogućih prethodnika nije moguće odrediti, jer taj podatak direktno ovisi da li će u konkretnom slučaj taj element biti prethodnik nekom elementu ili ne. U slučaju da promatrani element neće biti prethodnik nekog elementa, tada bi njegov krajnji rok

završetka mogao biti duži, nego pretpostavi li se da će biti nužan za izvršenje elementa kojem je potencijalni prethodnik. Pretpostavi li se da će svi elementi iz skupa mogućih prethodnika zaista biti prethodnici nekog elementa, to jest doda li ih se u graf kao preduvjete, dobivena vremena će biti restriktivnija nego što bi zaista mogla biti.

Način kako bi se problem mogao riješiti svodi se na bilježenje više vremena za pojedine elemente. Svi elementi bi imali krajnje vrijeme završetka koje bi bilo uvjetovano isključivo onim elementima kojima je taj element preduvjet, dok bi se za pojedini skup mogućih prethodnika bilježilo posebno vrijeme te bi svi elementi u skupu mogućih prethodnika imali upravo to vrijeme završetka, ali bi se to vrijeme odnosilo samo na situaciju kad se element pojavljuje kao dio tog skupa elemenata (poglavlje 5.3), a ne i na sva ostala pojavljivanja tog elementa. Ako bi se primijenilo navedeno rješenje to bi značilo da se prethodni algoritmi moraju modificirati na način da se za najraniji početak nekog elementa umjesto

$$t \coloneqq \max_{v \in P} t_{minF}\,(v)$$

koristi

$$t \coloneqq \max\{\max_{v \in P} t_{minF}\,(v)\,, \max\,(minimalna\ vremena\ skupova\ mogućih\ prethodnika)\}$$

Pritom pri izračunu krajnjeg vremena početka i završetka elementa osim uobičajenih vrijednosti za neke elemente treba evidentirati trojke oblika (skup mogućih prethodnika, t_{maxS}, t_{maxF}). Ova opaska primjenjuje se samo za one skupove mogućih prethodnika koji integracijom u zajednički graf ne tvore ciklus, jer u protivnom vremena nije moguće izračunati.

4.3 Problem raspoređivanja

Uvođenje vremenske komponente nije samo posebnost pojedinog elementa. Također, vremenska komponenta se može uvesti na razini cijelog konačnog modela, odnosno moguće je da se elementi iz modela trebaju podijeliti u nekoliko fiksnih vremenskih razdoblja (u daljnjem tekstu vremenskih okvira radi razlike u odnosu na vremenska razdoblja pridružena elementima), koji međusobno nemaju preklapanja, pri čemu pojedini elementi, u ovisnosti o postavljenim ograničenjima, mogu biti u nekoliko uzastopnih vremenskih okvira. Primjerice, radi li se o izradi studijskog programa, elementi mogu imati vremensku komponentu na način da se predaju samo u zimskom ili samo u ljetnom semestru ili su možda višesemestralni. U tom slučaju nije dovoljno samo implementirati vremensku komponentu na pojedinom elementu s vremenski uvjetovanim početkom i završetkom, već ih je potrebno i razdijeliti u vremenske okvire, koji u tom primjeru imaju značenje semestra, pritom poštujući pravila za pojedine semestre. Svaki od semestara predstavlja određeni vremenski okvir, koji ima svoja određena pravila. U ovom slučaju to može biti broj sati opterećenja, broj ECTS bodova, broj predmeta i slično. Gledajući općenito, svaki od vremenskih okvira može imati svoje posebnosti i svoja validacijska pravila.

Usporediti raspoređivanje elemenata iz integralnog modela u vremenske okvire s klasičnim raspoređivanjem po grupama nije jednostavno. U oba slučaja, radi se o NP problemu, a postojanje vremenske komponente može utjecati na smanjenje problema na način da se za neke elemente može unaprijed odrediti da moraju biti u točno određenoj grupi, to jest vremenskom okviru, čime se uz veći broj takvih ograničenja veličina problema može smanjiti u prihvatljive okvire (pri čemu se pod terminom prihvatljiv smatra dovoljno mala brojka da bi vremenski bilo izvedivo upotrijebiti algoritam koji bi provjeravao sve moguće kombinacije). Također, postojanje međusobne ovisnosti među elementima, povećava složenost algoritma zbog dodatnog zahtjeva za provjeru redoslijeda za svaki rezultat, jer dopušteni vremenski okviri ovise ne samo o pojedinom elementu, nego i vrijednostima koje su dodijeljene njegovim prethodnicima i sljedbenicima, čime provjera svakog elementa ukupno povećava složenost algoritma za jedan red veličine, to jest, konkretno u ovom slučaju za O(broj lukova u grafu), jer je zbog tranzitivnosti uvjeta preduvjeta dovoljno promatrati samo neposredne preduvjete. Neki klasični algoritmi, poput algoritma za problem naprtnjače, također nisu primjenjivi, jer primjerice, ako se radi o odabiru predmeta za pojedini semestar, uzimanjem onih predmeta koji pridonose maksimalnoj popunjenosti semestra, mogu uzrokovati da se kasnije neki elementi ne mogu rasporediti, jer nije preostalo dovoljno semestara. Kao suprotnost, favoriziranje elementa koji imaju najviše sljedbenika može dovesti do problema fragmentacije. Zbog svega navedenog, priroda problema sugerira upotrebu genetskog algoritma, pri čemu je potrebno algoritam konstruirati tako da omogućava rješavanje različitih problema uz minimalne promjene.

Model ostavlja mogućnost da se definira i neki drugačiji genetski algoritam, koji bi koristio drugačije načine prikaza potencijalnih rješenja (drugačiji zapis kromosoma), a samim time i drugačije operatore mutacije i križanja. Općenitije rješenje, koje bi dopuštalo zamjenu genetskog algoritma s nekim drugim načinom rješavanja optimizacijskih problema dano je u [Vanjak2006].

4.3.1 Genetski algoritmi

Genetski algoritmi su heuristička metoda optimiranja inspirirana prirodnim evolucijskim procesom. Analogija s prirodnom biološkom evolucijom temelji se na procesu selekcije i odgovarajućim genetskim operatorima nasljeđivanja i mutacije. Začeci ideje vežu se uz [Holland1975], a kroz proteklo vrijeme pokazali su se kao efikasan, a istovremeno dovoljno općenit i jednostavan način za rješavanje problema u različitim optimizacijskim problemima (traženje najkraćih putova, problem raspoređivanja, transportni problemi…). Kako genetski algoritmi pripadaju heurističkoj metodi, za dobivanje željenih rezultata potrebno je varirati parametre genetskog algoritma ovisno o pojedinom problemu te po potrebi izvesti genetski algoritam više puta.

Princip rada genetskog algoritma svodi se na rad s operacijama nad populacijom jedinki u nekoj određenoj generaciji. Svaka jedinka predstavlja potencijalno rješenje problema koji se obrađuje, primjerice jedinka u problemu raspoređivanja može predstavljati neki raspored elemenata (ne nužno i ispravan). Sve jedinke se predstavljaju istim tipom podatka: brojem,

nizom brojeva, matricom ili slično. Po analogiji s biološkim sustavom, jedinke se još nazivaju kromosomima. Svakoj jedinki pridružuje se mjera kvalitete, koja se naziva dobrota (eng. *fitness*). Funkcija koja određuje navedenu kvalitetu, naziva se funkcija dobrote ili funkcija cilja. Nakon što je svakoj jedinki u populaciji pridružena dobrota, slijedi formiranje nove populacije izdvajanjem, po određenim kriterijima, nekih jedinki iz stare populacije, pri čemu bolje jedinke (u smislu dobrote) imaju veću vjerojatnost da budu izdvojene i očuvane. Nakon toga, neke od novih jedinki podvrgnute su utjecaju genetskih operatora, koji iz njih stvaraju nove jedinke. Operatori nad jedinkama mogu biti unarni, kao što je *mutacija*, koja mijenja dio neke jedinke ili operatori višeg reda, kao što su *operatori križanja* koji stvaraju nove jedinke kombinirajući osobine više jedinki.

Postupak provedbe genetskog algoritma počinje stvaranjem početne populacije i pridjeljivanjem vrijednosti dobrote toj populaciji, a završava kada se zadovolji uvjet zaustavljanja. Uvjet zaustavljanja može biti pronalazak traženog rješenja (primjerice traži li se barem jedan mogući raspored) ili ako je proveden određeni broj iteracija. Na kraju postupka najbolji član trenutne populacije predstavlja traženo rješenje ili je po odabranom kriteriju za funkciju dobrote najbliže traženom rješenju u odnosu na ostale jedinke u populaciji. Genetski algoritmi se s obzirom na vrstu selekcije, dijele na generacijske i eliminacijske. Generacijski genetski algoritam u svakoj iteraciji stvara novu populaciju selekcijom najboljih jedinki iz prethodne populacije te njihovim križanjem i mutacijom. Eliminacijski genetski algoritam ne stvara čitavu novu populaciju već iz postojeće izbacuje najlošije jedinke i križanjem preostalih ih nadomješta. Tablica 5 prikazuje pseudokod eliminacijskog genetskog algoritma.

Tablica 5. Pseudokod eliminacijskog genetskog algoritma [Golub2004]

```
Eliminacijski genetski algoritam
{
  generiraj početnu populaciju
  sve dok nije zadovoljen uvjet završetka evolucijskog procesa
  {
    izračunaj vjerojatnost eliminacije za svaku jedinku;
    M puta jednostavnom selekcijom odaberi i izbriši jedinku
    parenjem preživjelih jedinki nadopuni populaciju
  }
  ispiši rješenje;
}
```

U [Kalpić2006] korišten je eliminacijski genetski algoritam s turnirskom selekcijom [Goldberg1991] te se pokazao dobar za probleme izrade satnice na poslijediplomskom studiju Fakulteta elektrotehnike i računarstva u Zagrebu te se koristi i u ovom radu za problem raspoređivanja elemenata. U svakoj od iteracija genetskog algoritma slučajnim odabirom biraju se 3 jedinke iz populacije. Najlošija jedinka među odabranim mijenja se jedinkom nastalom križanjem drugih dviju jedinki. Istovremeno, te dvije jedinke imaju određenu vjerojatnost mutacije kojom može doći do mutacije jednog ili više gena u kromosomu.

Navedeni princip selekcije korišten je i u ovom radu uz modifikaciju rada algoritma prilikom pojave jedinke koja već postoji u populaciji. U tom slučaju ista jedinka se odbacuje i zamjenjuje sa slučajno odabranom jedinkom, kako bi se izbjegla pojava duplikata i usporavanje algoritma [Golub2004].

4.3.2 Prikaz kromosoma i genetski operatori

Kromosom, kao genetski zapis pojedine jedinke, može biti bilo kakva struktura podataka koja sa što manje gena opisuje svojstva te jedinke te istovremeno ima definirane genetske operatore mutacije i križanja koji ne dovode do stvaranja novih jedinki koje bi predstavljale nemoguća rješenja, jer se time znatno smanjuje učinkovitost genetskog algoritma. Najčešći prikaz kromosoma je u obliku binarnog prikaza, međutim ovisno o problemu, prikaz može biti i drugačiji, poput matrica ili nizova cijelih ili realnih brojeva i slično.

Za problem provjere da li je problem postojanja međusobnih isključivanja i skupa mogućih prethodnika opisan u poglavlju 3.5 rješiv pogodan je binarni prikaz. Numeriraju li se elementi redom od 1 do broja elemenata, tada vrijednost 1 na mjestu *i* označava da je i-ti element dodan u konačni model, a vrijednost 0 da se i-ti element ne nalazi u konačnom modelu. Početna populacija stvara se slučajnim odabirom vrijednosti 0 ili 1 za svaku od pozicija u binarnom prikazu. Mogući operatori križanja su križanje u jednoj točki (eng. *single-point crossover*), križanje u više točaka (eng. *multi-point crossover*) i uniformno križanje (eng. *uniform crossover*). U slučaju križanja u jednoj točki, slučajnim odabirom se bira pozicija *p* unutar kromosoma, nakon čega jedinka nastala križanjem dviju jedinki na prvih *p* mjesta sadrži gene iz jednog roditelja, a nakon toga gene iz drugog roditelja. Križanje u više točaka naizmjence uzima gene iz roditelja između pojedinih slučajno odabranih točaka prekida. Kod uniformnog križanja svaki gen nove jedinke je nastao slučajnim odabirom gena jednog od roditelja. Operator mutacije svodi se na slučajnu promjenu određenog broja gena u kromosomu, što u ovom slučaju predstavlja promjenu vrijednosti 0 i 1 za neki broj pozicija u binarnom prikazu. Vjerojatnost mutacije i broj bitova za promjenu određeni su parametrima genetskog algoritma.

U problemu raspoređivanja elemenata u vremenske okvire, kromosom može biti predstavljen kao niz cijelih brojeva, pri čemu vrijednost k na mjestu i, označava da je i-ti element pridružen k-tom vremenskom okviru, pri čemu su elementi prethodno numerirani redom od 1 do broja elemenata i vremenski okviri od 1 do broja vremenskih okvira. Za ovaj problem raspoređivanja prikaz kromosoma pomoću binarnog zapisa nije prikladan. Kad bi svakom elementu pridružena vrijednost vremenskog okvira bila prikazana binarnim zapisom, za svaki takav zapis trebalo bi $\lfloor \log_2(T-1) \rfloor + 1$ binarnih znamenki, gdje je T broj vremenskih okvira. Takav prikaz je neprikladan za one vrijednosti broja T koje su tek nešto veće od neke potencije broja 2, jer u tom slučaju veliki postotak kombinacija daje vrijednost koja se ne nalazi u skupu kodiranih vrijednosti. Primjerice, 9 vremenskih okvira iziskuje 4 binarne znamenke za svaki element, što znači da 7 od 16 kombinacija tvori nepostojeću vrijednost. Drugi nedostatak je postojanje nedozvoljenih vrijednosti na razini pojedinog elementa, jer svaki element ima svoj skup dozvoljenih vrijednosti. Tako je, primjerice, moguće da samo dvije od 16 kombinacija imaju smisla i slično. Alternativno, binarni zapis ne

mora predstavljati redni broj vremenskog okvira, nego poziciju u nizu dozvoljenih vrijednosti, čime se veličina zapisa smanjuje, ali problem vrijednosti koje se ne mogu dekodirati (to jest pravilno interpretirati) i dalje ostaje isti.[16]

Kod stvaranje inicijalne populacije niz brojeva se puni slučajno odabranim brojevima, pri čemu se izbor slučajnih brojeva sužava samo na one vrijednosti koje predstavljaju valjane vremenske okvire za pojedini element. Operator križanja je isti kao i kod operatora križanja za binarni zapis, dok se operator mutacije svodi na zamjenu jedne ili više vrijednosti u kromosomu s onim vrijednostima koje su dozvoljene za pojedini element. Iz navedenih razloga za svaki element je potrebno odrediti skup dozvoljenih vrijednosti, to jest odrediti da li neki element može pripadati određenom vremenskom okviru. Time se za svaki element formira popis vremenskih okvira u kojima se može izvoditi te se samo vrijednosti s tog popisa mogu pojaviti u potencijalnom rješenju.

Provjera se vrši ispitivanjem odnosa vremenskih razdoblja pridruženih pojedinom elementu i vremenskih okvira za koje se vrši raspoređivanje. Trivijalni slučaj je onaj u kojem postoji takvo vremensko razdoblje da je u potpunosti sadržano unutar nekog vremenskog okvira. Budući da vremenski okviri i vremenska razdoblja pojedinih elemenata mogu biti neovisno definirani, moguće su i situacije u kojima trajanje pojedinog vremenskog razdoblja premašuje trajanje pojedinog vremenskog okvira. Primjer takve situacije je dvosemestralni predmet u studijskom programu. U tom slučaju provjera je složenija i ovisi o svojstvima vremenskog okvira i svojstvima samog elementa. Da bi element mogao pripadati takvom vremenskom okviru mora postojati barem jedno vremensko razdoblje pridruženo tom elementu takvo da vrijedi:

a) svojstvo *AllowLongElements* navedenog vremenskog okvira kao i svih okvira kroz koje se element proteže mora biti postavljeno na vrijednost *true*.
b) svojstvo *CanSpanAcrossMorePeriods* navedenog elementa mora biti *true*.
c) Vremensko razdoblje je u potpunosti sadržano unutar unije nekoliko uzastopnih vremenskih okvira

Ukoliko se neki element proteže kroz nekoliko vremenskih okvira, za redni broj okvira dodijeljenog tom elementu uzima se redni broj prvog okvira, a prilikom izračuna vrijednosti funkcije dobrote element se evidentira u nekim ili svim vremenskim okvirima u ovisnosti o svojstvu elementa. Nakon što se svakom elementu pridruži skup dozvoljenih vrijednosti može se prijeći na odabir početne populacije i na izvođenje genetskog algoritma.

[16] U slučaju da se ipak koristi neki od algoritama koji pojedinom elementu može dodijeliti nedozvoljenu vrijednost, funkcija dobrote bi takvo rješenje trebala označiti neprihvatljivim i „kazniti“ s malom vrijednošću dobrote, pa će u sljedećim križanjima takvo rješenje imati manju šansu za selekciju i preživljavanje.

4.3.3 Funkcija dobrote

Za svaku jedinku iz populacije potrebno je definirati funkciju dobrote te provjeriti predstavlja li jedinka ispravno rješenje. Za problem iz poglavlja 3.5 jedinka sadrži vrijednosti 0 ili 1 pri čemu vrijednost 1 na mjestu *i* označava da je i-ti element odabran za dodavanje u konačni model. U [Kalpić2006] za izračun funkcije dobrote korišten je broj kolizija u rasporedu, pa se po analogiji može koristiti suma broja međusobno isključivih parova koji su se pojavili u potencijalnom rješenju i broja neostvarenih uvjeta u skupovima mogućih prethodnika pri čemu veći broj predstavlja lošiju vrijednost funkcije dobrote.

Za problem raspoređivanja u vremenske okvire algoritam će završiti pri prvom pronalasku jedinke koja predstavlja valjano rješenje. Provjera da li su zadovoljeni svi uvjeti iz parcijalno definiranih modela provedena je prilikom stvaranja konačnog modela te je potrebno utvrditi da li je raspoređivanje pravilno izvedeno. O ispravnosti rasporeda ovisi da li neka jedinka predstavlja valjano rješenje i, ako ne predstavlja, koliko je to rješenje loše. Jedan od uvjeta koje treba provjeriti je taj da li su elementi u pravilnom poretku. Primjerice, neispravno rješenje je ono u kojem je redni broj vremenskog okvira dodijeljenog nekom elementu manji od rednog broja vremenskog okvira nekog od njegovih prethodnika. Stoga, potrebno je provjeriti koliko elemenata nije u pravilnom poretku.

Osim poretka, valjanost neke jedinke ovisi i domeni područja za koje se vrši raspoređivanje. Primjerice, radi li se raspored sati i ako se teži da su dnevna opterećenja približno jednaka, tada se potencijalno rješenje, koje cijeli raspored svede na jedan ili dva dana u tjednu, ne može smatrati dobrim. Sličan primjer je raspoređivanje predmetima po semestrima gdje se teži da se u svakom semestru nalazi predmeta u vrijednosti 30 bodova, pri čemu se manja odstupanja mogu smatrati ispravnim rješenjem. Jasno, spektar problema je različit i nije moguće unaprijed definirati funkciju dobrote za sva postojeća i buduća značenja elemenata i vremenskih okvira.

Iz tog razloga, algoritam treba izračunati ono što pouzdano zna, a to je poštivanje poretka, a značenje pojedinog vremenskog okvira i elemenata u njima prepustiti onome tko definira značenja elemenata. Konkretno, to povlači da je funkcija dobrote definirana iz dvije komponente K_1 i K_2, koje mogu imati različite težine λ i μ, pri čemu se vrijednosti težina mogu parametrizirati.

$$dobrota(jedinka) = \lambda \cdot K_1 + \mu \cdot K_2 \qquad \lambda + \mu = 1,\ \lambda > 0,\ \mu > 0$$

U konačnom modelu prikazanom pomoću usmjerenog grafa, svakom od vrhova moguće je dodijeliti vrijednost vremenskog okvira. Prva komponenta se tada može računati kao broj onih lukova u kojima izvorišni vrh ima veću vrijednost vremenskog okvira od odredišnog vrha.[17]

[17] Samo promatranje vrijednosti pridruženih pojedinim vrhovima po redoslijedu topološkog poretka ne bi bio dobar način, jer x<y ne povlači da postoji put od x do y, to jest x ne mora biti preduvjet od y.

Budući da može postojati nesrazmjer među redovima veličina u obje komponente, potrebno je obje komponente normirati.

Maksimalna vrijednost prve komponente bez normiranja po ovoj ideji jednaka je broju lukova u grafu, pa se tako prva komponenta može normirati na interval [0,1] pri čemu vrijednost 1 poprima ako ne postoji nijedan luk za kojeg poredak vrhova nije dobar, odnosno 0, ako su svi lukovi takvi da njegovi pripadajući vrhovi nisu u dobrom poretku. Prema ovome, označi li se sa *frame(v)* dodijeljeni redni broj vremenskog okvira vrhu v, komponenta K_1 se može zapisati kao

$$\mathrm{K}_1 = 1 - \frac{\sum_{(v,w)\in A} \delta(v,w)}{|A|}$$

pri čemu je

$$\delta(v,w) = \begin{cases} 1 & ako\ je\ frame(v) > frame(w) \\ 0 & ako\ je\ frame(v) < frame(w) \\ \sigma(v,w) & ako\ je\ frame(v) = frame(w) \end{cases}$$

Vrijednost $\sigma(v,w)$ može biti 0 ili 1 i služi za izračun specijalne situacije u kojoj su i vrhu v i vrhu w dodijeljeni isti vremenski okviri. U tom slučaju potrebno je usporediti vremenska razdoblja elemenata koji odgovaraju vrhovima v i w i promotriti odnos s dodijeljenim vremenskim okvirom kako bi se utvrdilo da li je takva situacija moguća ili ne. Jedan primjer takve situacije je organiziranje nastave u ciklusima pri čemu vremenski okviri predstavljaju pojedine semestre. U tom slučaju moguća je situacija da dva predmeta od kojih je prvi preduvjet drugom budu u istom semestru, odnosno u istom vremenskom okviru, što je valjana situacija, pa bi tada $\sigma(v,w)$ bio jednak 0.

Potrebno je primijetiti da prva komponenta broji samo neispravne poretke među vrhovima pojedinih lukova, a da ne broji neispravan poredak svih prethodnika nekog elementa. Formula za takav način brojanja bi u tom slučaju bila složenija, te bi u sumi u brojniku razlomka u gornjoj formuli trebalo gledati ne samo neposredne prethodnike, već sve prethodnike nekog vrha w.

Nazivnik formule bi se promijenio u sumu broja prethodnika svih elemenata. Osim što podiže složenost izračuna, praktičnim testom pokazalo se da takav izračun postiže lošije rezultate [18].

Druga komponenta sastoji se vektora d = (d_1, …, d_m), gdje je m broj vremenskih okvira i pri čemu se za svaku komponentu vektora računa funkcija dobrote te se normira na vrijednost od 0 do 1. Vrijednost cijele komponente tada se može izračunati kao prosječna vrijednost pojedinih komponenti vektora[19].

Vrijednost pojedine komponente d_i dobiva se pozivom odgovarajućeg postupka koji je definiran za određenu implementaciju elemenata. Kako bi se mogla normirati pojedina komponente vektora, to jest dobrota pojedinog vremenskog okvira, postupku je potrebno proslijediti podatke o elementima smještenim u navedeni vremenski okvir kao i popis ostalih elemenata koji sudjeluju u raspoređivanju kako bi se mogla dobiti maksimalna i minimalna vrijednost. Detaljni izračun ovisi o značenju elemenata i pojedinog okvira, a primjer izračuna dan je u poglavlju 7.2.

Kako je, osim funkcije dobrote, potrebno provjeriti da li jedinka predstavlja valjano rješenje, to je osim dobrote svakog od vremenskih okvira potrebno vratiti i vrijednosti c_i, koje poprimaju vrijednosti istina ili laž, a koje označavaju da li je raspored za pojedini okvir valjan ili ne. Jedinka predstavlja valjano rješenje, ako su sve vrijednosti c_i istina te ako su svi preduvjeti zadovoljeni, odnosno ako je K_1 = 1.

[18] Ideja da bi brojanje svih prethodnika, umjesto samo pojedinačnih lukova, mogla biti bolja može se objasniti sljedećim primjerom. Neka se promatra raspored trojke elemenata (A, B, C), gdje je A preduvjet od B i B preduvjet od C te neka je za tu trojku dobivena jedinka (8, 4, 6) pri čemu brojevi predstavljaju redne brojeve vremenskih okvira. Tada je broj lukova za koje raspored nije dobar jednak 1, pa je u ovom slučaju vrijednost prve komponente funkcije dobrote približno jednaka 0,67. Promjenom vrijednosti za element A iz 8 u 2, ova jedinka ima dobrotu 1 i postaje valjano rješenje. Za jedinku (4, 8, 6) koja ima jednaku dobrotu vrijedi slična opaska oko promjene jedne vrijednosti u jedinci.

Promatrajući jedinke (3,1,2) i (1,3,2), slične ranije navedenima, nije moguće povući jednaku paralelu. Naime, iako jedinke imaju jednaku dobrotu, u prvom slučaju potrebno je više promjena do ispravne jedinke nego u drugom slučaju. U tom slučaju bi brojanje svih prethodnika moglo predstavljati bolju mjeru, pa bi u tom slučaju dobrota prve jedinke bila 0,33 a druge 0,67.

Za praktični test odabrani su predmeti modula Programsko inženjerstvo na preddiplomskom studiju Fakulteta elektrotehnike i računarstva te je napravljen graf u kojima su opisani međusobni odnosi elemenata. Moguće vrijednosti za pojedini element nisu reducirane te se u obzir uzimala samo prva komponenta funkcije dobrote. Nakon 5000 izvođenja genetskog algoritma s 10000 iteracija brojanja lukova je pronašlo dobar raspored u približno 72% slučajeva, naspram 57% u slučaju kad su se brojali svi prethodnici.

Treba napomenuti da bi se pri brojanju prethodnika mogla napraviti određena poboljšanja, uključivanjem razlika pridijeljenih vrijednosti, jer je primjerice (4,2,3), koji je sličan (3,1,2), ipak bolji od (1,3,2), iako je broj nezadovoljenih preduvjeta veći. Takva modifikacija bi složenost brojanja prethodnika još više povećala u odnosu na brojanje lukova, pa je upitna korist takvog poboljšanja pri većem broju iteracija.

[19] Želi li se favorizirati one vektore s jako dobrim pojedinačnim komponentama, moguće je vrijednost računati kao $\frac{\|d\|}{\sqrt{m}}$.

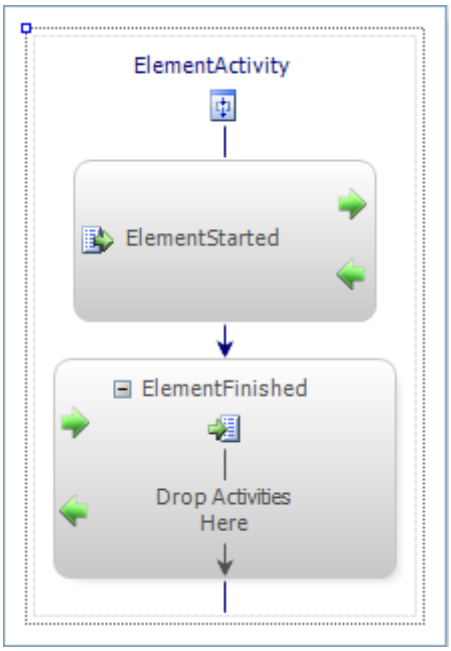
5. Pretvorba grafa konačnog modela u WF model

Kao što je već izneseno u poglavlju 2.3.1 jedan od načina da se sloj protoka poslova odvoji u zasebni (pod)sloj je izlaganje modela protoka poslova putem WCF servisa. Povezivanje s WCF servisom ne mora nužno biti ostvareno putem HTTP(S) protokola, ali može se pretpostaviti da će navedeni WCF servis biti izložen kao web servis, što omogućuje da se sloj protoka poslova nalazi na zasebnom fizičkom sloju i da susjedne slojeve tretira jednako kao bilo koje druge vanjske sustave. Stoga u daljnjem tekstu pojam vanjski sustav se odnosi na bilo koju aplikaciju ili njen dio koji se koristi protokom poslova izloženim kao web servis.

Prva aktivnost unutar WF modela, nastalog iz konačnog modela, mora biti aktivnost *ReceiveActivity* koja je vezana uz postupak *StartWorkflow* iz prethodno definiranog ugovora te omogućava pokretanje nove instance protoka poslova. Nakon toga slijede elementi oblikovani prema pravilima prikazanim u sljedećim odjeljcima pri čemu su svi postupci za pozive definirani ugovorom u poglavlju 6. Od vanjskog sustava očekuje se da implementira one postupke iz ugovora koji služe za primanje obavijesti od strane instance protoka poslova.

5.1 Pretvorba pojedinačnog elementa

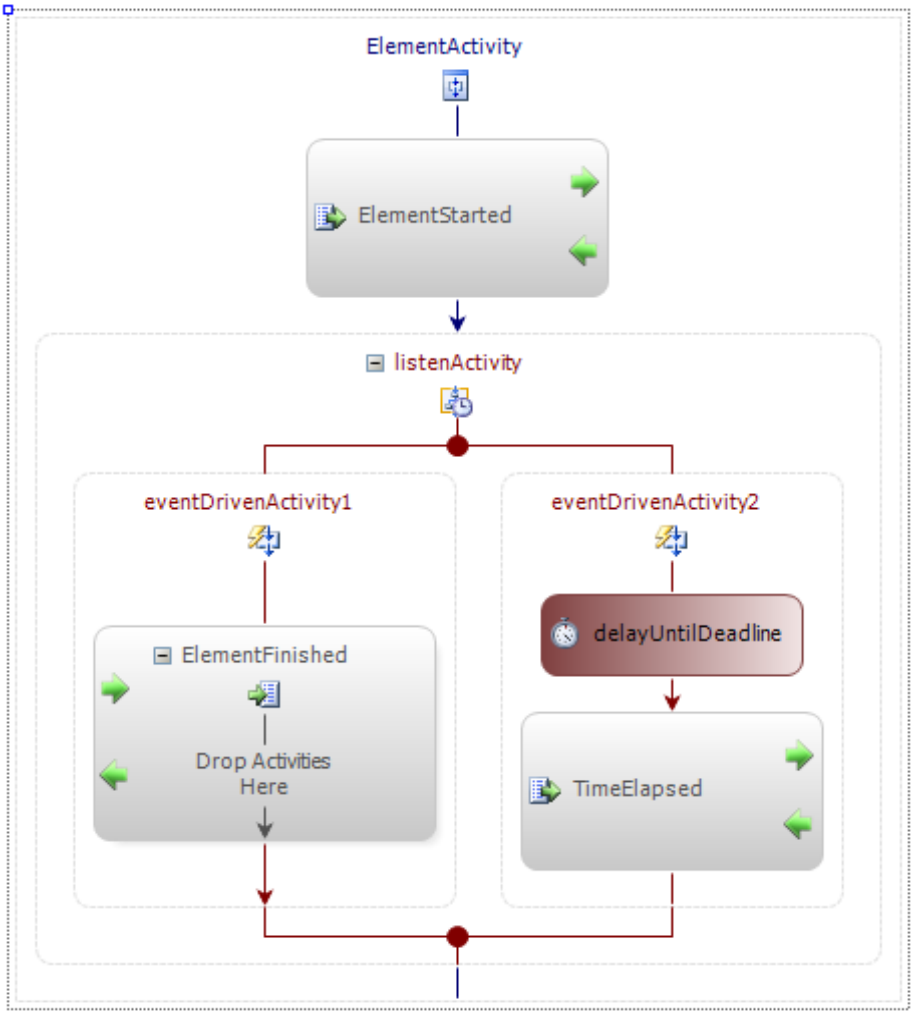
Promatra li se pojedinačni element bez vremenske komponente, tada se njegovo funkcioniranje može podijeliti na dvije WF aktivnosti (slika 20).



Slika 20. Implementacija pojedinačnog elementa bez vremenske komponente

Prva aktivnost implementirana kroz aktivnost *SendActivity* poziva postupak *ElementStarted* na WCF servisu na vanjskom sustavu. Svrha ove aktivnosti je obavijestiti vanjski sustav da se navedeni element pokrenuo, pri čemu treba poslati podatke o kojoj se instanci protoka poslova radi te koji se element aktivirao (i ako je to potrebno, nakon kojeg

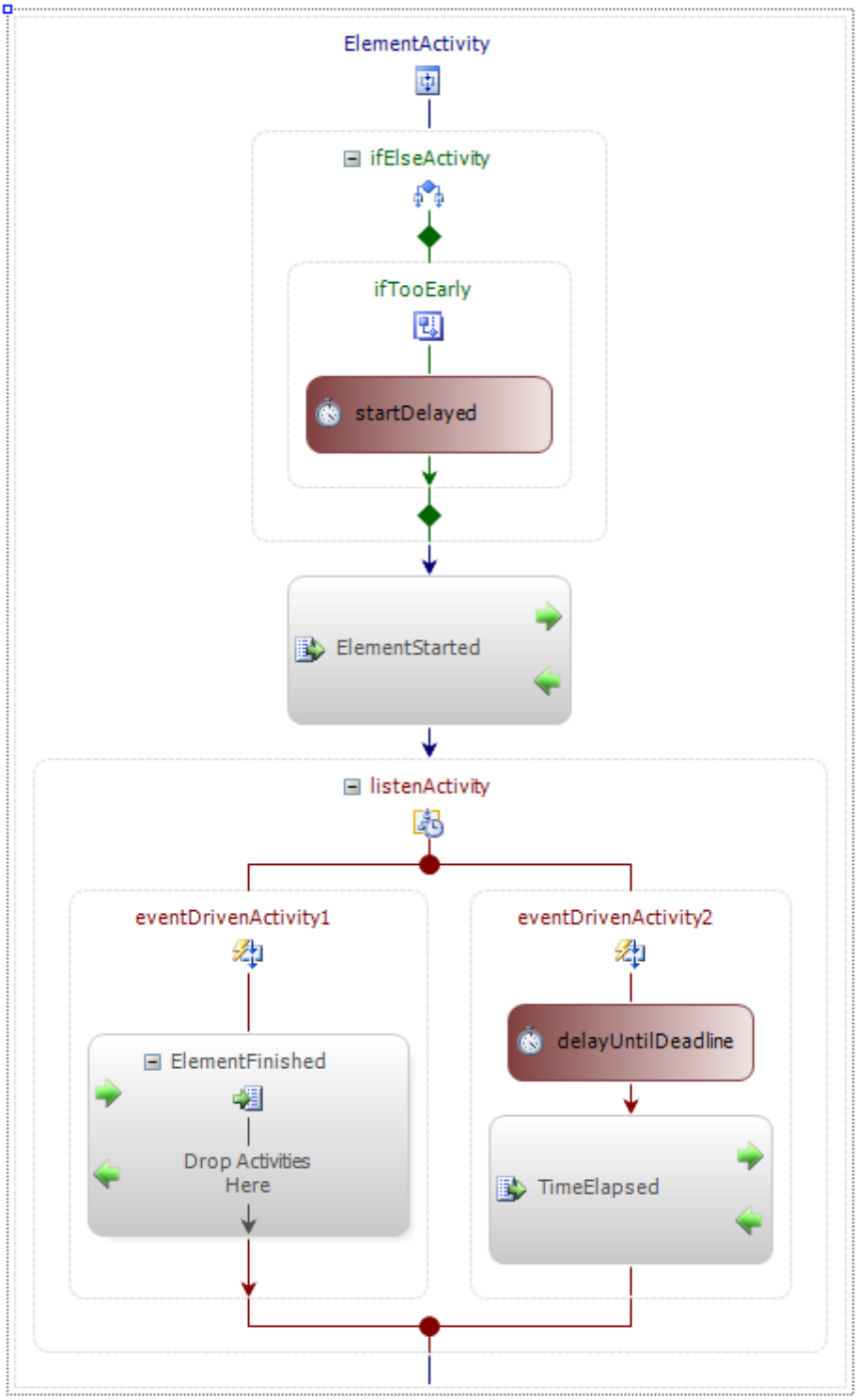
elementa se aktivirao). Sukladno značenju pojedinog elementa, vanjski sustav će obaviti sve što je potrebno za izvođenje tog elementa. Istovremeno, nastavak rada instance protoka poslova, barem što se tiče konkretnog elementa, stavlja se na čekanje. Po završetku elementa vanjski sustav pozivom postupka *ElementFinished* šalje obavijest instanci protoka o završetku elementa. Poziv postupka *ElementFinished* prosljeđuje se aktivnosti *ReceiveActivity*. Budući da se sve aktivnosti *ReceiveActivity* vežu uz isti postupak na WCF servisu, potrebno je imati mehanizam prosljeđivanja poruke upravo onoj aktivnosti za koju je namijenjena. Pri tom je potrebno poznavati identifikator pojedine aktivnosti, to jest identifikator konverzacije aktivnosti *ReceiveActivity* (*conversationId*) te identifikator instance protoka poslova (*instanceId*).



Slika 21. Omogućavanje prisilnog završetka nekog elementa

Za situacije u kojima je definirano krajnje vrijeme završetka pojedinog elementa, nakon obavijesti o početku pojedinog elementa izvršavanje aktivnosti vezanih uz navedeni element u instanci protoka poslova će se staviti na čekanje sve dok se ne dogodi jedan od događaja unutar aktivnosti *Listen*. Događaji u aktivnosti *Listen* u ovom primjeru su poruka dobivena iz vanjskog sustava (obavijest o završetku elementa) i aktivnost *Delay* za čekanje na istek određenog vremena. Vrijeme čekanja u aktivnosti *Delay* u ovom slučaju predstavlja

istek krajnjeg vremenskog roka u kojem element mora biti završen. U slučaju isteka vremena, instanca će pozvati postupak *TimeElapsed* na WCF servisu na vanjskom sustavu kako bi obavijestila vanjski sustav da je vrijeme za izvršavanje konkretnog elementa isteklo. Slika 21 prikazuje izgled elementa koji ima definirano krajnje vrijeme završetka.

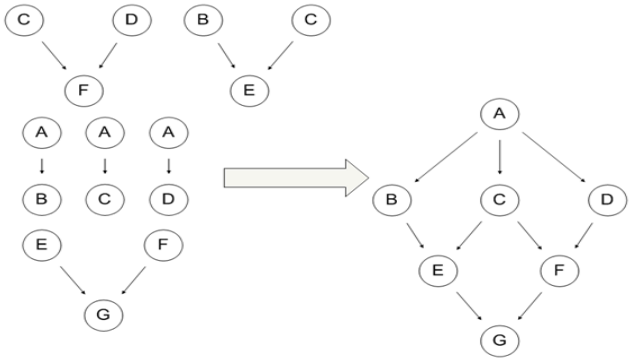


Slika 22. Onemogućavanje preranog početka pojedinog elementa, uz ograničenje na krajnji rok završetka

Za situacije u kojima je za neki element postavljeno vrijeme najranijeg mogućeg početka, prije aktivnosti koja signalizira početak elementa potrebno je postaviti aktivnost *Delay* za vremensku odgodu izvršavanja. Slika 22 prikazuje situaciju u kojoj je za neki element definirano i krajnje vrijeme završetka i najranije vrijeme početka. Poseban slučaj čine situacije u kojima je za pojedini element definirano više ponavljajućih vremenskih razdoblja. Ako nije drugačije definirano prilikom kreiranja modela, pri provjeri preuranjenog starta, izvođenje elementa će se odgoditi do naredne iteracije izvođenja.

5.2 Pretvorba preduvjeta

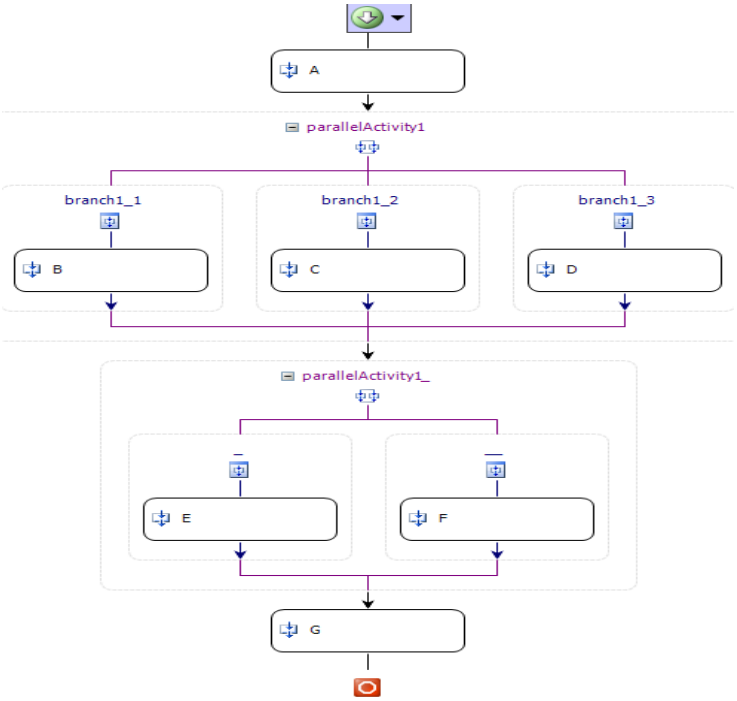
Nakon što su odabrani svi elementi i njihovi preduvjeti koji se trebaju naći u konačnom modelu, tako dobiveni integralni graf treba pretvoriti u konkretni WF model. Zadatak nije trivijalan što je ilustrirano narednim primjerom.



Slika 23. Skup preduvjeta integriran u zajednički graf

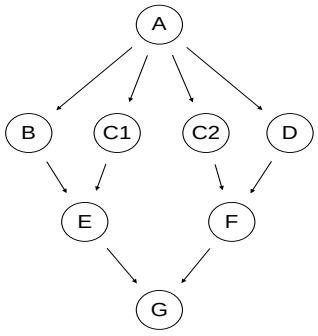
Desna strane Slika 23 prikazuje integralni graf nastao spajanjem parcijalnih modela s lijeve strane slike. Tako dobiveni graf ima jedno paralelno grananje nakon završetka elementa A, nakon čega se elementi B, C i D mogu izvršavati istovremeno te tri sinkronizacijske točke i to redom prije elementa E, prije elementa F i prije elementa G. Već ovako jednostavan primjer ukazuje na problem koji se ogleda u činjenici da paralelna grananja i sinkronizacijske točke nisu međusobno upareni. Jednom odvojene paralelne grane unutar WF modela, ne mogu imati međusobnih veza do sinkronizacijske točke što u ovom primjeru nije slučaj. Slika 24 prikazuje neispravan pokušaj pretvorbe grafa u WF model.

Iako vizualno sličan grafu sa slike 23, model je neispravan. Nakon elementa A dolazi do paralelnog grananja te se elementi B, C i D mogu izvršavati paralelno, međutim, elementi E i F moraju pričekati završetak svih triju elemenata B, C i D, da bi mogli početi sa svojim izvršavanjem, što nije u skladu s postavljenim uvjetima preduvjeta. Promotri li se početna situacija, vidljivo je da bi element E trebao ovisiti samo o elementima B i C, ali ne i o elementu D, odnosno element F bi trebao ovisiti samo o elementima C i D. Pokušaj da se elementi B i C odijele u zasebnu paralelnu granu iza koje će slijediti E rezultirao bi time da bi element F izgubio ovisnost o elementu C. Analogno, nije moguće to napraviti za element F, što sugerira da direktnu pretvorbu grafa sa slike 23 u WF model nije moguće izvesti bez dodatne prilagodbe.



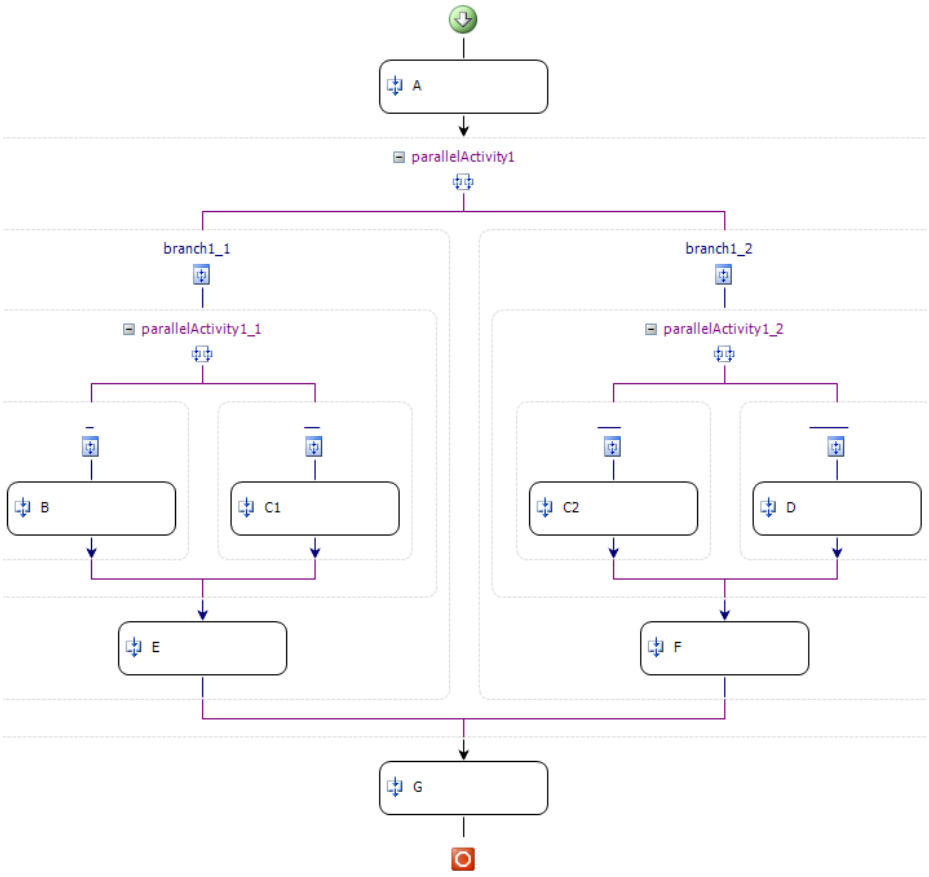
Slika 24. Neispravan pokušaj pretvorbe grafa sa slike 23 u WF model

Kao ideja za rješenje problema može poslužiti graf sa slike 25 koji je sličan grafu na slici 23, ali ne sadrži veze među paralelnim granama te stoga može biti direktno pretvoren u WF model, što je prikazano na slici 26.



Slika 25. Primjer grafa koji se može direktno pretvoriti u WF model

Razdvajanjem vrha C iz grafa sa slike 23 na dva vrha C1 i C2, klonove vrha C te spajanjem vrhova C1 i E, odnosno vrhova C2 i F moguće je dobiti graf iz primjera na slici 25, odnosno WF model na slici 26. Na ovaj način, kloniranjem elemenata koji čine veze među paralelnim granama, moguće je izvršiti transformaciju u WF model. Dodatno treba osigurati da prilikom izvođenja klonovi nekog elementa budu prikazani kao jedna instanca elementa te da događaj za završetak elementa budu pravilno dostavljen svim klonovima tog elementa. U sljedećem odjeljku dan je algoritam za pronalaženje elemenata koje treba klonirati te algoritam za pretvorbu u WF model. Rad s klonovima elemenata opisan je u poglavlju 5.4.



Slika 26. WF model za graf sa slike 25

5.2.1 Označavanje vrhova

Algoritam označavanja vrhova izvodi se na usmjerenom grafu $G = (V, A)$, gdje je V skup vrhova koji predstavljaju elemente iz parcijalnih modela protoka poslova i A skup usmjerenih lukova za koje moraju biti zadovoljeni sljedeći uvjeti:

- Graf G je povezan i ne sadrži cikluse
- Postoji samo jedan početni vrh $v \in V$ takav da je ulazni stupanj vrha v jednak nula ($\deg^-(v) = 0$)
- Postoji samo jedan završnih vrh $v \in V$ takav da je izlazni stupanj vrha v jednak nula ($\deg^+(v) = 0$)
- Za svaki par vrhova p i q takvih da postoji luk između njih ($a \in A$, $a = (p, q)$), ne postoji niti jedan drugi put $pv_1v_2 \ldots v_n q$ u grafu od p do q, gdje su $v_i \in V$.

Provjera postojanja ciklusa, opisana u poglavlju 3.2.2, te reduciranje lukova zbog tranzitivnosti relacije preduvjeta, opisano u poglavlju 3.2.3, kao i dodavanje početnog i završnog vrha (*Start* i *End* iz poglavlja 4.2.1) osigurat će da gore navedeni uvjeti budu ispunjeni.

Definicija: Oznaka vrha je niz znakova u kojem se nalaze samo prirodni brojevi i točke.

Skup svih mogućih oznaka označen je skupom $\mathcal{L}$. Svakom vrhu bit će pridijeljena jedna ili više oznaka. Neka je $\ell: V \rightarrow L$ preslikavanje koje će svakom vrhu dodijeli jednu ili više oznaka, pri čemu je V skup vrhova grafa, a $L \subset \mathcal{L}$ podskup skupa svih oznaka.

Napomena. U opisu algoritma koji slijedi termin prethodnik se odnosi na vrh koji u grafu prethodi nekom vrhu. U ovom poglavlju jedini prethodnici su preduvjeti, međutim proširenje algoritma se može izvesti i na skupove mogućih prethodnika uz opasku iz poglavlja 5.3.

Tablica 6. Algoritam označavanja vrhova

1. Završnom vrhu dodijeli oznaku '1' i dodaj ga u skup otvorenih vrhova
   ℓ (*End*) := {'1'}
   O := {*End*}

2. Neka je *curr* zadnji vrh po topološkom poretku u skupu O i neka je P skup njegovih prethodnika
   $curr$:= $w \in$ O takav da je $v < w$, $\forall\, v \in$ O, $v \neq w$
   P := { v | $\exists\, a \in$ A takav da je a = (v, *curr*)}

   ako je |P| = 1 tada
   $\ell(v)$:= $\ell(v) \cup \ell(curr)$ gdje je $v \in$ P
   ako je |P| > 1 tada
   za svaki vrh $v \in$ P
   $\ell(v)$:= $\ell(v) \cup concat(\ell(curr), '$.redniBroj') gdje je *concat* funkcija koja će za ulazni skup oznaka vratiti novi skup oznaka kojima je dopisan niz naveden kao drugi parametar funkcije, a redniBroj predstavlja redni broj prethodnika (moguće vrijednosti su od 1 pa do |P|)

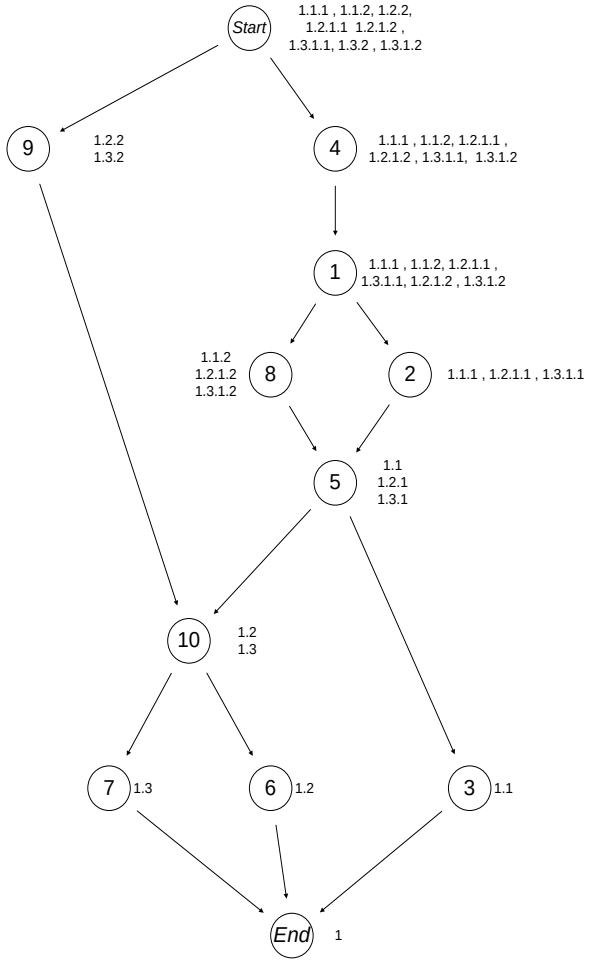
   O := (O ∪ P) \ {curr}

3. Ako je |O| > 0 ponovi korak 2.

Algoritam započinje sa završnim vrhom grafa kojem pridjeljuje oznaku 1 i dodaje ga u skup „otvorenih“ vrhova. Pojam „otvoren“ odnosit će se na one vrhove koji još nisu proslijedili sve svoje oznake svojim prethodnicima. Algoritam označavanja u svakom koraku uzima novi vrh iz skupa otvorenih vrhova te prosljeđuje svoje oznake prethodnicima prema sljedećim pravilima. Ako vrh ima samo jednog prethodnika, onda se sve oznake dodaju u prethodnikov skup oznaka bez ikakvih modifikacija. Ako vrh ima N prethodnika, u tom slučaju se pri kopiranju oznaka iz trenutnog vrha u svaki od prethodnika, na svaku oznaku dodaje određeni sufiks oblika *.i* gdje *i* predstavlja redni broj prethodnika. Na taj način prvom odabranom prethodniku su sve oznake trenutnog vrha dodane uz dopisivanje *.1*, drugom prethodniku uz sufiks *.2* i tako redom do N-tog prethodnika koji će dobiti oznake uz sufiks *.N*, nakon čega se trenutnih vrh briše iz skupa otvorenih vrhova, a prethodnici trenutnog

vrha postaju novi članovi skupa otvorenih vrhova. Kad bi redoslijed odabir vrhova bio nasumičan ili po algoritmu za pretraživanje u dubinu, u ovisnosti o redoslijedu odabira vrhova, pojedini vrh bi mogao biti u skupu otvorenih vrhova, proslijediti svoje trenutne oznake prethodnicima, zatim biti izbačen iz skupa, a zatim u nekom trenutku ponovno dodan u skup otvorenih vrhova, jer je bio prethodnik nekog vrha koji je kasnije razmatran. U tom slučaju bi se prilikom dodavanja oznaka u prethodnike dodavale samo one oznake koje već nisu bili prenesene. Kako bi se navedena situacija izbjegla, radi jednostavnosti, ali i radi efikasnosti algoritma, uzimanje vrhova vrši se silazno po topološkom poretku. Na taj način će se osigurati da su svi sljedbenici promatranog vrha već obrađeni i da se isti vrh neće ponovo naći u skupu otvorenih vrhova. Navedeni koraci mogu se formalno zapisati kao što je prikazano u tablici 6.

Za ilustraciju rada algoritma poslužit će primjer na slici 27.



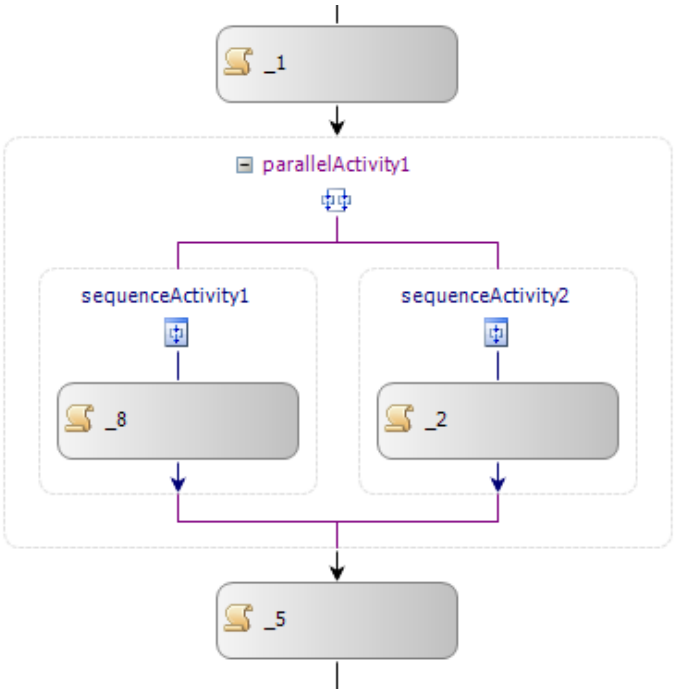
Slika 27. Primjer označavanja vrhova grafa

Neka je topološki poredak ovog grafa bio *Start*, 9, 4, 1, 8, 2, 5, 10, 7, 6, 3, *End*. U prvom koraku završnom vrhu dodijeljena je oznaka 1 te je dodan u skup otvorenih vrhova. Završni vrh ima 3 prethodnika i to redom vrhove 3, 6 i 7. S obzirom da vrh *End* ima više od jednog prethodnika njegova oznaka 1 dodaje se skupu oznaka vrha 3 sa sufiksom .1, vrhu 6 sa

sufiksom .2 te vrhu 7 sa sufiksom .3. Vrh *End* se briše iz skupa otvorenih vrhova, a dodaju se vrhovi 3,6 i 7. Po topološkom poretku posljednji je vrh 3 te se on uzima kao sljedeći vrh. On ima samo jednog prethodnika (vrh 5) te se oznaka 1.1 kopira u skup oznaka vrha 5 bez modifikacija. Vrh 3 se briše iz skupa otvorenih vrhova, a dodaje se vrh 5. Najveći po topološkom poretku sada je vrh 6, koji kopira svoju oznaku vrhu 10, a isto se događa i za vrh 7. U tom trenutku u skupu otvorenih vrhova nalaze se vrhovi 5 i 10, pri čemu je 10 veći po topološkom poretku, pa se korak algoritma odvija za njega. Vrh 10 ima dva prethodnika i svakom od njih kopira svoje oznake uz dopisivanje sufiksa, pa se tako vrhu 5 u dosadašnji skup oznaka {1.1} dodaju još oznake 1.2.1 i 1.3.1, a vrhu 9 se dodaju oznake sa sufiksom 2, to jest oznake 1.2.2 i 1.3.2. Postupak se ponavlja sve dok se ne dođe do početnog vrha, koji nema prethodnika te, nakon njegovog brisanja iz skupa otvorenih vrhova, taj skup postaje prazan i algoritam završava.

5.2.2 Reduciranje oznaka

Budući da kardinalni broj skupa oznaka pojedinog vrha predstavlja broj klonova koje treba napraviti prilikom izrade WF modela, sasvim je razumljivo da bi taj broj trebao biti što je manji mogući. Stoga, ako je to moguće, potrebno je reducirati broj oznaka. Pogleda li se primjer sa slike 27, uočit će se da ne postoji razlog zašto bi vrhovi 1 i 5 imali različite oznake. Niti jedan put prema završnom vrhu koji uključuje izlazne lukove iz vrha 1 nije takav da ne bi prolazio kroz vrh 5. Istovremeno, za svaki ulazni luk vrha 5, put od početka grafa koji sadrži takav luk zasigurno prolazi i kroz vrh 1. Gledajući u terminima WF modela ovo je situacija koja odgovara paralelnom grananju u vrhu 1 koje ima dvije paralelne grane s elementima 2 i 8, nakon čega slijedi točka sinkronizacije, a zatim element 5 kao što je prikazano na slici 28.



Slika 28. Odsječak WF modela koji predstavlja dio između vrhova 1 i 5 u grafu sa slike 27

Stoga, može se zaključiti da bi se skup oznaka vrha 1 mogao reducirati te da bi mogao imati jednake vrijednosti kao i skup oznaka vrha 5. Prije samog opisa algoritma koji će omogućiti reduciranje oznaka, potrebno je uvesti nekoliko definicija koje formaliziranju odnos između pojedinih oznaka.

Definicija: Funkcija level: $\mathcal{L} \rightarrow \mathbb{N}$, gdje je $\mathcal{L}$ skup svih mogućih oznaka se definira kao broj točaka unutar oznake i predstavlja razinu neke oznake.

Tako je, primjerice, prema navedenoj definiciji level(1) = 0, a level(1.3.1.1) = 3. Veza između oznaka uspostavit će se u terminima roditelj-dijete, odnosno predak-nasljednik, pa će tako oznaka 1.3.1.1 biti dijete oznake 1.3.1 i nasljednik oznake 1 i obrnuto, oznaka 1 je predak oznaka 1.3.1 i 1.3.1.1, dok je oznaka 1.3.1 roditelj oznake 1.3.1.1

Definicija: Oznaka l je roditelj oznake m, ako je level(m) = level(l)+1 te ako su m i l jednaki do posljednje točke unutar obje oznake.

Definicija: Oznaka l je predak oznake m, ako je l roditelj od m ili postoji niz oznaka $l_1 \dots l_n$ takvih da je l roditelj od l_1, l_1 roditelj od l_2,… i l_n roditelj od m.

Osnovna ideja algoritma za reduciranje oznaka je zamjena određenih oznaka s njihovim zajedničkim roditeljem. Zamjena bi se obavila u onim vrhovima koji sadrže svu djecu neke određene oznake. Primjerice, ako su u nekom grafu jedina djeca oznake 1.2 oznake 1.2.1, 1.2.2 i 1.2.3, tada bi u vrhu koji sadrže te tri oznake došlo do njihove zamjene s 1.2. S druge strane, u svim ostalim vrhovima, koji sadrže samo jednu ili dvije od tih triju oznaka ne bi bilo zamjene.

Algoritam reduciranja oznaka grupira oznake po razinama i počinje s provjerom oznaka s predzadnje razine, pa sve do skupa oznaka nulte razine, što je prema algoritmu označavanja vrhova jednočlani skup s oznakom 1. Za svaku oznaku l na trenutno promatranoj razini (skup oznaka na trenutnoj razini je u algoritmu iz tablice 7 označen skupom U), skup S predstavlja sve oznake kojima je oznaka l roditelj. Sljedeći korak je pronaći sve one vrhove (takav skup označen je s V') u grafu koji sadrže sve oznake iz skupa S. Time će se pronaći oni vrhovi koji sadrže svu djecu oznake l i kod takvih vrhova se može obaviti zamjena. Dakle, za svaki vrh iz skupa V', sve oznake iz skupa S se brišu iz skupa oznaka svakog od tih vrhova te se umjesto njih dodaje oznaka l. Algoritam nastavlja s radom sve dok se ne obrade sve oznake na trenutnoj razini, nakon čega ide jednu razinu više i tako dalje sve dok se ne obradi i posljednja oznaka, to jest oznaka 1. Formalno se koraci algoritma mogu zapisati kao što je prikazano u tablici 7.

Tablica 7. Algoritam reduciranja oznaka

1. curr_level := $\max_{l \in \mathcal{L}} level(l)$ - 1

2. Neka je U skup svih oznaka na trenutnoj razini
   U: = { $l \in \mathcal{L}$ | level(l) = curr_level}
   Za svaku oznaku l iz skupa U radi
   S := { m | $m \in \mathcal{L}$ takav da je l roditelj od m}
   Ako je S ≠ ∅ tada
   V' := { v | $v \in$ V takav da je $m \in \ell(v)$, $\forall m \in$ S}
   Za svaki $v \in$ V' radi
   $\ell(v)$:= ($\ell(v)$ \ S) ∪ { l }

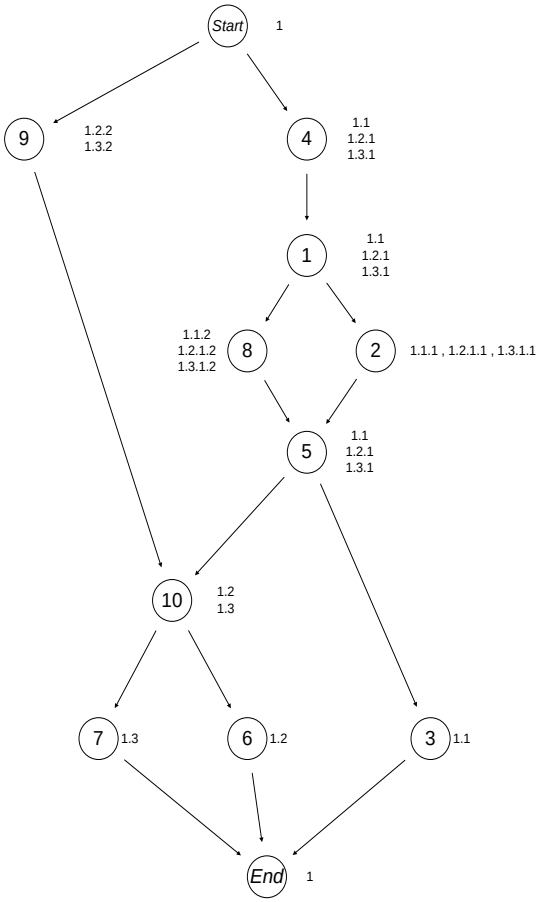
3. curr_level := curr_level - 1
   Ako je curr_level ≥ 0 ponovi korak 2.

Ilustracija rada algoritma bit će dana na primjeru sa slike 27. Najveća razina u navedenom grafu je 3 i čine je oznake 1.2.1.1, 1.2.1.2, 1.3.1.1 i 1.3.1.2. Algoritam kreće od razine 2 i skupa označenog s U kojeg čine oznake na drugoj razini, pa je U ={1.1.1, 1.1.2, 1.2.1, 1.2.2, 1.3.1, 1.3.2}. Za svaku od tih oznaka traže se vrhovi koji sadrže svu njihovu djecu. Oznaka 1.1.1 nema djece te je za tu oznaku skup S prazan skup pa se prelazi na sljedeću oznaku, 1.1.2, za koju vrijedi isto. Za oznaku 1.2.1 skup S je {1.2.1.1, 1.2.1.2} te se traže oni vrhovi koje sadrže sve oznake iz skupa S. Takvi vrhovi čine skup V' i u njemu su vrhovi *Start*, 4 i 1. Za svaki od tih vrhova, iz njihova skupa oznaka će se izbaciti oznake 1.2.1.1 i 1.2.1.2 i dodat će se oznaka 1.2.1.

Oznaka 1.2.2 nema djece, pa se prelazi na oznaku 1.3.1 čime se iz vrhova *Start*, 4 i 1 izbacuju djeca oznake 1.3.1 i dodaje oznaka 1.3.1, a isto vrijedi i za oznaku 1.3.2, nakon čega se prelazi na razinu 1, pa je skup U ={1.1, 1.2, 1.3}. Za oznaku 1.1 skup S je {1.1.1, 1.1.2}. Vrhovi koji sadrže sve oznake iz skupa S su vrhovi *Start*, 4 i 1. Za oznaku 1.2 skup S je {1.2.1, 1.2.2} i jedini vrh koji sadrže sve oznake iz S je vrh *Start*[20] te se iz njegovih trenutnih oznaka izbacuju oznake iz skupa S i dodaje se oznaka 1.2.

Algoritam na sličan način radi i s oznakom 1.3 nakon čega prelazi na razinu 0 i oznaku 1. Nakon kraja algoritma, vrh *Start* mora sadržavati samo oznaku 1. Slika 29 prikazuje graf nakon reduciranja oznaka.

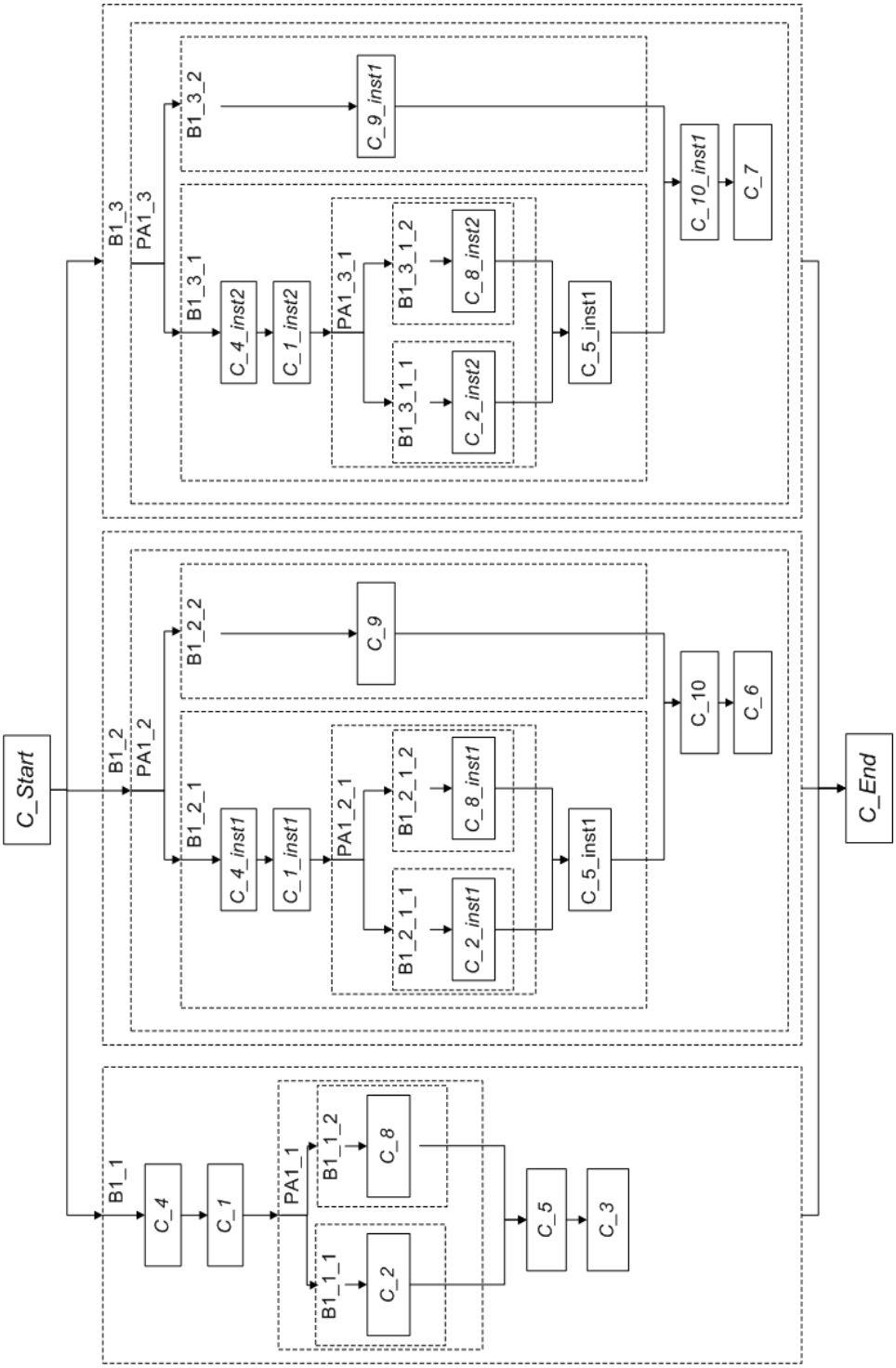
[20] Inicijalno vrh *Start* nije sadržavao oznaku 1.2.1, ali ju je u prethodnom koraku dobio umjesto 1.2.1.1 i 1.2.1.2



Slika 29. Graf sa Slika 27 nakon reduciranja oznaka

Nakon što je algoritam reduciranja oznaka dovršen, može se krenuti u stvaranje WF modela. Kao što je ranije izrečeno, prilikom kreiranja WF modela neki vrhovi će biti klonirani i njihovi elementi će se naći u nekoliko paralelnih grana. WF model koji će biti kreiran za navedeni primjer prikazan je na slici 30, gdje su elementi u modelu imenovani u obliku C_ {broj vrha}.[21] U slučaju da je potrebno napraviti klon nekog elementa, sufiks inst_{redni broj klona} će biti nadodan na ime elementa kako bi se očuvala jedinstvenost imena. U ovom primjeru svaka oznaka nekog vrha predstavlja poziciju u nekoj grani unutar WF modela. Princip izrade WF modela prikazan je u sljedećem odjeljku.

[21] Naziv pojedinog elementa mora slijediti standardne obrasce za imenovanje varijabla, pa stoga naziv ne može početi brojem. Nazivi pojedinih WF aktivnosti moraju biti jedinstveni, pa je iz tog razloga dodan odgovarajući sufiks za aktivnosti koje se vežu uz iste elemente. Formalno, nazivi su mogli biti i neki jedinstveni identifikatori na razini sustava, međutim radi jednostavnosti razumijevanja postupka, sadržani su nazivi koji odgovaraju rednim brojevima vrhova.



Slika 30. WF model[22] za graf sa slike 29

[22] Skica modela napravljena prema pravom WF modelu, inače preširokom za format stranice.

5.2.3 Izrada WF modela

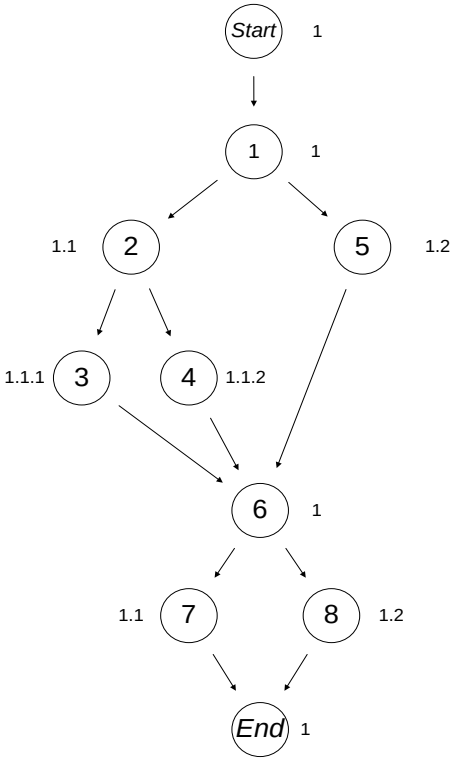
Osim aktivnosti koje nastaju prilikom pretvaranja pojedinačnog elementa u niz WF aktivnosti, što je opisano u poglavlju 5.1, u WF modelu će se nalaziti aktivnost koja omogućava paralelno izvršavanje[23] (*ParallelActivity*) niza aktivnosti u njenim paralelnim granama (eng. *branches*). Svaka grana paralelne aktivnosti je sekvenca (*SequenceActivity*) koja može sadržavati druge aktivnosti. Svaki WF model sastoji se od barem jednog slijeda elemenata, to jest glavne sekvence koja će sadržavati elemente koji u sebi sadrže oznaku 1. Prilikom pojave nekih drugih oznaka, stvarat će se nove paralelne aktivnosti i grane koje bi odgovarale tim oznakama. Primjerice, prilikom pojave elementa s oznakom 1.3.1.2, potrebno je dodati element u granu pridruženu oznaci 1.3.1.2. Ako takva grana ne postoji, potrebno ju je stvoriti kao granu paralelne aktivnosti koja odgovara oznaci 1.3.1. Ako takva paralelna aktivnost ne postoji, onda je potrebno i nju stvoriti po istom principu, pri čemu je postupak rekurzivan i završava najkasnije na glavnoj sekvenci označenoj s 1. Imena aktivnosti u modelu moraju biti jedinstvena i moraju poštivati pravila za imenovanje varijabli, pa će stoga njihova imena biti označena prefiksom te će tako paralelne aktivnosti imati prefiks PA_, a grane paralelne aktivnosti imat će prefiks B_.

Za primjer sa slike 30 oznaka može jednoznačno odrediti paralelnu granu unutar koje će se navedeni element nalaziti, međutim, bitno je naglasiti da će algoritam kreiranja WF modela biti napravljen tako da to ne mora biti slučaj, odnosno to će ovisiti o tome kakve su oznake pridijeljene vrhovima iz grafa. Pozitivna posljedica ovog načina je da će se skup oznaka moći dodatno reducirati. Kao primjer može se uzeti graf sa slike 31 i njegovim elementima pridjeljenje oznake.

Na ovaj način, oznaka ne identificira jednoznačno pojedinu paralelnu granu WF modela, nego služi kao mjera kad i koliko je ugniježđen pojedini element unutar modela gledajući od trenutne pozicije. Vrhovi grafa uzimaju se u topološkom poretku, čime se osigurava da je svaki vrh koji predstavlja točku sinkronizacije dodan nakon svojih prethodnika. Primjerice, za graf sa slike 31 vrhovi u topološkom redoslijedu mogu biti *Start*,1,2,5,3,4,6,7,8,*End*. Vrhovi *Start* i *End* se mogu dodati u WF model, ali nemaju posebno značenje, osim što su poslužili za postupak označavanja grafa kako bi se osigurao nužan uvjet da je graf povezan Algoritam počinje dodavanjem vrha 1 označenog oznakom 1. Element, u skladu s prethodnom najavljenim imenovanjem, nosi naziv C_1. Nakon toga slijedi vrh 2 s oznakom 1.1. Navedena oznaka povlači da mora postojati grana pridružena oznaci 1.1. Budući da takva grana u tom trenutku ne postoji, provjerava se postoji li paralelna aktivnost za oznaku 1. S obzirom da takva paralelna aktivnost ne postoji, ona se stvara i dodaje kao sljedeće dijete glavne sekvence, nakon čega se stvara grana za oznaku 1.1. te se u nju dodaje element C_2.

[23] S obzirom da se instanca protoka poslova izvršava unutar jedne dretve, može se reći da se radi o pseudoparalelizmu. Aktivnosti iz paralelnih grana izvršavaju se naizmjenično, a paralelizam dolazi do izražaja ako su aktivnosti u granama takve da čekanju na neki događaj.

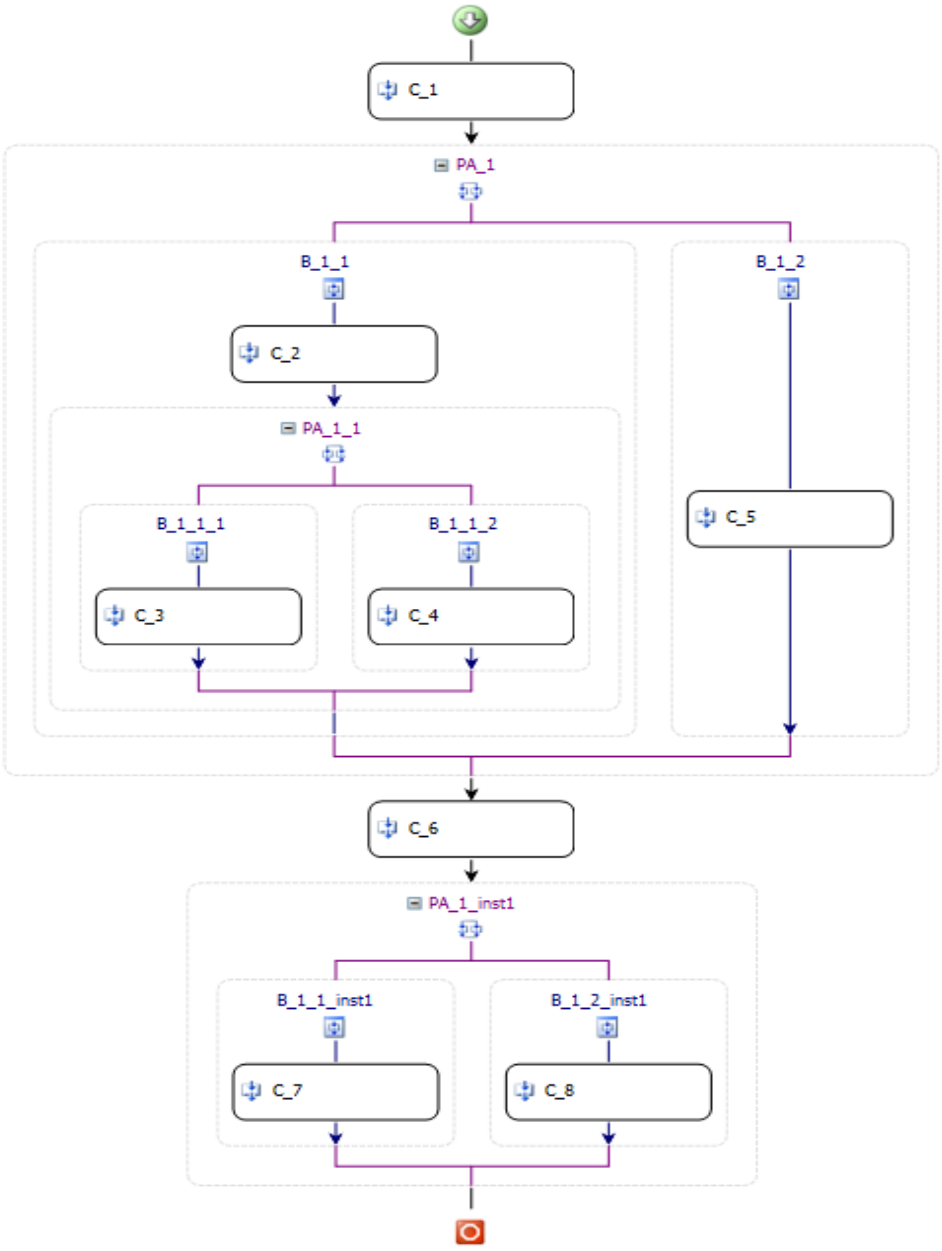
Na sličan način element 5 je dodan u granu s oznakom 1.2.[24] Vrh 3 ima oznaku 1.1.1, što povlači da treba postojati paralelna aktivnost pridružena oznaci 1.1. S obzirom da takva ne postoji stvara se paralelna aktivnost P_1_1 i grana B_1_1_1 u koju se dodaje element C_3. Sve paralelne aktivnosti i grane pridružene nekim oznakama ostaju evidentirane za neke buduće elemente s tim oznakama. Postupak je sličan i za ostale vrhove do vrha 6. Vrh 6 je mjesto spajanja grana 1.1 i 1.2, te će, budući da ima oznaku 1, biti dodan kao sljedeće dijete glavne sekvence (iza navedenih paralelnih grananja). Ono što je važno naglasiti da se pojavom oznake koja je predak neke od već korištenih postojećih oznaka, sva pridruživanja između oznaka nasljednica i paralelnih aktivnosti i njenih grana moraju obrisati. Vrhovi 7 i 8 imaju oznake koje su već postojale unutar grafa, međutim, neće biti dodani u „stare“, postojeće grane 1.1. i 1.2, već će biti stvorena nova paralelna aktivnost PA_1_inst1 s granama B_1_1_inst1 i B_1_2_inst1 koje će biti iza elementa koji predstavlja vrh 6. Novostvorene paralelne aktivnosti i grane pridružuju se oznakama 1.1, odnosno 1.2. i to pridruživanje je aktivno po prve sljedeće pojave oznake 1. WF model za graf sa slike 31 prikazan je na slici 32.



Slika 31. Graf s oznakama koje ne identificiraju jednoznačnu paralelnu granu u WF modelu

Ako vrh za kojeg se trenutno dodaju elementi ima više oznaka, onda se prethodni postupak ponavlja za svaku od oznaka. Pseudokod algoritma za kreiranje WF modela prikazan je u tablici 8.

[24] Poredak pri topološkom sortiranju je mogao biti takav da se prvo uzeo vrh 5, a zatim vrh 2, što ne bi promijenilo konačni izgled WF modela, jer bi se zbog postojanja 1.2 kreirala paralelna aktivnost P_1, a zatim odmah i grane za 1.1. i za 1.2. Grana za 1.1. bi bila kreirana, jer ako postoji 1.2, onda zasigurno postoji i 1.1.



Slika 32. WF model za graf sa Slika 31

Za rad algoritma iz tablice 8 potrebno je definirati sljedeća preslikavanja:

$p : \mathcal{L} \rightarrow$ PA, gdje je $\mathcal{L}$ skup svih oznaka, a PA skup svih paralelnih aktivnosti unutar WF modela je preslikavanje koje za neku oznaku vraća pridruženu paralelnu aktivnost,

$g : \mathcal{L} \rightarrow$ B, gdje je $\mathcal{L}$ skup svih oznaka, a B skup svih grana unutar WF modela je preslikavanje koje za neku oznaku vraća pridruženu granu neke paralelne aktivnosti,

pri čemu je inicijalno definirano *g(1)*, što se odnosi na glavnu sekvencu.

Tablica 8. Algoritam izrade WF modela na osnovu grafa s oznakama

Neka je graf $G(V, A)$ graf označen prema algoritmu iz poglavlja 5.2.1 i 5.2.2

Za svaki i:=1 do i≤ $|V|$ radi
    v := i-ti element iz topološkog poretka grafa G
    Za svaku oznaku $l \in \ell(v)$ radi
        D := { m | $m \in \mathcal{L}$ takav da je l predak od m}
        ako je D ≠ ∅ tada
            Za svaki $m \in$ D radi
                $p(m)$:= nedefinirano
                $g(m)$:= nedefinirano
        ako $g(l)$ nije definirano
            stvoriGranu(l)
        dodaj element C_v[25] u granu $g(l)$

Funkcija za stvaranje grane je rekurzivna i ovisi o funkciji za stvaranje paralelne aktivnosti. Navedene funkcije su opisane u tablici 9.

Tablica 9. Rekurzivni algoritam za stvaranje grane paralelne aktivnosti za neku oznaku

stvoriGranu(l)
    ako $p(roditelj(l))$ nije definirano
        stvoriParalelnuAktivnost($roditelj(l)$)
    $g(l)$:= stvori granu B_ l[26] kao granu paralelne aktivnosti $p(roditelj(l))$

stvoriParalelnuAktivnost(m)
    ako $g(m)$ nije definirano
        stvoriGranu(m)
    $p(m)$:= stvori paralelnu aktivnost PA_ m kao sljedeću aktivnost u grani $g(m)$

5.2.4 Ispravnost algoritma za kreiranje WF modela

Za dokaz ispravnosti algoritma potrebno je dokazati dvije glavne tvrdnje. Prva je da algoritmi integracije grafa i reduciranja lukova, a zatim i algoritmi označavanja vrhova i reduciranja oznaka zadržavaju sva pravila preduvjeta te da ne nastaju novi preduvjeti koji nisu postojali. Druga tvrdnja je da algoritam stvaranja modela iz označenog grafa pravilno kreira sekvence i paralelne aktivnosti.

Pogledaju li se koraci integracije parcijalnih modela u zajednički graf, lako se vidi da prilikom navedenog postupka ne dolazi ni do kakvih promjena modela, već se samo parcijalne definicije pretvaraju u usmjereni graf. Sve veze su u tom trenutku zadržane bez ikakvih promjena i/ili dodavanja. Algoritam reduciranja lukova, opisan u poglavlju 3.2.3 mijenja model, ali na način da uklanja tranzitivnosti među relacijama preduvjeta, što povlači

[25] Ime može sadržavati sufiks _instX, gdje je X broj već postojećih elemenata stvorenih za vrh v
[26] Ime može sadržavati sufiks _instX, gdje je X broj već postojećih grana prethodno stvaranih za oznaku l. Ista opaska vrijedi i za naziv paralelne aktivnosti.

da samo značenje nije promijenjeno, to jest, svi preduvjeti nekog elementa su sadržani, bilo u obliku neposrednih preduvjeta ili u obliku preduvjeta kroz niz tranzitivnih relacija preduvjeta. Na osnovu ovog ostaje pokazati sljedeće dvije tvrdnje za postupak označavanja vrhova, reduciranje oznaka i kreiranje WF modela

a) Svi vrhovi su u ispravnom poretku, što znači da su sve postojeće relacije preduvjeta zadovoljene
b) Izvršavanje svakog elementa ovisi samo o njegovim preduvjetima, to jest nema čekanja na neki element koji nije bio njegov preduvjet

Za svaki vrh iz označenog grafa u WF modelu se stvara jedan ili više klonova pretvorenih u odgovarajuće WF aktivnosti. Broj klonova nekog vrha jednak je kardinalnom broju skupa oznaka tog vrha. Promatrajući WF model, za neke klonove se može reći da se mogu odvijati paralelno (neovisno u odvojenim paralelnim granama) ili da se nalaze u istoj liniji izvršavanja. Iz dizajna WF modela slijedi da dva klona mogu biti u paralelnim granama samo ako njihove oznake nisu u nekoj vezi oblika predak-nasljednik, pri čemu obrat ne mora nužno vrijediti, kao s vrhovima 2 i 8 na slici 32. Tvrdnja se može iskazati sljedećim teoremom.

Teorem 1. Nužan uvjet za paralelno izvršavanje klona vrha X s oznakom l_x i klona vrha Y s oznakom l_y je nepostojanje odnosa predak-nasljednik među oznakama l_x i l_y i $l_x \neq l_y$.

Dokaz: Bez smanjenja općenitosti može se pretpostaviti da je vrh X u topološkom poretku prije vrha Y. Moguće je razlikovati dva slučaja. U prvom slučaju, u topološkom poretku između vrhova X i Y ne postoji niti jedan vrh s oznakom koja je zajednički predak oznaka l_x i $l_y$te slučaj u kojem između vrhova X i Y postoji vrh s oznakom koja je zajednički predak oznaka l_x i l_y.

Slučaj I. Prema algoritmu iz tablice 8 grane paralelnih aktivnosti stvaraju se na način da je grana B pridružena oznaci l grana paralelne aktivnosti PA pridružene oznaci $roditelj(l)$. Paralelna aktivnost PA je sadržana u grani pridruženoj oznaci $roditelj(l)$, koja je grana paralelne aktivnosti pridružene oznaci $roditelj(roditelj(l))$… Pri dodavanju klona vrha X s oznakom l_x stvorit će se (ako već ne postoji) grana pridružena oznaci l_x. U trenutku dodavanja vrha Y s oznakom l_y treba se stvoriti grana (ako već ne postoji) s oznakom l_y. U slučaju da je $l_x = l_y$ radi se o istoj grani te su X i Y u istom slijedu, to jest u istoj liniji izvršavanja. Ako je l_x predak oznake l_y tada je grana pridružena oznaci l_y sadržana u grani l_x prema prethodno opisanom principu dodavanja grana. Ako je l_y predak grane l_x to znači da grana pridružena oznaci l_y zasigurno već postoji, te se klon vrha Y dodaje na kraj te grane, što znači da je Y sinkronizacijska točka za postojeće grane u l_y, pa su X i Y sigurno u istoj liniji izvršavanja. U slučaju da l_x i l_y nisu u odnosu oblika predak-nasljednik tada se za slučaj I. X i Y mogu izvoditi paralelno.

Slučaj II. Pojavom oznake l takve da je zajednički predak oznaka l_x i l_y prema algoritmu iz Tablica 8 sva pridruživanja između grana i oznaka nasljednica oznake l se brišu. Na taj način

su obrisana i pridruživanja za oznake l_x i l_y. Klon vrha s oznakom l je sinkronizacijska točka za sve grane koje su u tom trenutku bile pridružene oznakama l_x i l_y. Ta činjenica povlači da će prilikom pojave elementa Y novostvorena grana biti sadržana u grani l, što znači da su X i Y u istoj liniji izvršavanja i na toj liniji izvršavanja, na putu od X do Y, se nalazi vrh s oznakom l.

■

Kao posljedica dokaza Teorema 1 direktno slijedi Korolar 2.

Korolar 2. Klon vrha X s oznakom l_x i klon vrha Y s oznakom l_y izvršavaju se paralelno ako je $l_x \neq l_y$, oznake l_x i l_y nisu u odnosu predak-nasljednik te u topološkom poretku između vrhova X i Y ne postoji vrh s oznakom koja je zajednički predak oznaka l_x i l_y.

Promatrajući ne samo pojedinačni klon nego sve klonove nekog vrha, može se reći da su dva vrha su istoj liniji izvršavanja ako za svaki klon jednog od vrhova postoji klon drugog vrha s kojim je u istoj liniji izvršavanja, a u protivnom vrhovi se mogu izvršavati neovisno, to jest paralelno. Primjenjujući prethodno izrečenu opasku oko oznaka na vrhove iz grafa, to povlači da su dva vrha u grafu u istoj liniji izvršavanja, ako se rečena tvrdnja može primijeniti za sve oznake jednog od tih vrhova. Primjerice, za graf sa slike 29, vrhovi 4 i 10 su istoj liniji izvršavanja, jer za svaku oznaku vrha 10 (oznake su {1.2, 1.3}), postoji oznaka vrha 4 ({1.1, 1.2.1, 1.3.1}) s kojom je oznaka iz vrha 10 u nekoj relaciji predak-nasljednik. Kao primjer drugačije situacije, u istom primjeru, vrhovi 9 i 3 nisu u istoj liniji izvršavanja. Lemom 3 pokazat će se da je dovoljno naći jedan par oznaka u odnosu predak-nasljednik za neka dva vrha kako bi se zaključilo da su u istoj liniji izvršavanja prije reduciranja oznaka.

U nizu tvrdnji koje slijede umjesto termina preduvjet koristi se pojam prethodnik, zbog odnosa prethodnik-nasljednik u terminima grafa.

Korolar 3. Za WF model kreiran odmah nakon postupka označavanja vrhova algoritmom iz Tablica 6, dva vrha X i Y su u istoj liniji izvršavanja ako i samo ako za svaki par oznaka $l_x \in \ell$ (X) i $l_y \in \ell$ (Y) vrijedi da je oznaka l_x jednaka oznaci l_y ili je oznaka l_x predak oznake l_y ili je oznaka l_y predak oznake l_x.

Dokaz: Prvi smjer tvrdnje slijedi direktno iz Korolara 2, jer u topološkom poretku između X i Y ne postoji vrh s oznakom koja je zajednički predak oznaka l_x i l_y pa l_x i l_y moraju biti u odnosu predak-nasljednik. Drugi smjer tvrdnje slijedi iz teorema 1, jer je nepostojanje odnosa predak-nasljednik među oznakama nužan uvjet da bi se neka dva klona mogla izvršavati paralelno. Ako su sve oznake u odnosu predak-nasljednik, onda nijedan par klonova vrhova X i Y ne može izvršavan paralelno, to jest vrhovi X i Y su istoj liniji izvršavanja

■

Lema 4. Nakon označavanja vrhova, a prije reduciranja oznaka, za svaka dva vrha p i q takve da je p prethodnik vrha q i za svaku oznaku $l_q \in \ell$ (q) postoji oznaka $l_p \in \ell$ (p) takva da je ili $l_q = l_p$ ili je l_q predak od l_p.

Dokaz: Slučaj I. Neka je p neposredni prethodnik od q. Ako je p jedini prethodnik vrha q tada se sve oznake vrha q dodaju u skup oznaka vrha p bez ikakvih modifikacija, pa je u tom slučaju $\ell(q) \subset \ell(p)$ iz čega odmah slijedi da za svaku oznaku $l_q \in \ell(q)$ postoji oznaka $l_p \in \ell(p)$ takva da je $l_q = l_p$. U slučaju da je vrh q imao više neposrednih prethodnika te je vrh p bio i-ti prethodnik, tada je na sve oznake vrha q prilikom dodavanja tih oznaka u skup oznaka vrha p dopisano *.i* što povlači da se za svaku dodanu oznaku $l_q \in \ell(q)$ u $\ell(p)$ pojavila oznaka l_q.i, koja je dijete oznake l_q.

Slučaj II. Neka p nije neposredni prethodnik od q. Tada postoji put u grafu p $v_1v_2..v_nq$, pri čemu se oznake iz vrha q kopiraju u v_n, iz vrha v_n u vrh v_{n-1} i tako sve do vrha p. U svakom od koraka vrijedi jedna od dviju mogućih situacija iz prvog slučaja iz čega slijedi da za svaku oznaku $l_q \in \ell(q)$ ili postoji oznaka $l_p \in \ell(p)$ takva da je $l_q = l_p$ (u slučaju da su svi vrhovi na tom putu imali samo po jednog neposrednog prethodnika) ili je l_q predak od l_p kao posljedica uzastopnih dopisivanja rednih brojeva pojedinih prethodnika.

■

Lema 5. Nakon postupka označavanja vrhova, dva vrha ne mogu biti u istoj liniji izvršavanja ako jedan vrh nije prethodnik drugog vrha.

Dokaz: Neka je pretpostavka da su vrhovi X i Y u istoj liniji izvršavanja, ali takvi da niti je X prethodnik od Y, niti je Y prethodnik od X. Neka je vrh Z po topološkom poretku prvi zajednički sljedbenik vrhova X i Y. Takav vrh mora postojati, jer je graf povezan te je vrh *End* barem jedan zajednički sljedbenik. Budući da su, po pretpostavci X i Y na odvojenim granama nasljeđivanja, nakon kopiranja oznaka iz vrha Z prema svojim prethodnicima, neka je grana prema X dobila oznake vrha Z s dopisanim *.i*, a linija u kojoj se nalazi vrh Y, neka je dobila oznake vrha Z uz dopisivanje *.j*. Budući da nije bilo reduciranja oznaka, vrhovi X i Y imaju oznake koje nisu u nikakvoj vezi oblika predak-nasljednik te se prema Korolaru 3 ne mogu nalaziti u istoj liniji izvršavanja što je u kontradikciji s pretpostavkom iz čega slijedi da X mora biti prethodnik od Y ili Y mora biti prethodnik od X.

■

Lema 4 dokazuje da su prije reduciranja oznaka svi prethodnici i svi sljedbenici nekog vrha (i prema Lemi 5 isključivo samo oni) u istoj liniji izvršavanja s tim vrhom zbog postojanja odnosa predak-nasljednik među njihovim oznakama. Preostaje pokazati da postupak reduciranja oznaka čuva redoslijed izvršavanja.

Lema 6. Dva vrha p i q su u istoj liniji izvršavanja nakon postupka reduciranja oznaka ako i samo ako su bili u istoj liniji izvršavanja prije postupka reduciranja oznaka.

Dokaz: ⇒Neka su dva vrha p i q u istoj liniji izvršavanja nakon reduciranja oznaka. Pretpostavi li se da p i q nisu bili u istoj liniji izvršavanja prije reduciranja oznaka, onda se njihove linije izvršavanja dijele na zajedničkom pretku i spajaju u prvom, po topološkom poretku, zajedničkom sljedbeniku. Budući da se od zajedničkog sljedbenika oznake kopiraju uz dopisivanje *.i* gdje je i redni broj prethodnika, nijedan element u granama različitih prethodnika, pa tako i p i q, ne mogu imati svu djecu neke oznake l iz zajedničkog sljedbenika i to vrijedi za svaku oznaku l iz sljedbenika. Stoga p i q nisu mogli reducirati svoje oznake da one postanu iste kao oznake iz one druge grane ili da budu preci, odnosno nasljednici oznaka iz suprotne grane te prema Lemi 3 znači da su bili u različitim granama te do reduciranja nije moglo doći.

⇐ Neka su p i q bili u istoj liniji izvršavanja prije postupka reduciranja oznaka te neka je, bez smanjenja općenitosti, p prethodnik od q. To povlači da je prije reduciranja oznaka svaka oznaka $l_q \in \ell(q)$ bila jednaka nekoj oznaci $l_p \in \ell(p)$ ili je l_q bila predak oznake l_p. Postupak reduciranja oznaka može promijeniti skup oznaka vrha p i/ili vrha q pri čemu se mogu promatrati tri moguće situacije u kojima se mogu nalaziti vrhovi koji su u istoj liniji izvršavanja prije postupka reduciranja oznaka.

U prvom slučaju svaki put od početnog vrha grafa do vrha q prolazi kroz p i svaki put od početnog vrha prema završnom vrhu koji prolazi kroz p, prolazi ujedno i kroz q (primjerice, vrhovi 1 i 5 sa slike 27). U tom slučaju i p i q će nakon reduciranja imati jednake oznake, jer p sadrži svu djecu oznaka nastalih kopiranjem oznaka vrha q njegovim prethodnicima.

Drugi slučaj je onaj u kojem svaki put od početnog vrha grafa do vrha q prolazi kroz p, ali postoji put od početnog vrha prema završnom vrhu koji prolazi kroz p, ali ne prolazi kroz q (primjerice, vrhovi 1 i 2 sa slike 27). Ako je vrh p sadržavao i one oznake koje imaju istog roditelja kao i oznake iz q, doći će do reduciranja oznaka iz p, na način da će takve oznake biti zamijenjene svojim roditeljima. U nizu takvih reduciranja može se dogoditi da za neku oznaku $l_q \in \ell(q)$ je odgovarajuća oznaka $l_p \in \ell(p)$ zamijenjena nekom oznakom koja je predak od l_p, pa je nova oznaka također u odnosu predak-nasljednik s oznakom l_q, pa su prema lemi 3 vrhovi i dalje u istoj liniji izvršavanja.

Treći slučaj je onaj u kojem postoji put od početnog vrha grafa do vrha q koji ne prolazi kroz p i postoji put od početnog vrha prema završnom vrhu koji prolazi kroz p, ali ne prolazi kroz q (primjerice, vrhovi 4 i 7 sa slike 27). U tom slučaju vrh p ne može reducirati oznaku $l_p \in \ell(p)$ koja bi bila nasljednik oznake $l_q \in \ell(q)$, jer postoji barem još jedna oznaka koja je nasljednik oznake l_q, a nije sadržana u vrhu p, pa se odnos između p i q ne mijenja.

■

Teorem 7. Svi klonovi vrhova grafa, pretvoreni u aktivnosti unutar WF modela ovise samo o svojim prethodnicima u grafu i niti jedan od klonova ne može biti izvršen prije nego što budu završeni oni klonovi koji odgovaraju prethodnicima iz grafa.

Dokaz: Lema 4 dokazuje da su sve oznake nekog vrha proslijeđene prethodnicima tog vrha te da niti jedan prethodnik pri tome nije ispušten, što znači da su svi prethodnici u istoj liniji izvršavanja prije postupka reduciranja oznaka. Lema 5 pokazuje da za svaki vrh ne može postojati vrh u istoj liniji izvršavanja, a da nije njegov prethodnik ili sljedbenik, a lema 6 pokazuje da reduciranje oznaka čuva liniju izvršavanja, što povlači da su svi prethodnici očuvani. Međusobni vertikalni poredak klonova stvorenih za vrhove iz grafa unutar WF modela osiguran je topološkim poretkom kao što je prikazano u poglavlju 5.2.3 (Međusobni poredak paralelnih grana može biti permutiran, no to ne utječe na vertikalni poredak).

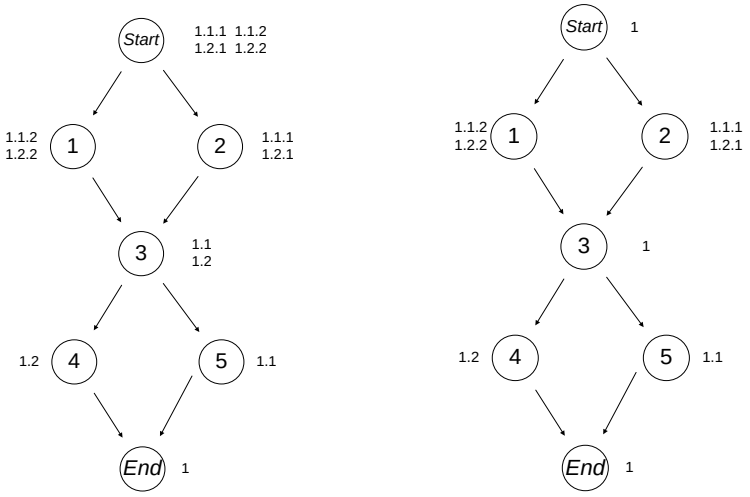
■

5.2.5 Poboljšanja algoritma

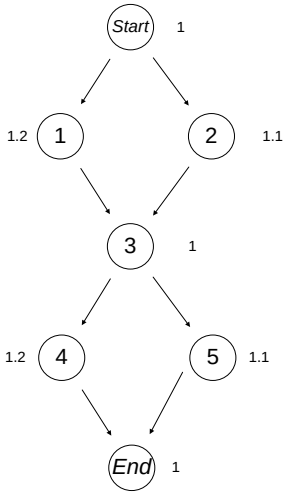
Iako algoritam prikazan u tablici 7 smanjuje broj klonova, postoje određeni nedostaci algoritma. Kao što se može vidjeti na slici 30, dvije krajnje desne paralelne grane s oznakama *B_1_2* i *B_1_3* su gotovo iste, osim aktivnosti *C_6* i *C_7* koje predstavljaju vrhove 6 i 7 iz grafa sa slike 29. Navedeni WF model mogao bi se poboljšati na način da se grananje ne odvija na početku grana *B_1_2* i *B_1_3* već tek prije aktivnosti *C_6* i *C_7.* U tom slučaju, svi dotad zajednički elementi mogli su biti dodani samo jednom u jednu zajedničku granu. Dakle, moguće poboljšanje sastoji se od spajanja dijelova pojedinih paralelnih grana koje su djeca istog paralelnog grananja. U ovom slučaju to bi značilo da oznake 1.2 i 1.3 mogu biti jednake za sve vrhove prije vrha 10, uključujući i vrh 10 te bi se za svaki takav vrh u svakoj oznaci koja počinje s 1.3 taj početak mogao zamijeniti s 1.2. Za sve vrhove iza vrha 10, oznaka 1.2 bi se trebala zamijeniti s 1.2.1, a 1.3. s oznakom 1.2.2.

Pokazuje se da što je graf „pravilniji“ u smislu da su paralelne grane adekvatno uparene s odgovarajućim sinkronizacijama, višak klonova je sve očitiji, kao u primjeru na Slika 33. Postavlja se pitanja zbog čega se ovo događa i može li se navedena situacija „popraviti“. Pogleda li se slika 33 može se primijeniti da je postupak reduciranja oznaka smanjio skup oznaka vrha 3 s {1.1, 1.2} na {1}, ali, prema algoritmu, ista stvar nije bila primjenjiva na vrhove 1 i 2. Međutim, kao što će biti pokazano, takvi skupovi oznaka mogu se smanjiti, ne zamjenom cijelih oznaka njihovim roditeljima, već zamjenom njihovih početaka, koji će u konačnici proizvesti graf kao na slici 34.

Dokaz za ove tvrdnje bazira se na dokazu sličnog poboljšanja, koje se može izvesti za one vrhove čiji skup oznaka nije reduciran, ali sadrži dvije ili više oznaka koje imaju istog roditelja, kao što je vrh 10 sa slike 29. Zamjena više oznaka koje imaju istog roditelja počinje od vrha koji sadrži te oznake te se zamjene prosljeđuju prema prethodnicima sve do početnog vrha grafa ili do prvog vrha koji ne sadrži navedene oznake.



Slika 33. Lijeva strana slike prikazuje graf nakon označavanja, a desna strana pokazuje graf nakon postupka reduciranja oznaka



Slika 34. Optimalno označavanje vrhove za lijevi graf sa slike 33

Teorem 8. Neka vrh v sadrži skup oznaka O={l_1, … ,l_n} koje imaju istog roditelja, pri čemu je l_1 u leksikografskom poretku prva takva oznaka. Ako se u vrhu v i svim njegovim prethodnicima svaka oznaka oblika l_iX zamijeni s l_1X te u svim sljedbenicima vrha v svaka oznaka oblika l_i zamijeni s $l_i.k$, gdje je k redni broj oznake unutar skupa O, linija izvršavanja nije narušena.

Dokaz: Bez smanjenja općenitosti, može se pretpostaviti da v sadrži dvije oznake oblike *l.i* i *l.j*, gdje je *l* zajednički prefiks navedenih oznaka te da je i<j. Svi prethodnici vrha v, pored mogućih drugih oznaka od svojih drugih sljedbenika, imaju oznake oblika *l.i.sufiks* i *l.j.suffiks*, gdje je sufiks nastao dopisivanjem uslijed postojanja više prethodnika. Opcionalno, v je mogao sadržavati i neke druge oznake, koje ne počinju s *l*, i za te će oznake među njegovim prethodnicima postojati oznake koje su nasljednici tih oznaka (uz dopisani sufiks) te je v u istoj liniji izvršavanja sa svojim prethodnicima. Zamjenom prefiksa *l.j* sa *l.i* u

vrhu v i svim njegovim prethodnicima, moguće je da će se skup oznaka nekog vrha smanjiti, ali neće se narušiti linija izvršavanja, jer će i dalje za svaku oznaku iz vrha v postojati oznaka u njegovim prethodnicima takva da je ta oznaka u odnosu predak-nasljednik s oznakom iz vrha v. Preostaje pokazati da zamjena prefiksa *l.i* s *l.i.1* te *l.j* s *l.i.2* u svim oznakama sljedbenicima vrha v ne mijenja liniju izvršavanja. Vrh v ima oznake *l.i* i *l.j* što znači da postoji zajednički sljedbenik vrha v koji ima oznaku *l*. Navedeni sljedbenik je imao barem još jednog prethodnika koji je sadržavao oznaku *l.k* (jer bi inače algoritam redukcije reducirao oznake *l.i* i *l.j* s oznakom *l* u vrhu v*).* Zamjena *l.i* s *l.i.1* te *l.j* s *l.i.2* povlači da će se grane u kojima se nalaze oznake *l.i.1* i *l.i.2* međusobno spojiti jedan korak prije, nakon čega slijedi spajanje s *l.k* i eventualnim ostalim granama.

U slučaju da je skup O imao više od dvije oznake, dokaz se odvija na isti način te vrijede iste tvrdnje.

■

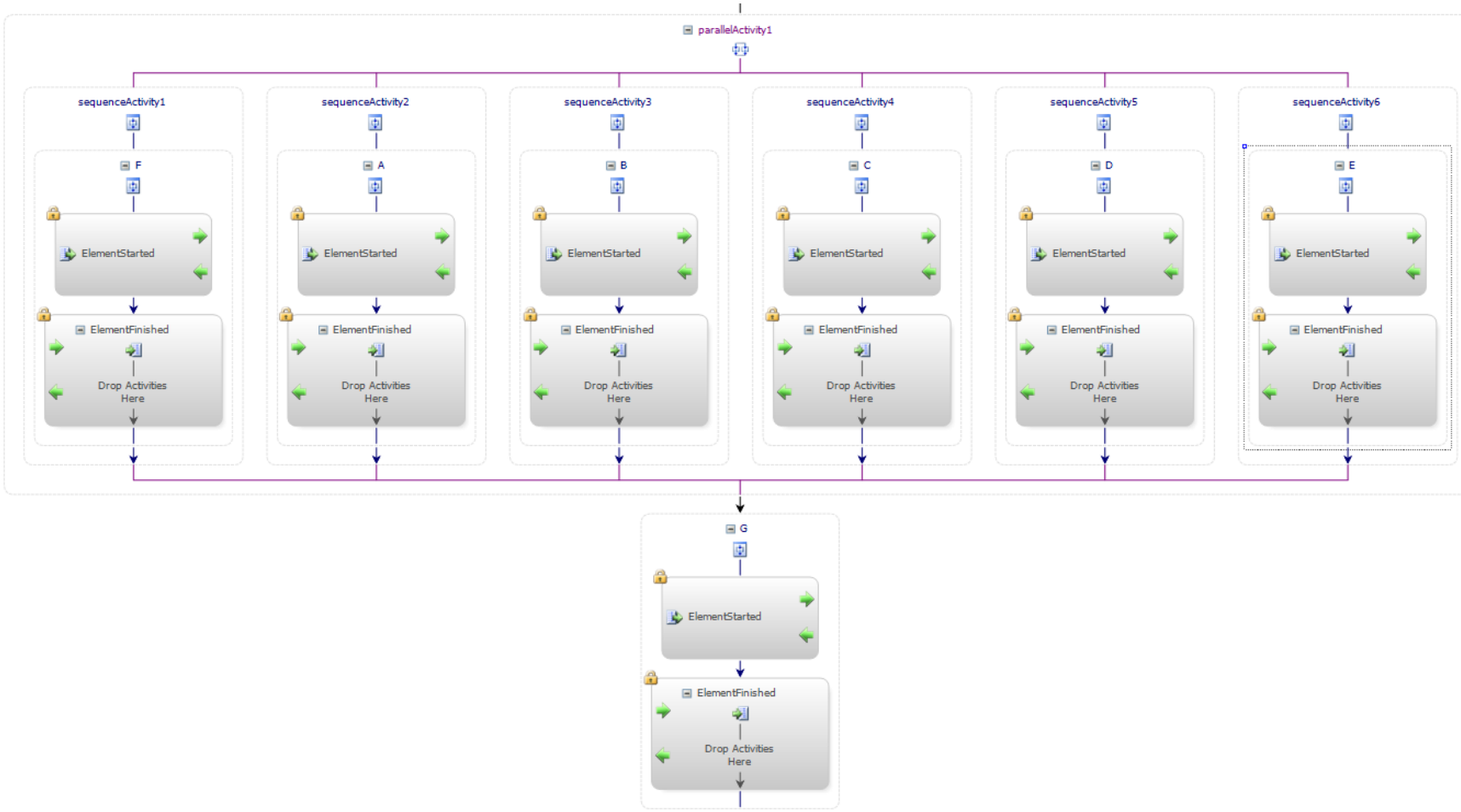
Reduciranje oznaka algoritmom iz tablice 7 u tom slučaju predstavlja specifičan slučaj opisan Teoremom 8. Za skup oznaka O={l_1, … ,l_n} koje imaju istog roditelja u prethodnicima vrha od kojeg zamjena počinje ne dolazi do zamjene prefiksa l_i s l_1 tako da oznake oblika l_iX postanu oblika l_1X već dolazi do zamjene prefiksa l_i s *roditelj*(l_i). Druga specifičnost je da ne dolazi do zamjene oznaka u sljedbenicima. U primjeru sa slike 33 to bi značilo da se zbog zamjene 1.1 i 1.2 s 1 u vrhu 3, u svim prethodnicima koji sadrže navedene prefikse, ti prefiksi mogu zamijeniti s 1, pa tako nastaje situacija kao na slici 34.

5.3 Pretvorba skupa mogućih prethodnika u WF model

Osim za slučaj u kojem se vrši raspoređivanje elemenata po vremenskim okvirima, pri čemu je prethodnike potrebno odabrati prije kreiranja modela čime oni postaju preduvjeti nekog elementa, moguće je da će izvršavanje nekog elementa biti uvjetovano završetkom jednog ili više skupova mogućih prethodnika. Realizacija takvog uvjeta složen je problem, jer WF nema posebnu aktivnost koja bi funkcionirala kao skup prethodnika, a istovremeno nudi manju slobodu u kontroli toka izvođenja za razliku od klasičnih programskih jezika. Za ilustraciju ideje rješenja problema skupa mogućih prethodnika poslužit će sljedeći primjer.

Neka element G kao preduvjet ima element F, te neka su definirani skupovi mogućih prethodnika {A, B, C} i {D, E} ranga 2, odnosno ranga 3. Kada bi se umjesto skupa mogućih prethodnika, radilo o preduvjetima te kada bi elementi bili prikazani bez vremenske komponente tada bi taj odsječak modela izgledao kao na slici 35.

Postavlja se pitanje može li se prikazani model modificirati da simulira skupove mogućih prethodnika. Primjerice, može li se u situaciji u kojoj su završeni elementi F, A, C i element E prekinuti čekanje na paralelne grane u kojima se nalaze elementi B i D? Gledano formalno, samo sa stanovišta WF-a, odgovor je ne. Brisanje aktivnosti koje su već pokrenute nije moguće. Aktivnosti tipa *ReceiveActivity* ostaju u stanju čekaju na poruku da je element završen.



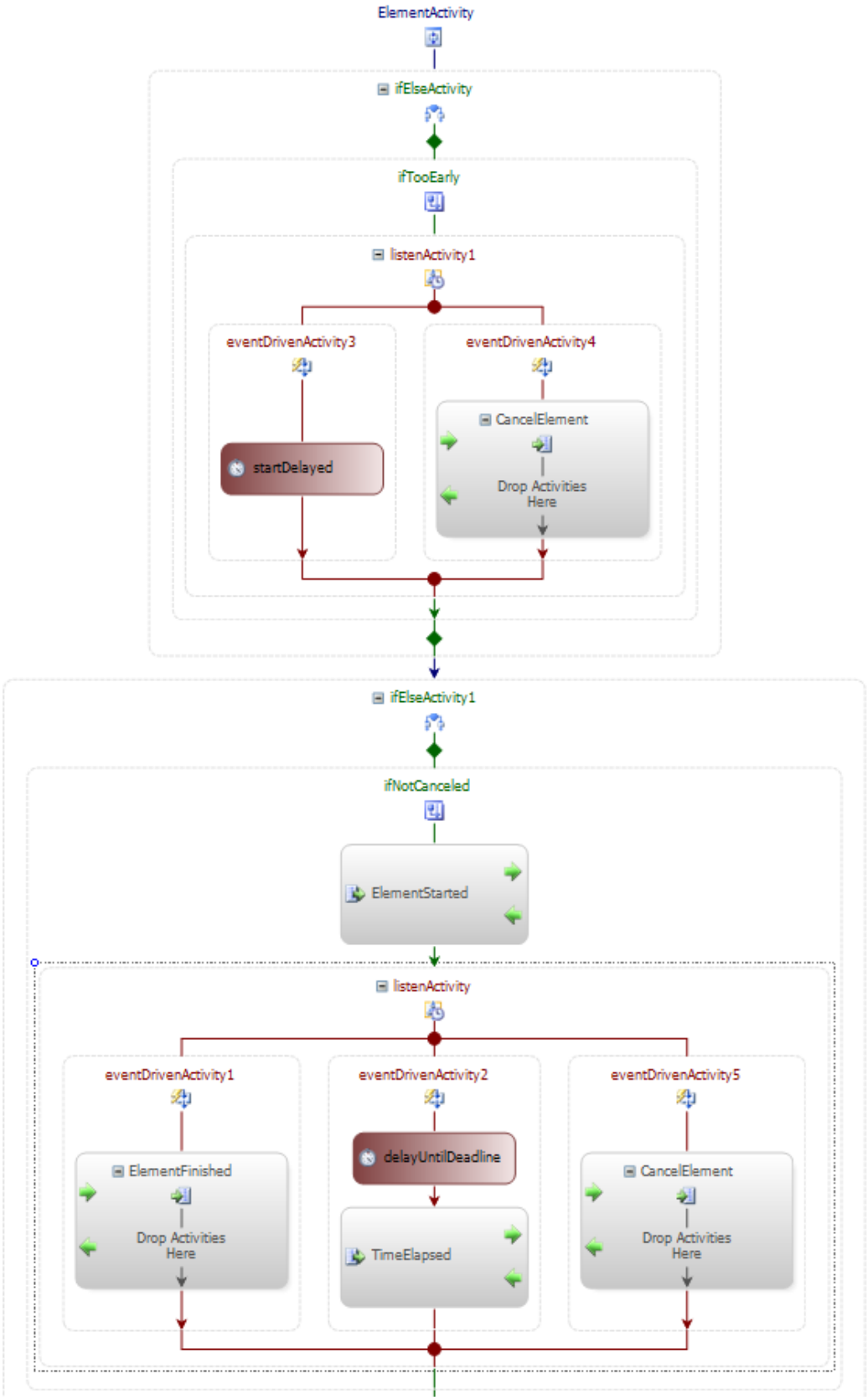
Slika 35. Mogući prethodnici i preduvjeti kao paralelne grane

Međutim, ono što je moguće je inicirati poziv postupka *CancelElement* za elemente u granama koje odgovaraju istom skupu mogućih prethodnika. U slučaju da se u pojedinoj grani nalazilo više elemenata, tada je, korištenjem dinamičke promjene instance modela protoka poslova (poglavlje 2.3.2) moguće obrisati sve aktivnosti u navedenoj grani koje slijedi iza aktivnosti *ReceiveActivity*. Kako inicirati poziv objašnjeno je u opisu djelovanja omotača sloja protoka poslova u sljedećem poglavlju. Eventualnom iniciranju poziva će prethoditi niz aktivnosti u kojima će se provjeriti da li je završetkom konkretne grane ispunjen uvjet skupa mogućih prethodnika te, ako je, inicirat će se poziv za završetak svih onih grana iz istog skupa mogućih prethodnika. Stoga je prilikom kreiranja modela za svaki element u paralelnoj grani koja predstavlja jedan element iz skupa mogućih prethodnika potrebno dodati mogućnost otkazivanja elementa.

Dodatni problem koji treba riješiti je odgađanje početka izvršavanja nekog elementa kao što je prikazano na slici 22. Jednom pokrenutu aktivnost čekanja nije moguće promijeniti, pa je tu aktivnost čekanja potrebno staviti kao dio aktivnosti *Listen* koja će se dovršiti kad se aktivira jedan od pridruženih događaja, što će u ovom slučaju biti istek vremena ili poziv postupka za otkazivanje elementa. Nakon navedenih promjena novi izgled pojedinog elementa s vremenskom komponentom i odgodom početka je prikazan na slici 36. Pri tom, ako se primijeni opaska iz 4.2.3 tada se za izračun krajnjeg vremena završetka elementa iz pojedinog skupa mogućih prethodnika koristi ono vrijeme završetka koje je vezano uz taj skup.

Navedene modifikacije pojedinačnog elementa te dodavanje provjere za iniciranje otkazivanja pojedinih elemenata omogućit će da se mogući prethodnici promatraju na gotovo isti način kao i preduvjeti. Nakon što se stvori zajednički graf u kojem će se nalaziti svi preduvjeti, u graf je potrebno dodati sve one elemente koji se nalaze u skupovima mogućih prethodnika, a nisu već unutar zajedničkog grafa. Nakon reduciranja lukova slijedi dodavanje prethodnika pojedinim elementima pri čemu je potrebno na luku označiti o kojem se skupu mogućih prethodnika radi i koji je njegov rang. Razlog zašto se reduciranje lukova vrši prije dodavanja prethodnika, a ne nakon, leži u činjenici da se u slučaju ispunjenja prethodnika preostali elementi u paralelnim granama istog skupa mogućih prethodnika brišu i neće se moći pokrenuti. Kako bi se osiguralo da takvi elementi ipak postoje barem na jednom mjestu u modelu, njihove originalne veze su sačuvane.

Prilikom pridjeljivanja oznaka nekom vrhu, ako se radi o luku koji ima oznaku skupa mogućih prethodnika i rang skupa, potrebno je taj podatak evidentirati kako bi se u omotaču mogla izvršiti provjera ispunjenja uvjeta skupa mogućih prethodnika.

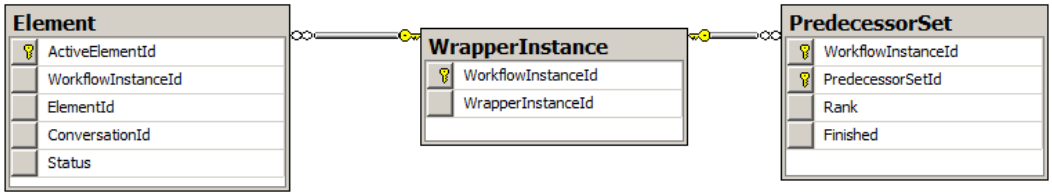


Slika 36. Model za pojedinačni element s ugrađenom mogućnosti otkazivanja

5.4 Omotač sloja protoka poslova

Prilikom pretvaranja grafa konačnog modela u WF model za one vrhove koji imaju dvije ili više oznaka u WF modelu se pojavljuje više klonova elemenata koji odgovaraju pojedinom vrhu. Korisniku aplikacije koja koristi navedeni WF model mora se pojavljivati samo jedan od klonova elementa te je potrebno konstruirati mehanizam rada tih klonova na način da se istovremeno korisniku prikazuje samo jedan klon, ali i da završetak prikazanog elementa uzrokuje završetak svih ostalih klonova tog elementa. Teorijski, problem klonova moguće je riješiti u bilo kojem sloju. Međutim, s obzirom da se radi o specifičnosti konkretnog modela protoka poslova, koje razvojnik ostalih slojeva ne mora biti svjestan, navedeni problem treba riješiti unutar sloja protoka poslova. Time se postiže jasnija podjele među slojevima i lakše održavanje, jer bi prosljeđivanjem problema susjednom sloju učinilo oba sloja složenijim i težim za održavanje.

Susjedni slojevi aplikacije sa slojem protoka poslova komuniciraju putem WCF servisa, pa je sloju protoka poslova svejedno komunicira li s vanjskim sustavom ili susjednim slojem. Stoga, kao idealno rješenje nameće se omotač sloja protoka poslova također izveden kao WF model i izložen kao WCF servis koji implementira iste ugovore kao i originalni model protoka poslova. Na taj način ostali slojevi neće biti ni svjesni postojanje posrednika prema sloju protoka poslova, koji će se brinuti za pravilan rad s klonovima pojedinih elemenata.



Slika 37. Tablice potrebne za rad omotača protoka poslova

Sam rad omotača je relativno jednostavan i zahtjeva tri tablice prikazane na slici 37. U tablici *WrapperInstance* bilježe se identifikatori originalne instance protoka poslova (one koja sadrži klonove) i instance omotača pridružene originalnoj instanci protoka poslova.

Početak nekog elementa u originalnoj instanci protoka poslova, prema opisu iz poglavlja 5.1, pozvat će postupak *StartElement*. Razlika je utoliko što se taj postupak neće pozivati na vanjskom sustavu nego na omotaču. Omotač će u tablici *Element* evidentirati o kojem elementu se radi (identifikator elementa) te će zabilježiti vrijednost parametra *ConversationId* koji predstavlja identifikator konverzacije, potreban da bi se prepoznao pravi element prilikom poziva postupka *ElementFinished*. U slučaju da se radilo o prvoj aktivaciji nekog elementa, a ne o nekom njegovom klonu, omotač će tada dinamički promijeniti svoj WF model dodavanjem nove aktivnosti *ReceiveActivity* koja će čekati na poruku o završetku elementa. Istovremeno, vanjskom sustavu će poslati podatke o svom identifikatoru instance i parametar *ConversationId* koji odgovara novostvorenoj aktivnosti. Za svaku sljedeću poruku iz originalne instance protoka poslova vezane za već aktivirani

element podaci će se zabilježiti u bazi, ali poruka se neće proslijediti prema vanjskom sustavu. Isti princip prosljeđivanja poruka vrijedi i u slučaju isteka vremena izvršavanja nekog elementa.

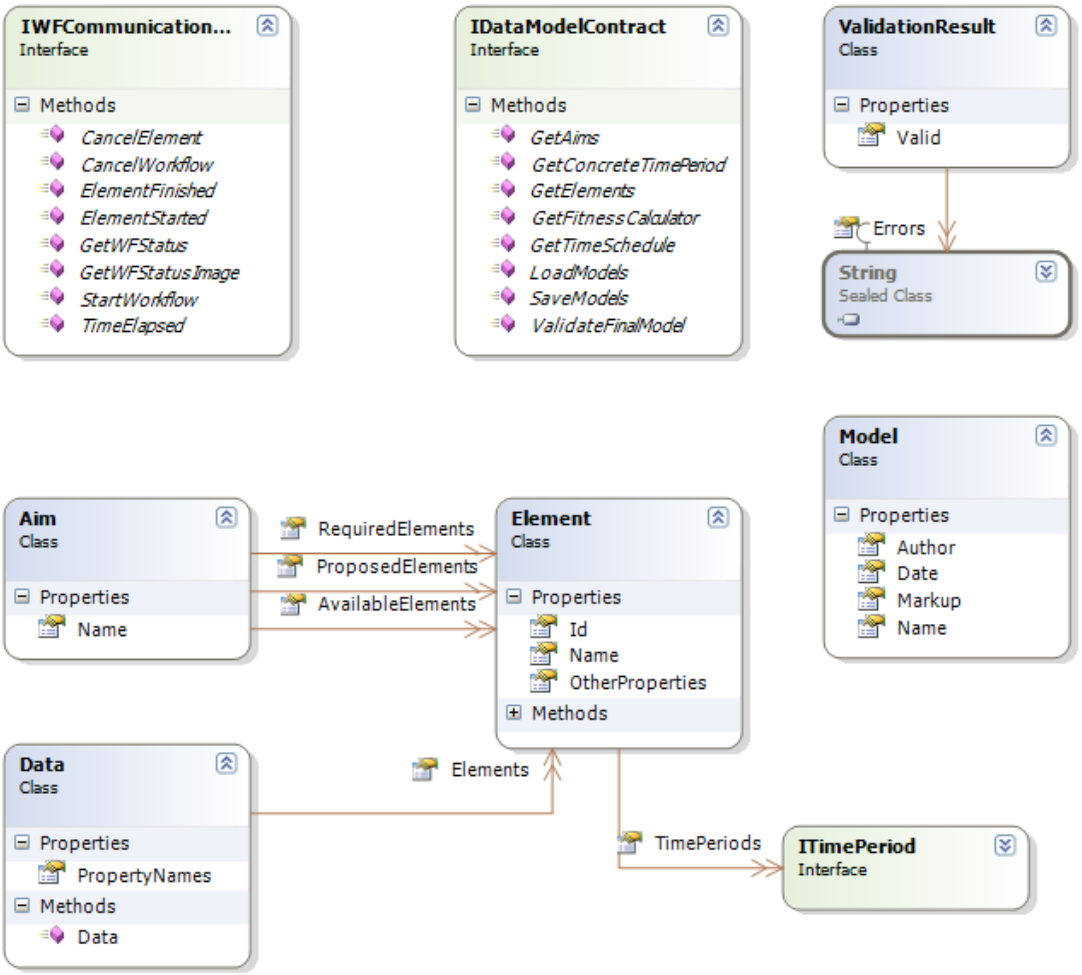
Prilikom početka elementa koji je sadržan u nekom skupu mogućih prethodnika, poruka će sadržavati i podatak o kojem se skupu mogućih prethodnika radi te koliki je rang tog skupa. U slučaju da je to prvi element pokrenut iz tog skupa omotač će u bazu pohraniti navedene podatke.

Po završetku nekog elementa vanjski sustav na osnovu parametra *ConversationId* i identifikatora instance protoka poslova u omotaču aktivira aktivnost *ReceiveActivity*. Nakon toga omotač u bazi traži podatke o svim klonovima tog elementa i za svaki element poziva postupke *ElementFinished* na originalnoj instanci protoka poslova. U slučaju da je dovršen element koji je bio član skupa mogućih prethodnika broj završenih prethodnika će se uvećati za 1. Postane li taj broj jednak rangu skupa mogućih prethodnika, doći će do dinamičke promjene modela otkazivanjem elemenata iz istog skupa mogućih prethodnika u onim granama koje u sebi imaju zapisan identifikator skupa mogućih prethodnika. Budući da se pojedina instanca protoka poslova unutar WF uvijek odvija u jednoj dretvi, nema potrebe za dodatnim mehanizmima sinkronizacije.

6. Sučelje za rad s parcijalnim modelima protoka poslova

Rad s parcijalnim modelima protoka poslova omogućen je implementacijom skupa programskih sučelja za razmjenu podataka između instanci protoka poslova, a izrada parcijalnih modela protoka omogućena je izradom vlastite aplikacije pri čemu se za dohvat elemenata i validacija konačnog modela odvija kroz za to predviđen skup WCF servisa.

Slika 38 prikazuje dijagram glavnih razreda i postupaka kojima su definirani ugovori za WCF servise.



Slika 38. Dijagram glavnih razreda i postupaka za ugovore u WCF servisima

Tablica 10 prikazuje odnos pojedinih komponenti i postupaka iz ugovora te se na osnovu takve matrice može vidjeti koja komponenta sustava koristi, to jest poziva, određeni postupak, odnosno koja komponenta implementira pojedine postupke iz ugovora. Tablica služi kao jasan prikaz zahtjeva na vanjski sustav u pogledu implementacije ugovora. Osim implementacije postupaka za primanje i slanje obavijesti oko početka i završetka nekog elementa, vanjski sustav koji se želi koristiti modelom protoka poslova izrađenim na osnovu parcijalno definiranih modela, mora implementirati postupke koji određuju značenje elemenata.

Tablica 10. Odnos komponenti i postupaka u ugovoru (P=Poziva, I=Implementira)

	Vanjski sustav	Omotač	Instanca modela protoka poslova	Aplikacija (izrada parcijalnih modela)	Aplikacija (izrada konačnog modela)
StartWorkflow	P	P/I	P/I		
CancelWorkflow	P	P/I	P/I		
ElementStarted	I	P/I	P		
ElementFinished	P	P/I	I		
TimeElapsed	I	P/I	P		
CancelElement		P	I		
GetWFStatus	P	P/I	I		
GetWFStatusImage	P	P/I	I		
GetAims	I				P
GetConcreteTimePeriod	I			P	P
GetElements	I			P	
GetFitnessCalculator	I				P
GetTimeSchedule	I				P
LoadModels	I			P	
SaveModels	I			P	
ValidateFinalModel	I				P

Rad aplikacije s grafičkim sučeljem za rad s parcijalnim modelima može se podijeliti u nekoliko ključnih dijelova: učitavanje podataka, stvaranje parcijalnih modela, pohrana modela te odabir elemenata za konačni model i stvaranje WF modela.

6.1 Izrada parcijalnih modela protoka poslova

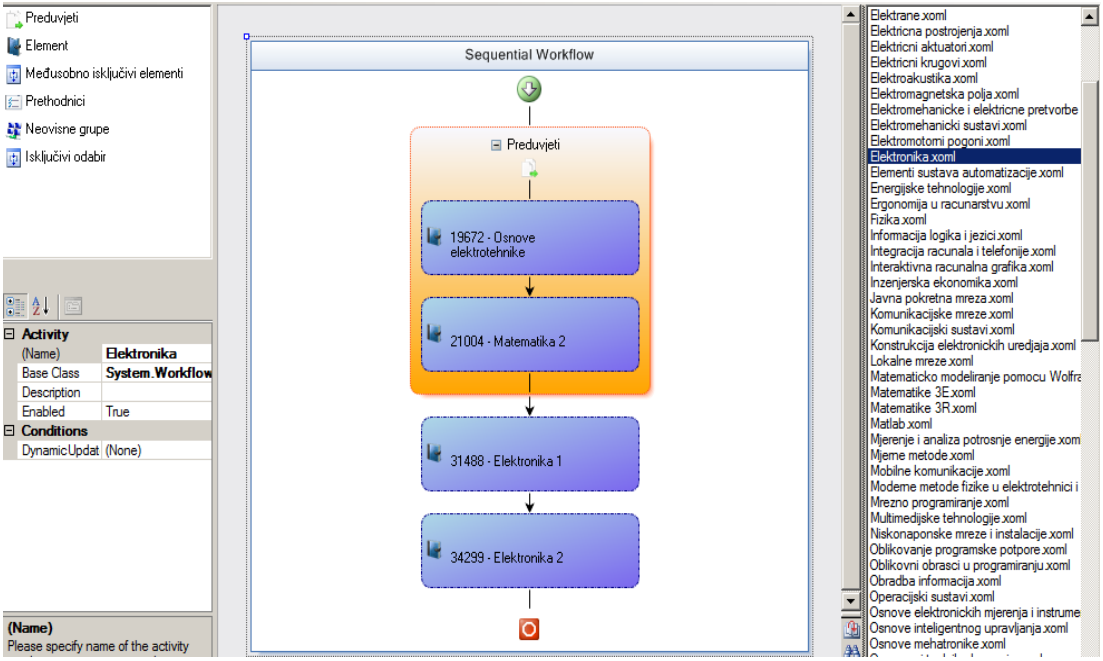
Prvi dio sastoji se od učitavanje podataka o elementima i eventualnog dohvata postojećih parcijalnih modela. Kako je aplikacija namijenjena za različita značenja elemenata, prvi korak je odabir adrese WCF servisa koji će omogućiti rad s parcijalnim modelima, to jest pružiti podatke o elementima i postojećim modelima. Servis se može nalaziti na bilo kojoj adresi te je jedino bitno da implementira postupke iz unaprijed definiranog ugovora. Po odabiru željene adrese, započinje aktivnost poziva postupka *GetElements* na WCF servisu čime se dohvaćaju podaci o elementima. Svaki element je definiran svojim jedinstvenim identifikatorom i imenom, a može sadržavati i proizvoljni broj ostalih vrijednosti pospremljenih u rječniku u svojstvu *OtherProperties*. Ključevi rječnika i očekivani tipovi vrijednosti zapisani su u svojstvu *PropertyNames* i služe da bi se forma za prikaz elemenata prilagodila za pojedino značenje elementa i prikazala sve potrebne podatke na formi. Za pojedini element može biti definirana vremenska komponenta što je zapisano u kolekciji *TimePeriods* u kojoj se mogu nalaziti vrijednosti svih triju tipova vremenskih razdoblja iz poglavlja 4.1.

Nakon dohvata svih elemenata potrebno je odabrati lokaciju na kojoj se nalazi repozitorij parcijalnih modela, što može biti mapa na korisnikovom računalu nakon čega slijedi odabir mape, ili prethodno definirani WCF servis na kojem će se pozvati postupak *LoadModels* te

će se modeli pohraniti u privremenu mapu kod korisnika. Modeli su pospremljeni kao *xoml* datoteke koje predstavljaju zapis WF modela protoka poslova u XML formatu. Po dohvatu podataka o modelima može se pristupiti izradi parcijalnih modela.

Slika 39 i slika 40 prikazuju izgled sučelja za izradu parcijalnih modela. U gornjem lijevom kutu nalaze se osnovne komponente od kojih se grade parcijalni modeli (preduvjeti, skupovi mogućih prethodnika, međusobno isključivi elementi, isključivi odabir) prošireni s dvije aktivnosti *Neovisne grupe*, koja omogućava da se na istom modelu ujedno definiraju i preduvjeti i skupovi mogućih prethodnika (slika 40). Dodavanje elementa na model vrši se povlačenjem odgovarajuće komponente na središnji dio. O kojem elementu se točno radi može se odrediti upisivanjem njegove šifre ili odabirom elemente iz forme s popisom elemenata.

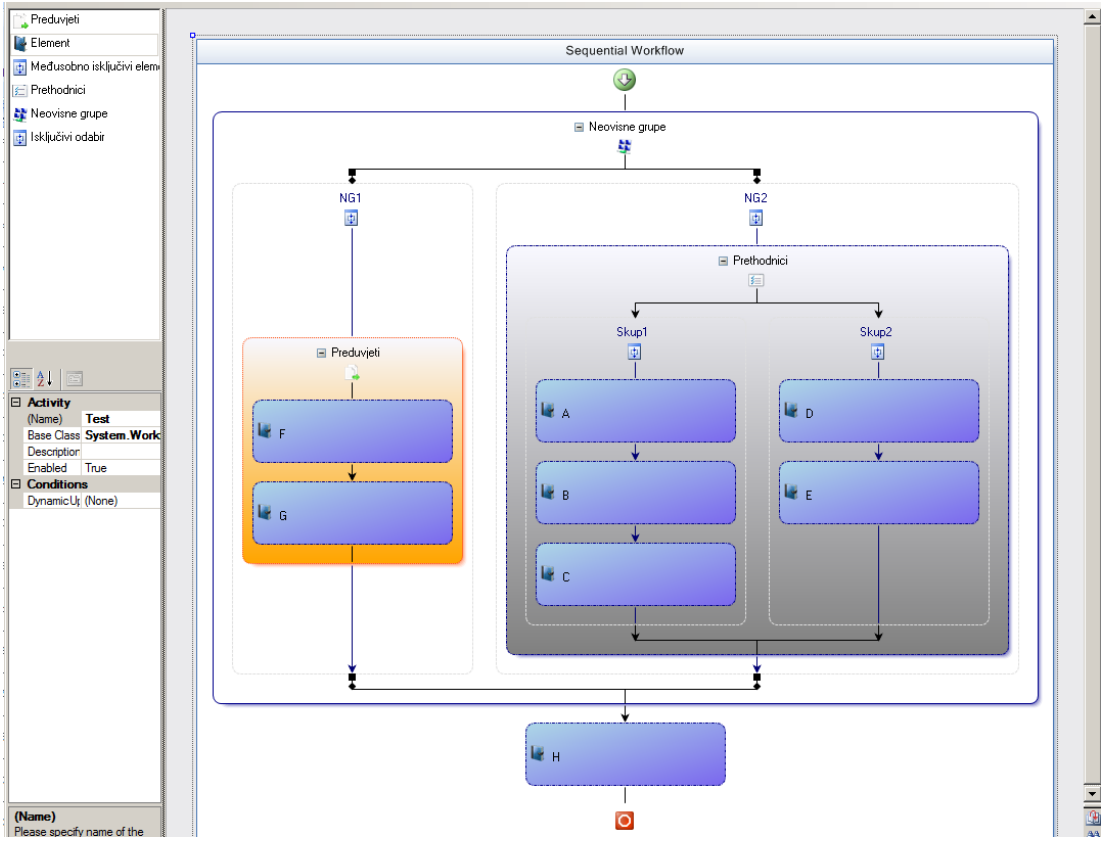
Desni dio aplikacije prikazuje popis dotad definiranih parcijalnih modela. Klikom na pojedini naziv, u središnjem dijelu će se prikazati konkretni parcijalni model.



Slika 39. Grafičko sučelje za definiranje parcijalnih modela

U fokusu promatranja nalazi se glavni element te njegovi preduvjeti i/ili skupovi mogućih prethodnika, odnosno isključujuće odluke koje slijede iza glavnog elementa. Pojedini parcijalni model ne mora nužno predstavljati samo jednu jednostavnu situaciju iz poglavlja 3. Čest je slučaj da ista osoba definira nekoliko elemenata koji su jedan drugome preduvjet, pa je radi jednostavnosti na jednom parcijalnom modelu moguće odmah definirati više međusobnih odnosa među elementima. Primjerice u slučaju na slici 39 *Elektronika 1* je preduvjet *Elektronike 2*, a *Elektronika 1* za preduvjet ima *Osnove elektrotehnike* i *Matematiku 2*. Forsiranjem parcijalnih modela protoka poslova koji bi sadržavali samo osnovnu definiciju morala bi se napraviti tri parcijalna modela; u prvom bi *Elektronika 1* bila preduvjet *Elektronike 2*, u drugom *Osnove elektrotehnike* preduvjet *Elektronike 1* i u trećem

Matematika 2 preduvjet *Elektronike 1*. Konačni ishod bi pri integriranju parcijalnih modela, bio potpuno jednak. Omogućavanjem ovakvih modela dobitak se očituje na brzini i preglednosti za pojedinog autora. Slika 40 predstavlja hipotetski primjer u kojem je na jednom parcijalnom modelu definirano više međusobnih odnosa među elementima, pa tako element H ima za preduvjete elemente F i G i dva skupa mogućih prethodnika, koje čine skup s elementima {A, B, C} i skup {D, E}. Rangovi pojedinih skupova mogućih prethodnika postavljeni su u svojstvima pojedinog skupa.



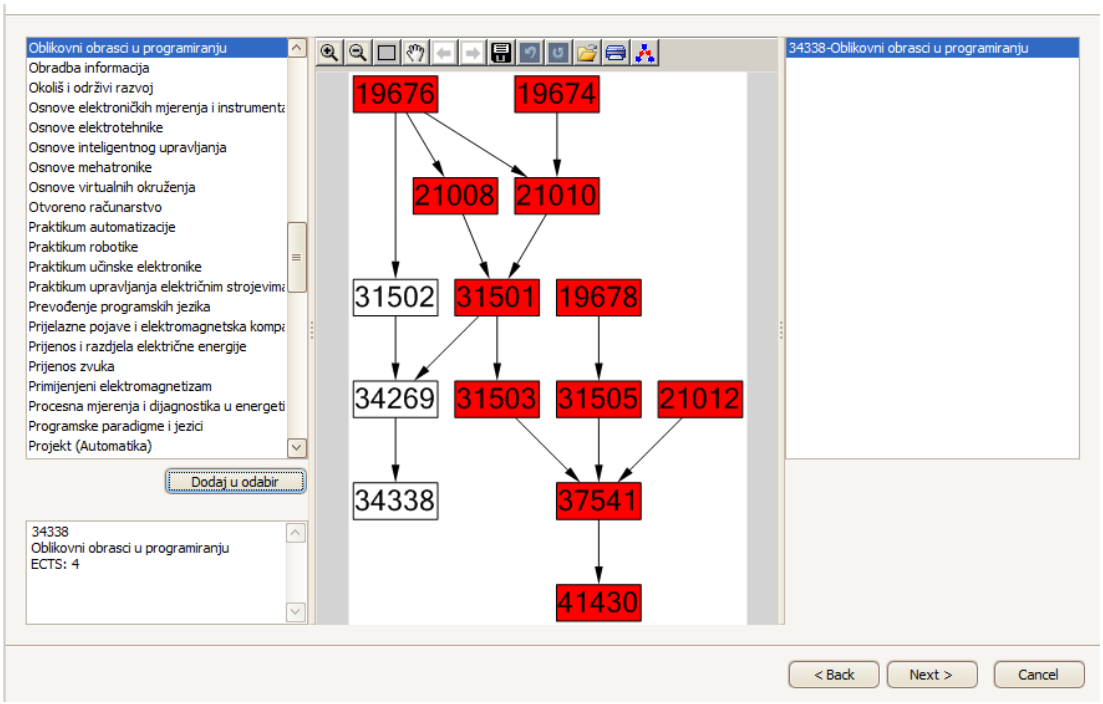
Slika 40. Hipotetski primjer za istovremeno definiranje preduvjeta i skupova mogućih prethodnika

Nakon što su parcijalni modeli stvoreni, slijedi njihova validacija prema pravilima iznesenim u poglavlju 3. Za elemente koji su imali vremensku komponentu moraju se izvršiti provjere iz poglavlja 4.2. U slučaju da je neki od elemenata imao definirano varijabilno vremensko razdoblje, povezivanje s fiksnim vremenskim razdobljima se može obaviti pozivom postupka *GetConcreteTimePeriod* za svako takvo vremensko razdoblje. Alternativno, korisniku se može dopustiti da sam unose konkretne datume. Kreirani parcijalni modeli mogu se pohraniti lokalno ili na server pozivom postupka *SaveModels*[27].

[27] Jednostavnosti radi, u primjeru nisu postavljena ograničenja nad mogućnostima promjene definiranih parcijalnih modela. Primjerice, može se definirati da pojedini korisnik može mijenjati samo one parcijalne modele koje je sam stvorio, za razliku od korisnika s administratorskim dozvolama. Provjeru je potrebno obaviti na strani WCF servisa prilikom pokušaja spremanja promijenjenih modela.

6.2 Odabir elemenata za konačni model i stvaranje WF modela

U postupku stvaranja modela mogu se razlikovati dvije situacije: stvaranje modela isključivo na osnovu veza definiranih u poglavlju 3 te stvaranje modela uz raspoređivanje elemenata po vremenskim okvirima. U oba slučaja slijedi odabir elemenata koji će sudjelovati u konačnom modelu. Odabir elemenata započinje odabirom jednog od ciljeva dostupnih pozivom postupka *GetAims*. Rezultat poziva sadrži naziv pojedinog cilja te tri popisa elemenata, *RequiredElements, SuggestedElements*, *AvailableElements* koji sadrže one elemente koji se moraju naći u konačnom modelu, elemente koji se korisniku sugeriraju kao preporuka za ostvarenje navedenog cilja i popis elemenata među kojima se može vršiti odabir. Prema poglavlju 3.4 u svakom sustavu mora postojati barem jedan cilj, koji može biti trivijalni cilj koji dopušta odabir svih elemenata na sustavu bez sužavanja skupa mogućih elemenata. Trivijalni cilj, primjerice nije pogodan u odabiru predmeta u studijskom programu, jer bi se moglo doći u situaciju da korisnik odabere samo neke predmete i da nikad ne dođe do određenog cilja, što bi u konkretnom slučaju bio završni rad iz nekog područja.



Slika 41. Dodavanje elemenata u konačni model

Slika 41 prikazuje formu za odabir elemenata u konačni model, pri čemu je konačni model prikazan usmjerenim grafom, Lukovi koji predstavljaju preduvjete iscrtani su punom strelicom, a skupovi mogućih prethodnika isprekidanom uz oznaku identifikatora i ranga skupa mogućih prethodnika. Crvenom bojom označeni su obvezni elementi i oni se ne mogu ukloniti iz grafa.

Dodavanjem pojedinog elementa, u konačni model se automatski dodaju i svi njegovi preduvjeti. Za izradu grafa korišten je alat Automatic Graph Layout [28] [Msagl] koji omogućava automatski razmještaj grafa prema više metoda među kojima je korišten razmještaj prema [Sugiyama1981].

Osim pravila iz poglavlja 3, dodatno se provjerava rezultat funkcije *valjan* za konačni model koja je za konkretno značenje elementa implementirana u postupku *ValidateFinalModel*. Po završetku odabira elemenata u model će se dodati i svi oni elementi koji su nužni za ispunjenje uvjeta skupova mogućih prethodnika. Ako je neki od elemenata imao definirano varijabilno vremensko razdoblje, povezivanje s fiksnim vremenskim razdobljima se mora obaviti pozivom postupka *GetConcreteTimePeriod* za svako takvo vremensko razdoblje. Ako je konačni model valjan aplikacija će izraditi WF model prema poglavlju 5.

Želi li se izvršiti raspoređivanje elemenata po vremenskim okvirima, korisnik sam mora eksplicitno odabrati željene elemente iz skupa mogućih prethodnika. Prije početka raspoređivanja potrebno je dohvatiti podatke o vremenskim okvirima (postupak *GetTimeSchedule*). Alternativno, korisniku se može dopustiti unos nekih hipotetskih vremenskih okvira.[29]

Prije pokretanja genetskog algoritma za problem raspoređivanja aplikacija će na osnovu dozvoljenih vremenskih okvira za pojedine elemente, podijeliti elemente u dva skupa: skup fiksnih elemenata, koji čine oni elementi koji moraju biti u točno određenom vremenskom okviru i koji ne sudjeluju u genetskom algoritmu (osim za izračun dobrote pojedinog vremenskog okvira) i skup elemenata koji mogu biti u više vremenskih okvira za koje će se tražiti jedinka koja odgovara traženim uvjetima.

Funkcija dobrote računa se na osnovu dvije komponente od koji je druga specifična za pojedino značenje elemenata. Udio pojedine komponente može se promijeniti prije poziva algoritma. Teorijski, druga komponenta može biti izvedena kao postupak na WCF servisu, međutim zbog performansi kod velikog broja iteracija, izračun će se izvoditi lokalno. Binarni sadržaj biblioteke za izračun funkcije dobrote bit će dohvaćen pozivom postupka *GetFitnessCalculator* i korišten lokalno. Biblioteka za izračun implementira sučelje *IFitnessCalculator* i za izračun dobrote za svaki pojedini okvir koristi funkciju koja prima identifikator vremenskog okvira, popis elemenata dodijeljen navedenom vremenskom okviru i popis svih elemenata, kako bi se moglo izvesti normiranje funkcije dobrote.

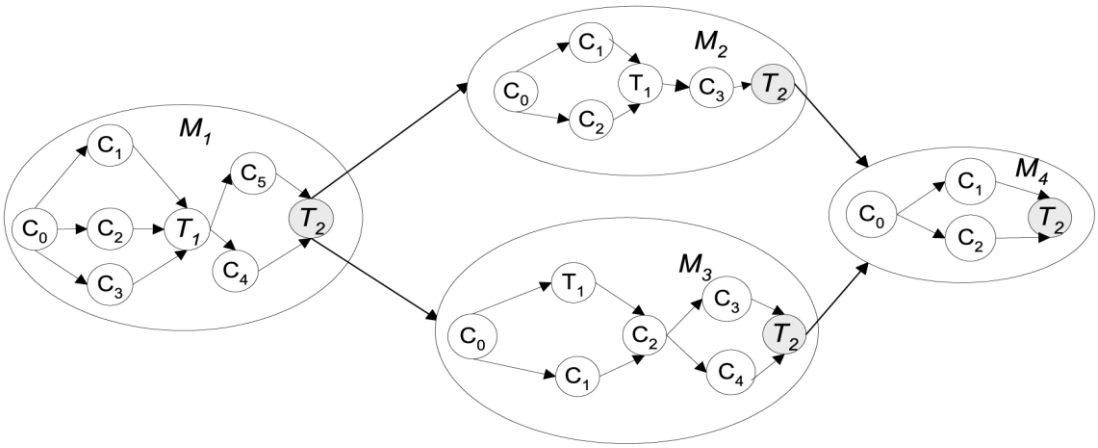
[28] Za potrebe ovog rada korištena je akademska licenca navedenog alata.
[29] Model dobiven s korisnički unesenim vremenskim okvirima može poslužiti za testiranje, a ispravan može postati budu li zaista postojali takvi vremenski okviri.

7. Primjena modela na analizu slučaja

U ovom poglavlju dane su dvije analize slučaja nad kojima se može primijeniti modeliranje protoka poslova na temelju parcijalnih modela. Prvi primjer, na kojem će biti i demonstriran rad aplikacije stvorene za potrebe ovog rada je modeliranje redoslijeda cjelina u sustavu za udaljeno učenje AHyCo. Drugi primjer je izrada modela protoka poslova za odabir predmeta i raspoređivanje predmeta po semestrima studijskog programa po smjeru FER-2 na Fakultetu elektrotehnike i računarstva.

7.1 Prijedlog ugradnje parcijalnih modela protoka poslova u sustav AHyCo

Sustav AHyCo (Adaptive Hypermedia Courseware) je prilagodljivi hipermedijski sustav za učenje i provjeru znanja te se koristi na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu i na Informatičkom odjelu Filozofskog fakulteta Sveučilišta u Rijeci. Prema [AHyCoRI] model domene koji opisuje strukturu sadržaja za učenje sastoji se od dvije razine. Jednu razinu čine elementarne granule znanja ili koncepti grupirani u module, a drugu razinu čine moduli grupirani u predmet učenja. Slika 42 prikazuje graf modela koji sadrži koncepte, koji mogu biti cjeline i provjere, podijeljene u module. Koncepti modela domene povezani su relacijom $\prec$ pri čemu je značenje relacije $\prec$ takvo da određeni koncept mora/treba biti poznat/naučen prije nekog drugog koncepta. U slučaju da se radi o nužnom uvjetu, takva relacija bi bila ekvivalentna preduvjetnoj relaciji iznesenoj u 3.2.



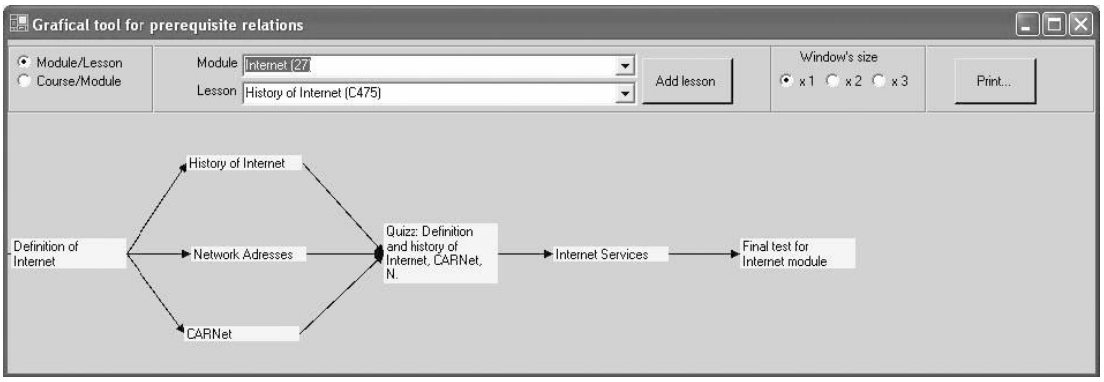
Slika 42. Cjeline i provjere (testovi) podijeljeni u module [Hoić2005]

Prema [Hoić2005] pedagoška ograničenja među cjelinama mogu varirati ovisno o domeni učenja te značenje relacije $\prec$ treba promatrati u kontekstu područja koje se uči, pa je tako prema [AHyCoRIa] kretanje među cjelinama potpuno slobodno te se korisniku ostavlja mogućnost da sam bira cjeline koje želi učiti. Pritom se popis cjelina koje može otvoriti dijeli na glavne cjeline, preporučene cjeline i cjeline koje se ne preporučuju. Među cjelinama koje se ne preporučuju nalaze se one cjeline za koje relacija $\prec$ još nije zadovoljena. Za cjeline u glavnim i preporučenim cjelinama relacija $\prec$ je zadovoljena uslijed neposredne povezanosti relacijom $\prec$ odnosno kroz niz tranzitivnih relacija $\prec$.

Model protoka poslova zasnovan na parcijalnim modelima prezentiran u ovom radu je neovisan o značenjima pojedinih elemenata. Stoga, može se pretpostaviti da se relacija $\prec$ može modelirati relacijom preduvjeta, a da se o samom načinu prikaza cjelina brine sam sustav AHyCo na način da raspolaže popisom svih cjelina te u interakciji s modelom protoka poslova raspoređuje cjeline u glavne i preporučene u ovisnosti o pojedinoj instanci modela protoka poslova.

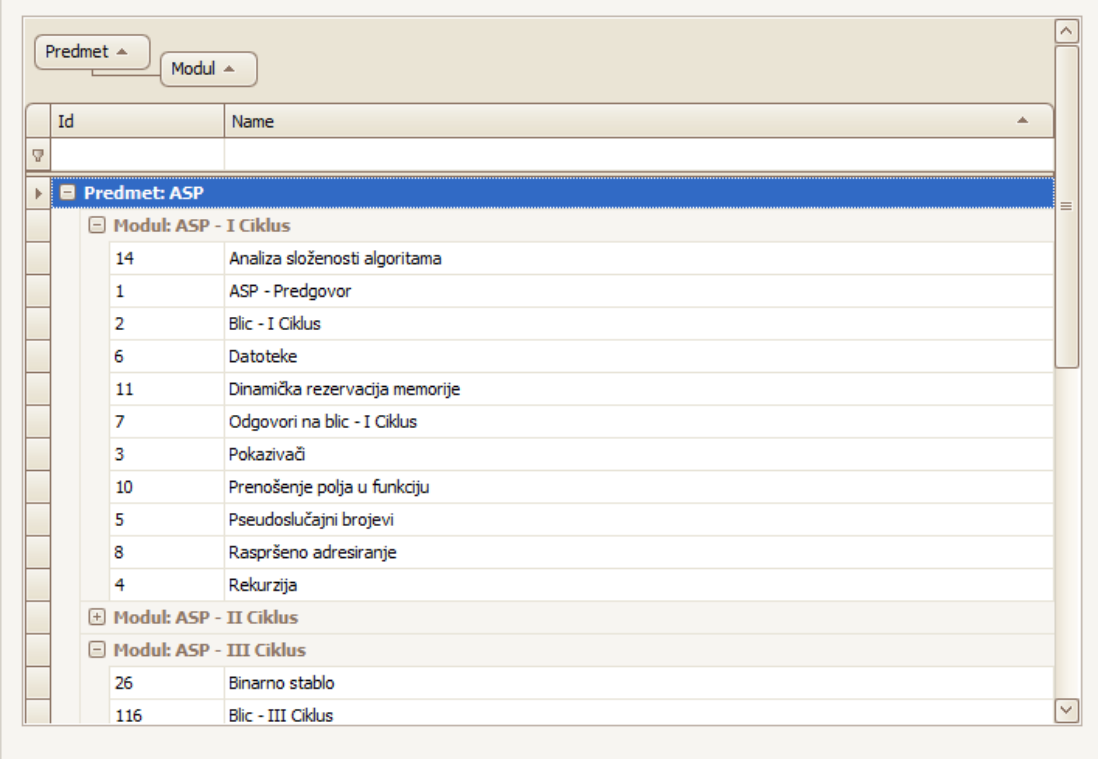
7.1.1 Modeliranje protoka poslova u sustavu AHyCo

Slika 43 prikazuje postojeću aplikaciju za modeliranja preduvjeta u sustavu AHyCo. Navedenim pristupom moguće je definirati relacije među cjelinama unutar pojedinog modula, odnosno relacije među modulima unutar predmeta. Princip definiranja odnosa nalik je definiranju parcijalnih modela. Svaki luk zapisuje se u bazu kao par (p,q), gdje je p identifikator cjeline koja prethodi cjelini čiji je identifikator q ili kao par (v,w) gdje je v identifikator modula koji prethodi modulu čiji je identifikator w. Mane ovog pristupa su nemogućnost definiranja direktnih veza između cjelina i modula, nužnost definiranja veza za sve cjeline u modulu (ne dozvoljavaju se nepovezani vrhovi) te nedostatak odgovarajućih provjera za valjanost relacija, to jest korektnost definiranih relacija je prepuštena autoru relacija. Potencijalni problem predstavlja i nedostatak automatskog razmještaja elemenata grafa nekim od za to predviđenih algoritama, primjerice [Sugiyama1981], što može doći do izražaja prilikom rada s većim brojem elemenata u grafu.

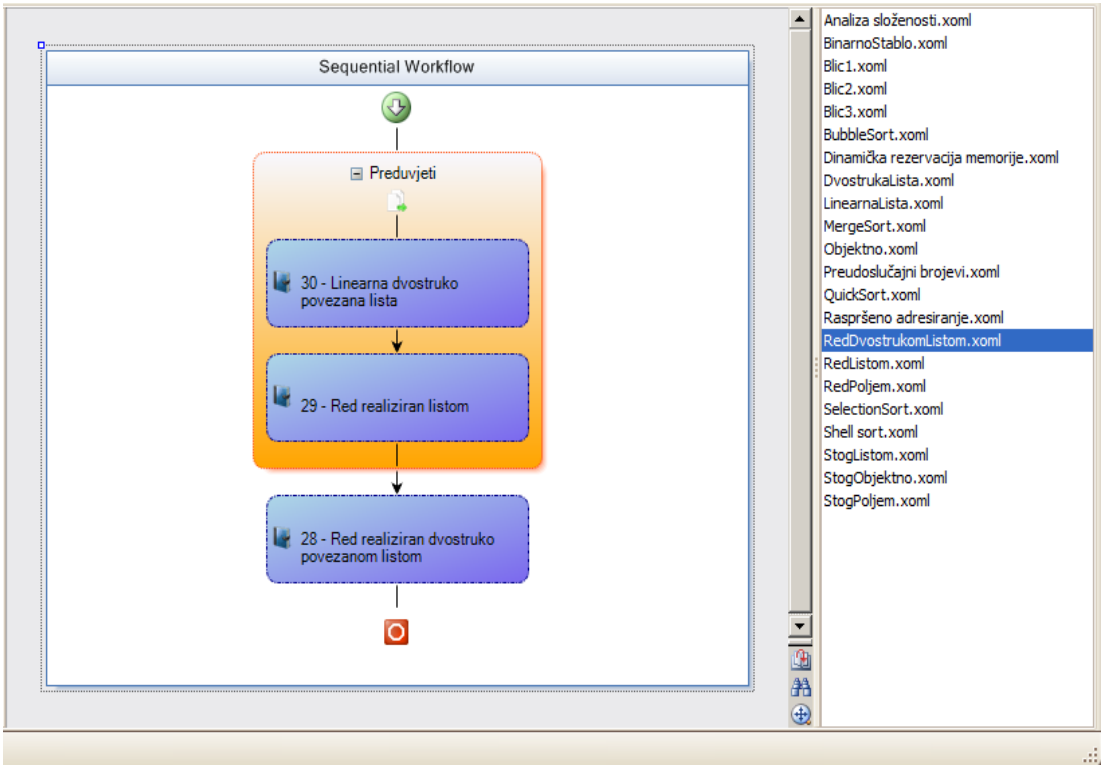


Slika 43. Aplikacija za modeliranje relacije preduvjeta u sustavu AHyCo [Hoić2005]

Za korištenje aplikacije s grafičkim sučeljem za modeliranje parcijalnih modela protoka poslova opisane u poglavlju 6, vanjski sustav treba pružiti podatke o elementima koji se koriste u modeliranju. Tablica 10 u stupcu *Vanjski sustav* sadrži popis postupaka koji Ahyco mora implementirati i izložiti putem web servisa kako bi aplikaciji pružio tražene podatke. Osim identifikatora elementa i naziva podaci koje bi bilo dobro poslati uz pojedini element su značenje elementa (modul/cjelina) i naziv modula kojem cjelina pripada, kako bi se podaci na formi lakše mogli filtrirati i grupirati. Za potrebe rada napravljen je prototip očekivanog web servisa koji vraća podatke o cjelinama za učenje u predmetu *Algoritmi i strukture podataka*. Podaci o cjelinama mogu se vidjeti na formi za odabir elementa (slika 44) koja se pojavljuje iz skočnog izbornika u aplikaciji za izradu parcijalnih modela protoka poslova (slika 45).



Slika 44. Forma za odabir cjelina



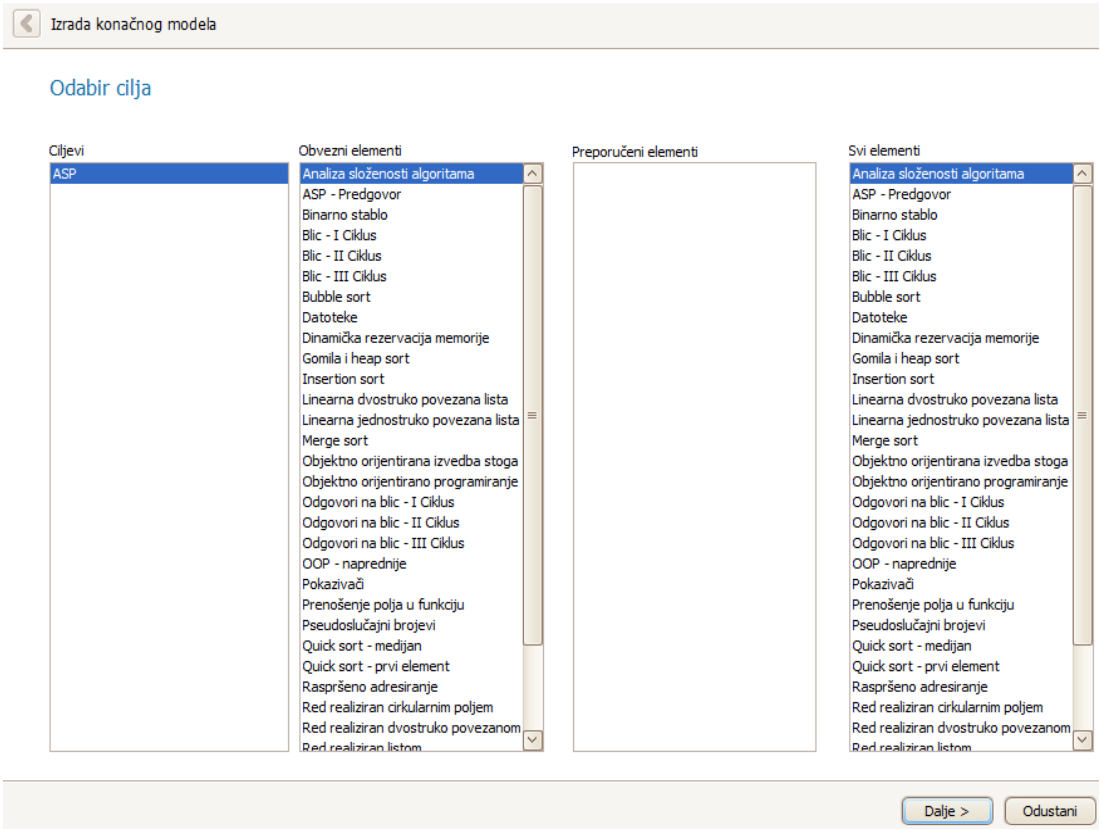
Slika 45. Definiranje parcijalnih modela preduvjeta unutar predmeta Algoritmi i strukture podataka

Napomena: Budući da za odnose među cjelinama koristi samo relacija preduvjeta, umjesto korištenjem aplikacije prikazane na slici 45, izrada parcijalnih modela može se izvesti (uz odgovarajuću pretvorbu prilikom pohrane) direktno pomoću grafa, bilo kroz skup podgrafova, bilo izradom cjelokupnog grafa odjednom.

Svaki parcijalni model sprema se kao WF model koji se pohranjuje u obliku XML datoteke s ekstenzijom xoml. Tablica 11 prikazuje sadržaj xoml datoteke za parcijalni model u kojem se opisuje da je cjelina *Rekurzija* s identifikatorom 4 preduvjet cjeline *Analiza složenosti algoritama* s identifikatorom 14. Navedena činjenica omogućava da se postojeće relacije iz sustava AHyCo pretvore u parcijalne modele bez potrebe za ponovnom ručnom izradom svih modela.

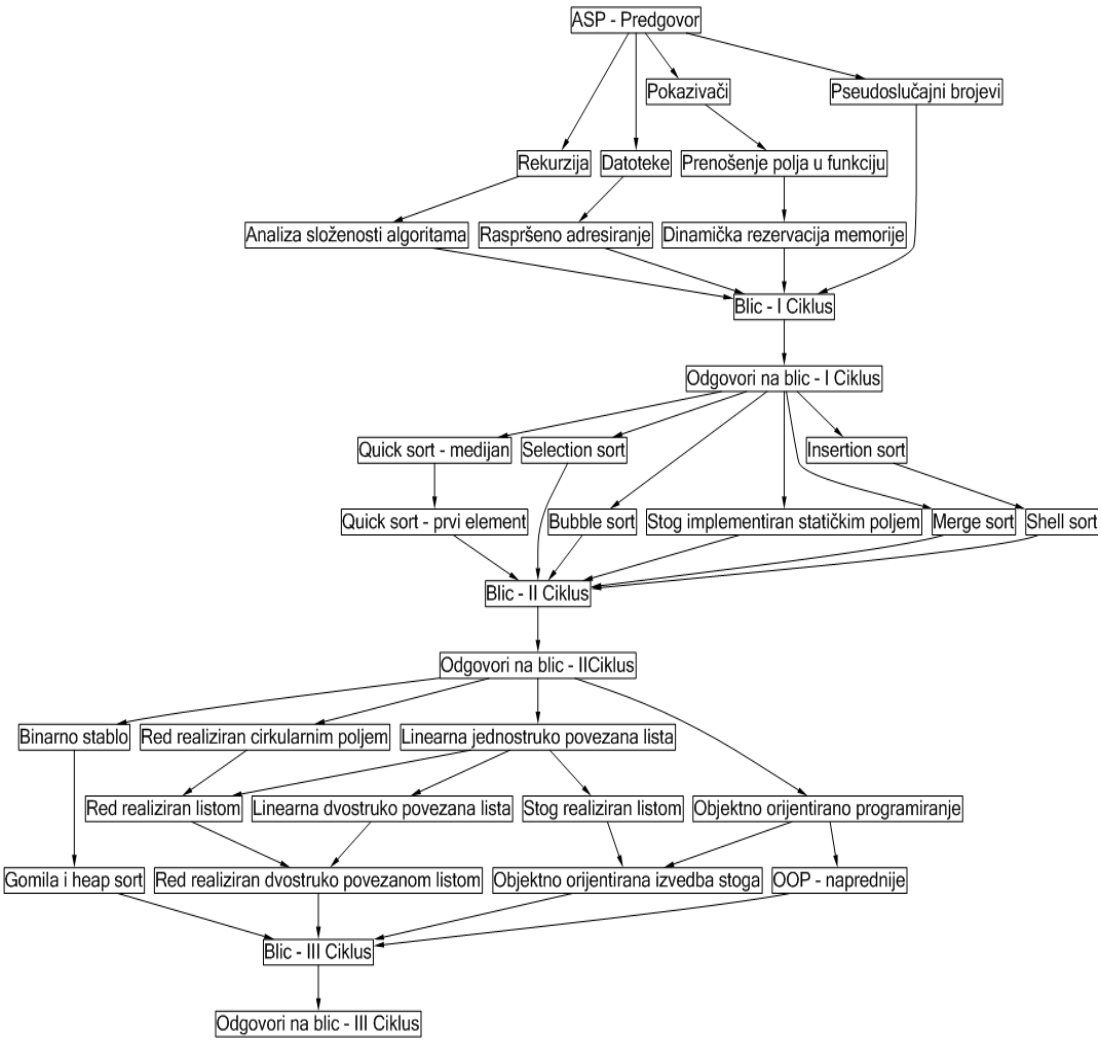
Tablica 11. Primjer sadržaja xoml datoteke za jedan od parcijalno definiranih modela

```
<?xml version="1.0" encoding="utf-8"?>
<SequentialWorkflowActivity x:Name="Analiza složenosti" xmlns:ns0="clr-
namespace:DesignerWFLibrary;Assembly=DesignerWFLibrary, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
<ns0:ElementActivity Sifra="4" x:Name="elementActivity2" Naziv="Rekurzija"
/>
<ns0:ElementActivity Sifra="14" x:Name="elementActivity1" Naziv="Analiza
složenosti algoritama" />
</SequentialWorkflowActivity>
```



Slika 46. Odabir ponuđenih ciljeva i prikaz elemenata u pojedinom cilju

Izrađeni parcijalni modeli pohranjuju se na serveru pozivom postupka *SaveModels*, nakon čega se može pristupiti izradi konačnog modela. Postupak počinje odabirom jednog od ciljeva, a ovom primjeru jedini dostupni cilj je onaj koji u popisu obveznih elemenata sadrži sve cjeline[30], pa se radi se o trivijalnoj situaciji u kojoj nije moguće mijenjati elemente koji će se naći u konačnom modelu. Po odabiru cilja, dolazi do dohvata parcijalno definiranih modela i rada algoritama vezanih uz parcijalnih modela. U skup odabranih elemenata dodaju se preduvjeti odabranih elemenata (ako već nisu bili odabrani) i provjerava se da li odabrani elementi mogu tvoriti konačni model. Za valjani konačni model pokazuje se graf konačnog modela, što u primjeru za *Algoritme i strukture podataka* izgleda kao na slici 47. Po odabiru svih elemenata slijedi izrada WF modela koji je u ovom slučaju isti za sve studente na predmetu *Algoritmi i strukture podataka*. Izrađeni WF model se smješta na server na kojem se nalazi sustav za upravljanje protokom poslova i omotač iz poglavlja 5.4, a podatak o lokaciji evidentira se u sustavu AHyCo za konkretnog korisnika.



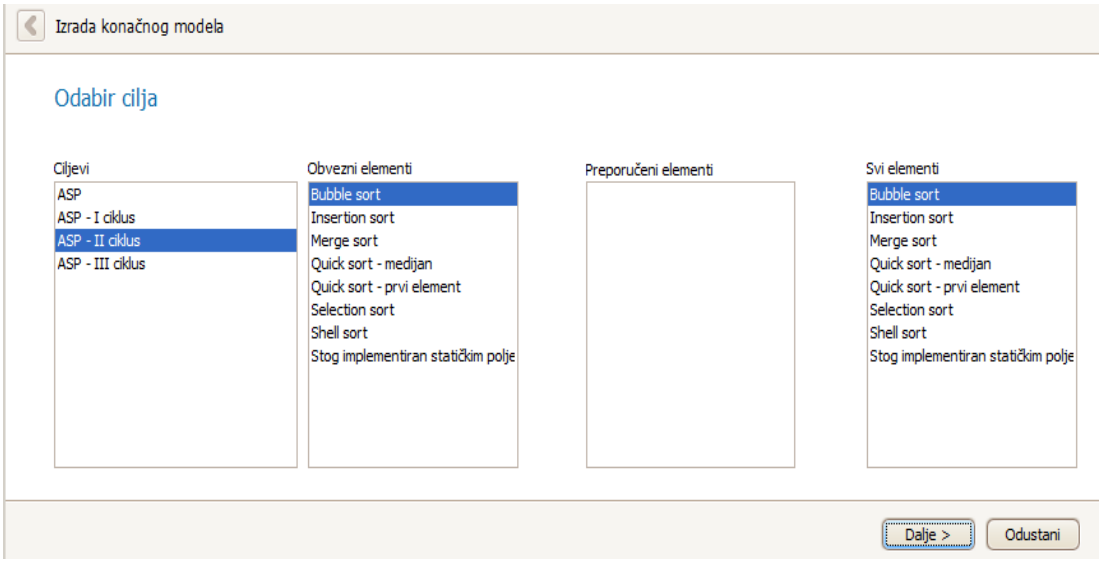
Slika 47. Graf konačnog modela za Algoritme i strukture podataka

[30] U ovom slučaju dovoljno je bilo staviti cjelinu *Odgovori na blic – III ciklus* u obavezne elemente, jer bi svi ostali elementi bili dodani zbog definiranih relacija preduvjeta

Promotri li se graf sa slike 47 može se vidjeti da je graf u gornjem dijelu, za prva dva ciklusa „pravilan“ i bez veza među paralelnim granama te će prema algoritmima za pretvorbu u WF model nastati klonovi tek za cjeline iz trećeg ciklusa i to za *Linearna jednostruka povezana lista* i *Objektno orijentirano programiranje*. Sve ostale će ili inicijalno dobiti samo jednu oznaku ili će uslijed algoritma reduciranja oznaka svesti svoj skup oznaka na samo jednu oznaku.

Činjenica da su cjeline grupirane u module (što se može primijetiti i na grafu sa slike 47) i činjenica da se parcijalni modeli zasnivaju na pretpostavci da je njihova izrada neovisna o značenjima pojedinih elemenata mogu se iskoristiti za pojednostavljenje izrade modela protoka poslova za *Algoritme i strukture podataka*. Pojednostavljenje će se manifestirati u obliku smanjenja potrebnog broja parcijalnih modela u kojima se početni elementi nekog ciklusa vežu uz završne elemente prethodnog ciklusa.

U popisu elemenata koji se mogu koristiti za definiranje parcijalnih modela protoka poslova potrebno je uključiti i module koji predstavljaju tri ciklusa unutar predmeta. Proširenjem broj ciljeva i ograničavanjem popisa elemenata u svakom od ciljeva na samo one koji pripadaju pojedinom ciklusu kao na Slika 48 odvojeno će se izraditi modeli protoka poslova za svaki od ciklusa.



Slika 48. Odabir cilja proširen s ciljevima za pojedini ciklus

Slika 49 prikazuje parcijalni model u kojem se kao elementi pojavljuju tri modula uključena u popis elemenata. Elementi sa slike 49 bit će jedini elementi glavnog cilja s nazivom ASP. Nakon što se izrade WF modeli za svaki od ciljeva, modeli za ciljeve kojima se definira pojedini ciklus bit će podmodeli glavnog modela protoka poslova. U trenutku kada instanca protoka poslova obavijesti AHyCo da je počeo element *Model za 1. Ciklus*. AHyCo će na omotaču pozvati postupak *StartWorkflow* za početak instance protoka poslova s nazivom *Model za 1. Ciklus* što će predstavljati podmodel glavnog modela protoka poslova.

Slika 49. Parcijalni model protoka poslova za Algoritme i strukture podataka

Ukoliko se u nekom trenutku pojavi potreba da se u protoku poslova unesu složeniji uvjeti, u popis elemenata osim cjelina i modula mogu se dodati i gotovi modeli protoka poslova koji mogu, a ne moraju koristiti elemente za koje postoje parcijalno definirani modeli. U slučaju da se u tim modelima koristite elementi za koje već postoje definirani parcijalno definirani modeli protoka poslova primjenjuje se opaska iz poglavlja 3.9.

7.1.2 Komunikacija između sustava AHyCo i modela protoka poslova

Budući da je model protoka poslova zasnovan na parcijalnim modelima, pri pretvorbi grafa konačnog modela u konkretni WF model (što je opisano u poglavlju 5) može se stvoriti više klonova (instanci) istog elementa te je potrebno osigurati da se pojedini elementi ne prikazuju više puta u prikazu unutar aplikacije, što je osigurano izgradnjom omotača opisanog u poglavlju 5.4. Budući da se komunikacija odvija putem WCF servisa, pojedine aplikacije, pa tako i AHyCo, navedenog problema višestrukih instanci neće biti uopće svjesne te neće uočiti razliku radi li se o direktnoj komunikaciji s instancom protoka poslova ili komunikaciji s omotačem kao posrednikom u navedenoj komunikaciji.

Iz tablice 10 vidljivo je koje metode iz ugovora na slici 38 bi AHyCo kao vanjski sustav trebao implementirati odnosno koji postupci su na raspolaganju za poziv. Kada pojedini korisnik prvi put započinje proces učenja iz nekog predmeta, AHyCo mora pozvati postupak *StartWorkflow* na WCF servisu (unutar omotača) navodeći pritom naziv modela protoka poslova pridruženog tom predmetu i tom korisniku, nakon čega se pokreće nova instanca modela protoka poslova. Naziv modela protoka poslova može biti evidentiran u sustavu AHyCo na razini predmeta ili na razini pojedinog korisnika, primjerice ako je konkretni model izrađen od strane samog korisnika. Omotač stvara novu instancu protoka poslova, evidentira originalni identifikator instance protoka poslova, a AHyCo-u prosljeđuje vlastiti identifikator instance. Taj podatak AHyCo mora pridružiti trenutnom korisniku za trenutni predmet, kako bi za sve sljedeće događaje mogao kontaktirati instancu protoka poslova upravo za tog korisnika i za taj predmet.

U međuvremenu, u novo pokrenutoj instanci protoka poslova dolazi do početka jednog ili više elemenata. Kao što je opisano u poglavlju 5.4 prvo pokretanje pojedinog elementa uzrokuje prosljeđivanje poziva postupka *ElementStarted* kojeg AHyCo mora implementirati. Parametri poziva postupka sadrže identifikator instance protoka poslova, identifikator elementa i identifikator konverzacije kako bi se poruka o završetku nekog elementa mogla dostaviti upravo pravoj aktivnosti unutar instance protoka poslova. Navedene podatke AHyCo treba sačuvati, kako bi ih mogao upotrijebiti prilikom završetka elementa. Budući da se radi o elementima koji su se pokrenuli odmah pri pokretanju instance protoka poslova, to znači da elementu nije prethodio niti jedan element s kojim bi u relaciji $\prec$ bio kao drugi element, što znači da svi tako pokrenuti elementi nemaju nikakvu informaciju o prethodnicima te ih AHyCo treba smjestiti u glavne cjeline.

Prema trenutnoj funkcionalnosti [AHyCoRIa], otvaranjem sadržaja pojedine cjeline, u opisu cjelina dostupnih za daljnju navigaciju, u kategoriji glavnih cjelina nalaze se one cjeline koje slijede iza navedene cjeline, uz uvjet da su i ostali preduvjeti tih cjelina zadovoljeni. U konkretnom slučaju uvjet kojim se zadovoljava neki preduvjet je otvaranje te cjeline za učenje. Takva funkcionalnost, u ovom slučaju, znači da prvim otvaranjem pojedine cjeline AHyCo treba pozvati postupak *ElementFinished* na omotaču s parametrima identifikatora instance i identifikatora konverzacije. Omotač će provjeriti postoji li više klonova navedenog elementa i svima će proslijediti poziv za završetak elementa.

Navedeni poziv će uzrokovati završetak jednog ili više elemenata u instanci protoka poslova. Završetkom pojedinog elementa može doći do pokretanja novih elemenata i novih poziva *ElementStarted* o čemu će AHyCo biti obaviješten. Budući da se međusobna komunikacija odvija asinkrono, prilikom prikaza sadržaja pojedine cjeline okvir u kojem se nalazi popis glavnih i preporučenih cjeline mora imati mogućnosti periodičkog osvježavanja, kako bi mogao prikazati promjene koje nastanu od trenutka otvaranja cjeline do trenutka aktivacije cjelina koje slijede. U modelu koji nema vremenskih ovisnosti, vremenski razmak od trenutka dostave poruke o završetku nekog elementa do trenutka učitavanje instance protoka poslova pohranjene korištenjem servisa za postojanost te vremenski razmak od završetka neke aktivnosti i početka novih aktivnosti ukupno je reda veličine nekoliko desetaka milisekundi do nekoliko sekundi. Stoga, potreba za osvježavanjem nakon nekoliko početnih sekundi je minimalna.

Želi li se za neku cjelinu promijeniti ponašanje na način da je za aktiviranje njenih sljedbenika potrebno ne samo otvoriti cjelinu, već i riješiti određeni test, tada model protoka poslova nije potrebno mijenjati, već je samo potrebno na sustavu AHyCo promijeniti trenutak u kojem se poziva postupak *ElementFinished* na način da se postupak poziva tek nakon uspješno riješenog testa.

7.2 Preddiplomski studij po programu FER-2

Kao primjer parcijalnih modela s vremenskom komponentom može se uzeti neki od studijskih programa, primjerice modul Programsko inženjerstvo na preddiplomskom studiju računarstva po studijskom programu FER-2. Na [FER2Rac] prikazana je preporučena shema studija po semestrima, a pokazat će se da je takva shema velikim dijelom određena vremenom predavanja pojedinog predmeta i definiranim relacijama preduvjeta među predmetima. Primjer je pogodan za ilustraciju problema raspoređivanja iz poglavlja 4.3 te problema pretvorbe u WF model iz poglavlja 5.2.

Treba ipak napomenuti da primjer nije u potpunosti reprezentativan, jer su vremenska razdoblja svakog predmeta jednaka nekom od vremenskih okvira, to jest u ovom slučaju semestrima. Međutim, promjenom načina studija uvođenjem slušanja predmeta u ciklusima, moglo bi se postići da se vremenski okviri i vremenska razdoblja predmeta ne podudaraju, čime bi algoritmi iz poglavlja 4 i 5 došli do punog izražaja.

7.2.1 Raspoređivanje predmeta po semestrima

Prema definicijama iz poglavlja 3.4 svaki konačni model ima jedan ili više ciljeva. U ovom slučaju jedan od ciljeva je *Programsko inženjerstvo* sa skupom obveznih predmeta, skupom predloženih predmeta (može biti prazan skup) i skupom svih mogućih elemenata koji se mogu koristiti za navedeni cilj. Funkcija *valjan* za konačni model u ovom slučaju je funkcija koja mora provjeriti da li je ukupna suma ECTS bodova koje je korisnik odabrao veća ili jednaka 180 bodova te manja ili jednaka od sume dozvoljenog broja bodova za svaki od semestara.

Tablica 12. Jedan od mogućih izbora predmeta na modulu Programsko inženjerstvo

Šifra	Naziv	ECTS	Semestar	Mogućnosti
19670	Matematika 1	7	Z	1
19671	Mathematica	1	Z	1,3,5
19672	Osnove elektrotehnike	7	Z	1,3
19674	Digitalna logika	6	Z	1
19676	Programiranje i programsko inženjerstvo	6	Z	1
19678	Vještine komuniciranja	3	Z	1,3
21004	Matematika 2	7	LJ	2
21005	Autocad	2	LJ	2,4,6
21006	Fizika 1	6	LJ	2,4
21008	Algoritmi i strukture podataka	6	LJ	2
21010	Arhitektura računala 1	6	LJ	2
21012	Menadžment u inženjerstvu	3	LJ	2,4
31487	Fizika 2	6	Z	3,5
31488	Elektronika 1	7	Z	3,5
31490	Upravljanje kakvoćom	3	Z	1,3,5
31491	Matlab	2	Z	3,5
31493	Vjerojatnost i statistika	5	LJ	4
31494	Signali i sustavi	6	LJ	4,6
31500	Matematika 3R	5	Z	3,5
31501	Operacijski sustavi	7	Z	3
31502	Uvod u teoriju računarstva	6	LJ	2,4
31503	Baze podataka	6	LJ	4
31505	Seminar-FER2	3	LJ	2,4
34269	Oblikovanje programske potpore	8	Z	5
34272	Komunikacijske mreže	4	Z	5
34275	Okoliš i održivi razvoj	2	Z	1,3,5
34280	Prevođenje programskih jezika	4	Z	5
34282	Programske paradigme i jezici	4	LJ	3,5
34283	Razvoj primijenjene programske potpore	4	LJ	6
34286	Otvoreno računarstvo	4	LJ	2,4,6
34294	Trgovačko pravo	2	LJ	2,4,6
34315	Teorija informacije	4	Z	5
34327	Ergonomija u računarstvu	4	LJ	2,4,6
34335	Mrežno programiranje	4	LJ	2,4,6
37541	Projekt iz programske potpore (Programsko inženjerstvo)	8	Z	5
41251	Inženjerska ekonomika	4	LJ	4,6
41430	Završni rad (Programsko inženjerstvo)	12	LJ	6

Također, potrebno je provjeriti da li je odabrano barem 8 ECTS bodova iz predmetima koji se smatraju izbornim predmetima[31]. Nakon što korisnik (student) odabere željene predmete[32] na način da je funkcija *valjan* istina, može se krenuti u raspoređivanje i kreiranja WF modela. Bez smanjenja općenitosti može se pretpostaviti da je jedan od odabira bio nalik predmetima u tablici 12.

Tablica 12 sadrži popis obveznih predmeta[33] na modulu Programsko inženjerstvo uz dodatna tri kolegija kako bi se zadovoljio uvjet odabira dovoljnog broja izbornih predmeta. Za svaki predmet navedeno je predaje li se u ljetnom ili zimskom semestru. Na taj način svakom je predmetu pridruženo jedno ponavljajuće vremensko razdoblje. Prije same izgradnje modela potrebno je upariti ponavljajuća vremenska razdoblja (ljetni i zimski semestar) s fiksnim vremenskim razdobljima, to jest konkretnim datumima za pojedinu akademsku godinu. Zadnji stupac tablice sadrži brojeve semestara (redne brojeve vremenskih okvira) u kojima se predmet može predavati s obzirom na postavljene uvjete preduvjeta i semestar u kojem se predmet održava. Vrijednosti su dobivene na osnovu algoritama iz poglavlja 4.2, gdje se početni skup mogućnosti sužava na osnovu prethodnika i sljedbenika u integralnom grafu. Parcijalni model koji opisuju preduvjetne relacije među predmetima stvoreni su na osnovu slike 50 na stranici 102.

U poglavlju 4.3.3 funkcija dobrote definirana je kao

$$dobrota(jedinka) = \lambda \cdot K_1 + \mu \cdot K_2 \quad , \quad \lambda + \mu = 1, \lambda > 0, \mu > 0$$

pri čemu je komponenta K_1 dobivena normalizacijom broja lukova u kojima izvorišni vrh nekog luka ima veću vrijednost vremenskog okvira od vrijednosti vremenskog okvira u odredišnom vrhu.

Komponenta K_2 ovisi o konkretnom značenju elemenata, a kako se u ovom primjeru radi o raspoređivanju predmeta po semestrima, računat će se na osnovu broja ECTS bodova u svakom od semestara. Kao što je vidljivo, neki predmeti nemaju mogućnost izbora semestra u kojem se mogu upisati, već su limitirani na točno određeni semestar. Takvi elementi neće ući u problem raspoređivanja, ali će biti uključeni u izračun vrijednosti komponente K_2.

[31] Izborni predmeti su u ovom slučaju predmeti koji ne spadaju u redovne (obvezne) predmete, a ne smatraju se vještinama.

[32] Inicijalno je potrebno odabrati predmete za cijeli studij, ali takav izbor ne mora biti konačan. Model se može dinamički mijenjati (poglavlje 2.3.2). Alternativno mogu se definirati ciljevi za pojedine godine pri čemu se elementi iz takvih ciljeva dodaju na već aktivni konačni model.

[33] U tablici se ne nalaze predmeti koji nose 0 ECTS bodova i koji se ne polažu, primjerice, Tjelesna i zdravstvena kultura.

Komponenta K_2 se tada može računati kao prosječna vrijednost funkcije dobrote za svaki od semestara, a dobrota d_i za svaki od semestara se može računati po sljedećoj formuli

$$d_i = \begin{cases} 0 & r_i \geq 2 \\ 2 - r_i & r_i \in \langle 1,2] \\ r_i & r_i \in [0,1] \end{cases}$$

pri čemu je $r_i = \frac{suma\ \ ECTS\ bodova\ \ u\ i{-}tom\ \ semestru}{30}$, a raspored za neki semestar je ispravan ako je suma ECTS bodova u intervalu [25,35].

Potencijalni nedostatak ovako definirane funkcije dobrote je jednaka vrijednost funkcije dobrote za situaciju u kojoj se u nekom semestru nalazi predmeta u iznosu od 60 ECTS bodova i predmeta u iznosu od 180 ECTS bodova. Alternativni način definicije funkcije dobrote bi uključivao izračun maksimalnog broja ECTS bodova koji se može u nekom trenutku pridijeliti nekom semestru te uspostavom odnosa između trenutno dodijeljenog broja bodova, najvećeg mogućeg bodova te idealnog broja bodova. Potreba za takvih načinom izračuna u ovom slučaju nije bila potrebna, jer nešto manje od polovice predmeta ima unaprijed određen semestar u kojem se može izvoditi i ne sudjeluje u raspoređivanju, pa je i vjerojatnost ekstremnih situacija zanemariva, a bit će pokrivena komponentom K_1.

Za testiranje uspješnosti ovakvog pristupa izvedena su dva testa. U prvom slučaju izvedeno je reduciranje mogućih vremenskih intervala kako bi se dobile vrijednosti u tablici 12. Ponavljanjem genetskog algoritma 500 puta uz 5 000 iteracija dobivena je gotova 100% uspješnost. Trajanje jednog izvođenja algoritma trajalo je oko 25 milisekundi[34] na računalu CoreDuo 1.83GHz. Rješenja u traženom broju iteracija nisu pronađena tek u 3 od 500 slučajeva pri vrijednosti $\lambda = 0{,}2$ i u 14 slučajeva za $\lambda = 0{,}1$ uz napomenu da je za veće vrijednosti μ naspram λ algoritam pronalazio traženo rješenje u većem broju iteracija, ali odnos broja potrebnih iteracija i vrijednosti λ nije linearan. Test u kojem je $\lambda=0$, $\mu=1$ pronašao je rješenje u 86,4% slučajeva, a test sa $\lambda=1$, $\mu=0$ našao je rješenje u tek 61,4% slučajeva. Također, broj potrebnih iteracija za pronalazak rješenja u slučajevima $\lambda=0{,}9$, $\mu=0{,}1$ i $\lambda=0{,}8$, $\mu=0{,}2$ je bio veći nego za $\lambda=0{,}7$, $\mu=0{,}3$. Iz navedenog proizlazi da treba izbjegavati rubne vrijednosti za λ i μ.

Kako bi se detaljnije utvrdio odnos između vrijednosti λ i μ, napravljen je drugi test u kojem su svakom od predmeta dopuštene sve moguće vrijednosti vremenskih okvira (1-6), bez uzimanja u obzir u kojem se semestru predmet predaje. Izvedeno je 5000 izvođenja genetskog algoritma uz 50 000 iteracija. Vrijeme za izvođenje jednog ponavljanja algoritma u ovom slučaju je bilo cca 25 puta duže nego u prvom slučaju[35], a rezultati su pokazali velike razlike za odabrane vrijednosti λ i μ, pri čemu je veća vrijednost λ pridonosila većoj uspješnosti algoritma.

[34] Podatak treba uzeti samo kao mjeru za red veličine brzine problema, zbog nepreciznosti testiranja na računalu na kojem su se odvijali i drugi procesi.

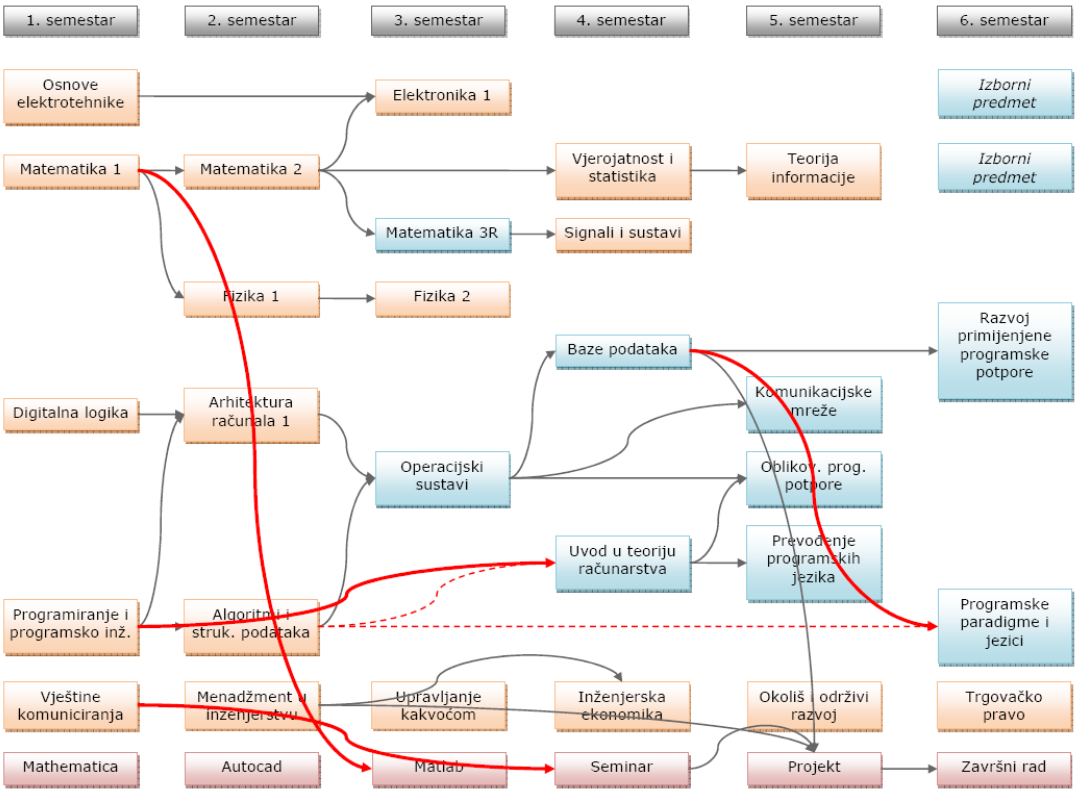
[35] Razlog povećanja je 10 puta veći broj iteracija, manji postotak pronađenih rješenja i veći broj potrebnih iteracija da bi se rješenje pronašlo.

Kao zanimljivi primjeri za vrijednost λ mogu se uzeti vrijednosti 0,1, 0,2, 0,5, 0,8 i 0,9. U prvom slučaju kad je λ=0,1 u 5000 izvođenja genetskog algoritma s 50 000 iteracija nije pronađeno niti jedno rješenje. U slučaju za λ=0,2 broj rezultata je bio 115, što predstavlja uspješnost pronalaska rješenja od tek 2,3%. Za vrijednost λ=0,5 rješenje je za isti broj ponavljanja i isti broj iteracija pronađeno u 64,78% slučajeva. Pri vrijednostima λ=0,8 i λ=0,9 postotak rješenja bio je 92,24%, odnosno 93,78%.

Zaključak koji se može izvesti je da vrijednosti za λ i μ u slučaju većeg problema mogu značajno utjecati na performanse i rezultate genetskog algoritma. U ovom konkretnom slučaju pokazuje se da je poredak elementa dominantni faktor u traženju dobrih jedinki u genetskom algoritmu. Pretpostavka je da bi za probleme u kojima je graf stabla primarno rastao u širinu veći značaj imala vrijednost μ, to jest funkcije dobrote pojedinih vremenskih okvira, dok je za slučajeve poput testiranog, u kojem je graf rastao u dubinu, potrebno povećati vrijednost λ.

7.2.2 Izrada parcijalnih modela protoka poslova i pretvorba u WF model

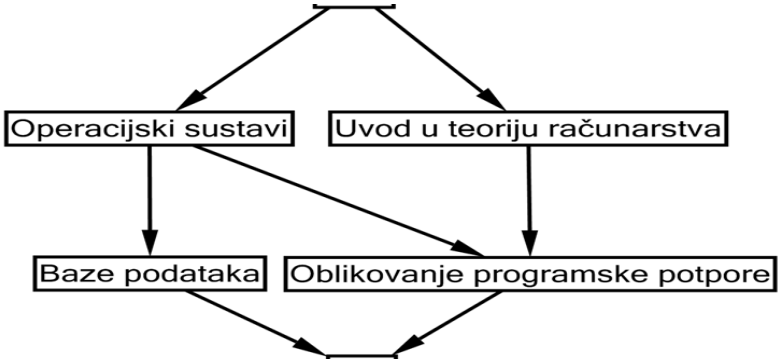
Drugi aspekt pogodnosti ovog primjera je način definiranja preduvjeta i izgradnje WF modela. Slika 50 prikazuje predmete i preduvjete uz promjene od akademske godine 2009/10. Isprekidanom linijom označeni su preduvjeti koji se brišu te zamjenjuju novim preduvjetima označenim crvenom linijom. Princip definiranja preduvjeta korišten na Slika 50 kao i slični tablični prikazi kojim se definiraju preduvjeti među predmetima polaze od pretpostavke da se za svaki predmet vodi evidencija samo o njegovim neposrednim preduvjetima. Posljedice tog principa su potreba za dodatnom pažnjom prilikom promjene neke od preduvjetnih relacija. Brisanje veze između *Algoritama i struktura podataka* i *Uvoda u teoriju računarstva* povlači da se mora dodati veza između *Programiranja i programskog inženjerstva* i *Uvoda u teoriju računarstva*, jer je *Programiranje i programsko inženjerstvo* preduvjet *Algoritama i struktura podataka*. Na sličan način briše se veza između *Algoritama i struktura podataka* i *Programskih paradigmi i jezika*, jer se relacija preduvjeta sada ostvaruje preko niza drugih preduvjeta. Pored potrebe za dodatnim brisanjima ili dodavanjima novih veza, ovakav način vođenje evidencije je nepraktičan, jer podrazumijeva da nositelj kolegija mora poznavati čitav niz preduvjeta prilikom definicije preduvjeta za svoj kolegij. Kao ekstremniji primjer može poslužiti hipotetski primjer u kojem bi se uklonila veza između *Algoritama i struktura podataka* i *Operacijskih sustava*. Posljedice te odluke odnose se i na *Baze podataka* i na *Razvoj primijenjene programske potpore*. Pretpostavi li se da su *Algoritmi i struktura podataka* uvjet za *Baze podataka*, tada je potrebno promijeniti i popis preduvjeta za *Baze podataka*. Postavlja se pitanje tko će napraviti te preinake, odnosno dolazi se do situacije da se vijest o brisanju preduvjeta mora proslijediti svim nositeljima predmeta koji slijede iza predmeta kojem se briše neki od preduvjeta.



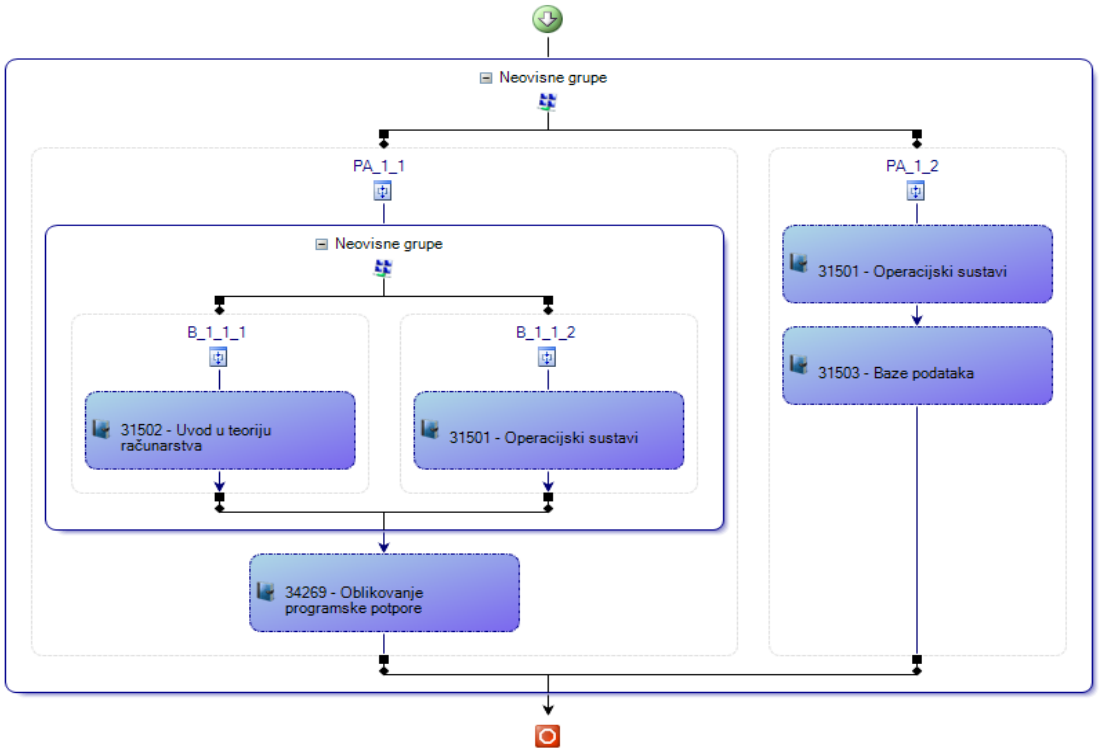
Slika 50. Grafički prikaz promjena predmeta i preduvjeta po FER-2 programu za ak.god. 2009/10 [FER2]

Za razliku od trenutnog korištenog principa za definiranje preduvjeta kojim se definiraju samo neposredni preduvjeti, poželjno je preduvjete definirati parcijalnim modelima prikazanim u poglavlju 3.2. Navođenjem svim relevantnih preduvjeta nekog predmeta, neovisno da li se radi o neposrednim preduvjetima ili preduvjetima koji će biti ostvareni putem tranzitivnosti relacije preduvjeta, vodit će prema situaciji u kojoj brisanje neke ranije veze preduvjeta u stablu ne zahtjeva promjene parcijalnih modela za predmete sljedbenike. Integralni model će se ponovo izgraditi na osnovu parcijalnih modela preduvjeta te će reduciranjem lukova opet nastati model koji sadrži samo one lukove koji predstavljaju neposredne preduvjete.

Međusobni odnos među predmetima u ovom studijskom programu je takav da bi se prilikom izrade modela protoka poslova u kojem su elementi vezani uz pojedine predmete pojavio problem opisan u poglavlju 5.2. Slika 51 prikazuje dio integralnog grafa u kojem se navedeni problem pojavljuje. Zanemari li se pretpostavka da je svaki predmet u svom semestru ili pretpostavi li se da se nastava odvija u ciklusima, ovakva situacija vodi ka problemu postojanja veza među paralelnim granama u modelu protoka poslova te se kao rješenje nameće rješenje predloženo u poglavlju 5. U tom slučaju odsječak WF modela za problem sa slike 51 izgledao bi kao na slici 52.



Slika 51. Isječak iz integralnog grafa studijskog programa koji sadrži problem veza među paralelnim granama



Slika 52. Skica odsječka WF modela za problem sa slike 51

Postupak odabira cilja i dodavanje elemenata u konačni model nalikuje onima opisanim u poglavljima 6.2 i 7.1. Izrada takvog modela protoka poslova općenita je do razine konkretne implementacije postupaka u ugovoru sa slike 38. Praktična posljedica ovog pristupa je da se za pojedinog studenta može koristiti jedan centraliziran WF model na osnovu kojeg se može pratiti njegov napredak, a da se s također centraliziranom implementacijom ugovora mogu postići razne primjene, primjerice otvaranje/zatvaranje korisničkih računa vezanih uz pojedini predmeta, slanja obavijesti i slično. Kao nedostatak može se pojaviti situacija u kojoj se konačni model mora ponovno izgraditi uslijed neispunjenja nekih od uvjeta čime se instanca protoka poslova neće moći jednostavno dinamički promijeniti već će morati biti zamijenjena novom instancom, pa bi povijest statusa napretka pojedinog studenta mogla biti razdvojena u više instanci protoka poslova.

8. Zaključak

Odvajanjem prezentacijske logike, poslovne logike i upravljanja protokom poslova u zasebne slojeve postiže se prilagodljivost sustava, olakšava se održavanje sustava i omogućava ponovna iskoristivost pojedinog sloja. Izlaganjem sloja za upravljanje protokom poslova kroz web servise dobivena je mogućnost upravljanja protokom poslova u različitim vrstama aplikacija na različitim platformama.

Modeliranje protoka poslova kod složenijih sustava često nadilazi mogućnosti jedne osobe ili jednog tima, zbog veličine i složenosti modela te čestih izmjena modela tijekom modeliranja. Cilj ovog rada bio je identificirati općenita svojstva modela protoka poslova, neovisno o vrsti postupka koji se modelira, svodeći modele na skup međusobno povezanih osnovnih gradivnih elemenata, gdje pojedini element interno zadržava svoju specifičnost, a zajednička svojstva izlaže kroz za to predviđeni skup sučelja. Na taj način omogućeno je stvaranje modela protoka poslova na osnovu parcijalno definiranih modela protoka poslova. Konačni model dobiven integracijom parcijalnih modela jednostavniji je za održavanje i prilagodbu te otporniji na pogreške u odnosu na ručno integrirani model.

U ovoj disertaciji osmišljeni su mehanizmi integracije parcijalnih modela, izneseni su problemi i predložena rješenja u obliku algoritama i programskih sučelja za rad s parcijalnim modelima protoka poslova. Parcijalni modeli protoka poslova uz opisana programska sučelja i algoritme time predstavljaju generički okvir za modeliranje parcijalnih modela protoka poslova te apstrahiraju konkretne implementacije parcijalnih modela.

Za potrebe demonstracije praktične upotrebe izrađen je prototip aplikacije za izgradnju parcijalnih modela protoka poslova. Prototip je demonstriran na primjeru izrade studijskog programa po programu FER-2 Fakulteta elektrotehnike i računarstva. Analiziran je sustav za udaljeno učenje Adaptive Hypermedia Courseware (AHyCo), koji se koristi za provjere znanja i za računalom podržano učenje na Fakultetu elektrotehnike i računarstva u Zagrebu i Odsjeku za informatiku Filozofskog fakulteta u Rijeci. Korištenjem izrađenog prototipa opisan je mogući način ugradnje parcijalnih modela u sustav AHyCo .

Modeliranje protoka poslova temeljem parcijalnih modela predstavlja kompromis između veličine skupa osnovnih gradivnih elemenata i jednostavnosti upotrebe i izrade modela protoka poslova. Disertacija otvara nekoliko problema koji mogu poslužiti kao smjernice za daljnja istraživanja na području parcijalnih modela protoka poslova. Neki od tih problema uključuju dinamičke promjene aktivnih modela bez potrebe za zamjenom aktivnog modela novim modelom, optimizaciju rada s parcijalnim modelima koji u sebi uključuju podmodele protoka poslova kao i mogućnost uvođenja nedeterminističkih uvjeta među parcijalnim modelima protoka poslova.

Literatura

[Aalst2002] van der Aalst, W.; Ter Hofstede, A. *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages*. Proceedings of the Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools; Aarhus, Danska, 2002. Str. 1-20

[Aalst2004] van der Aalst, W.; van Hee, K. *Workflow management. Models, Methods, and Systems*. Cambridge, Massachusetts, SAD: MIT Press, 2004.

[Aho1972] Aho, A.V.; Garey, M.R.; Ullman J.D. *The Transitive Reduction of a Directed Graph*. SIAM Journal on Computing, 1972.; 1(2): str. 131-137.

[AHyCoRI] Što *je AHyCo?* Portal AHyCo Informatičkog odjela Filozofskog fakulteta, Sveučilište u Rijeci. http://AHyCo.ffri.hr/portal/Glavna.aspx?IDClanka=21&IDKategorije=4 (30.3.2010.)

[AHyCoRIa] *Učenje i testiranje pomoću AHyCo sustava*. Portal AHyCo Informatičkog odjela Filozofskog fakulteta, Sveučilište u Rijeci. http://AHyCo.ffri.hr/portal/Glavna.aspx?IDClanka=23&IDKategorije=4 (30.3.2010.)

[AlZabir2007] Al Zabir, O. *Building a Web 2.0 Portal with ASP.NET 3.5*. Sebastopol, California, SAD: O'Reilly Media, 2007.

[Avgeriou2003] Avgeriou, P; Retalis, S; Papaspyrou, N. *Modelling learning technology systems as business systems*. Software and Systems Modeling, 2003.; 2: str. 120-133.

[BizTalk] *Microsoft BizTalk Server*. http://www.microsoft.com/biztalk/en/us/default.aspx (30.3.2010)

[Botički2008] Botički, I; Milašinović, B. *Knowledge Assessment at the Faculty of Electrical Engineering and Computing*. Proceedings of the 30th International Conference on Information Technology Interfaces, Cavtat, 2008. Str. 111-116.

[BPMResearch] *From Office Automation to Workflow Management*. BPM Research History: http://www.bpm-research.com/research/history-of-bpm-and-workflow-research/ (30.3.2010.)

[Bukovics2007] Bukovics, B: *Pro WF: Windows Workflow in .NET 3.0*, Newyork, SAD: Apress, 2007.

[Carbajal2006] Carbajal, D. *BizTalk Server 2006 vs Windows Workflow Foundation*. Daniel Carbajal's Weblog http://geekswithblogs.net/DanielCarbajal/archive/2006/09/10/90825.aspx (30.3.2010.)

[Carnet] *Standardi za izradu nastavnih materijala*. Carnet - Referalni centar za odabir alata za e-obrazovanje. http://wwww.carnet.hr/referalni/obrazovni/oca/formati (30.3.2010.)

[Cesarini2004] Cesarini, M.; Monga, M.; Tedesco, R.: *Carrying on the e-Learning process with a Workflow Management Engine*. Proceedings of the 2004 ACM symposium on Applied computing, Nicosia, Cipar, 2004. Str. 940-945.

[Elezović2010] Elezović, N. *Vjerojatnost i statistika: Diskretna vjerojatnost.* Zagreb: Element, 2010.

[Evdemon2006] Evdemon, J. *BizTalk and WF - Which To Use When?* http://blogs.msdn.com/jevdemon/archive/2006/06/19/637462.aspx (30.3.2010.)

[FER2] *Grafički prikaz predmeta i preduvjeta Preddiplomskog studija po FER-2 programu*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu. http://www.fer.hr/_download/repository/FER2-Preduvjeti-090630-Razlike.pdf (30.3.2010.)

[FER2Rac] *Studijski program Računarstvo*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu http://www.fer.hr/fer2/racunarstvo (30.3.2010.)

[Fertalj2009] Fertalj, K. *Razvoj informacijskih sustava – Predavanja*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2009.

[Fertalj2010]: Fertalj, K.; Hoić-Božić, N.; Jerković, H. *The Integration of Learning Object Repositories and Learning Management Systems*, Computer Science and Information Systems, 2010.; 7(2) (prihvaćen za objavljivanje)

[Fong2003] Fong, J.; Ng, M.; Kwan, I.; Tam, M. *Effective E-learning by Use of HCI and Web-Based Workflow Approach*. Springer, Lecture Notes in Computer Science, 2003.; 2783: str. 271-286.

[Goldberg1991] Goldberg, D.E.; Deb, K. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. Morgan Kaufmann Publishers, Foundations of Genetic Algorithms, San Mateo, SAD, 1991. Str. 69-93.

[Golub2004] Golub, M. *Genetski algoritam, Prvi dio*, skripta. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2004.

[Haeupler2008] Haeupler, B.; Telikepalli, K.; Mathew, R.; Siddhartha S.; Tarjan, R.E. *Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance*.[36] Proceedings 2008 of the 35th International Colloquium on Automata, Languages, and Programming (ICALP), Reykjavik, Island, 2008.

[Hammer1990] Hammer, M. *Re-engineering Work: Don't automate, Obliterate*. Harvard Business Review, 1990. Str. 104-112.

[36] Referenciran je rad u postupku objavljivanja (http://www.cs.princeton.edu/~sssix/papers/dto-journal.pdf) . Originalni naslov s konferenciji je *Faster Algorithms for Incremental Topological* (http://www.cs.princeton.edu/~sssix/papers/dto-conf.pdf). Zadnji pristup navedenim adresama: 30.3.2010.

[Hlaoui2009] Hlaoui, Y.B.; Ayed, L.J.B. *An Interactive Composition of UML-AD for the Modelling of Workflow Applications*. Ubiquitous Computing and Communication Journal Journal, 2009.; 4(3): str.599-608.

[Hoić2005] Hoić-Božić, N. Mornar, V. *AHyCo: A Web-Based Adaptive Hypermedia Courseware System*. Journal of Computing and Information Technology, 2005.; 13(3): str. 165-176.

[Hyatt] Hyatt, K. *N-Tier Application Development with Microsoft.NET*, http://www.microsoft.com/belux/msdn/nl/community/columns/hyatt/ntier1.mspx (30.3.2010.)

[JavaWf] *Open Source Workflow Engines in Java*. Open Source Software in Java. http://java-source.net/open-source/workflow-engines (30.3.2010.)

[Johnson1975] Johnson, D.B. *Finding all the elementary circuits of a directed graph*. SIAM Journal on Computing, 1975.; 4(1): str. 77–84.

[Kalpić2006] Kalpić, D., Rajnović, T., Mornar, V. *Minimisation of Collisions in Scheduling of Lectures*. WSEAS Transactions on Information Science and Applications, 2006.; 1790-0832(3): str. 365-371

[Kiepuszewski2003] Kiepuszewski, B. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. Doktorska disertacija, Faculty of Information Technology, Queensland University of Technology, Australija, 2003.

[Knuth1973] Knuth, D.E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 1973.

[Knuth1974] Knuth, D.E.; Szwarcfiter, J.L. *A structured program to generate all topological sorting arrangements.* Information Processing Letters, 1974.; 2(6): str. 153-157.

[Kreher1998] Kreher, D.L.; Stinson, D.R. *Combinatorial Algorithms: Generation, Enumeration, and Search*, Boca Raton, Florida, SAD: CRC Press, 1998.

[Lhotka2008] Lhotka, R. *Expert C# 2008 Business Objects*, Newyork, SAD: Apress, 2008.

[Lin2001] Lin, J, Ho, C., Sadiq, W., Orlowska, M.E. *On workflow enabled e-learning services*. Proceedings of the IEEE International Conference on Advanced Learning Technologies,Madisaon, SAD, 2001. Str. 349-352

[Lin2002] Lin, J, Ho, C., Sadiq, W., Orlowska, M.E. *Using Workflow Technology to Manage Flexible e-Learning Services*. Educational Technology & Society, 2002.; 5(4): str. 116-123.

[Mendling2006] Mendling, J.; Simon, C. *Business Process Design by View Integration.* Springer Lecture Notes in Computer Science, 2006.; 4103: str. 55-64.

[Milašinović2006] Milašinović, B. *Prilagodljivi sustav za upravljanje protokom poslova*. magistarski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2006.

[Milašinović2007] Milašinović, B.; Fertalj, K.; Nižetić, I. *On some Problems while Writing an Engine for Flow Control in Workflow Management Software*. Proceedings of the 29th International Conference on Information Technology Interfaces, Cavtat, 2007. Str. 489-494.

[Milašinović2009] Milašinović, B.; Fertalj, K. *Using partially defined workflows for course modelling in a learning management system*. ICIT-2009, The 4th International Conference on Information Technology, Amman, Jordan.

[Moodle] Moodle: open-source community-based tools for learning. http://moodle.org/ (30.3.2010.)

[Muehlen2004] zur Muehlen, M. *Workflow-based Process Controlling. Foundation, Design and Application of workflow-driven Process Information Systems*, Logos, Berlin, 2004.

[Msagl] Microsoft Research: *Automatic Graph Layout*: http://research.microsoft.com/en-us/projects/msagl/default.aspx (30.3.2010.)

[MsdnWFAct] *Windows Worflow Foundation Activities*. MSDN Library. http://msdn.microsoft.com/en-us/library/ms733615.aspx (30.3.2010.)

[MsAppArch] *Microsoft Patterns and Practices: Microsoft Application Architecture Guide*, 2nd Edition, Redmond, SAD: Microsoft Press, 2009.[37]

[MsAppArch7] *Business Layer Guidelines*. Microsoft Patterns and Practices: Microsoft Application Architecture Guide, Design Fundamentals, Chapter 7 http://msdn.microsoft.com/en-us/library/ee658103.aspx (30.3.2010.)

[Nishizawa1996] Nishizawa, K. *A method to Find Elements of Cycles in an Incomplete Directed Graph and Its Applications – Binary AHP and Petri Nets*, Computers & Mathematics with Applications, 1997.; 33(9): str. 33-46.

[Raposo2004] Raposo, A.B.; Pimentel, M.G.; Gerosa, M.A.; Fuks, H.; Lucena, C.J.P.D.: *Prescribing e-Learning Activities Using Workflow Technologies*. Proceedings of the 1st International Workshop on Computer Supported Acitivity Coordination, Porto, Portugal, 2004. Str. 71-80.

[RegExLib] Regular Expression Library, Dates and Times. http://regexlib.com/DisplayPatterns.aspx?cattabindex=4&categoryId=5 (30.3.2010.)

[Rodriguez2009] Rodriguez, R.P., Rodriguez, M.C., Rifon, L.A.: *Enabling Process-Based Collaboration in Moodle by Using Aspectual Services*, The 9th IEEE International Conference on Advanced Learning Technologies, ICALT 2009, Riga, Latvija, 2009. Str. 301-302.

[Sadiq2002] Sadiq, S., Sadiq, W., Orlowska, M.: *Workflow Driven e-Learning – Beyond Collaborative Environments*. International NAISO Congress on Networked Learning in a Global Environment, Challenges and Solutions for Virtual Education. Berlin, Njemačka, 2002.

[37] Dostupno na http://msdn.microsoft.com/en-us/library/dd673617.aspx .Zadnji pristup navedenoj adresi: 30.3.2010.

[SATWiki] *Boolean satisfiability problem*. Wikipedia.
http://en.wikipedia.org/wiki/Boolean_satisfiability_problem (30.3.2010.)

[Scorm] *SCORM*. Advanced Distributed Learning.
http://www.adlnet.gov/Technologies/scorm/default.aspx (30.3.2010.)

[ScormSeqNav] *Scorm Sequencing and Navigation*. JCA Solutions - SCORM Development Tools. http://www.scormsoft.com/scorm/sn/overview (30.3.2010.)

[Scribner2007] Scribner, K. *Microsoft Windows Workflow Foundation Step by Step*. Redmond, SAD: Microsoft Press, 2007.

[Sharp2008] Sharp, A.; McDermott, P. *Tools for Process Improvement and Application Development, Second Edition*. Boston, SAD: Artech House, 2008.

[Sugiyama1981] Sugiyama, K; Tagawa, S. Toda, M. *Methods for Visual Understandings of Hierarchical System Structures*. IEEE Transactions in Systems, Man, and Cybernetics, 1981.; 11(2): str. 109-125.

[Sun2006] Sun, S.; Kumar, A.; Yen, J. *Merging workflows: A new perspective on connecting business processes*, Decision Support Systems, 2006.; 42: str. 844-858.

[Tarjan1972] Tarjan. R.E. *Depth-first search and linear graph algorithms.* SIAM Journal on Computing, 1972.; 1(2): str. 146–160.

[Vanjak2006] Vanjak, Z. *Okruženje za rješavanje optimizacijskih problema*, doktorska disertacija, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2006.

[Vantroys2001] Vantroys, T.; Peter, Y. *A WMF-based workflow for e-Learning.* Proceedings of the European Research Seminar on Advances in Distributed Systems, Bertinoro, Italija, 2001.

[Vantroys2002] Vantroys, T.; Rouillard, J. *Workflow and Mobile Devices in Open Distance Learning*. Proceedings of the IEEE International Conference on Advanced Learning Technologies, Kazan, Rusija, 2002. Str. 123-127.

[Vantroys2003] Vantroys, T.; Peter, Y. *COW, a Flexible Platform for the Enactment of Learning Scenarios*. Springer Lecture Notes in Computer Science, 2003.; 2806: str. 168-182.

[Vossen2003] Vossen, G.; Westerkamp, P. *E-learning as aWeb service (extended abstract)*. Proceedings oft he 7th International Conference on Database Engineering and Applications, Hong Kong, Kina, 2003.; Str. 242-249.

[WCF] Windows Communication Foundation. .NET Framework Developer Center http://msdn.microsoft.com/en-us/netframework/aa663324.aspx (30.3.2010.)

[Weisstein] Weisstein, E.W. *Topological Sort.* MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/TopologicalSort.html (30.3.2010.)

[WfMC] Workflow Management Coalition, http://www.wfmc.org/ (30.3.2010.)

[WfMC-TC-1011] Workflow Management Coalition Terminology & Glossary (Document No. WFMC-TC-1011), http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html (30.3.2010.)

[WfPat-Eval] Workflow Patterns: Evaluations http://www.workflowpatterns.com/evaluations/index.php (30.3.2010.)

[WFScenario] *WF Scenarios Guidance: Human Workflow*. MSDN Library http://msdn.microsoft.com/en-us/library/cc709416.aspx (30.3.2010.)

[WikiFramework].*NET Framemork Version History*. Wikipedia. http://en.wikipedia.org/wiki/.NET_Framework#.NET_Framework_3.0, (30.3.2010.)

[Xi2007] Xi, S.; Yong, J.; *New Data Integration Workflow Design for e-Learning*. Springer Lecture Notes in Computer Science, 2007.; 4402: str. 699-707.

[Ye1997] Ye, Y.; Roy, K. *A graph-based synthesis algorithm for AND/XOR networks.* Proceedings of the 34th Annual Design Automation Conference, Anaheim, California, SAD, 1997.; Str. 107-112.

[Yong2005] Yong, J. *Workflow-based e-learning platform*. Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, Coventry, UK, 2005.; 2: Str. 1002-1007.

Popis kratica

AHyCo	*Adaptive Hypermedia Courseware*
BPM	*Business process management*
BPR	*Business process reengineering*
BPx	*Zajednička kratica za BPM i BPR*
ECTS	*European Credit Transfer System*
FER	*Fakultet elektrotehnike i računarstva*
FER-2	*Studijski program na Fakultetu elektrotehnike i računarstva*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
SCORM	*Sharable Content Object Reference Model*
WF	*Windows Workflow Foundation*
WfMC	*Workflow Management Coallition*
WCF	*Windows Communication Foundation*
XML	*Extensible Markup Language*
XOML	*Extensive Object Markup Language*

Sažetak

Uslojavanje prezentacijske logike, poslovne logike te upravljanja protokom poslova u složenim sustavima preduvjet je za postizanje prilagodljivosti sustava. Ustanovljavanjem općih svojstava modeliranog postupka omogućuje se stvaranje modela protoka poslova na osnovu manjih parcijalno definiranih modela protoka poslova čime integralni model postaje elastičniji i otporniji na pogreške, a postupak izrade istovremeno postaje učinkovitiji u odnosu na ručnu izgradnju integralnog modela.

U radu su opisani načini definiranja parcijalnih modela protoka poslova te je omogućena njihova verifikacija i integracija, a zatim i pretvorba u operativni model protoka poslova nastao integracijom. Uočeni su potencijalni problemi te su osmišljena odgovarajuća programska sučelja i algoritmi za rad s parcijalnim modelima protoka poslova. Predloženi parcijalni modeli, programska sučelja i algoritmi apstrahiraju konkretne implementacije parcijalnih modela te tako tvore generički okvir za modeliranje parcijalno definiranih modela protoka poslova.

U sklopu rada razvijen je prototip aplikacije za izgradnju parcijalnih modela protoka poslova demonstriran na primjeru izrade rasporeda cjelina u sustavu za učenje Adaptive Hypermedia Courseware (AHyCo) i na izradi studijskog programa po programu FER-2 Fakulteta elektrotehnike i računarstva.

Disertacija završava zaključkom u kojem se iznosi kritički osvrt na prikazana rješenja te se iznose potencijalne smjernice budućeg istraživanja u području rada s parcijalnim modelima protoka poslova.

Ključne riječi

- modeliranje protoka poslova
- parcijalni definirani model
- preduvjet
- protok poslova
- studijski program
- računalom podržano učenje
- udruživanje protoka poslova
- upravljanje protokom poslova
- Windows Workflow Foundation

Generic framework for workflow management based on partially defined models

Abstract

Decoupling the presentation logic, business logic and workflow in a complex system is essential to keep the system flexible. By identifying common properties of a workflow model, the integral workflow model could be built using partially defined workflow models that would make integral workflow model more flexible, less prone to errors, and more time efficient than building an integrated workflow manually.

In the thesis, partially defined workflow models were introduced and the potential problems during the process of integration, verification and transformation to operative model were noted down. Novel algorithms with appropriate programming interfaces for those problems have been presented. Proposed partially defined workflow models, algorithms and interfaces form a generic framework and represent an abstraction over the models concrete meanings.

An application prototype has been developed, demonstrated in two case studies: course prerequisites modelling in a study curriculum and maintaining lessons order in AHyCO, a learning management system used in current work environment.

Thesis ends with critical evaluation and guidelines for the future work related to partially defined workflow models.

Keywords

- e-learning
- partially defined model
- prerequisite
- study curriculum
- Windows Workflow Foundation
- workflow
- workflow management
- workflow merge
- workflow modelling

Životopis

Boris Milašinović rođen je 6. prosinca 1976. godine u Metkoviću. Nakon završene 3. gimnazije u Splitu 1995. godine upisuje se na Matematički odjel Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu te u srpnju 2001. godine diplomira na profilu diplomirani inženjer matematike, smjer računarstvo s izvrsnim uspjehom.

Od lipnja 2000. do rujna 2001. zaposlen je u tvrtki Multimedia Lab kao programer na poslovima elektroničke distribucije sadržaja, a u rujnu 2001. godine postaje znanstveni novak na Matematičkom odjelu Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu.

U listopadu 2002. kao znanstveni novak zapošljava se na Fakultetu elektrotehnike i računarstva na Zavodu za primijenjeno računarstvo te upisuje poslijediplomski studij računarstva, smjer Primijenjeno računarstvo. Magistarski rad pod nazivom „Prilagodljivi sustav za upravljanje protokom poslova“ pod vodstvom mentora prof.dr.sc. Krešimira Fertalja obranio je 4. svibnja 2006. godine. U listopadu 2006. godine upisuje poslijediplomski doktorski studij računarstva.

U okviru nastavnih djelatnosti na FER-u sudjelovao je u nastavi iz predmeta *Algoritmi i strukture podataka*, *Primjena računala*, *Programiranje i programsko inženjerstvo*, *Programske paradigme i jezici*, *Razvoj informacijskih sustava* i *Razvoj primijenjene programske potpore*. Sudjelovao je u izvođenju nastave na drugim visokoškolskim ustanovama: Matematičkom odjelu Prirodoslovno-matematičkog fakulteta u Zagrebu na predmetima *Operacijski sustavi*, *Programiranje* i *Uvod u računarstvo* te na Tehničkom veleučilištu u Zagrebu na predmetu *Osnove programiranja*.

Autor je ili koautor više znanstvenih i stručnih radova te nastavnih materijala, a znanstveni interes uključuje upravljanje protokom poslova i softversko inženjerstvo. Od stranih jezika govori i piše engleski, a pasivno se služi njemačkim jezikom.

Biography

Boris Milašinović was born December 6, 1976, in Metković. After graduating from secondary school (the 3rd General Education High School in Split) in 1995 he started studying at the Department of Mathematics of the Faculty of Science, University of Zagreb. He graduated on profile dipl.ing. of mathematics, in the field Computer Science in July 2001 with honours.

From June 2000 to September 2001 he had worked as a Software Developer at Multimedia Lab. From September 2001 to October 2002 he was employed at the Department of Mathematics of the Faculty of Science, University of Zagreb as a research and teaching assistant.

Since October 2002 he has been employed at the Department of Applied Computing of the Faculty of Electrical Engineering and Computing, University of Zagreb as a research and teaching assistant. In October 2002 he enrolled in the Postgraduate Study at the Faculty of Electrical Engineering and Computing. He defended his Master Thesis titled "Adaptable Workflow Management System" on May 6, 2006, under the supervision of prof.dr.sc. Krešimir Fertalj. In October 2006 he enrolled in the Doctoral Study at the Faculty of Electrical Engineering and Computing.

At the Faculty of Electrical Engineering and Computing he participated as teaching assistant on courses *Applied computing*, *Algorithms and Data Structures*, *Development of Information Systems*, *Development of Software Applications* and *Programming and Software Engineering*. Also, he participated in courses *Introduction to Computer Science*, *Operating systems* and *Programming* at the Department of Mathematics of the Faculty of Science, University of Zagreb and *Programming* at the Zagreb Polytechnic for Technical Sciences.

He is author or co-author of a number of scientific, professional and educational publications in his field. His scientific interests are Workflow Management and Software Engineering. He is fluent in English and has passive knowledge of German.