UNIVERZITET U PRIŠTINI
FAKULTET TEHNIČKIH NAUKA

Julijana Lekić

RAZVOJ INORMACIONIH SISTEMA ZA UPRAVLJANJE POSLOVNIM PROCESIMA DEMONSTRACIJOM

Doktorska disertacija

Kosovska Mitrovica, 2015. godina

Sadržaj

I Uvod

1.1 Predmet i cilj rada

Razvoj računarske tehnologije i digitalnih komunikacija i njihovo korišćenje doveli su do drastičnih promena u organizaciji rada i načinu poslovanja. Inovacije u računarstvu i komunikacijama su i dalje glavni pokretači promena u poslovnim procesima. Samim tim, poslovni procesi su postali složeniji, u velikoj meri se oslanjaju na informacione sisteme i često obuhvataju više organizacija. Modeli procesa pomažu u upravljanju složenošću jer pružaju uvid u strukturu procesa i dokumentuju procedure njegovog izvršavanja. Kao rezultat toga, modeli poslovnih procesa imaju široku primenu u današnjim organizacijama, posebno u donošenju odluka u vezi sa procesima za potrebe kontrole i planiranja, zbog čega je modelovanje procesa postalo veoma značajno.

Razvoj softvera pomoću modela je disciplina koja pretpostavlja upotrebu modela kao centralnog i osnovnog proizvoda razvoja softvera iz kojeg implementacija može nastati i neposredno. Modelovanje poslovnih procesa je disciplina u razvoju informacionih sistema u kojoj se poslovni procesi, kao skupovi parcijalno uređenih aktivnosti koje se izvršavaju, modeluju odgovarajućim jezicima za modelovanje, kao što su modeli aktivnosti na jeziku UML (*Unified Modeling Language*) [1], ili BPMN (*Business Process Model and Notation*) [2].

Osnovni zadatak razvoja softvera je da se transformišu neformalni zahtevi u formalne specifikacije koje računar može interpretirati. Transformacija zamišljenih uzoraka strukture ili ponašanja na generalizovane formalne specifikacije, često je dugotrajan proces, podložan greškama, jer programer može da previdi neke detalje s obzirom na to da razmatra uzorke koji nisu manifestovani na konkretan, vidljiv i opipljiv način.

Umesto takvog pristupa, gde programer prvo mapira svoju zamišljenu ili skiciranu strukturu i ponašanje sistema na formalne specifikacije (modele) koji upućuju računar kako da obradi opšte slučajeve, ponekad je zgodnije da to uradi direktno. Programer može da demonstrira željeno ponašanje direktno na računaru, a računar može da reprodukuje demonstrirano ponašanje na drugim slučajevima u toku eksploatacije izgrađenog sistema. Ova metafora je poznata kao *programiranje pomoću demonstracije* (engl. *programming by demonstration* - PBD) ili *programiranje pomoću primera* (engl. *programming by example* - PBE).

Predmet ovog rada jeste ispitivanje primenljivosti paradigme *programiranja pomoću demonstracije* na domen razvoja informacionih sistema za upravljanje poslovnim procesima. U radu je

koncipiran pristup ovakvom razvoju i predložena je tehnika pomoću koje se on može ostvariti. Jedan od glavnih ciljeva ove disertacije jeste pronalaženje eventualnog načina za interaktivnu konstrukciju modela poslovnih procesa korišćenjem demonstracije, na osnovu interaktivnog odigravanja različitih scenarija izvršavanja poslovnog procesa od strane korisnika. U radu su prikazani rezultati kojima je ovaj cilj ostvaren, ograničen na posebnu vrstu poslovnih procesa, a to su *paralelni poslovni procesi*.

Prednost korišćenja tehnika PBE i PBD jeste da korisnik ne mora da zna nekakav "poseban" programski jezik da bi automatizovao zadatke koji se ponavljaju, on pokazuje svoje programe putem korisničkog interfejsa. Za ostvarivanje pomenutog cilja, kreiran je sopstveni demonstracioni korisnički interfejs, koji omogućava korisniku odigravanje različitih scenarija izvršavanja aktivnosti procesa korišćenjem direktne manipulacije, što će u radu biti detaljno prikazano.

Polazna osnova istraživanja predstavljenog u ovom radu su tehnike poznate kao *play-in* i *play-out*, ali koje su primenjene na poseban domen - razvoj reaktivnih sistema. Čitav pristup i tehnika kojom je omogućeno ostvarivanje interaktivne konstrukcije blok-struktuiranih modela paralelnih poslovnih procesa (koji će biti definisani u radu), zasnovani su na konceptima i idejama tehnike otkrivanja modela procesa i α-algoritma, koja pripadaju naučnoj disciplini *pronalaženje modela* (engl. *Process Mining* - PM).

Za ostvarivanje ideja i ciljeva istraživanja predstavljenih ovim radom, izvršena je modifikacija tehnike otkrivanja modela procesa i α-algoritma [88], što će u radu biti prikazano. U svrhu praktične potvrde dobijenih rezultata, napravljena su tri priključka (engl. *plug-in*) postojećem alatu ProM, koji se koristi za pronalaženje modela procesa, koji će u okviru rada takođe biti predstavljeni. Prednost korišćenja predložene modifikovane metode i algoritma, i njihova primena na interaktivnu konstrukciju modela paralelnih poslovnih procesa, biće komentarisani i demonstrirani na konkretnim primerima. Za potrebe interaktivnog kreiranja modela paralelnih poslovnih procesa napravljen je grafički korisnički interfejs, koji će u radu biti predstavljen. Upotrebljivost predloženog rešenja biće testirana na izabranom reprezentativnom uzorku paralelnih poslovnih procesa, a rezultati izvršene eksperimentalne analize biće analizirani i komentarisani.

1.2 Struktura i sadržaj rada

Rad je podeljen na osam poglavlja. Posle ovog uvodnog poglavlja sledi poglavlje Definicija problema. U tom poglavlju je ukratko opisan glavni predmet ovog istraživanja, a to je problem neusklađenosti specificiranog modela sistema sa stvarnim ponašanjem sistema. Dat je kratak opis oblasti i tema u okviru kojih je istraživanje sprovedeno ili na koje je bilo upućeno. Uvedeni su neki osnovni pojmovi koji su od značaja za sam rad, i date su precizne definicije nekih pojmova koje čine polaznu osnovu za ovaj rad.

U trećem poglavlju ukratko je dat pregled postojećih rešenja datog problema i oblasti koje su u bližoj vezi sa temom ovog rada, a koji su bili dostupni iz literature. Osim toga, detaljnije su opisana dva pristupa koji su od izuzetnog značaja za ovo istraživanje: *Play-in/Play-out* i *Pocess Mining*. Tehnike *play-in* i *play-out*, koje se primenjuju u razvoju reaktivnih sistema, poslužile su kao polazna osnova u ovom istraživanju, ali primenjene na sasvim drugi domen - razvoj informacionih sistema za upravljanje poslovnim procesima. Predstavljena je i disciplina *process mining*, koja se bavi otkrivanjem modela procesa na osnovu evidencije zapisane u dnevnicima događaja tokom izvršavanja procesa. U okviru toga, predstavljen je ProM - alat otvorenog koda za otkrivanje modela procesa, koji je korišćen u ovom radu. Na konceptima i idejama ove discipline zasnovan je čitav pristup i tehnika kojom je omogućeno ostvarivanje interaktivne konstrukcije modela paralelnih poslovnih procesa predstavljenih u radu.

Četvrto poglavlje detaljno opisuje ideje predloženog originalnog rešenja. Najpre je predstavljena modifikovana PM metoda sa novim uvedenim relacijama između aktivnosti procesa, koje se mogu ustanoviti iz zapisa u dnevnicima događaja. Dat je i prikaz $\alpha^{\|}$-algoritma, i način na koji se došlo do njega modifikacijom postojećeg α-algoritma, kojim se na osnovu zapisa u dnevniku događaja može rekonstruisati originalna mreža.

Kako je korišćenje osnovnog α-algoritma uslovljeno time da dnevnici događaja na kojima se primenjuje moraju ispunjavati uslove tzv. *kompletnosti*, u ovom radu je rađeno na tome da se uslovi kompletnosti relaksiraju [89]. U tom cilju, najpre su definisani tzv. *kauzalno kompletni* dnevnici događaja, koji ne moraju ispunjavati uslove kompletnosti, a ipak se primenom $\alpha^{\|}$-algoritma iz njih može rekonstruisati originalna mreža paralelnog poslovnog procesa. Za verifikaciju dobijenih rezultata, kreirani su odgovarajući priključci postojećem ProM alatu. U postupku otkrivanja modela najpre je potrebno odrediti *bazičnu kauzalnu relaciju*, definisanu u okviru modifikovane PM metode, što je omogućeno priključkom: *Modified Alpha algorithm helper plug-in-find basic causality relation*, čiji je programski kod priložen u Prilogu A. Za otkrivanje originalne mreže primenom modifikovanog $\alpha^{\|}$-algoritma na kauzalno kompletan dnevnik događaja kreiran je priključak: *Mine for a Petri net using modified Alpha-algorithm*, čiji je programski kod priložen u Prilogu B. U radu je detaljno prikazana upotreba ovih priključaka za otkrivanje modela paralelnog poslovnog procesa na jednom demonstrativnom primeru.

Da bi se ustanovio odnos između neophodnih veličina kompletnih i kauzalno kompletnih dnevnika događaja, izraženih u minimalnom broju tragova u dnevnicima potrebnih za otkrivanje originalne mreže, u okviru postojećeg ProM okruženja napravljen je priključak pod nazivom: *Modified Alpha-algorithm - Minimal Logs*, čiji je programski kod dat u Prilogu C. Pomenuti priključak je

poslužio za izvođenje eksperimentalne analize koja je obavljena na uzorku od 100 realnih primera paralelnih poslovnih procesa, do kojih se došlo pretragom Interneta i odabirom javno dostupnih modela poslovnih procesa, a koji su dati u Prilogu D. Rezultati eksperimentalne analize su detaljno prikazani i statistički potvrđeni primenom testa rangova, odnosno *Wilcoxon-Mann-Whitney* testa, na njih.

U četvrtom poglavlju su definisani i *slabo kompletni* dnevnici događaja koji takođe ne moraju ispunjavati uslove kompletnosti, a iz kojih se primenom $\alpha^{\|}$-algoritma može rekonstruisati originalna mreža paralelnog poslovnog procesa. Za verifikaciju rezultata korišćeni su isti priključci: *Modified Alpha algorithm helper plug-in-find basic causality relation* i *Mine for a Petri net using modified Alpha-algorithm*, čija je primena na slabo kompletne logove u radu detaljno predstavljena na demonstracionom primeru. Predstavljeni su problemi na koje se nailazi prilikom otkrivanja originalnih mreža iz slabo kompletnih logova (viseći čvorovi), kao i rešenje tih problema. Zadatak eksperimentalne analize u ovom delu rada bio je da se ustanovi odnos između neophodnih veličina kompletnih i slabo kompletnih dnevnika, kao i odnos između minimalnih veličina kauzalno kompletnih i slabo kompletnih dnevnika događaja, potrebnih za otkrivanje originalne mreže. U tu svrhu, takođe, je korišćen priključak: *AlphaMiner_mod Find Minimal Logs from Any Log*, koji je primenjen na dnevnike događaja 100 realnih primera paralelnih poslovnih procesa datih u prilogu D. Rezultati i ove eksperimentalne analize su detaljno prikazani i statistički potvrđeni *Wilcoxon-Mann-Whitney* testom.

U petom poglavlju je prikazan predloženi postupak kreiranja blok-struktuiranih modela paralelnih poslovnih procesa pomoću demonstracije. U tu svrhu kreiran je sopstveni grafički korisnički interfejs, koji omogućava korisniku odigravanje različitih scenarija izvršavanja aktivnosti procesa korišćenjem direktne manipulacije, čiji je listing aplikacije dat u Prilogu E. Postupak interaktivnog kreiranja modela detaljno je prikazan na primeru kreiranja modela procesa ubrzavanja obrade podataka paralelizmom pomoću FPGA procesora. I u ovom delu rada izvršena je eksperimentalna analiza, čiji je cilj bio da se istraži da li može da se ustanovi minimalan broj odigravanja scenarija neophodan za dobijanje konačnog modela, i od čega to zavisi. Eksperimentalna analiza je, takođe, sprovedena na uzorku od 100 realnih primera paralenih poslovnih procesa iz baze primera, koji su dati u Prilogu D. Dobijeni rezultati su detaljno analizirani i komentarisani.

U šestom poglavlju iznet je zaključak, sa rekapitulacijom urađenog i predstavljenog u ovom radu i analizom rezultata. Ukazano je na doprinos ovog rada, ali i na njegove nedostatke. Date su smernice za dalji mogući razvoj na temeljima ideja koje su u ovom radu ostvarene.

Sedmo poglavlje donosi spisak korišćene literature.

U osmom poglavlju su dati prilozi, a u okviru njih se nalaze:

- Prilog A - paket *Alphaminer_mod_basic_relation*, u kome se nalazi programski kod priključka *Modified Alpha algorithm helper plug-in-find basic causality relation*.
- Prilog B - paket *AlphaMiner_mod*, u kome se nalazi programski kod priključka *Mine for a Petri net using modified Alpha-algorithm*.
- Prilog C - paket *AlphaMiner_mod Find Minimal Logs from Any Log*, u kome se nalazi programski kod priključka *Modified Alpha-algorithm - Minimal Logs*.
- Prilog D - baza od 100 realnih primera paralelnih poslovnih procesa.
- Prilog E - *app.js* - listing aplikacije grafičkog korisničkog interfejsa.

II Definicija problema

2.1 Problem otkrivanja modela

Razumevanje ponašanja složenih informacionih sistema je od suštinske važnosti kako bi sistem mogao da se modifikuje, održava, unapredi. Modeli poslovnih procesa igraju važnu ulogu u tome, jer analiza modela procesa u sistemu doprinosi boljem razumevanju sistema, čime se omogućuje iznalaženje načina za poboljšanje i unapređenje sistema. Kada se u poslovno okruženje uvode novi informacioni sistemi i kada se poslovni procesi redizajniraju, modeli poslovnih procesa se koriste u različite svrhe. *Neformalni* modeli se koriste za neformalnu analizu i dokumentaciju, dok se *formalni* modeli koriste za formalnu analizu i proveru, ili za stvarno izvršavanje procesa. Način kreiranja modela može uticati na usklađenost modela i toga kako se poslovne aktivnosti obavljaju u stvarnosti. Često specifikacija namenjenog ponašanja sistema nije u potpunosti u skladu sa njegovim stvarnim ponašanjem, već pruža samo idealizovan ili poželjan pogled na poslovne procese koji se obavljaju u sistemu. Predmet i cilj ovog istraživanja jeste pronalaženje modela poslovnih procesa koji oslikavaju stvarno izvršavanje procesa, i pronalaženje načina da se takvi modeli kreiraju interaktivno.

Na sreću, mnogi informacioni sistemi imaju mogućnost da zapišu svoje izvršenje, i na taj način generišu trag o događajima koji oslikavaju stvarno ponašanje sistema. Izdvajanje znanja iz tragova zapisanih u dnevnicima događaja (eng. *event log*) koji su dostupni u današnjim informacionim sistemima, i upotreba tog znanja za otkrivanje, praćenje i poboljšanje modela poslovnih procesa, osnova su za nastanak različitih tehnika oblasti zvane *pronalaženje procesa* - PM [3]. *Process mining* je relativno mlada istraživačka disciplina koja se nalazi između oblasti mašinskog učenja i pronalaženja podataka (engl. *data mining*) sa jedne, i analize i modelovanja poslovnih procesa (engl. *business process modeling*) sa druge strane. Tehnike PM pomažu organizacijama da otkriju modele svojih stvarnih poslovnih procesa na osnovu dnevnika obavljenih aktivnosti i pružaju nove načine za unapređenje poslovnih procesa u različitim domenima aplikacija. Podaci snimljeni pomoću informacionih sistema, odnosno dobijeni iz zapisnika stvarno odigranih aktivnosti tokom realnih poslovnih procesa, mogu se koristiti za pružanje boljeg uvida u stvarne procese. Na taj način se odstupanja stvarnih od modelovanih procesa mogu analizirati, a kvalitet modela poboljšati.

Algoritmi za otkrivanje modela procesa, poput α-algoritma [3], [4], [5], mogu automatski da generišu model procesa iz dnevnika događaja. Takvi pristupi otkrivanju modela procesa su primeri tzv.

play-in tehnika [6], [7], čiji je cilj konstruisanje modela na osnovu primera ponašanja, što se često naziva "zaključivanje" (eng. *inference*) ili *programiranje pomoću primera* ili *demonstracije*.

Iako u osnovi jednostavan, α-algoritam se nije pokazao naročito praktičnim zbog mnogih problema koje nije u stanju da prevaziđe [3]. Međutim, mnoge njegove ideje ugrađene su u složenije i robusnije tehnike [4], [8-18]. Tako su nastale različite varijacije α-algoritma kao što su: *Genetic process mining* [15], *Fuzzy mining* [19], *Flexible heuristics miner* [20], regioni zasnovani na stanjima [21], [22], regioni zasnovani na jezicima [8], [18] i drugi. Većina ovih algoritama je nastala iz potrebe za prevazilaženjem problema na koje se nailazilo primenom osnovnog α-algoritma, ali problem kompletnosti dnevnika događaja (o čemu će više reči biti kasnije) još nije prevaziđen, i još uvek ostaje izazov za buduće istraživače. Kao takav, postao je predmet posmatranja i modifikacije, u okviru jednog šireg istraživanja mogućnosti pronalaženja modela poslovnih procesa pomoću primera/demonstracije prikazanog u ovom radu.

Primena osnovnog α-algoritma u pokušaju da se dođe do interaktivnog kreiranja modela poslovnih procesa nije dala željene rezultate. To je bio jedan od osnovnih razloga za modifikacijom α-algoritma koja je izvedena u sklopu ovog istraživanja, u cilju pronalaženja algoritma kojim bi se mogla ostvariti početna ideja: kreiranje modela poslovnih procesa pomoću demonstracije. Sprovedenim istraživanjem se došlo do tehnike i algoritma koji, kada se primene na određenu vrstu procesa - paralelne procesa, daju željene rezultate, odnosno omogućuju interaktivno kreiranje blok-struktuiranih modela paralelnih poslovnih procesa, što će u radu biti prikazano.

2.2 Polazne osnove

U ovoj sekciji su date definicije nekih pojmova na kojima se bazira tehnika otkrivanja modela kod PM-a, a posebno α-algoritam koji se primenjuje na podacima zapisanim u dnevniku događaja, preuzeti iz [3], [4] i [5], s obzirom na to da su na istim osnovama zasnovani i modifikovana PM tehnika i $\alpha^{\|}$-algoritam, o čemu će kasnije biti reči.

2.2.1 Petri mreže

α-algoritam iz evidencije o izvršavanju procesa zapisane u vidu tragova u dnevniku događaja konstruiše Petri mrežu [23], [24] kojom se predstavlja ponašanje zabeleženo u evidenciji, bez korišćenja bilo kakvog dodatnog znanja. Pri tome se koristi varijanta klasičnih Petri mreža, tj. mreže Mesto/Tranzicija (engl. *Place/Transition* (P/T) *nets*).

Definicija 2.1. (*Mreže Mesto/Tranzicija - P/T*) P/T mreža je uređena torka $N = (P, T, F)$, gde je:
1. P konačan skup mesta (engl. *place*),
2. T je konačan skup tranzicija tako da $P \cap T = \emptyset$,
3. $F \subseteq (P \times T) \cup (T \times P)$ je skup usmerenih grana nazvanih *relacijama toka* (engl. *flow relation*).

Markirana P/T mreža je par (N, s), gde je $N = (P, T, F)$ P/T mreža, a s je multiskup nad P koji označava markiranje mreže. Skup svih markiranih P/T mreža označava se sa $\mathcal{N}$.

Neka je $N = (P, T, F)$ markirana P/T mreža. Elementi $P \cup T$ se zovu *čvorovi*. Čvor x je *ulazni čvor* drugog čvora y akko postoji usmerena linija od x do y (tj. $(x, y) \in F$). Čvor x je *izlazni čvor* od y akko $(y, x) \in F$. Za bilo koje $x \in (P \cup T)$, skup prethodnika čvora x je skup $\bullet x = \{y \mid (y, x) \in F\}$, i skup sledbenika čvora x je skup $x\bullet = \{y \mid (x, y) \in F\}$.

Dinamičko ponašanje markiranih P/T mreža je definisano *pravilom okidanja*.

Definicija 2.2. (*Pravilo okidanja*) Neka je $(N = (P, T, F), s)$ markirana P/T mreža. Tranzicija $t \in T$ je omogućena, što se označava sa $(N, s)[t\rangle$, akko $\bullet t \leq s$. Pravilo okidanja $\_[\ \_\ \rangle\_ \subseteq \mathcal{N}\times T\times\mathcal{N}$ je najmanja relacija koja je zadovoljena za bilo koju $(N = (P, T, F), s) \in \mathcal{N}$ i bilo koje $t \in T$, $(N, s)[t\rangle \Rightarrow (N, s)[t\rangle\ (N, s - \bullet t + t\bullet)$.

Ponekad je potrebno znati sekvencu tranzicija koje se okidaju kako bi se dostiglo određeno markiranje.

Definicija 2.3. (*Dostupno* (engl. *reachable*) *markiranje*) Neka je (N, s_0) markirana P/T mreža u $\mathcal{N}$. Markiranje je *dostupno* od početnog markiranja s_0 akko postoji sekvenca omogućenih tranzicija čije okidanje vodi od s_0 do s. Skup dostupnih markiranja mreže (N, s_0) označava se sa $[N, s_0\rangle$.

Sekvenca dužine n, za neki prirodan broj $n \in \mathrm{I}N$ nad alfabetom A je funcija $\sigma : \{0, \ldots, n\text{-}1\} \rightarrow A$. Skup svih sekvenci proizvoljne dužine nad alfabetom A označava se A*.

Definicija 2.4. (*Sekvenca okidanja*) Neka je (N, s_0) sa $N = (P, T, F)$ markirana P/T mreža. Sekvenca $\sigma \in T^*$ zove se *sekvenca okidanja* mreže (N, s_0) akko, za neki prirodan broj $n \in \mathrm{I}N$, postoje markiranja $s_1, \ldots, s_n$ i tranzicije $t_1, \ldots, t_n \in T$ tako da $\sigma = t_1 \ldots t_n$, i za svako i gde $0 \leq i < n$, $(N, s_i)[t_{i+1}\rangle$, i $s_{i+1} = s_i - \bullet t_{i+1} + t_{i+1}\bullet$. Za sekvencu σ se kaže da je omogućena markiranjem s_0, što se označava $(N, s_0)[\sigma\rangle$. Sekvenca okidanja σ rezultuje markiranjem s_n, što se označava $(N, s_0)[\sigma\rangle\ (N, s_n)$.

Definicija 2.5. (*Povezanost*) Mreža $N = (P, T, F)$ je *slabo povezana*, akko za svaka dva čvora x i y u $P \cup T$, $x\ (F \cup F^{-1})^* y$, gde je R^{-1} inverzija a R^* refleksivno i tranzitivno zatvaranje relacije R. Mreža N je *strogo povezana* akko za svaka dva čvora x i y važi xF^*y.

Kod tehnike otkrivanja modela procesa, podrazumeva se da su sve mreže slabo povezane i da imaju najmanje dva čvora.

Definicija 2.6. (*Ograničenost, sigurnost*) Markirana mreža $(N = (P, T, F), s)$ je *ograničena* akko je skup markiranja koji se mogu dostići $[N, s\rangle$ konačan. Ona je *sigurna* akko za bilo koje $s' \in [N, s_0\rangle$, i bilo koje $p \in P$, $s'(p) \leq 1$.

Treba napomenuti da osobina sigurnosti nalaže i ograničenost mreže. Kod sigurnih mreža ni u jednom mestu se ne može pojaviti više od jednog tokena.

Definicija 2.7. (*Mrtve* (engl. *dead*) *tranzicije*, *živost* (engl. *liveness*)) Neka je $(N = (P, T, F), s)$ markirana P/T mreža. Tranzicija $t \in T$ je *mrtva* u (N, s) akko ne postoji dostupno markiranje $s' \in [N, s\rangle$ tako da $(N, s')[t\rangle$. (N, s) je *živa* akko za svako dostupno markiranje $s' \in [N, s\rangle$ i $t \in T$ postoji dostupno markiranje $s'' \in [N, s'\rangle$ tako da $(N, s'')[t\rangle$.

2.2.2 *Workflow* mreže

α-algoritam je u stanju da otkrije model poslovnog procesa iz potklase Petri mreža koje modeluju dimenziju toka kontrole (engl. *control-flow*) procesa, poznate kao *WorkFlow mreže* (WF-mreže) [25].

Definicija 2.8. (*Workflow mreže*) Neka je $N = (P, T, F)$ P/T mreža i $\bar{t}$ identifikator koji ne pripada $P \cup T$. N je *workflow* mreža (WF-mreža) akko je ispunjeno:

1. objekat kreiranja: P sadrži ulazno mesto i takvo da $\bullet i = \emptyset$,
2. objekat kompletiranja: P sadrži izlazno mesto o takvo da $o\bullet = \emptyset$, i
3. povezanost: $\overline{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ je strogo povezana.

Dakle, WF-mreža je Petri mreža sa tačno jednim početnim mestom procesa, tačno jednim mestom završetka procesa, i svim čvorovima dostupnim od početnog mesta (tj. kada je samo početno mesto markirano jednim tokenom „•"). WF-mreža je prirodna potklasa Petri mreže koja odslikava dinamičko ponašanje izdvojenog slučaja procesa, zbog čega je prikladna za modelovanje i analizu operativnih procesa. α-algoritam je algoritam zasnovan na činjenici da za većinu WF-mreža važi da su dve aktivnosti povezane ako i samo ako se njihova kauzalnost može detektovati iz dnevnika događaja [4], [5]. Pretpostavlja se da su mreže povezane, ograničene i sigurne.

Čak i ako mreža zadovoljava sve uslove postavljene definicijom 2.8, odgovarajući proces može ispoljiti greške kao što su zastoji, zadaci koji nikad ne postaju aktivni, ostaci procesa posle prekida i slično. Zbog toga se defniše sledeći kriterijum ispravnosti:

Definicija 2.9. (*Ispravnost*) Neka je $N = (P, T, F)$ WF-mreža sa ulaznim mestom i i izlaznim mestom o. N je *ispravna* (eng. *sound*) akko su ispunjeni uslovi:

1. sigurnost: $(N, [i])$ je sigurna,
2. pravilan završetak: za bilo koje markiranje $s \in [N, [i]\rangle$, $o \in s$ nalaže da je $s = [o]$,
3. mogućnost završetka: za bilo koje markiranje $s \in [N, [i]\rangle$, $[o] \in [N, s\rangle$,
4. odsustvo „mrtvih" tranzicija: $(N, [i])$ ne sadrži mrtve tranzicije.

Skup svih sound WF-mreža označava se sa $\mathcal{W}$.

2.2.3 Problem ponovnog otkrivanja procesa

Cilj algoritma otkrivanja procesa je mogućnost da se ponovo otkrije model (tj. WF-mreža) na osnovu tragova (engl. *trace*) u dnevniku događaja. Dnevnik događaja se definiše na sledeći način:

Definicija 2.10. (*Trag*, *dnevnik događaja*) Neka je T skup aktivnosti, $\sigma \in T^*$ je *trag* i $L \in \mathcal{P}(T^*)$[1] *je dnevnik događaja*.

Da bi se pojednostavilo korišćenje dnevnika događaja i sekvenci, uvode se dodatne notacije:

[1] $\mathcal{P}(T^*)$ je partitivni skup (engl. *powerset*) od T^*, tj. $L \subseteq T^*$

Definicija 2.11. ($\in$, *prvi*, *poslednji*) Neka je A skup, $a \in A$, i $\sigma = a_1\, a_2 \ldots a_n \in A^*$ sekvenca nad A dužine n. $\in$, *prvi* i *poslednji* su definisani na sledeći način:

1. $a \in \sigma$ akko $a \in \{a_1, a_2, \ldots, a_n\}$,
2. *prvi* $(\sigma) = a_1$, ako $n \geq 1$, i
3. *poslednji* $(\sigma) = a_n$ ako $n \geq 1$.

Definicija 2.12. (*Dnevnik događaja* (engl. *event log*)) Neka je $\mathcal{A}$ skup imena aktivnosti. *Trag* σ je sekvenca aktivnosti, tj. $\sigma \in \mathcal{A}^*$, dok je dnevnik događaja L multi skup tragova nad $\mathcal{A}$, tj. $L \in \mathbb{B}(\mathcal{A}^*)$.

U mreži mogu postojati mesta koja ne utiču na ponašanje procesa - implicitna mesta, i ona se ne mogu otkriti izdvajanjem iz dnevnika događaja.

Definicija 2.13 (*Implicitna mesta*) Neka je $N = (P, T, F)$ P/T mreža sa početnim markiranjem s. Mesto $p \in P$ se naziva *implicitnim* u (N, s) akko za svako dostupno markiranje $s' \in [N, s\rangle$ i tranziciju $t \in p\bullet$ važi $s' \geq \bullet t \setminus \{\mathrm{p}\} \Rightarrow s' \geq \bullet t$.

S obzirom da se ovakva mesta ne mogu detektovati u dnevniku događaja, istraživanje je ograničeno na mreže bez implicitnih mesta, a to su tzv. *strukturirane workflow mreže* (engl. *structured workflow* - SWF *nets*).

Definicija 2.14 (*SWF mreža*). WF mreža je SWF (*strukturirana workflow mreža*) akko:

1. Za svako $p \in P$ i $t \in T$ gde $(p, t) \in F$: $|\mathrm{p}\bullet| > 1$ nalaže da je $|\bullet t| = 1$.
2. Za svako $p \in P$ i $t \in T$ gde $(p, t) \in F$: $|\bullet t| > 1$ nalaže da je $|\bullet p| = 1$.
3. Ne postoje implicitna mesta.

SWF mreže zadovoljavaju jednu interesantnu osobinu, koja sledi direktno iz definicije SWF mreže i ukazuje na to da ne postoje dve tranzicije koje su povezane višestrukim mestima.

Osobina 2.1. Neka je $N = (P, T, F)$ SWF mreža. Za bilo koje $a, b \in T$ i $p_1, p_2 \in P$: ako $p_1 \in a\bullet \cap \bullet b$ i $p_2 \in a\bullet \cap \bullet b$, onda je $p_1 = p_2$.

Otkrivanje procesa je jedan od najizazovnijih zadataka discipline pronalaženja modela. Generalni problem otkrivanja procesa može biti definisan na sledeći način:

Definicija 2.15 (*Opšti problem otkrivanja procesa*) Neka je L dnevnik događaja nad $\mathcal{A}$, tj. $L \in \mathbb{B}(\mathcal{A}^*)$. *Algoritam otkrivanja procesa* je funkcija koja mapira L na model procesa, tako da je model „reprezentativan“ u pogledu ponašanja zaključenog iz dnevnika događaja.

Kako se za odredišni model uzima Petri mreža, cilj je da se pronađe Petri mreža koja može „ponovo da odigra“ (*replay*) dnevnik događaja L. S obzirom na to, problem otkrivanja procesa se može preformulisati i učiniti konkretnijim sledećom definicijom koja je prilagođena specifičnosti mreže:

Definicija 2.16 (*Specifičan problem otkrivanja procesa*) *Algoritam otkrivanja procesa* je funkcija γ koja mapira dnevnik događaja $L \in \mathbb{B}(\mathcal{A}^*)$ na markiranu Petri mrežu $\gamma(L) = (N, M)$. Idealno, N je ispravna (sound) WF-mreža, i svi tragovi u L odgovaraju mogućoj sekvenci okidanja (N, M).

Funkcija γ definiše takozvanu *play-in* tehniku.

III Pregled postojećih rešenja

Kao što je u uvodnom delu rečeno, predmet ove disertacije je pronalaženje eventualnog načina za interaktivnu konstrukciju modela poslovnih procesa, u kojem projektant sam definiše modele demonstracijom primera koji su deo opisa poslovnih procesa u praksi, a sistem sam ili uz pomoć projektanta formira takve modele. S obzirom na to da je čitav rad zasnovan na temeljima PBD i PBE paradigme, u ovom poglavlju će biti prikazan kratak pregled i opisi postojećih tehnika i alata baziranih na PBD i PBE u hronološkom redosledu.

Pored toga, detaljnije će biti predstavljene tehnike *play-in* i *play-out* D. Harela [6], [7], [26-31] i *pronalaženje procesa* W.M.P van der Aalsta [3], [4], [5], s obzirom na to da se celokupno istraživanje prikazano u ovom radu u velikom delu bazira na pomenutim tehnikama, modifikujući ih i prilagođavajući ih domenu od interesa, a to su modeli poslovnih procesa.

3.1 Programiranje pomoću demonstracije i programiranje pomoću primera

U istraživanju softverskog razvoja, tehnike *programiranje pomoću primera* i *pogramiranje pomoću demonstracije* su se pojavile kao način za definisanje niza operacija bez potrebe za učenjem programskog jezika. Uobičajena razlika u literaturi između ova dva termina jeste što u PBE korisnik daje prototipski proizvod kompjuterskog izvršenja (kao što je npr. red u željenom rezultatu upita kod Query by Example - QBE), dok u PBD korisnik izvodi niz akcija koje računar mora da ponavlja, generalizujući ih kako bi bile upotrebljive na različitim skupovima podataka. Ova dva termina su najpre bila nerazdvojiva, ali vremenom je tehnika PBE bila usvojena od strane softverskih istraživača, a PBD od strane istraživača oblasti robotike. Danas PBE predstavlja potpuno nov koncept, podržan novim programskim jezicima sličnim simulatorima [32].

Programiranje primerima i demonstracijom je veoma stara ideja primenjena u različitim oblastima, nastala kao potreba za prevazilaženjem složenosti konvencionalnog programiranja. Konvencionalni programski jezici su teški za učenje i korišćenje, i zahtevaju znanje i veštinu koju mnogi korisnici računara ne poseduju. U pokušaju da se programiranje učini lakšim zadatkom istraživanja su bila usmerena ka korišćenju grafike, što je rezultovalo *grafičkim programiranjem* [33].

Grafičko programiranje ne koristi konvencionalne tekstualne programske jezike, već programe izražava korišćenjem dvodimenzionalnih grafičkih objekata.

Među prvim istraživanjima u tom smeru bilo je istraživanje koje je sproveo D. C. Smith opisujući u svojoj disertaciji sistem *Pygmalion* [34] koji kombinuje dve moćne ideje: *grafičko programiranje* i *programiranje pomoću primera*. Korisnici Pygmaliona koriste ikone koje predstavljaju operacije i operande, kojima se može manipulisati ručno. Vrednosti podataka se mogu pomerati u ikone i van njih, preko ikona se može zahtevati izvršenje operacija, korisnici mogu kreirati nove ikone. Operatori ikona su tipični za programske jezike: aritmetički, relacioni i logički operatori.

Slično kao kod Pygmaliona, u sistemu koji je kreirao G. A. Curry [35] korisnik manipuliše ikonama, ali samo ograničenim skupom ikona (memorijskim ćelijama i operatorima), i ne može da kreira nove ikone. Ovaj sistem je zasnovan na metodologiji programiranja zvanoj *programiranje pomoću apstraktne demonstracije* (engl. *programming by abstract demonstration* - PAD). U tom sistemu korisnik može da piše programe pomoću primera koji sadrže samo konstante, koristeći konkretne uzorke podataka. Međutim, korisnik takođe može da zapisuje programe koristeći apstraktne vrednosti podataka.

B. Shneiderman [36] je smatrao da grafički prikaz sam po sebi ne može garantovati bolje performanse zbog toga što grafički prikaz, kao sintaksa jezičkih komandi, ima aspekte koje bi trebalo naučiti, ikone mogu biti zbunjujuće ili nejasne, grafičke oznake mogu dovesti do lažnih zaključaka. Stoga je zamenio složenu sintaksu jezičkih komandi *direktnom manipulacijom* objektima od interesa. Centralna ideja direktne manipulacije je vidljivost objekata na ekranu, kao i brze, reverzibilne i inkrementalne aktivnosti.

Neki sistemi vizuelnog programiranja su uspešno pokazali da neprogrameri mogu kreirati veoma složene programe sa vrlo malo obuke. Jedan od njih je *SmallStar* [37] koji predstavlja sistem sa direktnom manipulacijom. Kao početnu tačku u svom istraživaju programiranja pomoću primera D. C. Halbert [37] je koristio Xerox Star 8010 poslovni informacioni sistem kao višekorisnički informacioni sistem opšte namene, kome je dodao notaciju *generalizacije* i jednostavan mehanizam *skupa iteracija*. SmallStar omogućava funkcionalnost, ali je i programabilan tako da može automatizovati procedure za određene aplikacije. U sistemu SmallStar korisnik izvršava aktivnosti zadataka koje želi da budu zabeležene. Za to vreme sistem kreira program praveći transkript izvršenih aktivnosti. Za pisanje programa pomoću primera korisnik zahteva od sistema da zabeleži ove aktivnosti. SmallStar ne beleži program samo interno, već program predstavlja i u korisniku čitljivoj formi, koja je vidljiva kada se otvori ikona programa. Ukoliko je ikona programa otvorena dok korisnik zapisuje program, svaka korisnička aktivnost se dodaje u program inkrementalno i prikazuje se na ekranu.

U svojim ranim radovima B. A. Myers [33] govori o korišćenju grafike kao pomoćnog sredstva u programiranju, debagovanju i razumevanju kompjuterskih programa. On uvodi pojmove *vizuelno programiranje* (engl. *Visual Programming* - VP) i *vizuelizacija programa* (engl. *Program Visualization* - PV), kao tehnike koje koriste primere kao način za prevazilaženje složenosti programiranja, i naziva ih *programiranjem pomoću primera*. Vizuelno programiranje se odnosi na bilo koji sistem koji dopušta korisniku da specificira program na dvodimenzionalan ili višedimenzionalan način. Tehnika VP uključuje konvencionalne dijagrame toka i grafičke programske jezike. U tehnici VP program je sam

po sebi grafički, dok je u tehnici PV program specificiran na konvencionalan, tekstualan način, a grafika se koristi za ilustraciju nekih aspekata programa ili njegovog izvršenja.

Za ilustraciju kako neprogrameri mogu da kreiraju korisnički interfejs pomoću demonstracije, B. A. Myers je razvio sistem *Peridot* (*Programming by Example for Real-time Interface Design Obviating Typing*), koji koristi ograničenja podataka, programiranje pomoću primera i vizuelno programiranje [38], [39]. Grafika i ograničenje podataka se koriste za održavanje važnih relacija, programiranje pomoću primera se koristi kako bi se ove relacije automatski zaključile iz primera, a vizuelno programiranje dopušta dizajneru da kreira korisnički interfejs grafički i da vidi kako se interfejs razvija.

Dalji rad B. A. Myersa rezultovao je sledećim alatima i korisničkim interfejsima:

- *Garnet* je sveobuhvatno razvojno okruženje korisničkog interfejsa za X/11 [40], [41]. Garnet pruža mogućnost kreiranja grafičkih visoko-interaktivnih korisničkih interfejsa sa direktnom manipulacijom.

- *Amulet* je razvojno okruženje korisničkog interfejsa koje olakšava programerima kreiranje visoko-interaktivnih korisničkih interfejsa softvera za Unix, Windows i Macintosh. Ključna karakteristika Amulet dizajna je da se svi grafički objekti i ponašanje ovih objekata eksplicitno predstavljaju u vreme izvršenja, tako da sistem može obezbediti veći broj ugrađenih (engl. *built-in*) funkcija višeg nivoa, uključujući automatski prikaz, editovanje objekata i spoljašnu analizu i kontrolu interfejsa [42];

- *Gamut* je PBD alat za izradu interaktivnog softvera kao što su igre, simulacije i edukacioni softver [43].

Allen Cypher je u svojoj knjizi [44] dao sveobuhvatan pregled tada postojećih PBD sistema, opisujući 18 sistema u hronološkom redosledu*: Pygmalion*, *Tinker*, *A Predictive Calculator*, *Rehearsal World*, *SmallStar*, *Peridot*, *Metamouse*, *TELS*, *Eager*, *Garnet*, *The Turvy Experience*, *Chimera*, *The Geometer's Sketchpad*, *Tourmaline*, *A History-Based Macro by Example System***,** *Mondrian*, *Triggers* i *The AIDE Project*.

Baveći se problemom kako u tehnici PBE opisati akcije i objekte selektovane od strane korisnika u korisničkom interfejsu, i na taj način odrediti koja vrsta generalizacije je moguća, H. Lieberman i saradnici [45] predlažu *vizuelnu generalizaciju*. Za opisivanje namera korisnika kod vizuelne generalizacije se koriste vizuelne karakteristike samih elemenata interakcije, kao što su veličina, oblik, boja i izgled grafičkih objekata. Primena vizuelne generalizacije konkretizovana je, između ostalog, kod sistema *Tinker* [46], *Mondrian* [47] i *CADUI* (*Computer-aided Design of User Interfaces*) [48].

Danas je tehnika PBD najšire zastupljena u oblasti robotike [49], [50], [51], gde se na osnovu primera ili demonstracije kreira strategija koja omogućava robotu da izvrši određenu aktivnost, što je u literaturi poznato kao *Learning from Demonstration* - LfD [52], [53], [54]. U okviru tehnike LfD, u toku demonstriranja željenog ponašanja robota, definiše se i zapisuje niz parova stanje-akcija. LfD algoritmi koriste taj skup primera za razvoj strategije kojom robot odabira određenu akciju na osnovu trenutnog stanja, i na taj način reprodukuje demonstrirano ponašanje.

3.2. *Play-in/Play-out* pristup

D. Harel je sa saradnicima razvio pristup u kome se posebno profilisani modeli interakcije sa formalnom semantikom izvršavanja, nazvani *Live Sequence Charts* - LSC, mogu specificirati demonstracijom ponašanja sistema zasnovano na scenarijima (koju oni nazivaju *play-in*) [6], [7]. Nakon specifikacije LSC modela na ovaj način, sistem može sam reprodukovati definisane scenarije u konkretnim slučajevima (tehnika koju nazivaju *play-out*) [27], [28], [30], [31]. Ovaj pristup namenjen je prvenstveno za razvoj reaktivnih sistema opšte namene pokretanih događajima (engl. *event-driven*). Složenost kojom su se bavili kod takvih sistema ne potiče od složenih proračuna ili kompleksnih podataka, nego od komplikovane interakcije ka-i-od između sistema i njegovog okruženja i između delova samog sistema.

Kada se razmišlja o reaktivnim sistemima misli prirodno vode ka scenarijima ponašanja. Drugim rečima, mnogo je prirodnije da se opiše i diskutuje reaktivno ponašanje sistema pomoću scenarija ponašanja koje sistem omogućava, nego preko reaktivnosti zasnovane na stanju svake njegove komponente. Sa druge strane, za implementaciju sistema, za razliku od navođenja njegovog potrebnog ponašanja, modelovanje zasnovano na stanju je potrebno, pri čemu za svaku komponentu mora da se specificira kompletan niz mogućnosti za ulazne događaje i promene, i reakcije komponenti na njih.

Zapravo, radi se o dualnosti dva pristupa: *inter-objektnog* i *intra-objektnog*. Inter-objektni pristup karakteriše opis ponašanja zasnovan na scenariju, koji prolazi kroz granice komponenata (ili objekata) sistema, radi pružanja koherentnog i sveobuhvatnog opisa scenarija ponašanja. Kod intra-objektnog pristupa postoje opisi ponašanja zasnovani na stanju, koji ostaju unutar komponente ili objekta i baziraju se na pružanju kompletnog opisa reaktivnosti svakog od njih. Prvi pristup je intuitivniji i prirodniji za razumevanje, i zbog toga se uklapa u faze analize zahteva i testiranja u ciklusu razvoja sistema. Drugi pristup je međutim potreban za implementaciju, gde se traži da se svaka komponenta ili objekat sistema snabde kompletnom reaktivnošću tako da zaista može da radi ili da se izvršava.

Baveći se ponašanjem zasnovanom na scenarijima, D. Harel i saradnici pružaju različite modalitete specifikacija scenarija. Oni prave razliku između scenarija koji se *mogu* desiti i onih koji *moraju* da se dese, kao i između onih koji se dešavaju spontano i onih za koje je potreban „okidač“ da bi izazvao njihovo dešavanje. Osim toga, oni definišu *antiscenarija*, tj. zabranjena scenarija, u smislu da ukoliko se oni dese onda je nešto pogrešno: ili u specifikaciji nešto nije urađeno kako treba ili implementacija ne zadovoljava pravilno specifikaciju.

Osnovna ideja D. Harela i saradnika je da se intra-objektno modelovanje zasnovano na stanju, koje dovodi do implementacije trošeći pri tome vreme, energiju i novac, zameni pristupom inter-objektnog ponašanja koje je izražajno, prirodno i izvršivo. Da bi to ostvarili, osmislili su način kojim se zabeleženo ponašanje zasnovano na scenarijima može izvršiti simulirajući razvoj sistema upravo kao da je specificiran na konvencionalan način zasnovan na stanjima. U tom cilju predložili su jezik, dve tehnike sa detaljnim algoritmom i alat.

Čitav pristup je omogućen jezikom LSC koji je proširen i dopunjen na mnogo načina, što je rezultovalo veoma izražajnim sredstvom za prikazivanje ponašanja zasnovanog na scenarijima. Prva tehnika, *play-in*, uključuje korisniku blizak i prirodan način odigravanja ponašanja zasnovanog na scenarijima direktno putem grafičkog korisničkog interfejsa (engl. *graphical user interface* - GUI), u toku kojeg se LSC generiše automatski. Druga tehnika, *play-out*, omogućava da se na osnovu odigranog ponašanja izvrši sistem u skladu sa informacijama zasnovanim na scenarijima ponašanja. Ove ideje su u celini podržane alatom nazvanim *Play-Engine* [29].

U osnovi, rad D. Harela i saradnika razmatra mogućnost alternativnog načina programiranja ponašanja reaktivnih sistema koje je u potpunosti zasnovano na scenarijima i inter-objektnoj prirodi. Ideja je da LSC može stvarno da predstavlja implementaciju sistema sa play-out algoritmima, i Play-Engine kao vrste „univerzalnog reaktivnog mehanizma“ koji izvršava LSC kao da predstavlja konvencionalnu implementaciju. Na taj način, specifikacija ponašanja reaktivnog sistema ne uključuje bilo koje intra-objektno modelovanje ili programiranje.

3.2.1 Vizuelni jezik Live Sequence Charts - LSC

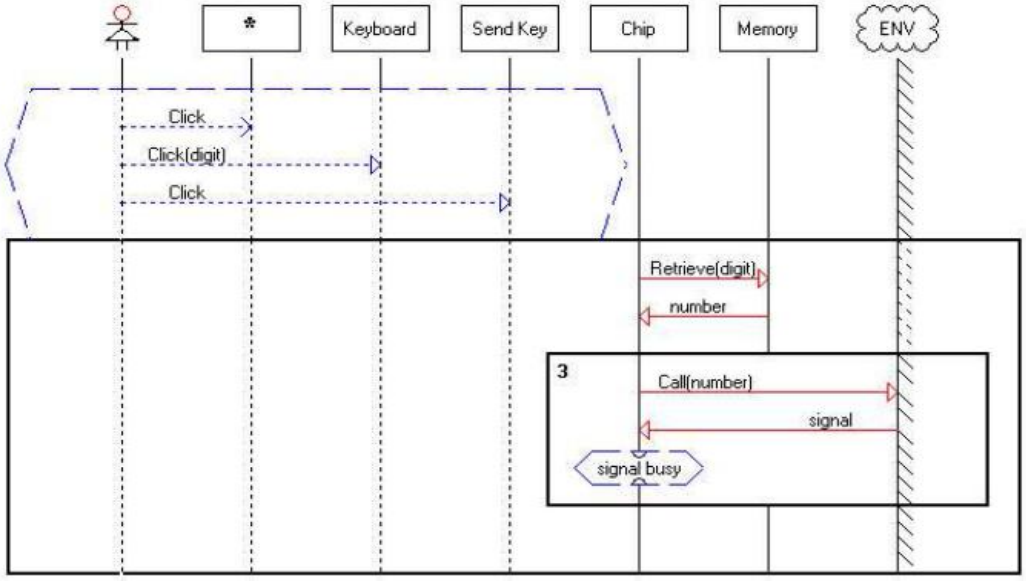
Važnost izvršivih modela je ne samo u njihovoj mogućnosti vođenja do konačne implementacije, nego i u testiranju i otklanjanju grešaka, osnova čega su definisani *zahtevi* postavljeni na početku razvoja sistema. Specificirani zahtevi predstavljaju ograničenja, potrebe, želje koje se odnose na sistem koji se razvija. Notacija za izražavanje zahteva u ponašanju preuzeta je iz jedinstvenog jezika modelovanja UML [1] koji se koristi kod objektno orijentisanog modelovanja [55], posebno iz dela jezika UML koji specificira nedvosmislene, izvršive modele koji se mogu implementirati, tzv. *izvršivog* (engl. *executable*) UML [56].

Zahtevi mogu biti *formalni* (rigorozni i precizno definisani) ili *neformalni* (npr. pisani prirodnim jezikom ili pseudokodom). Jedan od načina za opisivanje zahteva u ponašanju na višem nivou je ideja *slučaja upotrebe* (engl. *use case*) preuzeta iz jezika UML. Slučaj upotrebe je neformalni opis skupa mogućih scenarija, koji obuhvata sistem koji se razmatra i njegove spoljne aktere. Slučaj upotrebe opisuje vidljive reakcije sistema na događaje pokrenute od strane njegovih korisnika. Obično je opis slučaja upotrebe podeljen na glavni, najčešće korišćen scenario, i sporedne scenarije na koje se glavni grana i koji opisuju ponašanja koja nisu centralna (npr. moguće greške, prekidi operacija pre kompletiranja itd.). Međutim, s obzirom na to da su slučajevi upotrebe neformalni po prirodi, ne mogu poslužiti za formalno testiranje i verifikaciju. Za podršku potpunijem i rigoroznijem razvojnom ciklusu, slučajevi upotrebe moraju biti translirani na sasvim detaljne zahteve pisane na nekom formalnom jeziku.

Način za formalnu specifikaciju zahteva koji se primenjuje u oblasti objektno orijentisanih sistema jeste *Message Sequence Chart* - MSC, koji se koristi za specifikaciju scenarija kao sekvence interakcije porukama između instanci objekata. Ovaj vizuelni jezik se u okviru UML jezika manifestuje kao *dijagram sekvenci*. MSC se dobro kombinuje sa slučajevima upotrebe, jer može da

specificira scenarije koji instanciraju slučajeve upotrebe. Na taj način se oslikavaju željene međusobne relacije između procesa, zadataka, komponenti ili instanci objekata, kao i između njih i okruženja. Drugim rečima, projektant koristi MSC za formalnu vizuelizaciju stvarnih scenarija pre nego apstraktnih. Ukupan efekat takvog korišćenja vizuelnog jezika MSC jeste specifikacija scenarija ponašanja koja se sastoji od poruka koje se prosleđuju između objekata i ispunjenih postavljenih uslova duž puta.

Live Sequence Charts je proširena verzija vizuelnog jezika MSC koja ima mogućnost da specificira „životnost“ (engl. *liveness*), tj. događaje koji moraju da se dese. Tehnički, LSC omogućava prikazivanje razlike između mogućeg i neophodnog ponašanja, i to globalno na nivou celog dijagrama, i lokalno kada određuje događaje, uslove i napredovanje tokom vremena unutar dijagrama. Na slici 3.1 dat je primer korišćenja vizuelnog jezika LSC.



Slika 3.1: Live sequence chart – LSC.

LSC sadrži dva tipa dijagrama: *univerzalni* (oivičen punom linijom) i *egzistencijalni* (oivičen isprekidanom linijom) (slika 3.1). Univerzalni dijagram se koristi za specifikaciju ponašanja zasnovanih na scenarijima koja se odnose na sva moguća izvršavanja sistema. Univerzalni dijagram ima deo *preddijagram* (engl. *prechart*) koji specificira scenario koji, ako je zadovoljen, prisiljava sistem da takođe ispuni „telo“ dijagrama, tzv. *glavni* dijagram. Na taj način LSC izaziva odnos akcije i reakcije između scenarija koji se pojavljuje u preddijagramu i onog koji se pojavljuje u glavnom dijagramu, što ga pretvara u neku vrstu konstrukcije: ako-onda (engl. *iff-then*). Zbog toga, uslovi u univerzalnom dijagramu moraju biti zadovoljeni u bilo kom trenutku izvršavanja sistema.

Egzistencijalni dijagram se koristi za specifikaciju primera interakcije između sistema i njegovog okruženja, i mora biti zadovoljen bar u toku jednog izvršavanja sistema. Na taj način, ne zahteva od sistema da se ponaša na određen način u svim slučajevima, već pre ukazuje na to da postoji bar jedan skup okolnosti pod kojima se određeno ponašanje dešava. Egzistencijalni dijagrami mogu da

se koriste za specifikaciju testova sistema, ili jednostavno za ilustraciju dužih scenarija koji omogućavaju širu sliku o mogućim ponašanjima sistema koji se posmatra.

Jedan od aspekata vizuelnog jezika LSC koji doprinosi njegovoj fleksibilnosti je činjenica da ponašanja sistema mogu biti data u odvojenim dijagramima, i da svaki od njih može opisati fragment složenijeg ponašanja. Ti fragmenti postaju relevantni za vreme izvršavanja i izvršavaju se kao posledica njihovih preddijagrama, na taj način na dijagramu nije unapred nametnut eksplicitan redosled izvršavanja.

Unutar dijagrama, živi elementi nazvani „vrući“ (engl. *hot*) označavaju događaje koji *moraju* da se dese, i mogu se koristiti za specifikaciju različitih modaliteta ponašanja, uključujući antiscenarija. Drugi elementi, nazvani „hladni“ (engl. *cold*) označavaju događaje koji *mogu* da se dese, i oni mogu da se koriste za određivanje kontrolnih struktura poput granjanja i iteracije.

S obzirom na to da je izražajniji od jezika MSC, LSC omogućava bolji uvid u reaktivno ponašanje, odnosno u relacije između inter-objektnog pogleda na postavljene zahteve i intra-objektnog pogleda na implementaciju modela.

3.2.2 Tehnike *Play-in* i *Play-out*

Za kompletiranje rigoroznog ciklusa razvoja sistema potrebno je premostiti jaz između slučajeva upotrebe i mnogo formalnijih jezika koji se koriste da opišu različite scenarije. S obzirom na to da je slučaj upotrebe neformalan i na višem nivou, ne postoji neka generalna tehnika kojom bi se automatski sintetizovao LSC iz slučaja upotrebe, što dovodi do toga da se LSC konstruiše ručno. LSC predstavlja formalni (mada vizuelni) jezik, i njegova izrada zahteva veštinu rada u apstraktnom okruženju i detaljno poznavanje sintakse i semantike jezika. U cilju da sve bude što je moguće više automatizovano, poželjno je da proces kreiranja LSC dijagrama bude prikladniji i prirodniji, i dostupan širokom spektru ljudi.

Ovim problemom se bavio D. Harel u [7], gde je predložio i skicirao pristup rešavanju problema specifikacije ponašanja zasnovanih na scenarijima na višem nivou, nazvajući ih pri tome *play-in scenarijima*. Metodologija podržana od alata nazvanog *Play-Engine* predstavljena je detaljnije u [27]. Osnovna ideja tehnike play-in je da podigne nivo apstrakcije u inženjerskim zahtevima i da omogući rad sa verzijama nalik sistemu koji se razvija. To omogućava i onima koji ne poznaju LSC, ili koji ne žele da rade sa takvim formalnim jezicima direktno, da specificiraju zahteve u ponašanju sistema koristeći intuitivne, korisnicima bliske i razumljive mehanizme na višem nivou. Oni mogu biti eksperti domena, inženjeri aplikacija, inženjeri zahteva, pa čak i potencijalni krajnji korisnici.

Kod tehnike play-in projektant sistema (koji se često zove *korisnik*, ali ga treba razlikovati od krajnjeg korisnika sistema koji se razvija, koji se u literaturi ponekad naziva *akterom*) prvo postavlja stanje gradeći grafički korisnički interfejs sistema bez ponašanja ugrađenog u njega, sa samo osnovnim metodama podržanih od svakog GUI objekta. To je dato u alatu Play-Engine. U sistemima u kojima je bitan raspored skrivenih objekata korisnik može izgraditi grafički prikaz tih objekata. U sistemima koji su „manje GUI“ koriste se dijagrami modela objekata kao GUI. U bilo kojem slučaju, korisnik

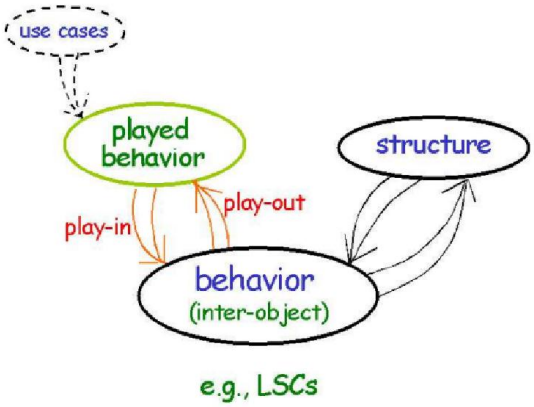
odigrava dolazne događaje putem grafičkog korisničkog interfejsa pritiskom na dugmad i šalje poruke (tj. poziva funkcije) skrivenim objektima na intuitivan tzv. „drag & drop“ način. Kod sistema koji koriste dijagram modela objekta kao interfejs korisnik odabira objekte i/ili metode i parametre. U suštini, Play-Engine radi kontinualno dve stvari: nalaže grafičkom korisničkom interfejsu da pokaže svoj trenutni status korišćenjem grafičkih mogućnosti ugrađenih u njega, i konstruiše odgovarajuću LSC specifikaciju automatski. Play-Engine pita GUI aplikaciju (koja je izgrađena od strane korisnika) za svoju strukturu i metode i interaguje sa njom. Na taj način manipuliše informacijama ubačenim od strane korisnika i kreira i prikazuje odgovarajuću formalnu verziju ponašanja.

Posle odigravanja (play-in) ponašanja (ili dela ponašanja), prirodna stvar je da se proveri da li to tačno odražava ono što je korisnik zamislio. Umesto da se to uradi na konvencionalan način kroz izradu intra-objektnog modela ili prototipa implementacije i izvršavanjem modela da bi ga testirali, play-out pristup omogućava testiranje inter-objektnog ponašanja direktno. Na taj način se mogućnosti metodologije intezivnog odigravanja putem grafičkog korisničkog interfejsa proširuju, tako da postaje moguće ne samo da se odredi i prikaže potrebno ponašanje, već da se ono i testira i potvrdi.

U tehnici play-out, koja je prvi put opisana u [27], korisnik jednostavno odigrava GUI aplikaciju kao što bi se ona izvršavala preko modela sistema ili konačnog sistema, ograničavajući se na akcije krajnjeg korisnika i spoljnog okruženja. Prilikom tog dešavanja, Play-Engine čuva trag o akcijama i inicira dešavanje drugih akcija i događaja, kako je to predviđeno univerzalnim dijagramom u specifikaciji. Takođe, Play-Engine interaguje sa GUI aplikacijom i koristi je da prikaže stanje sistema u svakom trenutku. Ovaj proces, kada korisnik izvršava GUI aplikaciju a Play-Engine je izaziva da reaguje prema specifikaciji, ima efekat rada sa izvršivim modelom.

LSC specifikacija zajedno sa play-in/play-out pristupom može se posmatrati ne samo u domenu sistemskih zahteva, nego i kao njegova konačna implementacija. Umereniji cilj bi bio da se najpre napravi prototip sistema kreiranjem GUI aplikacije, a zatim da se odigra ponašanje umesto da se programira. Dakle, play-in tehnika se koristi za specifikaciju inter-objektnog ponašanja zasnovanog na scenarijima, a play-out predstavlja mehanizam za izvršavanje takvog ponašanja. Play-out je proces testiranja ponašanja sistema na osnovu dobijenih korisničkih akcija, i provere sistemskih odgovora na njih.

Slika 3.2 pokazuje futuristički razvojni ciklus, gde odigrana ponašanja nisu posmatrana samo kao zahtevi, već se stvarno koriste kao jedino ponašanje koje bi trebalo da bude specificirano za sistem.



Slika 3.2: Futuristički razvojni ciklus.

3.3 PM tehnika otkrivanja modela procesa i *α*-algoritam

3.3.1 *Process mining*

Zadatak PM tehnika je otkrivanje, praćenje i poboljšanje modela stvarnih (ne pretpostavljenih) poslovnih procesa, izdvajanjem znanja iz dnevnika događaja koji su dostupni u današnjim informacionim sistemima. PM tehnike upravo polaze od stanovišta da postoji čvrst odnos između modela procesa i "stvarnosti" zabeležene u dnevniku događaja u vidu tragova [3]. Ukoliko se analizom tih tragova može otkriti model ponašanja sistema, onda se taj model može pouzdano koristiti za praćenje, održavanje i modifikaciju sistema.

Ideja otkrivanja modela procesa nije nova. Jedan od najranijih primera korišćenja PM tehnika u kontekstu workflow menadžmenta prikazan je u [9], koji se zasnivao se na workflow grafovima. Cook i Wolf su istraživali slične stvari u kontekstu procesa softverskog inženjerstva, posmatrajući posebno sekvencijalno [10] i paralelno ponašanje procesa [57]. Baveći se sekvencijalnim procesima, u [10] opisuju tri metode za otkrivanje procesa od kojih jedna koristi neuronske mreže, druga je sasvim zasnovana na algoritamskom pristupu, a treća koristi Markovljev pristup. U [57] Cook i Wolf proširuju svoj rad na konkurentne procese, gde predlažu određene mere za otkrivanje modela, ali ne omogućuju pristup za generisanje eksplicitnih modela procesa. Herbst [58], [59], [60] se takođe bavio otkrivanjem modela procesa u kontekstu workflow menadžmenta koristeći induktivan pristup, posmatrajući odvojeno sekvencijalne [60] i paralelne modele [58], [59]. Bitna razlika njegovog rada od ostalih pristupa je što se u njegovim workflow modelima isti zadatak može pojaviti više puta, tj. pristup dopušta duple zadatke. Schimm [61], [62] je izgradio alat pogodan za otkrivanje hijerarhijske strukture workflow procesa.

Mada su se mnogi istraživači bavili idejama discipline PM, najkompletnija sveobuhvatna studija predstavljena je u radovima W.M.P. van der Aalst-a i njegovih saradnika [3], [4], [5], [15], [17], [23], [24], [25], [63 - 69]. Za razliku od njihovog dotadašnjeg rada, istraživanje opisano u [67] je bilo fokusirano na workflow procese sa konkurentim ponašanjem. U [4] i [5] je dat detaljan opis formalizacije tehnika za otkrivanje procesa iz workflow dnevnika događaja, i definisan je *α*-algoritam za izdvajanje modela iz takvih dnevnika događaja. Pregled najbolje prakse i izazova predstavljen je u [70], koji čini rad grupe eksperata oformljene u cilju unapređenja i promocije istraživanja, razvoja i razumevanja PM tehnika kao i njihove implementacije i evolucije.

Za otkrivanje modela procesa iz tragova zapisanih u dnevnicima događaja predložene su mnoge tehnike [4], [5], [8-18]. Mnoge od ovih tehnika koriste Petri mreže [23], [24] u procesu otkrivanja i za predstavljanje dobijenog modela procesa. Međutim, koriste se i drugi vrlo različiti pristupi: heuristika [10], [17], [20], induktivno logičko programiranje [14], regioni zasnovani na stanjima [21], [22], jezički zasnovani regioni [8], [18] i genetički algoritmi [15], [71], [72].

Veliki broj postojećih sistema zapisuju ogromne količine podataka u dnevnicima događaja, ćuvajući na taj način detaljne informacije o svojim aktivnostima. Dnevnici događaja obiluju informacijama kao što su: podaci o vremenu odigravanja aktivnosti, resurs koji je inicirao aktivnost,

podaci koji se dobijaju kao rezultat izvršenja aktivnosti i drugi. Primeri sistema za upravljanje procesima koji proizvode dnevnike događaja sa informacijama o izvršenim aktivnostima su:

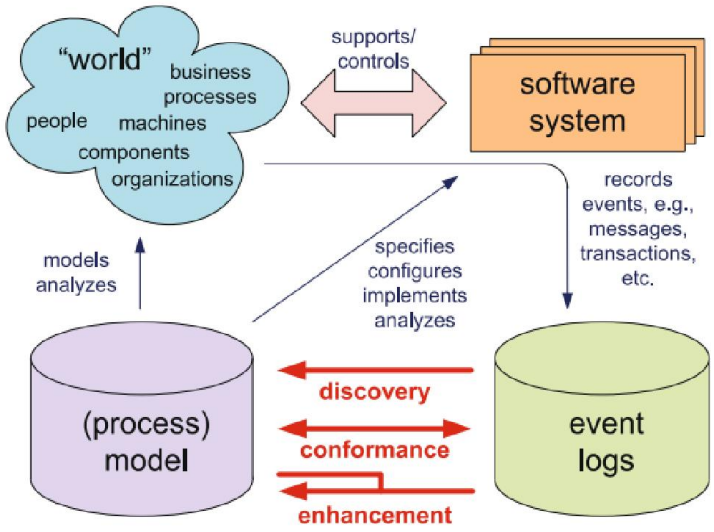
- Klasični sistemi za upravljanje poslovima (Workflow Menagment - WFM): Staffware i COSA.
- Sistemi za upravljanje poslovnim procesima (Bussiness Process Menagment - BPM): BPM| One - Pallas Athena, SmartBPM - Pegasystems, FileNet, Global360, Teamwork - Lombardi Software.
- Sistemi za upravljanje resursima preduzeća (Enterprise Resource Planning - ERP): SAP Buisiness Suite, Oracle E - Business Suite, Microsoft Dynamics NAV.
- Sistemi za upravljanje vezama sa klijentima (Customer Relationship Menagment - CRM): Microsoft Dynamics CRM, SalesForce.
- Infomacioni sistemi u bolnicama: Chipsoft, Siemens Soarian.

U disciplini *process mining* dnevnik događaja se koristi za sprovođenje tri tehnike: *otkrivanje* (engl. *discovery*) [4], [5], [10], [65], [66], [67], *usaglašenost* (engl. *conformance*) [63], [73] i *unapređenje* (engl. *enhancement*) *modela* [3], [64] (slika 3.3). Na slici 3.3 se mogu videti odnosi između stvarnih procesa i njihovih podataka sa jedne i modela procesa sa druge strane.

Tehnikom *otkrivanja* modela poslovnih procesa se iz datog dnevnika događaja kreira model procesa bez bilo kakvih apriori informacija.

Tehnikom *usaglašenost* se postojeći model procesa upoređuje sa dnevnikom događaja istog procesa, kako bi se utvrdilo u kojoj se meri oni podudaraju. Na taj način se mogu utvrditi i locirati eventualne nepravilnosti u sistemu, kao i njihov obim i značaj.

Ideja tehnike *unapređenja* je da se planski proširi ili poboljša postojeći model procesa koristeći informacije o stvarnom procesu zapisane u dnevniku događaja. Unapređenje sistema se može ostvariti *modifikacijom*, kako bi model bolje odražavao stvarnost, ili *proširivanjem*, tj. dodavanjem nove perspektive modelu procesa u korelaciji sa dnevnikom događaja.



Slika 3.3: Mapiranje realnih procesa i događaja u dnevnike događaja i dobijanje modela različitim PM tehnikama.

Tehnika otkrivanja modela je od posebnog interesa za istraživanje prikazano u ovom radu, jer je poslužila kao osnova za ostvarenje zacrtanog cilja istraživanja, a to je interaktivna konstrukcija modela procesa. Tom tehnikom se konstruišu modeli poslovnih procesa samo na osnovu tragova zapisanih u dnevniku događaja.

Da bi dobijeni model bio pravi odraz ponašanja sistema, preduslov je da postoji valjan dnevnik događaja na osnovu kojeg se model kreira. Događaji od interesa su aktivnosti koje se regularno odvijaju u sistemu. One su značajne za analizu upotrebljivosti sistema i vremena potrebnog za njihovo izvršavanje, kao i efekata koje njihovo izvršavanje proizvodi u sistemu. Pri tome se moraju uzeti reprezentativni događaji. To znači da se ne smeju izostaviti važni događaji, jer bi se dobila nerealna predstava o toku procesa, u kom slučaju bi model bio suviše opšti (engl. *underfitting*). Sa druge strane, ne smeju se uzeti ni suviše specifični događaji koji se retko pojavljuju, jer bi sam model koji se dobija oslikavao specifično ponašanje sistema u opštem slučaju, odnosno bio bi isuviše specifičan (engl. *overfitting*). Uzrok oba problema je nedovoljna korelacija između dnevnika događaja i realnog ponašanja sistema.

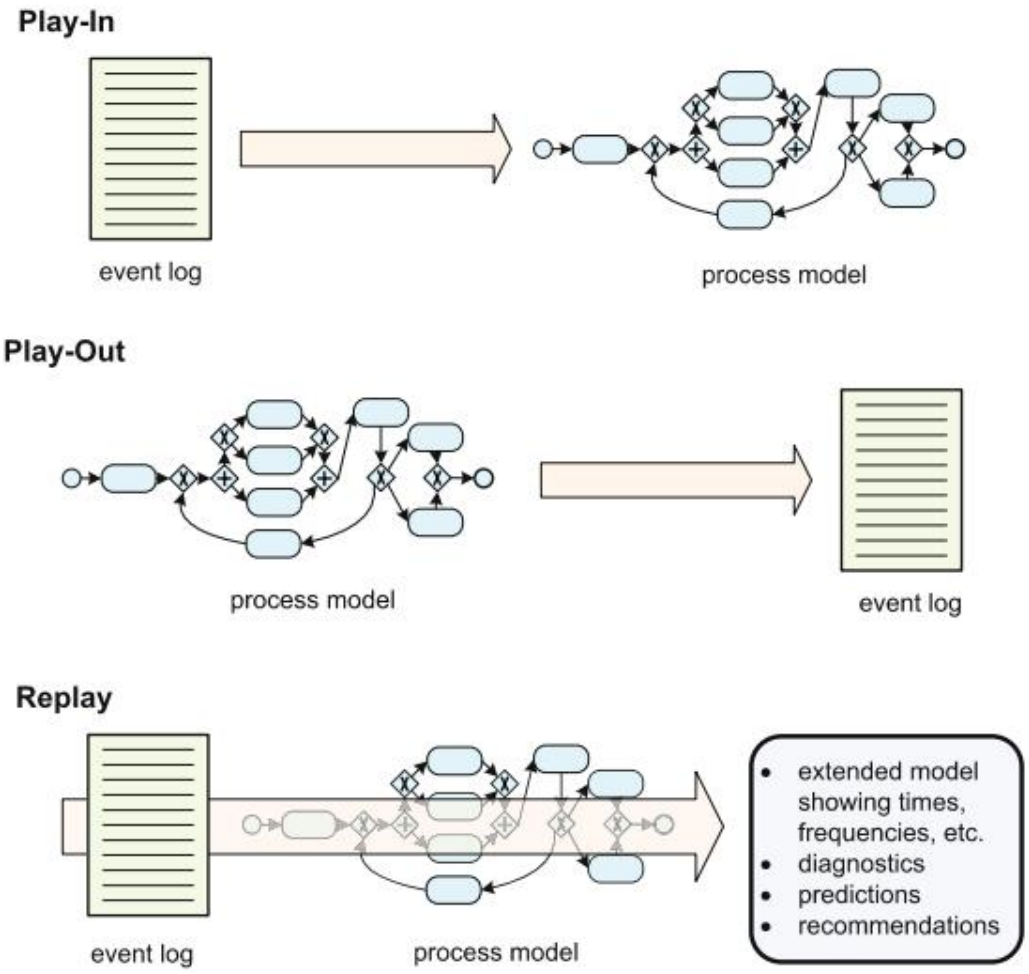
U principu, da bi model procesa bio pravi „predstavnik“ ponašanja procesa zabeleženog u dnevniku događaja, trebalo bi da zadovolji sledeća četiri kriterijuma kvaliteta:

- *sposobnost*: otkriveni model bi trebalo da omogući ponašanje viđeno u dnevniku događaja;
- *preciznost*: otkriveni model ne bi trebalo da omogući ponašanje koje nije u potpunosti vezano za ono što je viđeno u dnevniku događaja;
- *generalizacija*: otkriveni model bi trebalo da generalizuje primer ponašanja viđenog u dnevniku događaja;
- *jednostavnost*: otkriveni model bi trebalo da bude što je moguće jednostavniji.

Kada se terminologija koju koristi D. Harel u kontekstu vizuelnog jezika LSC primeni na disciplinu PM, dobijaju se tehnike kojima se odražava odnos između realnog sistema i njegovog modela:

- *Play-out* tehnika se odnosi na klasično korišćenje modela procesa gde se polazeći od modela (Petri mreže), prateći redosled izvršavanja aktivnosti procesa koji model dozvoljava, dobija dnevnik događaja. Tehnika play-out može biti korišćena i za analizu i za odigravanje poslovnih procesa. Workflow mašina se može posmatrati kao „Play-out mašina“ koja kontroliše slučajeve, dopuštajući samo „pomeranja“ (odigravanja) koja su moguća s obzirom na model. Zbog toga, tehnika play-out može biti korišćena za odigravanje operativnih procesa koristeći neki izvršivi model.
- *Play-in* je tehnika suprotna od play-out, tj. cilj je da se konstruiše model (Petri mreža) na osnovu primera ponašanja zapisanog u dnevniku događaja. Tehnika play-in se često naziva „zaključivanjem“. *α*-algoritam i drugi pristupi otkrivanju procesa su play-in tehnike.
- *Replay* je tehnika gde se i dnevnik događaja i model procesa koriste kao ulaz, tako što se dnevnik „propušta“ kroz model više puta, dobijaju se različiti pogledi na sistem, a potom se proverava podudarnost.

Na slici 3.4 su prikazana tri načina povezanosti dnevnika događaja (ili nekih drugih izvora informacija koji sadrže primere ponašanja) i modela procesa.



Slika 3.4: Tri načina povezanosti dnevnika događaja i modela procesa: *Play-in*, *Play-out* i *Replay*.

3.3.2 Otkrivanje procesa, *α*-algoritam

α-algoritam [3], [4], [5] i modifikovani *α*-algoritam, koji će u ovom radu biti predstavljeni i upoređivani, predstavnici su play-in tehnike. Kod ovih algoritama se na osnovu dnevnika događaja rekonstruiše tok dešavanja u sistemu u vidu Petri mreže.

Pretpostavlja se da je poznat skup aktivnosti poslovnog procesa označen sa $\mathcal{A}$. Cilj modelovanja procesa jeste da se odredi *koje aktivnosti* treba da se izvrše i u *kom mogućem (parcijalnom) redosledu*. Aktivnosti mogu biti izvršene sekvencijalno, opcionalno ili konkurentno, a moguće je i ponavljanje izvršenja nekih aktivnosti.

Model procesa opisuje životni ciklus jednog *slučaja* (engl. *case*) kao instance procesa. Posmatra se dnevnik događaja koji sadrži filtrirane podatke vezane za samo jedan proces. Svaki trag (scenario) u dnevniku pripada jednoj instanci procesa, odnosno jednom izvršavanju posmatranog procesa (slučaju), pri čemu se podrazumeva da se događaji odnose na izvršavanje neke aktivnosti. Događaji koji

pripadaju jednom slučaju su potpuno uređeni, što odgovara vremenskom uređenju njihovog stvarnog odigravanja. Redosled tragova u dnevniku nije bitan, dok redosled događaja u tragu jeste.

Ulaz za α-algoritam je jednostavan dnevnik događaja L, kao multiskup tragova nad nekim skupom aktivnosti $\mathcal{A}$, tj. $L \in \mathbb{B}(\mathcal{A}^*)$ [4], [5]. Svaki trag odgovara jednom odigranom slučaju. Elementi skupa $\mathcal{A}$ su aktivnosti koje odgovaraju tranzicijama u dobijenoj Petri mreži, i obeležavaju se malim slovima (tj. $a, b, c, ... \in \mathcal{A}$), a skupovi aktivnosti se obeležavaju velikim slovima (tj. $A, B, C, ... \subseteq \mathcal{A}$).

Za otkrivanje modela procesa na osnovu dnevnika događaja, kod α-algoritma se razlikuju četiri relacije uređenja zasnovane na dnevniku događaja, koje imaju za cilj da prikažu relevantne obrasce u dnevniku. One su određene sledećom definicijom [4], [5]:

Definicija 3.1. (*Relacije uređenja zasnovane na logu*) Neka je L dnevnik događaja nad $\mathcal{A}$, tj. $L \in \mathbb{B}(\mathcal{A}^*)$. Neka su $a, b \in \mathcal{A}$ neke dve aktivnosti. Tada je, po definiciji:

- $a >_L b$ akko postoji trag $\sigma = \langle t_1, t_2, t_3, \dots, t_n \rangle$ i $i \in \{1, \dots, n-1\}$ tako da $\sigma \in L$ i $t_i = a$ i $t_{i+1} = b$
- $a \rightarrow_L b$ akko $a >_L b$ i nije $b >_L a$
- $a \#_L b$ akko nije $a >_L b$ i nije $b >_L a$
- $a \|_L b$ akko $a >_L b$ i $b >_L a$.

Prema ovim definicijama, za svaki dnevnik događaja L nad $T \subseteq \mathcal{A}$ i bilo koji par aktivnosti $a, b \in T$, važi osobina [4], [5]:

Osobina 3.1. Neka je L dnevnik događaja nad skupom aktivnosti $T \subseteq \mathcal{A}$. Za bilo koje $a, b \in T$ važi: $a \rightarrow_L b$, ili $b \rightarrow_L a$, ili $a \#_L b$, ili $a \|_L b$. Relacije $\rightarrow_L$, $\rightarrow_L^{-1}$, $\#_L$ i $\|_L$ su međusobno isključive i dele $T \times T$, tako da je: $T \times T = \rightarrow_L \cup \rightarrow_L^{-1} \cup \#_L \cup \|_L$.

Pri tome je: $\rightarrow_L = (>_L \setminus (>_L^{-1}))$,

$\rightarrow_L^{-1} = (>_L^{-1} \setminus >_L)$,

$\#_L = (T \times T) \setminus (>_L \cup >_L^{-1})$,

$\|_L = (>_L \cap >_L^{-1})$.

Zbog toga ove relacije mogu biti zapisane u matrici, koja predstavlja „otisak“ (engl. *footprint*) dnevnika događaja, pri čemu su relacije između bilo koje dve aktivnosti $a, b \in T$ definisane na način kako je to predstavljeno Tabelom 3.1.

	$\lceil a > b$	$a > b$
$\lceil b > a$	$a \# b$	$a \rightarrow b$
$b > a$	$b \rightarrow a$	$a \| b$

Tabela 3.1: Tabelarni prikaz definisanja relacija između aktivnosti.

Zasnovan na relacijama definisanim na prikazani način, α-algoritam je opisan sledećom definicijom [3], [4], [5]:

Definicija 3.2. (*α-algoritam*) Neka je L dnevnik događaja nad $T \subseteq \mathcal{A}$. $\alpha(L)$ je definisan kao:

(1) $T_L = \{t \in T \mid (\exists \sigma \in L)\ t \in \sigma\}$

(2) $T_I = \{t \in T \mid (\exists \sigma \in L)\ t = first(\sigma)\}$

(3) $T_O = \{t \in T \mid (\exists \sigma \in L)\ t = last(\sigma)\}$

(4) $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall a \in A)\ (\forall b \in B)(a \rightarrow_L b) \wedge (\forall a_1, a_2 \in A)\ (a_1 \#_L a_2) \wedge (\forall b_1, b_2 \in B)\ (b_1 \#_L b_2)\}$

(5) $Y_L = \{(A, B) \in X_L \mid (\forall (A', B') \in X_L)\ (A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B'))\}$

(6) $P_L = \{p_{(A, B)} \mid (A, B) \in Y_L\} \cup \{i_L, o_L\}$

(7) $F_L = \{(a, p_{(A, B)}) \mid (A, B) \in Y_L \wedge a \in A\} \cup \{(p_{(A, B)}, b) \mid (A, B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$

(8) $\alpha(L) = (P_L, T_L, F_L)$

U koraku (1) formira se skup aktivnosti koje se pojavljuju u dnevniku događaja (T_L). One će odgovarati tranzicijama generisane WF-mreže. T_I je skup početnih aktivnosti, tj. svih aktivnosti koje se pojavljuju kao prve u nekom tragu (korak (2)). T_O je skup krajnjih aktivnosti, tj. svih aktivnosti koje se pojavljuju kao poslednje u nekom tragu (korak (3)). Koraci (4) i (5) čine jezgro α-algoritma. Cilj je konstruisanje mesta imenovanih $p_{(A, B)}$ (korak (6)), tako da je A skup ulaznih tranzicija ($\bullet p_{(A, B)} = A$), a B je skup izlaznih tranzicija ($p_{(A, B)}\bullet = B$) od $p_{(A, B)}$. Kao rezultat dobija se P/T mreža (korak (8)) $\alpha(L) = (P_L, T_L, F_L)$ koja opisuje ponašanje procesa zaključeno na osnovu dnevnika događaja L [3].

α-algoritam je u mogućnosti da otkrije veliku klasu workflow mreža [25] na osnovu ponašanja zapisanog u dnevniku događaja, pod glavnom ograničavajućom petpostavkom da je dnevnik događaja kompletan [4], [5]. Pretpostavka o kompletnosti dnevnika događaja nalaže da se u tragovima zapisanim u njemu nalaze sve relacije direktne sledbenosti koje mogu postojati između aktivnosti posmatranog poslovnog procesa.

Definicija 3.3. (*Kompletan dnevnik događaja*) Neka je $N = (P, T, F)$ ispravna WF-mreža, odnosno $N \in \mathcal{W}$. L je dnevnik događaja od N akko $L \in (T)$, i svaki trag $\sigma \in L$ je okidajuća sekvenca za N koja počinje u stanju $[i]$ i završava se u $[o]$, tj. $(N, [i])[\sigma\rangle\ (N, [o])$. L je *kompletan* dnevnik događaja od N akko:

1) za bilo koji dnevnik događaja L' od N važi $>_{L'} \subseteq >_L$, i

2) za bilo koje $t \in T$ postoji $\sigma \in L$ tako da $t \in \sigma$.

S obzirom na prethodno rečeno, primena α-algoritma na otkrivanje originalne mreže na osnovu kompletnog dnevnika događaja, opisana je sledećom definicijom:

Definicija 3.4. (*Mogućnost ponovnog otkrivanja*) Neka je $N = (P, T, F)$ ispravna WF-mreža, tj. $N \in \mathcal{W}$, i neka je α algoritam koji mapira dnevnike od N na ispravne WF-mreže, tj. $\alpha: \mathcal{P}(T^*) \rightarrow \mathcal{W}$. Ako za bilo koji kompletan dnevnik događaja L od N algoritam vraća N, onda je α algoritam u mogućnosti da ponovo otkrije N.

3.3.3 Odnos relacija i mesta povezivanja

U okviru discipline PM ustanovljene su veze između pojedinih relacija i mesta povezivanja tranzicija u P/T mreži, i formulisane su u vidu teorema [4], [5]. Kod modifikovane PM tehnike otkrivanja procesa dobijene ovim istraživanjem, neke od tih teorema se koriste u izvornom obliku, a neke su prilagođene relacijama i definicijama modifikovane PM tehnike. Stoga će u ovoj sekciji biti predstavljene teoreme koje su od značaja za razumevanje i dokazivanje nekih pojmova, odnosa, definicija, teorema i osobina datih u okviru modifikovane PM tehnike, koja će biti predstavljena kasnije. Dokazi ovih teorema su izostavljeni, a mogu se naći u [5].

Ukoliko se na osnovu dnevnika događaja ustanovi postojanje kauzalne relacije između dve tranzicije, onda postoji mesto u mreži koje povezuje te dve tranzicije.

Teorema 3.1. Neka je $N = (P, T, F)$ ispravna SWF mreža i neka je L kompletan dnevnik događaja od N. Za bilo koje $a, b \in T$: $a \rightarrow_L b$ podrazumeva i nalaže da $a\bullet \cap \bullet b \neq \emptyset$.

Postojanje mesta povezivanja između dve tranzicije „često“ podrazumeva kauzalnu relaciju između njih. Da bi se do ovoga došlo, prvo se definiše odnos mesta povezivanja tranzicija i relacije direktne sledbenosti između njih sledećom teoremom:

Teorema 3.2. Neka je $N = (P, T, F)$ ispravna SWF mreža i neka je L kompletan dnevnik događaja od N. Za bilo koje $a, b \in T$: $a\bullet \cap \bullet b \neq \emptyset$ podrazumeva i nalaže da $a >_L b$.

S obzirom na prethodno navedene teoreme dobija se:

Teorema 3.3. Neka je $N = (P, T, F)$ ispravna SWF mreža i neka je L kompletan dnevnik događaja od N. Za bilo koje $a, b \in T$: $a\bullet \cap \bullet b \neq \emptyset$ i $b\bullet \cap \bullet a = \emptyset$ podrazumeva da $a \rightarrow_L b$.

3.3.4 ProM okruženje

ProM (skraćeno od engl. *Process Mining Framework*) je alat otvorenog koda (engl. *open source*) [74], [75] za otkrivanje modela procesa. Internet prezentacija ovog alata se može naći na

http://www.processmining.org. ProM je zapravo okruženje koje pruža podršku za različite tehnike otkrivanja modela procesa, u vidu učitavanja i parsiranja dnevnika događaja, vizuelizacije rezultata i drugih operacija koje su im zajedničke, dok je svaka tehnika otkrivanja procesa implementirana kao zaseban priključak. Prednost ovakvog pristupa je u jednostavnom dodavanju novog priključka, koji može da implementira funkcionalnost novog algoritma, a sa konforom da se koriste pogodnosti okruženja koje se tiču unosa podataka iz dnevnika događaja, prikaza rezultata u vidu Petri mreže i slično.

Prva funkcionalna verzija ovog alata (ProM 1.1) pojavila se 2004. godine i imala je 29 priključaka. Već verzija ProM 5.2 iz 2009. godine imala je 289 priključaka. Format ulaznog fajla u tim verzijama bio je MXML. Od verzije ProM 6, koja se pojavila 2010. godine, za format ulaznog fajla se koristi XES format, koji je naslednik MXML formata. Korišćenje OpenXES standarda doprinelo je boljoj skalabilnosti i efikasnosti okruženja. Od ove verzije moguće je distribuirano izvršavanje priključaka na više računara. Korisnički interfejs je unapređen da radi istovremeno sa više priključaka, dnevnika događaja i prikaza modela procesa. Priključci su raspoređeni u takozvane pakete. Okruženje poseduje rukovodioca paketima pomoću kojeg se paketi mogu dodati i ukloniti, tako da korisnik raspolaže funkcionalnostima koje su mu zaista potrebne, bez opterećivanja sistema onim nepotrebnim. Naime, kada se uveze dnevnik događaja, korisnik može da izvrši jedan od trenutno instaliranih priključaka koji su za taj dnevnik dostupni. Okruženje se može prilagoditi specifičnim potrebama jedne oblasti rada, ili čak jedne organizacije.

U ovom radu korišćena je verzija ProM 6.2, čiji je prikaz okruženja dat na slici 3.5.

Slika 3.5: Prikaz ProM 6.2 okruženja.

IV Opis rešenja

Iako je originalni α-algoritam u mogućnosti da otkrije veliku klasu WF-mreža, postoje problemi sa kojima se on ne može izboriti: nekompletnost dnevnika događaja, retka ponašanja, složene konstrukcije rutiranja i drugi [3]. Kao posledica toga, nastao je veći broj algoritama koji prevazilaze nedostatke osnovnog α-algoritma [76]. Neki od njih su varijante originalnog α-algoritma, kao što je npr. $\alpha+$ algoritam [77], a drugi koriste potpuno različit pristup, kao što su: *Heuristic mining* [10], *Genetic mining* [15], [71], [72], *Fuzzy mining* [19], proces otkrivanja zasnovan na regionima [8], [18], [21], [22] ili *Flexible Heuristics Miner* [20].

Jedna od osnovnih ograničavajućih pretpostavki α-algoritma jeste da dnevnici događaja na kojima se primenjuje moraju posedovati svojstvo kompletnosti [4], [5]. Pretpostavka o kompletnosti dnevnika događaja zahteva da u tragovima zapisanim u njemu postoje sve direktne zavisnosti relacija koje mogu postojati između aktivnosti posmatranog poslovnog procesa. Osobina kompletnosti često zahteva veliki broj tragova u dnevniku događaja na osnovu kojih može biti konstruisan „reprezentativni“ model ponašanja sistema, zabeleženog u njima. Zbog toga je u sprovedenom istraživanju pravi izazov bio da se pronađu dnevnici događaja sa što je moguće manjim brojem tragova, koji mogu da ne budu kompletni, ali koji su dovoljno validni da se na osnovu evidencije zapisane u takvim dnevnicima odgovarajućim algoritmom mogu dobiti „reprezentativni“ modeli ponašanja sistema. Istraživački rad u tom smeru rezultovao je tzv. *modifikovanom PM tehnikom otkrivanja modela* poslovnog procesa i modifikovanom verzijom α-algoritma (nazvanoj $\alpha^{||}$*-algoritam*), kao i novim vrstama dnevnika događaja, tzv. *kauzalno kompletni* i *slabo kompletni* dnevnici, što će u nastavku biti prikazano.

4.1 Modifikovana PM tehnika za otkrivanje modela procesa

Sprovedeno istraživanje je bilo usmereno ka jednom od najvećih izazova u procesu otkrivanja modela poslovnih procesa na osnovu primera ponašanja zapisanog u dnevniku događaja, a to je problem *kompletnosti* dnevnika događaja. Kod osnovnog α-algoritma pretpostavlja se da je dnevnik događaja *kompletan* s obzirom na relaciju uređenja $>_L$. Pretpostavka o kompletnosti dnevnika L nalaže da ako model procesa koji se otkriva dozvoljava da aktivnost b može da sledi iza aktivnosti a, onda se u dnevniku mora nalaziti bar jedan trag u kome je b neposredno iza a, odnosno mora da važi $a >_L b$. U ovom slučaju se kaže da *b direktno* (ili *neposredno*) *sledi iza a*.

S obzirom na to da postojanje takvih relacija u dnevniku događaja utiče na definisanje relacija paralelnosti ($a \|_L b$ akko $a >_L b$ i $b >_L a$), cilj je bio da se proveri kako postojanje relacija indirektne sledbenosti utiče na zaključivanje relacija paralelnosti, i kako se to odražava na efikasnost otkrivanja modela poslovnog procesa. Zbog toga je uvedena relacija *indirekcije* kao još jedna osnovna relacija između aktivnosti procesa koja se može ustanoviti iz evidencije zapisane u dnevniku događaja.

Rezultati istraživanja su pokazali da se uopštavanjem relacije konkurentnosti s obzirom na indirektne sledbenosti između aktivnosti zapisanih u dnevniku događaja, relacija konkurentnosti, a samim tim i model poslovnog procesa, može dobiti iz manjih dnevnika, što omogućava brže otkrivanje modela poslovnog procesa i iz nekompletnih dnevnika događaja. Nekompletni dnevnici događaja iz kojih se mogu proizvesti modeli procesa mogu biti prilično oskudni u pogledu broja tragova koji je neophodan da bi se mogao dobiti validan model, ali moraju imati određena svojstva kako bi ispunili definisane uslove koji će biti predstavljeni. To daje posebno dobre rezultate kod procesa sa puno aktivnosti koje se mogu obaviti u međusobno nezavisnom redosledu, tj. konkurentno.

4.1.1 Relacije uređenja kod modifikovane PM tehnike otkrivanja modela procesa

Uvođenje relacije indirekcije, kao još jedne osnovne relacije između aktivnosti zapisanih u dnevniku događaja, dovelo je do znatno bržeg otkrivanja relacija konkurentnosti između aktivnosti. Osnovna ideja je u sledećem: da bi se zaključilo da su neke dve aktivnosti a i b nezavisne, odnosno konkurentne, dovoljno je da se u dnevniku nalazi neki trag u kome a direktno ili indirektno prethodi b, i neki drugi trag u kome b direktno ili indirektno prethodi a. Kod osnovnog α-algoritma, da bi se zaključila relacija konkurentnosti između a i b, u dnevniku moraju da postoje tragovi u kome su a i b neposredno jedan iza drugog u oba poretka. Ukoliko bi u dnevniku bili samo tragovi u kojima su a i b samo indirektno jedan iza drugog, dnevnik bi bio nekompletan i α-algoritam, zasnovan na relacijama uređenja datim definicijom 3.1, ne bi mogao da otkrije model (tačnije, model koji bi se dobio primenom ovog algoritma ne bi bio korektan u opštem slučaju).

Iz tog razloga uvedena je nova osnovna relacija $a >>_L b$, koja označava da *b indirektno sledi iza a*, a koja je definisana na sledeći način: $a >>_L b$ akko u L postoji trag $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ i $i + 2 \leq j$

gde $i, j \in \{1, \ldots, n\}$ tako da $\sigma \in L$ i $t_i = a$ i $t_j = b$, i nije $a >_L b$.

Uvođenjem ove relacije, broj relacija uređenja zasnovanih na dnevniku događaja, kod modifikovane PM tehnike otkrivanja modela procesa, povećao se na šest i došlo je do promena u definisanju poslednje tri relacije iz definicije 3.1, što je prikazano sledećom definicijom:

Definicija 4.1. (*Relacije uređenja zasnovane na dnevniku događaja kod modifikovane PM tehnike otkrivanja procesa*) Neka je L dnevnik događaja nad $\mathcal{A}$, tj. $L \in \mathbb{B}(\mathcal{A}^*)$. Neka su $a, b \in \mathcal{A}$ neke dve aktivnosti. Tada, po definiciji:

- $a >_L b$ akko postoji trag $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ i $i \in \{1, \ldots, n-1\}$ tako da $\sigma \in L$ i $t_i = a$ i $t_{i+1} = b$
- $a >>_L b$ akko postoji trag $\sigma = \langle t_1, t_2, t_3, \ldots, t_n \rangle$ i postoje $i, j \in \{1, \ldots, n\}$ takvi da je $i + 2 \leq j$, gde $\sigma \in L$ i $t_i = a$ i $t_j = b$, i nije $a >_L b$.
- $a \rightarrow_L b$ akko $a >_L b$ i nije $b >_L a$ i nije $b >>_L a$
- $a \Rightarrow_L b$ akko $a >>_L b$ i nije $b >_L a$ i nije $b >>_L a$
- $a \#_L b$ akko nije $a >_L b$ i nije $b >_L a$, i nije $a >>_L b$ i nije $b >>_L a$
- $a \|_L b$ akko $a >_L b$ i $b >_L a$, ili $a >_L b$ i $b >>_L a$, ili $a >>_L b$ i $b >_L a$, ili $a >>_L b$ i $b >>_L a$

Prema ovim definicijama, za svaki dnevnik događaja L nad $T \in \mathcal{A}$, i bilo koji par aktivnosti $a, b \in T$, važi:

Osobina 4.1. Neka je L dnevnik događaja nad T. Za bilo koje $a, b \in T$ važi: $a \rightarrow_L b$, ili $b \rightarrow_L a$, ili $a \Rightarrow_L b$, ili $b \Rightarrow_L a$, ili $a \#_L b$, ili $a \|_L b$. Relacije $\rightarrow_L$, $\rightarrow_L^{-1}$, $\Rightarrow_L$, $\Rightarrow_L^{-1}$, $\#_L$ i $\|_L$ su međusobno isključive i dele $T \times T$.

Pri čemu su relacije: $>_L^{-1}$, $>>_L^{-1}$ i $\Rightarrow_L^{-1}$ inverzne relacije, definisane slično kao što je to dato za $\rightarrow_L^{-1}$ prethodno u sekciji 3.3.2:

$>_L^{-1}$ je inverzna relacija relaciji $>_L$, tj. $>_L^{-1} = \{(y, x) \in T \times T \mid x >_L y\}$

$>>_L^{-1}$ je inverzna relacija relaciji $>>_L$, tj. $>>_L^{-1} = \{(y, x) \in T \times T \mid x >>_L y\}$

$\rightarrow_L^{-1}$ je inverzna relacija relaciji $\rightarrow_L$, tj. $\rightarrow_L^{-1} = \{(y, x) \in T \times T \mid x \rightarrow_L y\}$

$\Rightarrow_L^{-1}$ je inverzna relacija relaciji $\Rightarrow_L$, tj. $\Rightarrow_L^{-1} = \{(y, x) \in T \times T \mid x \Rightarrow_L y\}$

S obzirom na to, definisane relacije se mogu prikazati u obliku:

$\rightarrow_L = (>_L \setminus (>_L^{-1} \cup >>_L^{-1}))$,

$\rightarrow_L^{-1} = (>_L^{-1} \setminus (>_L \cup >>_L))$,

$\Rightarrow_L = (>>_L \setminus (>_L^{-1} \cup >>_L^{-1}))$,

$\Rightarrow_L^{-1} = (>>_L^{-1} \setminus (>_L \cup >>_L))$,

$\#_L = (T \times T) \setminus (>_L \cup >_L^{-1} \cup >>_L \cup >>_L^{-1})$,

$\|_L = ((>_L \cap >_L^{-1}) \cup (>_L \cap >>_L^{-1}) \cup (>_L^{-1} \cap >>_L) \cup (>>_L \cap >>_L^{-1}))$

zbog čega se dobija da je: $T \times T = \rightarrow_L \cup \rightarrow_L^{-1} \cup \Rightarrow_L \cup \Rightarrow_L^{-1} \cup \#_L \cup \|_L$.

Definisane relacije mogu biti zapisane u matrici, koja predstavlja *otisak* (engl. *footprint*) dnevnika događaja, pri čemu su relacije između bilo koje dve aktivnosti a, $b \in T$, kod modifikovane PM tehnike otkrivanja procesa, definisane na način kako je to predstavljeno Tabelom 4.1.

	$\urcorner a > b, \urcorner b >> a$	$\urcorner a > b, b >> a$	$a > b, \urcorner b >> a$	$a > b, b >> a$
$\urcorner b > a, \urcorner a >> b$	$a \# b$	$b \Rightarrow a$	$a \rightarrow b$	$a \parallel b$
$\urcorner b > a, a >> b$	$a \Rightarrow b$	$a \parallel b$	N/A[2]	N/A
$b > a, \urcorner a >> b$	$b \rightarrow a$	N/A	$b \parallel a$	N/A
$b > a, a >> b$	$b \parallel a$	N/A	N/A	N/A

Tabela 4.1: Tabelarni prikaz definisanja relacija između aktivnosti kod modifikovane PM metode otkrivanja procesa.

Zbog međusobne isključivosti relacija, kako je to dato osobinom 4.1, u bilo kojem polju *otiska* može se naći samo jedna od relacija definisanih Tabelom 4.1, što će se moći videti i u primeru koji će kasnije biti prikazan.

4.1.2 Paralelni procesi i blok-strukturirani modeli paralelnih procesa

Konkurentnost poslovnog procesa podrazumeva potencijalno paralelno izvršavanje poslovnih aktivnosti u okviru izvršenja jednog slučaja procesa. Zaključivanje relacija konkurentnosti iz tragova zapisanih u dnevniku događaja je jedan od ključnih elemenata za pronalaženje modela procesa, posebno kod procesa sa velikim brojem aktivnosti koje se mogu obavljati konkurentno (potencijalno paralelno). Na primer, ako u nekom poslovnom procesu postoji m nezavisnih aktivnosti koje se mogu obavljati paralelno, broj različitih tragova u dnevniku događaja u kojima se one mogu pojaviti u međusobno nezavisnom redosledu je $m!$. Da bi se zaključile sve relacije konkurentnosti primenom α-algoritma potrebno je da u dnevniku događaja postoji najmanje $m(m-1)$ različitih tragova [3]. Iako je broj potrebnih tragova u dnevniku događaja u odnosu na broj mogućih tragova znatno smanjen kod korišćenja α-algoritma, on je još uvek relativno veliki. Svako smanjenje broja tragova u dnevniku događaja neophodnih za zaključivanje relacija paralelnosti, dovodi do efikasnijeg pronalaženja modela i to na osnovu dnevnika događaja koji ne mora biti kompletan, što je upravo modifikovanom PM tehnikom otkrivanja modela omogućeno.

Za potrebe ilustracije efekta primene modifikovane PM tehnike na mogućnost i brzinu otkrivanja modela procesa, definisana je posebna klasa poslovnih procesa, a to su *paralelni procesi*.

[2] N/A is „not applicable“

Definicija 4.2. (*Paralelan proces*) Paralelan proces $N^{\|} = (P, T, F)$ je ispravna WF-mreža tj. $N^{\|} \in \mathcal{W}$, gde se kod izvršavanja svakog mogućeg pojedinačnog slučaja procesa koji se završava markiranjem izlaznog mesta, svaka aktivnost iz skupa $t_i \in T$ izvršava jednom i samo jednom.

Uslov iz definicije 4.2 da se svaka aktivnost iz $t_i \in T$ mora izvršiti, nalaže da se svaka aktivnost paralelnog poslovnog procesa mora pojaviti u svakom tragu $\sigma \in L$. Kao posledica toga, između aktivnosti paralelnog procesa ne postoji alternativnost, odnosno ne postoji relacija $a_1 \#_L a_2$ (pri čemu je $a_1 \neq a_2$), zbog čega se kod paralelnih procesa aktivnosti izvršavaju sekvencijalno ili paralelno, ali ne i opcionalno.

Ograničenje iz definicije 4.2 da se svaka aktivnost iz $t_i \in T$ može izvršiti najviše jedanput eliminiše mogućnost pojave iteracija u modelu paralelnog procesa.

S obzirom na definiciju 4.2 i prethodno rečeno, dolazi se do sledeće teoreme:

Teorema 4.1. Neka je L dnevnik događaja nad $T \subseteq \mathcal{A}$, i neka su $a, b, t \in T$ neke aktivnosti takve da $a \neq b \neq t$. Kod paralelnih procesa važi:

1. Ako $a, b \in T$, onda za bilo koje $a \in T$ ne postoji $b \in T$ takvo da je $a \neq b$ i $a \#_L b$.
2. Ako $a, b, t \in T$: $a \rightarrow_L t$, $b \rightarrow_L t$ onda je $a \|_L b$.
3. Ako $a, b, t \in T$: $t \rightarrow_L a$, $t \rightarrow_L b$ onda je $a \|_L b$.

Dokaz:

1. Ova tvrdnja se zasniva na uslovu iz definicije 4.2 da se svaka aktivnost iz $t_i \in T$ mora izvršiti, što nalaže da se svaka aktivnost paralelnog poslovnog procesa mora pojaviti u svakom tragu $\sigma \in L$, i na definiciji 4.1 kojom su definisane relacije između aktivnosti.
2. Pretpostavimo da $a \rightarrow_L t$, $b \rightarrow_L t$ i da nije $a \|_L b$. Treba dokazati da ovo dovodi do kontradikcije. Ako u nekom tragu dnevnika događaja $a \rightarrow_L t$, onda prema definiciji 4.2 da se svaka aktivnost izvršava jednom i samo jednom, između a i b u tom tragu može postojati jedna od sledećih relacija: $a >>_L b$, $b >_L a$ ili $b >>_L a$. Ako u nekom drugom tragu tog istog dnevnika $b \rightarrow_L t$, onda prema definiciji 4.2 između a i b u tom tragu može postojati jedna od sledećih relacija: $b >>_L a$, $a >_L b$ ili $a >>_L b$. Posmatrajmo odvojeno moguće slučajeve:

   - ako $a >_L b$ i $b >_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$.

   - ako $a >_L b$ i $b >>_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$. Isto važi i za slučaj $a >>_L b$ i $b >_L a$.

   - ako $a >>_L b$ i $b >>_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$.

   - ako $a >>_L b$ a u nekom drugom tragu $a >_L b$ onda, s obzirom na to da nema relacija odigranih u suprotnom smeru, prema definiciji 4.1 $a \rightarrow_L b$. Ako $a \rightarrow_L t$ i $a \rightarrow_L b$, s obzirom da aktivnost a nosi samo jedan token to bi značilo da postoji opciono izvršavanje $a \rightarrow_L t$ ili $a \rightarrow_L b$ što je u suprotnosti

sa definicijom 4.2 prema kojoj kod paralelnih procesa nema opcionog izvršavanja. Isto važi i za slučaj $b >>_L a$ odnosno $b >_L a$.

- ako $a >>_L b$ onda u slučaju kada $a \rightarrow_L t$ postojala bi neka od relacija $t >_L b$ ili $t >>_L b$, što bi uz $b \rightarrow_L t$ prema definiciji 4.1 značilo da je $t \|_L b$, a to je u suprotnosti sa polaznom pretpostavkom da $b \rightarrow_L t$. Slično važi i u slučaju kada $b >>_L a$.

3. Pretpostavimo da $t \rightarrow_L a$, $t \rightarrow_L b$ i da nije $a \|_L b$. Treba dokazati da ovo dovodi do kontradikcije. Ako u nekom tragu dnevnika događaja $t \rightarrow_L a$, onda, prema definiciji 4.2 da se svaka aktivnost izvršava jednom i samo jednom, između a i b u tom tragu može postojati jedna od sledećih relacija: $a >_L b$, $a >>_L b$ ili $b >>_L a$. Ako u nekom drugom tragu tog istog loga $t \rightarrow_L b$, onda prema definiciji 4.2 između a i b u tom tragu može postojati jedna od sledećih relacija: $b >_L a$, $b >>_{Lc} a$ ili $a >>_L b$. Posmatrajmo odvojeno moguće slučajeve:

   - ako $a >_L b$ i $b >_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$.

   - ako $a >_L b$ i $b >>_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$. Isto važi i za slučaj $a >>_L b$ i $b >_L a$.

   - ako je u nekom tragu $a >_L b$ a u nekom drugom tragu $a >>_L b$ onda, s obzirom na to da nema relacija odigranih u suprotnom smeru, prema definiciji 4.1 $a \rightarrow_L b$. Kako $t \rightarrow_L b$ i kako aktivnost b može da primi token samo od jedne aktivnosti, to bi značilo opciono izvršavanje $a \rightarrow_L b$ ili $t \rightarrow_L b$, što je u suprotnosti sa definicijom 4.2 prema kojoj kod paralelnih procesa nema opcionog izvršavanja. Isto važi i za slučaj kada je $b >>_L a$ odnosno $b >_L a$.

   - ako $a >>_L b$ i $b >>_L a$ onda je prema definiciji 4.1 $a \|_L b$, što je u suprotnosti sa početnom pretpostavkom da nije $a \|_L b$.

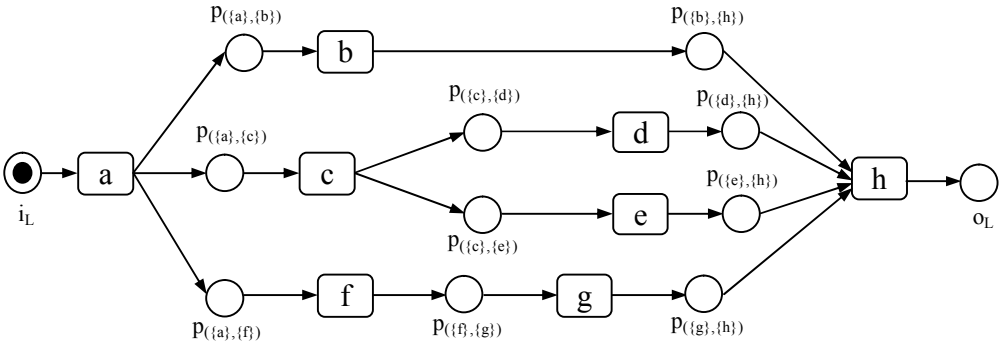
   - ako $a >>_L b$ onda zbog $t \rightarrow_L b$ postojala bi neka od relacija $a >_L t$ ili $a >>_L t$, što bi uz $t \rightarrow_L a$ prema definiciji 4.1 značilo da je $t \|_L a$, a to je u suprotnosti sa polaznom pretpstavkom da $t \rightarrow_L a$. Slično važi i u slučaju kada $b >>_L a$.

Blok-strukturirani modeli paralelnih procesa

Blok-strukturirana WF-mreža je hijerarhijska WF-mreža koja može rekurzivno biti podeljena na delove koji imaju jednu ulaznu i jednu izlaznu tačku. Stablo procesa je kompaktan apstraktan prikaz blok-strukturirane WF-mreže: ukorenjeno stablo u kojem su listovi označeni aktivnostima a svi ostali čvorovi su označeni operatorima [78]. S obzirom na definisanje blok-strukturiranih WF-mreža i osobina paralelnih procesa datih definicijom 4.2, modeli procesa koji se otkrivaju modifikovanom PM tehnikom čine podvrstu blok-strukturiranih modela procesa koji su nazvani *blok-strukturiranim modelima paralelnih procesa*. To su modeli procesa sa dva operatora: $\rightarrow$ koji opisuje sekvencijalnu kompoziciju, i $\wedge$ koji opisuje paralelnu kompoziciju.

Na slici 4.1 dat je primer blok-strukturiranog modela paralelnog poslovnog procesa predstavljenog u obliku P/T mreže, koji predstavlja demonstrativni primer u ovom radu za potrebe

prikazivanja prednosti korišenja modifikovane PM tehnike i modifikovanog algoritma u otkrivanju modela paralelnih procesa iz nekompletnih dnevnika događaja.



Slika 4.1: Primer blok-strukturiranog modela paralelnog poslovnog procesa kojem odgovara stablo procesa: $\rightarrow (a, \wedge (b, \rightarrow (f, g), \rightarrow (c, \wedge (d, e))), h)$.

4.1.3 $\alpha^{\|}$-algoritam

S obzirom na to da se α-algoritam zasniva na relacijama $\#_L$ i $\rightarrow_L$ [3], [4], [5], gubitak relacije $\#_L$ između različitih aktivnosti će dovesti do promene i u samom α-algoritmu u njegovoj primeni na otkrivanju modela paralelnih poslovnih procesa. Nedostatak relacije $\#_L$ između različitih aktivnosti paralelnog poslovnog procesa direktno utiče na definisanje koraka (4) u definiciji 3.2, koji čini jezgro α-algoritma, kao i na korak (5) koji je zasnovan na definiciji koraka (4). Gubitak relacije $\#_L$ između različitih aktivnosti relativno malo utiče direktno na definicije koraka (6), (7) i (8) u definiciji 3.2, ali znatno utiče na rezultate dobijene izvršavanjem pomenutih koraka, s obzirom na to da oni zavise od rezultata izvršavanja koraka (4) i (5), kod kojih je došlo do promena. Koraci (1), (2) i (3) ostaju nepromenjeni. Promene u α-algoritmu do kojih dolazi njegovom primenom na paralelne procese, definisane su lemama koje slede.

Lema 4.1. Kod paralelnih procesa ne postoje relacije $a_1 \#_L a_2$ ni $b_1 \#_L b_2$ definisane u koraku (4) α-algoritma, gde je $a_1 \neq a_2$ i $b_1 \neq b_2$.

Dokaz. Kod mesta u WF-mreži označenog sa $p_{(A, B)}$, A je skup ulaznih tranzicija ($\bullet p_{(A, B)} = A$) i $a_1, a_2 \in A$, a B je skup izlaznih tranzicija ($p_{(A, B)} \bullet = B$), i $b_1, b_2 \in B$. Kako kod paralelnih procesa između bilo koje dve tranzicije iz skupa tranzicija T ne postoji relacija $\#_L$, onda relacija $\#_L$ ne postoji ni između tranzicija ulaznog (odnosno izlaznog) skupa.

Lema 4.2. Za paralelne procese važi:
$X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall a \in A)\ (\forall b \in B)\ (a \rightarrow_L b)\}$.

Dokaz. Kod α-algoritma korak (4) izgleda: $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall a \in A)\ (\forall b \in B)\ (a \rightarrow_L b) \wedge (\forall a_1, a_2 \in A)\ (a_1 \#_L a_2) \wedge (\forall b_1, b_2 \in B)\ (b_1 \#_L b_2)\}$. Kako kod paralelnih procesa s obzirom na lemu 4.1 ne postoji $a_1 \#_L a_2$ ni $b_1 \#_L b_2$, članovi koji su definisani na osnovu tih relacija se gube iz izraza za X_L.

Lema 4.3. Za paralelne procese važi: $Y_L = X_L$.

Dokaz. Kod α-algoritma korak (5) izgleda: $Y_L = \{(A, B) \in X_L \mid (\forall(A', B') \in X_L)\ (A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B'))\}$, gde $(A', B') \in X_L$ može biti nemaksimalan par (dobijen na osnovu relacije $a \rightarrow_L b$), ili maksimalan par (dobijen na osnovu relacija $a_1 \#_L a_2$ i/ili $b_1 \#_L b_2$). $(A, B) \in Y_L$ su maksimalni parovi izdvojeni iz X_L. S obzirom da kod paralelnih procesa ne postoji $a_1 \#_L a_2$ ni $b_1 \#_L b_2$, ostaju samo parovi (A', B') koji su dobijeni na osnovu relacije $(a \rightarrow_L b)$, koji su na taj način i maksimalni parovi, tj. (A, B). Kako je $A = A'$i $B = B'$, to je i $((A, B) \in Y_L) = ((A', B') \in X_L)$, odnosno $Y_L = X_L$.

Kako kod paralelnih procesa, prema lemi 4.3, važi da je $Y_L = X_L$, to za ove procese α-algoritam ima jedan korak manje. Taj modifikovani α-algoritam obeležen je sa $\alpha^{\|}(L)$, gde „$^{\|}$“ oslikava svojstvo da se algoritam odnosi na paralelne poslovne procese. S obzirom na prethodno izvršene modifikacije, $\alpha^{\|}$-algoritam se može opisati sledećom definicijom:

Definicija 4.3 ($\alpha^{\|}(L)$ *algoritam*) Neka je L dnevnik događaja nad $T \subseteq \mathcal{A}$. $\alpha^{\|}(L)$ je definisan kao:
(1) $T_L = \{t \in T \mid (\exists\sigma \in L)\ t \in \sigma\}$
(2) $T_I = \{t \in T \mid (\exists\sigma \in L)\ t = first(\sigma)\}$
(3) $T_O = \{t \in T \mid (\exists\sigma \in L)\ t = last(\sigma)\}$
(4) $X_L = \{(A, B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge (\forall a \in A)\ (\forall b \in B)\ (a \rightarrow_L b)\}$
(5) $P_L = \{p_{(A, B)} \mid (A, B) \in X_L\} \cup \{i_L, o_L\}$
(6) $F_L = \{(a, p_{(A, B)}) \mid (A, B) \in X_L \wedge a \in A\} \cup \{(p_{(A, B)}, b) \mid (A, B) \in X_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\}$
$\cup\ \{(t, o_L) \mid t \in T_O\}$
(7) $\alpha^{\|}(L) = (P_L, T_L, F_L)$

Primena $\alpha^{\|}(L)$ algoritma na otkrivanje modela paralelnog procesa opisana je sledećom definicijom:

Definicija 4.4. (*Mogućnost ponovnog otkrivanja modela paralelnog procesa*) Neka je $N^{\|} = (P, T, F)$ paralelan proces, i neka je $\alpha^{\|}(L)$ algoritam koji mapira dnevnike od $N^{\|}$ na ispravne WF-mreže, tj. $\alpha^{\|}$: $\mathcal{P}(T^*) \rightarrow \mathcal{W}$. Ako za bilo koji kompletan dnevnik događaja L od $N^{\|}$, $\alpha^{\|}$-algoritam vraća $N^{\|}$, onda je $\alpha^{\|}$-algoritam u mogućnosti da ponovo otkrije $N^{\|}$.

4.2 Otkrivanje modela paralelnih poslovnih procesa na osnovu kauzalno kompletnih dnevnika događaja

Modifikacija PM tehnike uvođenjem novih relacija i redefinisanjem postojećih relacija prema definiciji 4.2, dovodi do mogućnosti otkrivanja blok-strukturiranih modela paralelnih poslovnih procesa na osnovu dnevnika događaja koji ne ispunjavaju uslov kompletnosti definisan u [3], [4], [5]. Za razliku od originalne PM tehnike otkrivanja modela procesa, gde se uslov kompletnosti zasniva na relaciji $>_L$, uslov kompletnosti kod modifikovane PM tehnike otkrivanja modela vezuje se za relaciju $\rightarrow_L$, i u skladu sa time definiše se novi oblik kompletnosti dnevnika događaja: *kauzalna kompletnost*.

4.2.1 Kauzalno kompletan dnevnik događaja

Za dnevnik događaja se kaže da je kauzalno kompletan (engl. *causally complete*) L_c, ukoliko sve aktivnosti koje se potencijalno mogu izvršiti neposredno jedna iza druge u samo jednom smeru zapravo kauzalno slede jedna iza druge u nekom tragu u dnevniku.

Kako je već rečeno, uslov kompletnosti dnevnika događaja [3], [4], [5] definisan kod α-algoritma, nalaže da sve aktivnosti koje se u posmatranoj mreži potencijalno mogu izvršiti neposredno jedna iza druge, zapravo, slede neposredno jedna iza druge u nekom tragu kompletnog dnevnika događaja. Stoga je relacija $>_L$, koja oslikava neposredno izvršavanje aktivnosti u mreži, dobijena iz bilo kojeg kompletnog dnevnika događaja za jednu mrežu ista. Kako se iz relacije $>_L$, izdvajanjem elemenata koji pripadaju relaciji paralelnosti $\|_L$, dobija relacija kauzalnosti $\rightarrow_L$, to se iz istih relacija $>_L$ za bilo koji kompletan dnevnik događaja jedne mreže dobija ista relacija kauzalnosti $\rightarrow_L$. Takva relacija kauzalnosti mreže naziva se *bazičnom kauzalnom relacijom*, i obeležava se sa $\rightarrow^B_N$.

Definicija 4.5. (*Bazična kauzalna relacija*) Neka je $N = (P, T, F)$ ispravna WF-mreža, odnosno $N \in \mathcal{W}$, i neka je L kompletan dnevnik događaja od N. $\rightarrow^B_N$ je *bazična kauzalna relacija* mreže N akko je $\rightarrow^B_N = \rightarrow_L$.

S obzirom na ovako definisanu bazičnu kauzalnu relaciju, kauzalno kompletan dnevnik događaja je definisan na sledeći način.

Definicija 4.6. (*Kauzalno kompletan dnevnik događaja*) Neka je $N = (P, T, F)$ ispravna WF-mreža, odnosno $N \in \mathcal{W}$, i neka je $\rightarrow^B_N$ bazična kauzalna relacija. L_c je kauzalno kompletan dnevnik događaja od N akko:

1) $\rightarrow_{Lc} = \rightarrow^B_N$, i

2) za bilo koje $t \in T$ postoji $\sigma \in L_c$ tako da $t \in \sigma$.

Na osnovu ovako definisanog dnevnika događaja L_c može se dobiti originalna mreža paralelnog poslovnog procesa, pa stoga važi sledeća definicija:

Definicija 4.7. (*Mogućnost ponovnog otkrivanja modela paralelnog procesa na osnovu* L_c) Neka je $N^{||} = (P, T, F)$ paralelan proces, i neka je $\alpha^{||}$ algoritam koji mapira logove od $N^{||}$ na ispravne WF-mreže, tj. $\alpha^{||}$: $\mathcal{P}(T^*) \rightarrow \mathcal{W}$. Ako za bilo koji kauzalno kompletan dnevnik događaja L_c od $N^{||}$, $\alpha^{||}$ algoritam vraća $N^{||}$, onda je $\alpha^{||}$ u mogućnosti da ponovo otkrije $N^{||}$.

Kako se elementi relacije $\rightarrow_L$ dobijaju iz $>_L$, izuzimanjem iz $>_L$ elemenata koji pripadaju relaciji $||_L$, i kako se kod originalne PM tehnike elementi relacije $||_L$ mogu otkriti samo iz kompletnog dnevnika događaja (jer je prema definiciji 3.1 $a \,||_L\, b$ samo ako $a >_L b$ i $b >_L a$), to znači da se kod originalne PM tehnike $\rightarrow^B_N$ može dobiti samo iz kompletnog dnevnika događaja. Za razliku od toga, kod modifikovane PM tehnike, korišćenjem relacija datih definicijom 4.1, elementi relacije $||_L$ se mogu otkriti iz relacija $>_L$ i $>>_L$ (jer je $a \,||_L\, b$ ako $a >_L b$ i $b >_L a$, ili $a >_L b$ i $b >>_L a$, ili $b >_L a$ i $a >>_L b$, ili $a >>_L b$ i $b <<_L a$), tako da se elementi relacije $||_L$ mogu otkriti i iz nekompletnog dnevnika događaja s obzirom na uslov kompletnosti postavljen u [3], [4], [5]. Na taj način, izuzimanjem elemenata relacije $||_L$ iz $>_L$ može se dobiti $\rightarrow_L = \rightarrow^B_N$ i kod nekompletnog dnevnika događaja, upravo zbog mogućnosti zaključivanja konkurentnosti i bez odigravanja svih relacija tipa $a >_L b$ i $b >_L a$, koji je u tom slučaju kauzalno kompletan dnevnik događaja L_c. Primeri su pokazali da kod bilo koje mreže do iste bazične kauzalne relacije $\rightarrow^B_N$ se može doći iz različitih kauzalno kompletnih dnevnika događaja sa različitim tragovima, koji imaju različite relacije $>_{Lc}$. Pri tome, što se veći broj elemenata u relaciji $||_{Lc}$ može dobiti iz relacije $>>_{Lc}$, to je manji broj elemenata neophodan u relaciji $>_{Lc}$, odnosno, kauzalno kompletan dnevnik događaja može biti oskudniji u pogledu broja tragova.

4.2.2 Primer otkrivanja originalne mreže iz kauzalno kompletnog dnevnika događaja

Rezultati istraživanja su pokazali da se do otkrivanja originalne mreže predstavljenom modifikovanom PM tehnikom može doći iz bilo kog nekompletnog dnevnika događaja kod kojeg je $\rightarrow_L = \rightarrow^B_N$, tj. iz bilo kojeg kauzalno kompletnog dnevnika događaja, što će primer koji sledi pokazati.

Posmatra se paralelan proces čiji je blok-strukturirani model prikazan na slici 4.1, i dnevnik događaja L_1 sa zapisima dobijenim posle nekoliko izvršavanja procesa. U dnevniku su prikazani samo različiti tragovi, bez naznake broja njihovog pojavljivanja, s obzirom na to da učestalost njihovog pojavljivanja nije relavantna za ovaj deo istraživanja.

$$L_1 = [< a, b, c, d, e, f, g, h >, < a, f, g, b, c, e, d, h >, < a, c, d, e, f, g, b, h >, < a, b, f, g, c, d, e, h >]$$

Bazična kauzalna relacija za mrežu prikazanu na slici 4.1 jeste:
$\rightarrow^{B}_{N} = \{(a, b), (a, c), (a, f), (b, h), (c, d), (c, e), (d, h), (e, h), (f, g), (g, h)\}$

Relacije uređenja dobijene na osnovu zapisa u dnevniku događaja L_1 su:
$>_{L1} = \{(a, b), (a, c), (a, f), (b, c), (b, f), (b, h), (c, d), (c, e), (d, e), (d, h), (e, d), (e, f), (e, h), (f, g), (g, b), (g, c), (g, h)\}$

$>>_{L1} = \{(a, d), (a, e), (a, g), (a, h), (b, d), (b, e), (b, g), (c, b), (c, f), (c, g), (c, h), (d, b), (d, f), (d, g), (e, b), (e, g), (f, b), (f, c), (f, d), (f, e), (f, h), (g, d), (g, e)\}$

$\|_{L1} = \{(b, c), (c, b), (b, d), (d, b), (b, e), (e, b), (b, f), (f, b), (b, g), (g, b), (c, f), (f, c), (c, g), (g, c), (d, e), (e, d), (d, f), (f, d), (d, g), (g, d), (e, f), (f, e), (e, g), (g, e)\}$

$\rightarrow_{L1} = \{(a, b), (a, c), (a, f), (b, h), (c, d), (c, e), (d, h), (e, h), (f, g), (g, h)\}$

$\Rightarrow_{L1} = \{(a, d), (a, e), (a, g), (a, h), (c, h), (f, h)\}$

$\#_{L1} = \{(a, a), (b, b), (c, c), (d, d), (e, e), (f, f), (g, g), (h, h)\}$

Može se primetiti da je relacija kauzalnosti dnevnika događaja L_1 jednaka bazičnoj kauzalnoj relaciji, tj. $\rightarrow_{L1} = \rightarrow^{B}_{N}$, što dnevnik događaja L_1 čini kauzalno kompletnim. *Otisak* dnevnika događaja L_1 dat je u Tabeli 4.2.

	a	b	c	d	e	f	g	h
a	#	$\rightarrow$	$\rightarrow$	$\Rightarrow$	$\Rightarrow$	$\rightarrow$	$\Rightarrow$	$\Rightarrow$
b	$\leftarrow$	#	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\rightarrow$
c	$\leftarrow$	$\|\|$	#	$\rightarrow$	$\rightarrow$	$\|\|$	$\|\|$	$\Rightarrow$
d	$\Leftarrow$	$\|\|$	$\leftarrow$	#	$\|\|$	$\|\|$	$\|\|$	$\rightarrow$
e	$\Leftarrow$	$\|\|$	$\leftarrow$	$\|\|$	#	$\|\|$	$\|\|$	$\rightarrow$
f	$\leftarrow$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	#	$\rightarrow$	$\Rightarrow$
g	$\Leftarrow$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\leftarrow$	#	$\rightarrow$
h	$\Leftarrow$	$\leftarrow$	$\Leftarrow$	$\leftarrow$	$\leftarrow$	$\Leftarrow$	$\leftarrow$	#

Tabela 4.2: *Otisak* dnevnika događaja L_1.

Primenom $\alpha^{\|}$ algoritma na posmatrani dnevnik događaja L_1 dobija se:
(1) $T_{L1} = \{a, b, c, d, e, f, g, h\}$
(2) $i_{L1} = a$
(3) $o_{L1} = h$
(4) $X_{L1} = \{(\{a\},\{b\}), (\{a\},\{c\}), (\{a\},\{f\}), (\{b\},\{h\}), (\{c\},\{d\}), (\{c\},\{e\}), (\{d\},\{h\}), (\{e\},\{h\}), (\{f\},\{g\}), (\{g\},\{h\})\}$

(5) $P_{L1} = \{p_{(\{a\},\{b\})}, p_{(\{a\},\{c\})}, p_{(\{a\},\{f\})}, p_{(\{b\},\{h\})}, p_{(\{c\},\{d\})}, p_{(\{c\},\{e\})}, p_{(\{d\},\{h\})}, p_{(\{e\},\{h\})}, p_{(\{f\},\{g\})}, p_{(\{g\},\{h\})}, i_{L1}, o_{L1}\}$

(6) $F_{L1} = \{(a, p_{(\{a\},\{b\})}), (p_{(\{a\},\{b\})}, b), (a, p_{(\{a\},\{c\})}), (p_{(\{a\},\{c\})}, c), (a, p_{(\{a\},\{f\})}), (p_{(\{a\},\{f\})}, f), (b, p_{(\{b\},\{h\})}), (p_{(\{b\},\{h\})}, h), (c, p_{(\{c\},\{d\})}), (p_{(\{c\},\{d\})}, d), (c, p_{(\{c\},\{e\})}), (p_{(\{c\},\{e\})}, e), (d, p_{(\{d\},\{h\})},), (p_{(\{d\},\{h\})}, h), (e, p_{(\{e\},\{h\})}), (p_{(\{e\},\{h\})}, h), (f, p_{(\{f\},\{g\})}), (p_{(\{f\},\{g\})}, g), (g, p_{(\{g\},\{h\})}), (p_{(\{g\},\{h\})}, h), (i_{L1}, a), (h, o_{L1})\}$

(7) $\alpha^{\|}(L_1) = (P_{L1}, T_{L1}, F_{L1})$

Dobijenim rezultatima odgovara Petri mreža prikazana na slici 4.1.

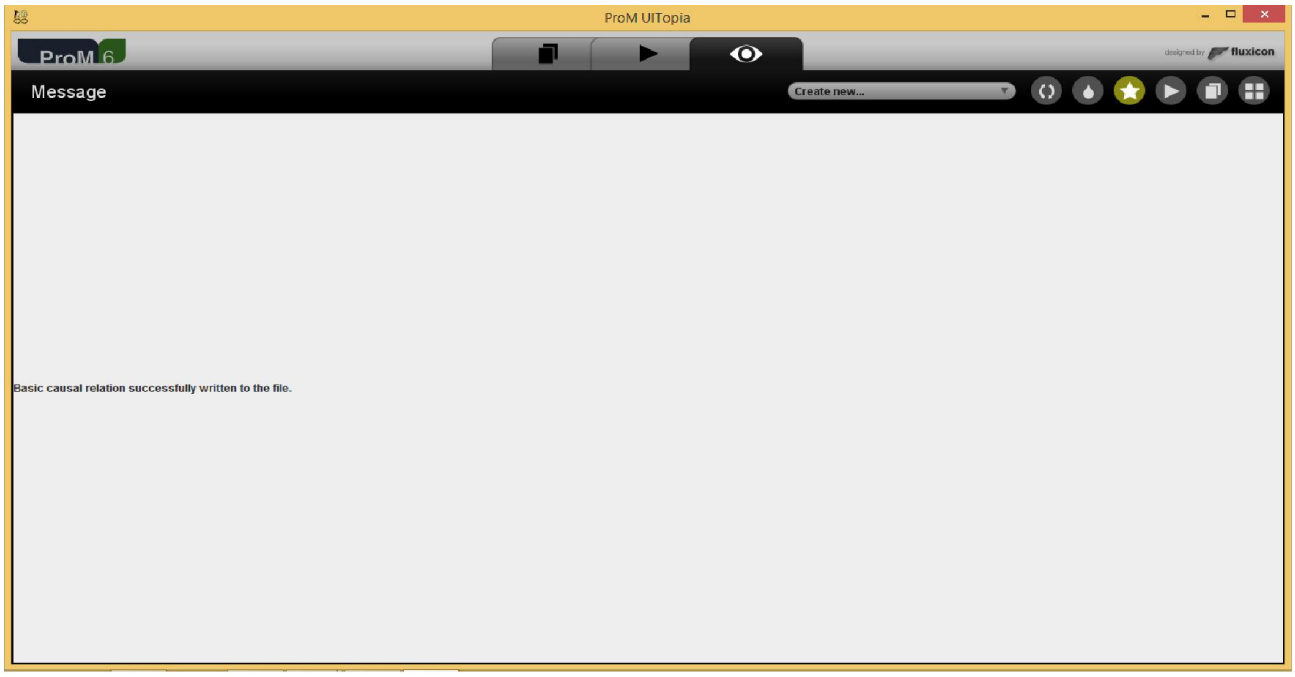
Otkrivanje originalne mreže iz kauzalno kompletnog dnevnika događaja pomoću alata ProM

Za potrebe otkrivanja originalnih blok-strukturiranih modela paralelnih poslovnih procesa modifikovanom PM tehnikom i $\alpha^{\|}$-algoritmom na osnovu kauzalno kompletnih dnevnika događaja, u okviru postojećeg ProM okruženja [74], [75] napravljena su dva priključka (slike 4.2 i 4.4). Za realizaciju priključaka korišćena je verzija ProM 6.2. Priključci su implementirani u Eclipse razvojnom okruženju podešenom prema uputstvima iz [79]. Potreban kod je preuzet uz pomoć SVN alata za praćenje verzija softvera sa adrese [80]. Kada se uveze dnevnik događaja, kreirani priključci postaju vidljivi na listi priključaka kojim se dnevnik može procesirati.

Na slici 4.2 dat je prikaz okruženja kada je aktivan jedan od pomenutih priključaka pod nazivom: *Modified Alpha algorithm helper plug-in-find basic causality relation*. Programski kod ovog priključka nalazi se u zasebnom paketu: *Alphaminer_mod_basic_relation*, koji je priložen u Prilogu A.

Slika 4.2: Prikaz ProM okruženja sa aktivnim priključkom:
Modified Alpha algorithm helper plug-in-find basic causality relation.

U procesu otkrivanja originalne mreže modifikovanom PM tehnikom na osnovu kauzalnih dnevnika događaja, prvo se startovanjem aktivnog priključka na slici 4.2 iz uvezenog kompletnog dnevnika događaja izdvaja bazična kauzalna relacija. Ukoliko je to uspešno urađeno na ekranu se pojavljuje poruka: „*Basic causal relation successfully written to the file*“ (slika 4.3).



Slika 4.3: Poruka koja se dobija posle uspešnog izdvajanja bazične kauzalne relacije.

Po dobijanju takve poruke na ekranu, iz uvezenog kauzalno kompletnog dnevnika događaja se može pronaći originalna mreža pomoću priključka: *Mine for a Petri net using modified Alpha-algorithm* prikazanog na slici 4.4. Programski kod ovog priključka nalazi se u zasebnom, novom paketu, *AlphaMiner_mod*, koji je priložen u Prilogu B.

Alat ProM koristi datoteke u XES formatu za unos dnevnika događaja. XES je standard za formiranje dnevnika događaja baziran na XML-u. Svrha ovog standarda je da pruži opšte prihvaćeni okvir za kreiranje dnevnika događaja sa širokim spektrom oblasti primene i pogodnim za upotrebu u različitim alatima. Ovaj standard ima generički pristup koji je u stanju da obuhvati različite dnevnike sa što je moguće manje narušavanja semantike realnih događaja. To podrazumeva da standard definiše elemente zajedničke za praktično sve scenarije, a sve ostale informacije su dostupne kroz opcionalne atribute.

Slika 4.4: Prikaz ProM okruženja sa uvezenim kauzalno kompletnim dnevnikom i aktivnim priključkom: *Mine for a Petri net using modified Alpha-algorithm*.

Na slici 4.5 dat je izgled kauzalno kompletnog dnevnika događaja L_1 iz razmatranog primera u .xes formatu.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
   <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext"/>
   <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
   <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
   <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
   <extension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/semantic.xesext"/>
   <global scope="trace">
      <string key="concept:name" value="__INVALID__"/>
   </global>
   <global scope="event">
      <string key="concept:name" value="__INVALID__"/>
      <string key="lifecycle:transition" value="complete"/>
   </global>
   <classifier name="MXML Legacy Classifier" keys="concept:name lifecycle:transition"/>
   <classifier name="Event Name" keys="concept:name"/>
   <classifier name="Resource" keys="org:resource"/>
```

```
  <string key="source" value="Rapid Synthesizer"/>
  <string key="concept:name" value="Kauzalno_kompletan_dnevnik_dogadjaja.xes"/>
  <string key="lifecycle:model" value="standard"/>
<trace>
    <string key="concept:name" value="Case4.0"/>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
```

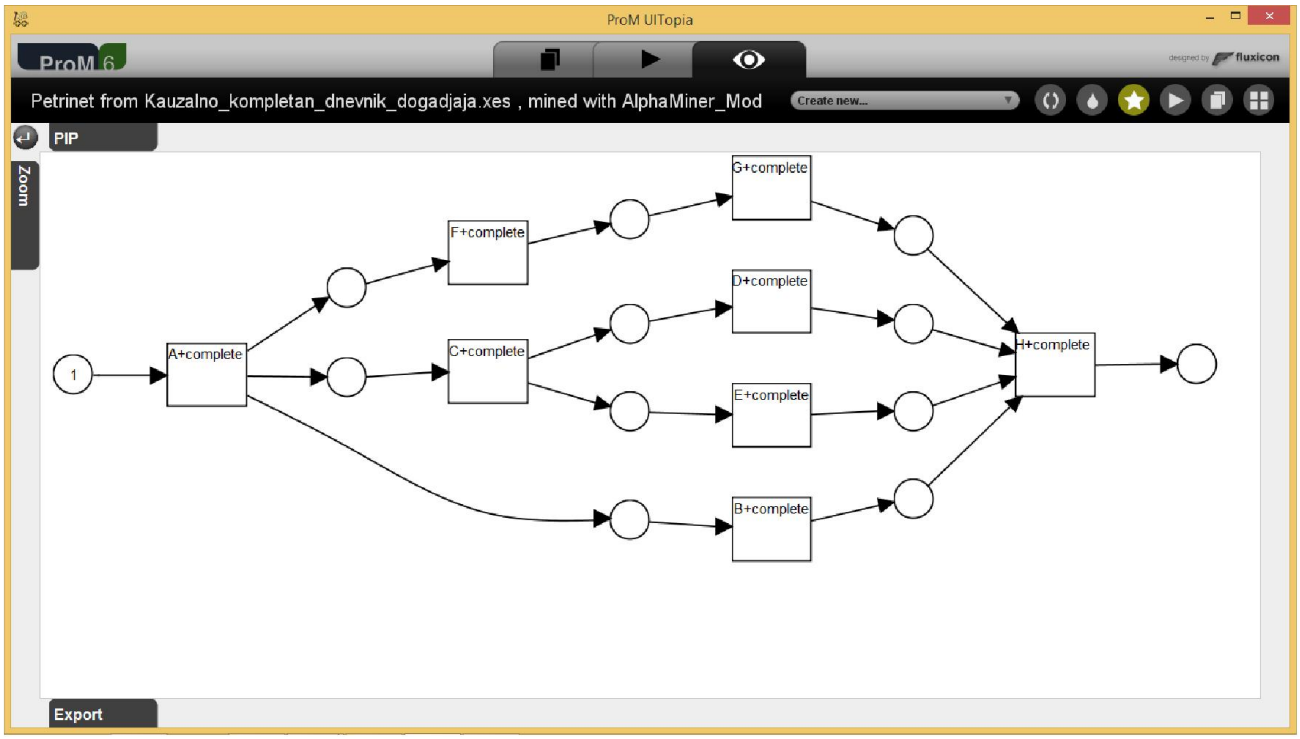
```
</trace>
<trace>
   <string key="concept:name" value="Case3.0"/>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
   <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
   </event>
 </trace>
        <trace>
   <string key="concept:name" value="Case2.0"/>
```

```
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
  </trace>
  <trace>
    <string key="concept:name" value="Case1.0"/>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
```

```
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
     </event>
     <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
     </event>
  </trace>
</log>
```

Slika 4.5. Izgled dnevnika događaja L_1 u .xes formatu

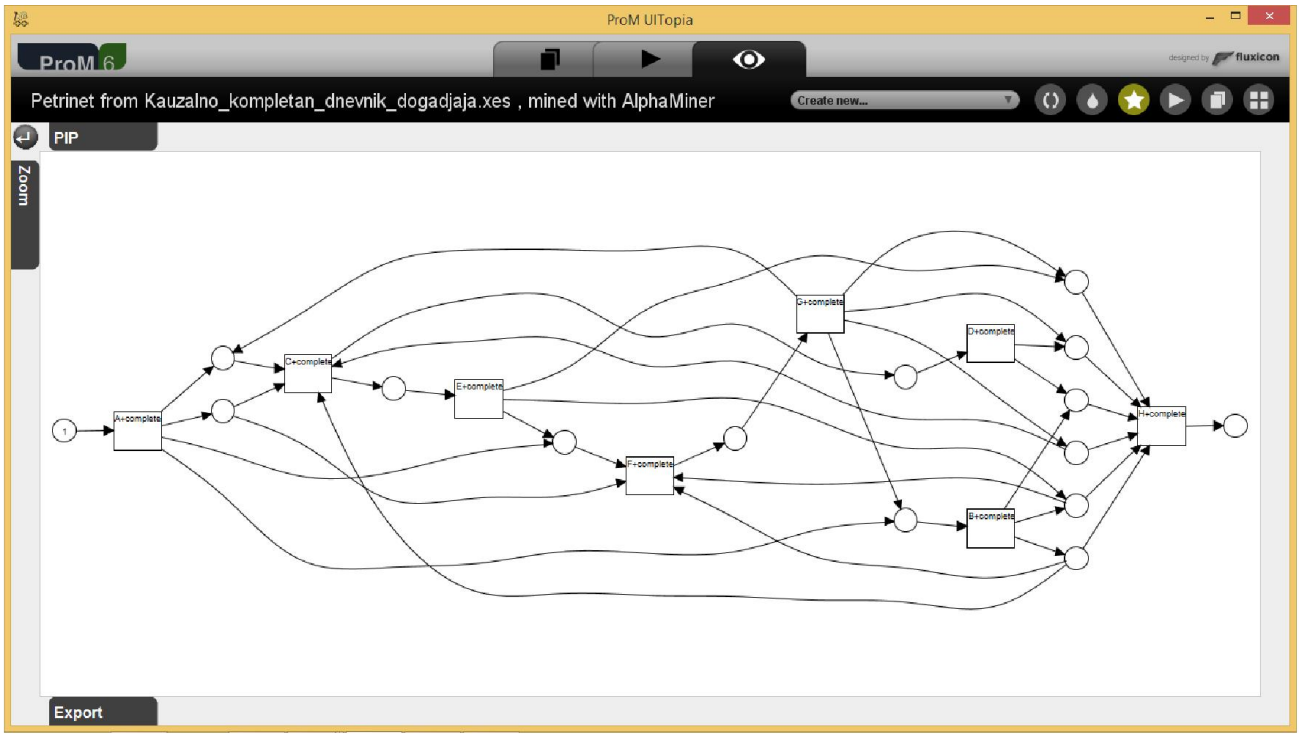
Kada se nad dnevnikom događaja L_1 (prikazanog na slici 4.5) startuje aktivan priključak *Mine for a Petri net using modified Alpha-algorithm*, kao što je prikazano na slici 4.4, dobija se model procesa prikazan u obliku Petri mreže (slika 4.6) koji je jednak mreži razmatranog primera procesa prikazanoj na slici 4.1.



Slika 4.6: Mreža dobijena na osnovu dnevnika L_1, primenom $\alpha^{\|}$-algoritma, tj. $N_1^{\|} = \alpha^{\|}(L_1)$.

Log L_1 nije kompletan, jer u relaciji $>_{L1}$ nedostaju elementi: $b > d$, $b > e$, $b > g$, $c > b$, $c > f$, $c > g$, $d > b$, $d > f$, $d > g$, $e > b$, $e > g$, $f > b$, $f > c$, $f > d$, $f > e$, $g > d$ i $g > e$, što bi potencijalno moglo da se obavi na osnovu modela procesa datog na Slici 4.1, i dobijene WF-mreže $N_1^{\|} = \alpha^{\|}(L_1)$. Log L_1 je kauzalno kompletan, jer je ispunjen uslov iz definicije 4.6, tj.: $\rightarrow_{L1} = \rightarrow^{B}_{N}$.

Ukoliko se nad ovim dnevnikom događaja startuje aktivan priključak *Mine for a Petri net using Alpha-algorithm* (prikazan na slici 3.5), koji za pronalaženje originalne mreže koristi osnovni α-algoritam, dobija se model prikazan na slici 4.7. Može se primetiti da mreža $N_1 = \alpha(L_1)$ prikazana na slici 4.7 nije jednaka originalnoj mreži razmatranog primera prikazanoj na slici 4.1.



Slika 4.7: Mreža dobijena na osnovu dnevnika L_1, primenom α-algoritma, tj. $N_1 = \alpha(L_1)$.

4.2.3 Eksperimentalna analiza kauzalno kompletnih dnevnika događaja

Osnovni zadatak u okviru eksperimentalne analize u ovom delu istraživanja jeste da se utvrdi da li su kauzalno kompletni dnevnici manji od kompletnih dnevnika događaja u opštem slučaju, i kakav je odnos njihovih veličina.

Eksperimentalna analiza je izvršena na realnim primerima, kod kojih su upoređene veličine minimalnih kompletnih i minimalnih kauzalno kompletnih dnevnika događaja, potrebnih za otkrivanje originalne mreže paralelnih procesa. Da bi se to ostvarilo, u okviru postojećeg ProM okruženja [74], [75] napravljen je priključak pod nazivom: *Modified Alpha-algorithm - Minimal Logs* (slika 4.8). Programski kod priključka nalazi se u zasebnom, novom paketu, *AlphaMiner_mod Find Minimal Logs from Any Log*, koji je dat u Prilogu C.

Analiza je sprovedena na uzorku od 100 realnih primera dobijenih pretragom Interneta i odabirom javno dostupnih modela poslovnih procesa, koji se uklapaju u uslove blok-strukturiranih modela paralelnih procesa. Razmatrani primeri su prikazani u Prilogu D, a mogu se naći i na adresi: http://фтнкм.срб/fakultet/zaposleni/julijana-lekic/Examples-Block-Structured-Parallel-Process [81].

Slika 4.8: Prikaz ProM okruženja sa uvezenim kompletnim dnevnikom i aktivnim priključkom: *Modified Alpha-algorithm - Minimal Logs*.

Izdvajanje minimalnih kompletnih dnevnika i minimalnih kauzalno kompletnih dnevnika događaja

Pomoću gore pomenutog priključka (slika 4.8), iz uvezenog kompletnog dnevnika događaja izdvajaju se kompletni i kauzalno kompletni dnevnici sa najmanjim mogućim brojem tragova, i upoređuju se njihove međusobne veličine.

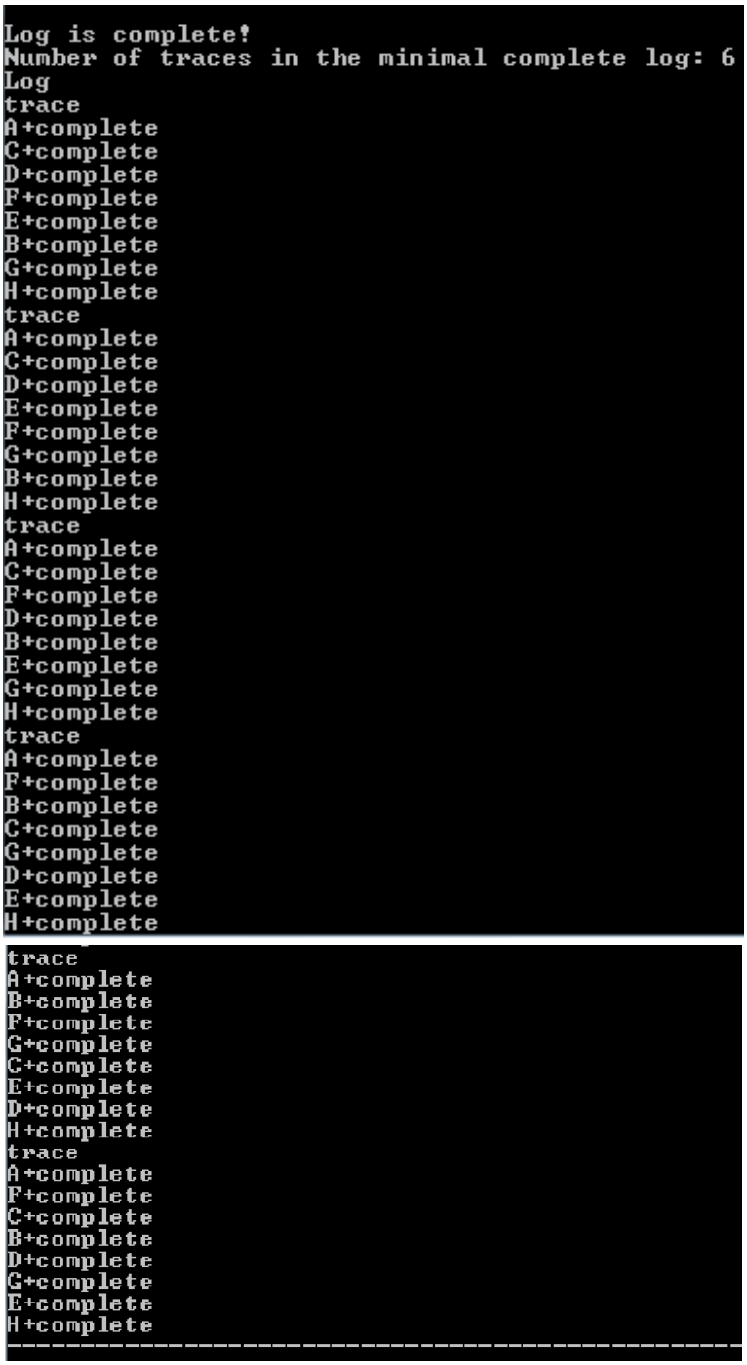
U postupku dobijanja minimalnog kompletnog i minimalnog kauzalno kompletnog dnevnika iz kompletnog dnevnika događaja, prvo se pronalazi bazična kauzalna relacija $\rightarrow^{B}_{N}$ aktiviranjem priključka: *Modified Alpha algorithm helper plug-in-find basic causality relation* (slika 4.2). Po dobijanju poruke na ekranu o uspešnom izdvajanju bazične kauzalne relacije (slika 4.3), nad uvezenim kompletnim dnevnikom startuje se aktivan priključak: *Modified Alpha-algorithm - Minimal Logs* (slika 4.8). Kao rezultat toga, na ekranu se dobijaju veličine i izgledi minimalnog kompletnog dnevnika i minimalnog kauzalno kompletnog dnevnika, izdvojenih iz uvezenog kompletnog dnevnika događaja.

Posmatrajmo dnevnik događaja L sa zapisima dobijenim posle nekoliko izvršavanja procesa čiji je model prikazan na slici 4.1.

$L = [< a, b, c, d, e, f, g, h >, < a, f, g, b, c, e, d, h >, < a, c, d, e, f, g, b, h >, < a, b, f, g, c, d, e, h >, < a, c, b, d, e, f, g, h >, < a, c, b, e, d, f, g, h >, < a, f, b, g, c, d, e, h >, < a, c, f, b, g, e, d, h >, < a, f, c, g, b, e, d, h >, < a, f, g, c, d, b, e, h >, < a, b, f, c, d, g, e, h >, < a, c, e, b, d, f, g, h >, < a, b, c, f, e, g, d, h >,< a, c, f, d, g, e, b, h >]$

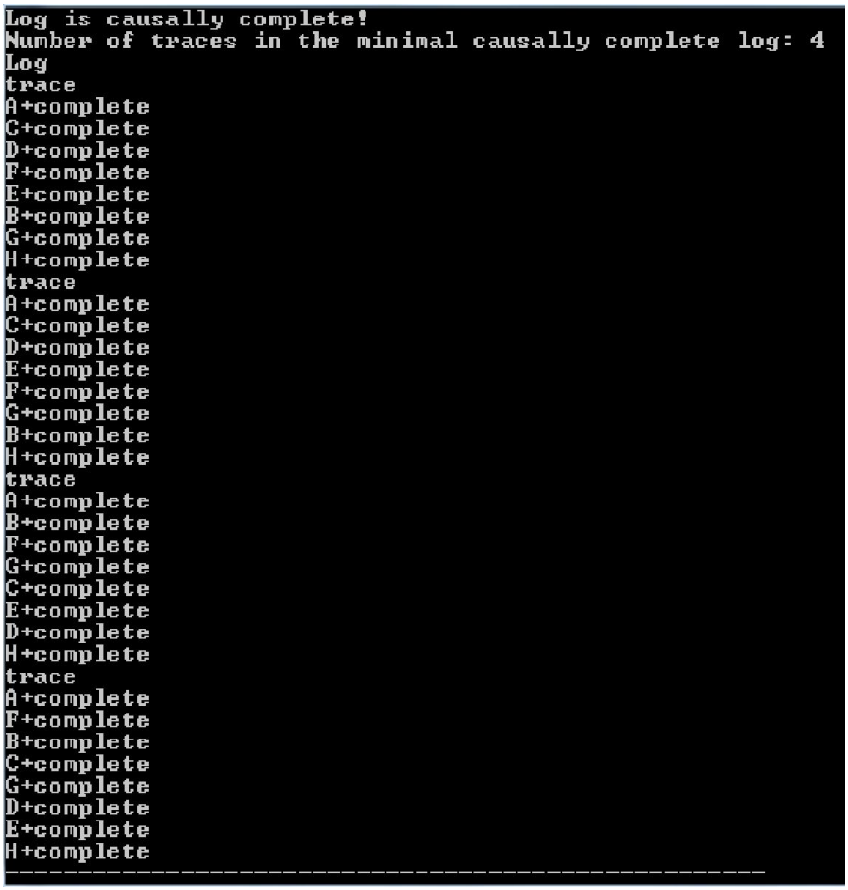
U dnevniku su prikazani samo različiti tragovi, bez naznake broja njihovog pojavljivanja, s obzirom na to da učestalost njihovog pojavljivanja nije relavantna za ovo istraživanje. Dnevnik *L* ima 14 tragova i ispunjava uslove kompletnosti (date definicijom 3.3) za model na slici 4.1.

Na slikama 4.9 i 4.10 prikazan je rezultat izdvajanja minimalnog kompletnog i minimalnog kauzalno kompletnog dnevnika, respektivno, iz uvezenog dnevnika događaja *L*. Sa slike 4.9 se može videti da minimalni kompletan dnevnik događaja izdvojen iz dnevnika *L* ima 6 tragova.



Slika 4.9: Rezultat izdvajanja minimalnog kompletnog.
dnevnika događaja iz dnevnika *L*

Sa slike 4.10 se može videti da minimalni kauzalno kompletan dnevnik događaja izdvojen iz dnevnika *L* ima 4 traga.



Slika 4.10: Rezultat izdvajanja minimalnog kauzalno kompletnog dnevnika događaja iz dnevnika *L*.

Karakteristike analiziranih primera

U Tabelama 4.2 i 4.3 date su neke karakteristike koje oslikavaju strukturu mreža analiziranih primera prikazanih u Prilogu D. U Tabeli 4.2 prikazan je broj aktivnosti u mreži za svaki primer, kao i prosečan broj aktivnosti po primeru. U Tabeli 4.3 prikazan je broj tranzicija u mreži za svaki primer, kao i prosečan broj tranzicija po primeru.

Broj aktivnosti	Broj primera
5	3
6	16
7	21
8	13
9	14
10	10
11	4
12	3
13	6
14	3
15	4
16	2
17	1
Prosečan broj aktivnosti po primeru	Ukupno
8,97	100

Tabela 4.2: Broj aktivnosti po primeru.

Broj tranzicija	Broj primera
6	4
7	20
8	22
9	17
10	5
11	5
12	7
13	3
14	7
15	4
16	1
17	2
18	2
23	1
Prosečan broj tranzicija po primeru	Ukupno
9,74	100

Tabela 4.3: Broj tranzicija po primeru.

Rezultati eksperimentalne analize

U Tabeli 4.4 su prikazani rezultati izvršene eksperimentalne analize i rezultati uporedne analize minimalnih veličina kompletnih i kauzalno kompletnih dnevnika događaja, potrebnih za otkrivanje originalnih mreža razmatranih primera opisanim postupkom. Eksperimentalna analiza je pokazala da su veličine kauzalno kompletnih dnevnika, iz kojih se mogu otkriti originalne mreže posmatranih paralelnih poslovnih procesa, manji ili (retko) jednaki veličini kompletnih dnevnika događaja.

Za lakše razumevanje tabela i slika koje slede, koriste se sledeće oznake:

- N_a označava ukupan broj aktivnosti u paralelnim granama
- N_b označava broj paralelnih grana u mreži
- N_{mcl} označava broj tragova u minimalnom kompletnom dnevniku događaja
- N_{mccl} označava broj tragova u minimalnom kauzalno kompletnom dnevniku događaja.

Iz Tabele 4.4 se može videti da su kod 99 primera (od razmatranih 100 primera), minimalni kauzalno kompletni dnevnici događaja manji od minimalnih kompletnih dnevnika, i to u proseku za 37,55%, a kod jednog primera njihove vrednosti su jednake. Ni u jednom od posmatranih primera broj tragova u minimalnom kompletnom dnevniku događaja nije manji od broja tragova u minimalnom kauzalno kompletnom dnevniku.

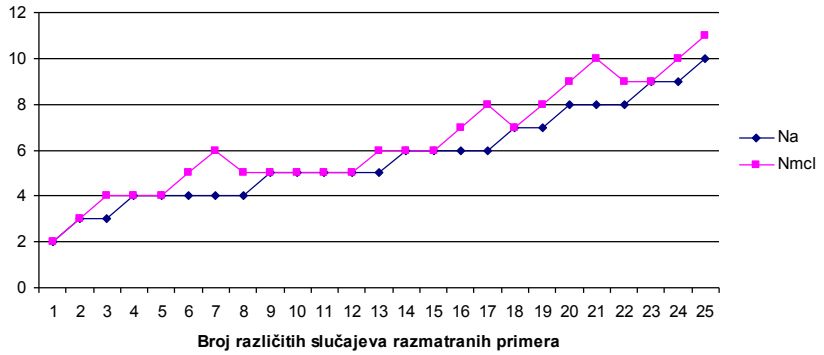
Broj primera	Broj tragova u dnevnicima događaja		Minimalni kauzalni dnevnici manji od minimalnih kompletnih dnevnika
	N_{mcl}	N_{mccl}	
1	2	2	0,00%
12	3	2	33,33%
13	4	2	50,00%
41	4	3	25,00%
5	5	2	60,00%
4	5	3	40,00%
4	5	4	20,00%
3	6	2	66,67%
3	6	3	50,00%
1	6	5	16,67%
3	7	3	57,14%
3	8	4	50,00%
1	9	2	77,78%
2	9	4	55,56%
2	10	3	70,00%
1	10	5	50,00%
1	11	3	72,73%
Ukupno			U proseku manji za
100			37,55%

Tabela 4.4: Rezultati uporedne analize minimalnih veličina kompletnih i kauzalno kompletnih dnevnika događaja dobijenih eksperimentalnom analizom.

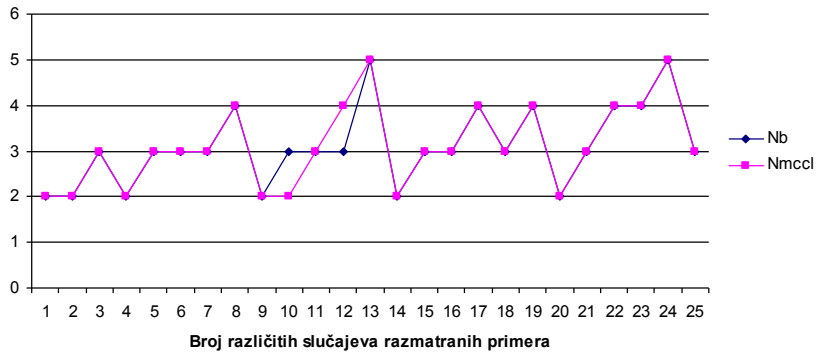
Posmatrajući strukturu mreža u razmatranim primerima, rezultati eksperimentalne analize su pokazali da veličine dnevnika događaja iz kojih se mogu otkriti originalne mreže zavise od broja paralelnih grana u mreži, kao i od ukupnog broja aktivnosti u međusobno paralelnim granama. Iz Tabele 4.5 se može videti da je veličina minimalnih kompletnih dnevnika srazmerna ukupnom broju aktivnosti u međusobno paralelnim granama (slika 4.11), a da je veličina minimalnih kauzalno kompletnih dnevnika događaja srazmerna broju paralelnih grana u mreži (slika 4.12). Shodno tome, razlika u veličinama minimalnih kompletnih i minimalnih kauzalno kompletnih dnevnika događaja, izražena u broju tragova, srazmerna je razlici između ukupnog broja aktivnosti u paralelnim granama i broju paralelnih grana u mreži (slika 4.13).

Broj primera	N_a	N_b	$N_a - N_b$	N_{mcl}	N_{mccl}	$N_{mcl} - N_{mccl}$
1	2	2	0	2	2	0
12	3	2	1	3	2	1
39	3	3	0	4	3	1
11	4	2	2	4	2	2
6	4	3	1	4	3	1
2	4	3	1	5	3	2
1	4	3	1	6	3	3
3	4	4	0	5	4	1
4	5	2	3	5	2	3
1	5	3	2	5	2	3
2	5	3	2	5	3	2
1	5	3	2	5	4	1
1	5	5	0	6	5	1
3	6	2	4	6	2	4
2	6	3	3	6	3	3
1	6	3	3	7	3	4
1	6	4	2	8	4	4
2	7	3	4	7	3	4
1	7	4	3	8	4	4
1	8	2	6	9	2	7
1	8	3	5	10	3	7
1	8	4	4	9	4	5
1	9	4	5	9	4	5
1	9	5	4	10	5	5
1	10	3	7	11	3	8

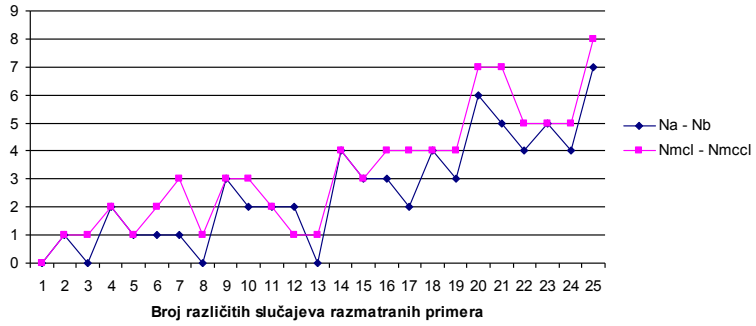
Tabela 4.5: Uticaj broja paralelnih grana u mreži i ukupnog broja aktivnosti u međusobno paralelnim granama na veličine minimalnih kompletnih i kauzalno kompletnih dnevnika događaja.



Slika 4.11: Odnos između N_a - ukupnog broja aktivnosti u paralelnim granama i N_{mcl} - broja tragova u minimalnom kompletnom dnevniku.



Slika 4.12: Odnos između N_b - broja paralelnih grana i N_{mccl} - broja tragova u minimalnom kauzalno kompletnom dnevniku.



Slika 4.13: Odnos između razlike: N_a - N_b i razlike: N_{mcl} - N_{mccl}.

Test rangova (Wilcoxon-Mann-Whitney test)

Da bi se statistički potvrdila hipoteza da su kauzalno kompletni dnevnici događaja manji od kompletnih dnevnika događaja, na rezultate prikazane u Tabeli 4.4 biće primenjen test rangova, odnosno *Wilcoxon-Mann-Whitney test* [82].

Ako su: X = minimalni kauzalno kompletni dnevnici događaja, i Y = minimalni kompletni dnevnici događaja, treba testirati nultu hipotezu da su raspodele dva obeležja jednake, tj. H_0: $F_X = F_Y$, protiv alternativne hipoteze H_1: ”Minimalni kauzalno kompletni dnevnici X su manji od minimalnih kompletnih dnevnika Y ”.

Odgovarajuća kritična oblast C u tom slučaju je:

H_0	H_1	C
$F_X = F_Y$	X je manje od Y	$z_0 \leq -z_{0,5-\alpha}$

U tu svrhu, dobijeni rezultati eksperimentalne analize prikazani u Tabeli 4.4, predstavljaju se u obliku:

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4

X	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Y	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

X	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	3	3	3	3
Y	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5

X	4	4	4	4	2	2	2	3	3	3	5	3	3	3	4	4	4	2	4	4	3	3	5	3
Y	5	5	5	5	6	6	6	6	6	6	6	7	7	7	8	8	8	9	9	9	10	10	10	11

Sređivanjem rezultata u jedinstven neopadajući niz i dodeljivanjem 0 svakom elementu X, a 1 svakom elementu Y, dobija se niz:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.

Na osnovu ovog niza testira se hipoteza H_0 protiv alternativne H_1, pri čemu se dobijaju sledeće vrednosti:

$n_1 = 100,\quad n_2 = 100,\quad n = n_1 + n_2 = 200$

$V = 12 + 12 +12 +12 +12 +12 +12 +12 +12 + 66 + 66 = 240$

$E(V) = n_1 n_2/2 = 4900{,}5$

$(V) = E(V)\,(\mathrm{n} + 1)/6 = 162533{,}25$

$z_0 = (V - E(V))/\,[D(V)]^{1/2} = -11{,}56$

Za nivo značajnosti $\alpha = 0{,}05$, kritična oblast ovog testa je $C = (-\infty, -1{,}645]$. S obzirom na to da realizovana vrednost test statistike z_0 pripada kritičnoj oblasti C, to se nulta hipoteza H_0 odbacuje u korist alternativne H_1. Drugim rečima, na nivou značajnosti $\alpha = 0{,}05$ može se zaključiti da su minimalni kauzalni kompletni dnevnici događaja manji od minimalnih kompletnih dnevnika događaja.

4.3 Otkrivanje modela paralelnih poslovnih procesa na osnovu slabo kompletnih dnevnika događaja

Dalje istraživanje je pokazalo da se primenom modifikovane PM tehnike mogu otkriti modeli paralelnih poslovnih procesa iz dnevnika događaja koji su još oskudniji po broju tragova od kauzalno kompletnih dnevnika. U potrazi za dnevnicima događaja sa što je moguće manjim brojem tragova iz kojih se može otkriti model paralelnog poslovnog procesa, kao i uslova koji takvi dnevnici moraju da ispune, došlo se do tzv. *slabo kompletnih* dnevnika događaja.

Kako je već predstavljeno u prethodnoj sekciji, primenom modifikovane PM tehnike kod paralelnih procesa do originalne mreže se može doći iz bilo kog dnevnika događaja kod kojeg je $\rightarrow_L = \rightarrow^B_N$. Dakle, osnovni zadatak je pronaći dnevnik čija je relacija kauzalnosti jednaka bazičnoj kauzalnoj relaciji, a zatim primeniti $\alpha^{\|}$-algoritam, i na taj način se dolazi do originalne mreže. Postoje dnevnici kod kojih se može otkriti da je njihova relacija kauzalnosti jednaka $\rightarrow^B_N$, ali se do te relacije kauzalnosti ne može doći samo iz evidencije zapisane u tragovima dnevnika, već se pojedini elementi relacije kauzalnosti mogu zaključiti u samom postupku primene $\alpha^{\|}$-algoritma na takve dnevnike. Dnevnik sa takvim svojstvom nazvan je *slabo* (engl. *weakly*) *kompletan dnevnik događaja* - L_w. Slabo kompletni dnevnici po svojoj obimnosti mogu biti znatno oskudniji kako od kompletnih dnevnika tako i od kauzalno kompletnih dnevnika. U ovom odeljku će biti prikazano koje uslove mora da ispuni dnevnik događaja da bi bio slabo kompletan, kako se iz takvog dnevnika može otkriti originalna mreža, problemi do kojih se dolazi u tom postupku otkrivanja modela, kao i način da se ti problemi prevaziđu.

4.3.1 Slabo kompletan dnevnik događaja

Primeri su pokazali da se do originalne mreže može doći i na osnovu nekompletnog dnevnika događaja L za koji važi: $\rightarrow^B_N \subset (\rightarrow_L \cup \Rightarrow_L)$, i $\rightarrow_L \subset \rightarrow^B_N$, koji se u tom slučaju naziva *slabo kompletan dnevnik događaja* L_w.

Definicija 4.8. (*Slabo kompletan dnevnik događaja*) Neka je $N = (P, T, F)$ ispravna WF-mreža, odnosno $N \in \mathcal{W}$, i neka je $\rightarrow^B_N$ bazična kauzalna relacija za mrežu N. Dnevnik L_w je *slabo kompletan* dnevnik događaja od N akko:

1) $\rightarrow^B_N \subset (\rightarrow_{Lw} \cup \Rightarrow_{Lw})$, i $\rightarrow_{Lw} \subset \rightarrow^B_N$, i
2) za bilo koje $t \in T$ postoji $\sigma \in L_w$ tako da $t \in \sigma$.

Iako se kod slabo kompletnih dnevnika događaja na osnovu tragova u dnevniku ne može zaključiti da je $\rightarrow_{Lw} = \rightarrow^B_N$ (nego $\rightarrow_{Lw} \subset \rightarrow^B_N$), elementi relacije kauzalnosti koji se ne nalaze u $\rightarrow_{Lw}$, a nalaze se u $\rightarrow^B_N$, mogu se naknadno zaključiti iz *otiska* dnevnika. Ti elementi čine relaciju kauzalnosti nazvanu *zaključenom* (engl. *inferred*) *kauzalnom relacijom*, koja se obeležava sa $\rightarrow^I_{Lw}$. Relacija

kauzalnosti koja se unosi u konačan izgled *otiska* dnevnika (označena sa $\rightarrow_{Lf}$), i nad kojom se primenjuje $\alpha^{\|}$-algoritam, postaje: $\rightarrow_{Lf} = \rightarrow_{Lw} \cup \rightarrow^{I}_{Lw}$ čime se dobija da je $\rightarrow_{Lf} = \rightarrow^{B}_{N}$, što će u nastavku biti prikazano. Pronalaženje relacije kauzalnosti $\rightarrow_{Lf}$ koja je jednaka bazičnoj kauzalnoj relaciji $\rightarrow^{B}_{N}$ je dovoljan uslov da se na osnovu $\rightarrow_{Lf}$ primenom algoritma $\alpha^{\|}$ otkrije originalna mreža, na način kako je to prikazano u prethodnom odeljku.

Na osnovu ovako definisanog dnevnika događaja L_w može se dobiti originalna mreža paralelnog poslovnog procesa, pa stoga važi sledeća definicija:

Definicija 4.9. (*Mogućnost ponovnog otkrivanja modela paralelnog procesa na osnovu* L_w) Neka je $N^{\|} = (P, T, F)$ paralelan proces, i neka je $\alpha^{\|}$ algoritam koji mapira logove od $N^{\|}$ na ispravne WF-mreže, tj. $\alpha^{\|}$: $\mathcal{P}(T^{*}) \rightarrow \mathcal{W}$. Ako za bilo koji slabo kompletan dnevnik događaja L_w od $N^{\|}$, $\alpha^{\|}$ algoritam vraća $N^{\|}$, onda je $\alpha^{\|}$ u mogućnosti da ponovo otkrije $N^{\|}$.

4.3.2 Odnos relacija i mesta povezivanja kod modifikovane PM tehnike otkrivanja procesa

α-algoritam je zasnovan na činjenici da za većinu WF-mreža važi da su dve aktivnosti povezane ako i samo ako se njihova kauzalnost može detektovati iz dnevnika događaja [4]. Na istoj osnovi počiva i modifikovana PM tehnika otkrivanja modela procesa. Pri tome, postoje mesta koja ne utiču na ponašanje procesa pa se ne mogu ni detektovati iz dnevnika, takozvana *implicitna mesta* (definicija 2.13). S obzirom na to da je za ponovno otkrivanje modela procesa od ključne važnosti da struktura WF-mreže jasno reflektuje njeno ponašanje, i u modifikovanoj PM tehnici se koriste *ispravne strukturne WF-mreže* bez implicitnih mesta i mrtvih tranzicija, kako je definisano u sekciji 2.2.3.

Većina teorema kojima su definisani odnosi relacija i mesta povezivanja datih u [4], [5] i dokazanih u [5] važe i kod modifikovane PM tehnike. Međutim, u definicjama tih teorema stoji da se radi o kompletnim dnevnicima događaja, ali su one jednako primenljive i na kauzalno kompletne i na slabo kompletne dnevnike. Ta konstatacija proizilazi iz činjenice da se u dokazima tih teorema [5] ne posmatraju dnevnici događaja, pa samim tim ni njihovo svojstvo kompletnosti, već samo relacije između aktivnosti, njihova povezanost u mreži i, shodno tome, okidanje aktivnosti u toku izvršavanja procesa čiji su deo. Stoga, kod modifikovane PM tehnike otkrivanja modela procesa važe teoreme 3.1, 3.2 i 3.3 (predstavljene u sekciji 3.3.3) primenjene na kauzalno kompletne i slabo kompletne dnevnike događaja.

S obzirom da je kod modifikovane PM tehnike uvedena relacija indirekcije trebalo bi definisati i odnos između indirektnih zavisnosti aktivnosti i mesta njihovih povezivanja. Ukoliko postoji indirektna zavisnost između dve aktivnosti, zaključena na osnovu dnevnika događaja, onda ne postoji mesto u SWF-mreži koje povezuje te dve aktivnosti, što je iskazano sledećom teoremom:

Teorema 4.2. Neka je $N = (P, T, F)$ ispravna SWF-mreža i neka je L dnevnik događaja od N. Za bilo koje $a, b \in T$: $a >>_L b$ nalaže da $a\bullet \cap \bullet b = \emptyset$.

Dokaz. Pretpostavimo da $a >>_L b$ i $a\bullet \cap \bullet b \neq \emptyset$. Treba pokazati da ovo dovodi do kontradikcije i na taj način dokazati teoremu. Ako je $a\bullet \cap \bullet b \neq \emptyset$, to prema Teoremi 3.2 nalaže da $a >_L b$, zbog čega po definiciji 4.1 ne postoji $a >>_L b$, što je u suprotnosti sa polaznom pretpostavkom. Time je teorema dokazana.

Uzimajući u obzir odnos između relacije indirektne sledbenosti i mesta povezivanja u mreži, definisanje relacija dato definicijom 4.1, kao i definiciju dostupnog markiranja (definicija 2.3), indirektna zavisnost se može izraziti definicijom koja sledi.

Definicija 4.10. (*Indirektna zavisnost*) Neka je $N = (P, T, F)$ ispravna SWF-mreža sa ulaznim mestom i i izlaznim mestom o. Za bilo koje $a, b \in T$, postoji *indirektna zavisnost* između a i b akko važi:

1. $a\bullet \cap \bullet b = b\bullet \cap \bullet a = a\bullet \cap b\bullet = \bullet a \cap \bullet b = \emptyset$,
2. ne postoji neko dostupno markiranje $s \in [N, [i]\rangle$ tako da $(N, s)[a\rangle$ i $(N, s - \bullet a + a\bullet)[b\rangle$,
3. postoji neko dostupno markiranje $s \in [N, [i]\rangle$ tako da $(N, s)[a\rangle$, i postoji neko dostupno markiranje $s' \in [N, s - \bullet a + a\bullet\rangle$ tako da $(N, s')[b\rangle$.

S obzirom na relacije date definicijom 4.1, odnosi relacije paralelnosti $\|_L$ između aktivnosti i mesta povezivanja između njih, mogu se definisati sledećom teoremom:

Teorema 4. 3. Neka je $N = (P, T, F)$ ispravna SWF mreža, i neka je L dnevnik događaja od N.

1. Za bilo koje $a, b \in T$, ako je $a \|_L b$, onda važi $a\bullet \cap \bullet b = b\bullet \cap \bullet a = \emptyset$.
2. Za bilo koje $a, b \in T$, ako je $a \|_L b$, onda važi $a\bullet \cap b\bullet = \bullet a \cap \bullet b = \emptyset$.
3. Ako $a, b, t \in T$ i $a \rightarrow_L t$, $b \rightarrow_L t$ i $a \|_L b$, onda je $a\bullet \cap b\bullet \cap \bullet t = \emptyset$.
4. Ako $a, b, t \in T$ i $t \rightarrow_L a$, $t \rightarrow_L b$ i $a \|_L b$, onda je $\bullet a \cap \bullet b \cap t\bullet = \emptyset$.

Dokaz. Dokaz će biti dat za svaki pojedinačan slučaj.

1) Pretpostavimo da $a, b \in T$ i $a \|_L b$ i da je, na primer, $a\bullet \cap \bullet b \neq \emptyset$ i $b\bullet \cap \bullet a = \emptyset$. Treba dokazati da ovo dovodi do kontradikcije. Ako je $a\bullet \cap \bullet b \neq \emptyset$ i $b\bullet \cap \bullet a = \emptyset$, teorema 3.3 nalaže da je $a \rightarrow_L b$, što je u suprotnosti sa polaznom pretpostavkom da je $a \|_L b$.

2) Pretpostavimo da $a, b \in T$ i $a \|_L b$ i da je, na primer, $a\bullet \cap b\bullet = \emptyset$ i $\bullet a \cap \bullet b \neq \emptyset$. Treba dokazati da ovo dovodi do kontradikcije. Neka je $p_1 \in P$ jedino ulazno mesto aktivnosti a, odnosno $p_1 \in \bullet a$, i neka je $p_2 \in P$ jedino ulazno mesto aktivnosti b, odnosno $p_2 \in \bullet b$. Ako je $\bullet a \cap \bullet b \neq \emptyset$, to znači da je $p_1 \cap p_2 \neq \emptyset$ tj. $p_1 = p_2$, odnosno da aktivnosti a i b imaju isto ulazno mesto $p_1 = p_2 = p$, gde $p \in P$. Neka je (N, s) stanje mreže N neposredno pre okidanja aktivnosti a i b. U stanju s makirano je

mesto p čime su omogućene aktivnosti i a i b. Neka je s' stanje mreže posle izvršavanja b u stanju s, odnosno $(N, s)[b\rangle\ (N, s')$. Prilikom okidanja b troši se jedini token iz p, s obzirom na to da je N sigurna WF mreža. Posle okidanja b ne vraća token u p, s obzirom na to da je $a \|_L b$ a prema stavki 1 ove teoreme $b\bullet \cap \bullet a = \emptyset$, odnosno b ne proizvodi token za ulazno mesto za a. Kako a u stanju s' ostaje bez tokena u ulaznom mestu p, a nije u mogućnosti da se izvrši, što je u suprotnosti sa polaznom pretpostavkom u dokazu da je $a \|_L b$.

3) Pretpostavimo da $a \rightarrow_L t$, $b \rightarrow_L t$ i $a \|_L b$, i da je $a\bullet \cap b\bullet \cap \bullet t \neq \emptyset$. Treba dokazati da ovo dovodi do kontradikcije. Ako $a \rightarrow_L t$ to prema teoremi 3.1 nalaže da je $a\bullet \cap \bullet t \neq \emptyset$, što znači da postoji neko mesto $p_1 \in P$ takvo da $p_1 \in a\bullet$ i $p_1 \in \bullet t$, odnosno $p_1 = a\bullet \cap \bullet t$. Ako $b \rightarrow_L t$ onda prema teoremi 3.1: $b\bullet \cap \bullet t \neq \emptyset$, što znači da postoji neko mesto $p_2 \in P$ takvo da $p_2 \in b\bullet$ i $p_2 \in \bullet t$, odnosno $p_2 = b\bullet \cap \bullet t$. Ako je $a\bullet \cap b\bullet \cap \bullet t \neq \emptyset$ onda je $p_1 \cap p_2 \cap \{p_1, p_2\} \neq \emptyset$, što znači da je $p_1 = p_2$. Neka je $p_1 = p_2 = p$ i neka je (N, s) stanje mreže N neposredno pre okidanja aktivnosti t. Kako $a \rightarrow_L t$ i $b \rightarrow_L t$ u stanju s su izvršene i tranzicija a i tranzicija b, i to u proizvoljnom redosledu izvršavanja s obzirom na to da je $a \|_L b$. Da bi t bilo omogućeno u stanju s, zbog $a \rightarrow_L t$, a smešta token u mesto p, i zbog $b \rightarrow_L t$, b smešta token u mesto p, što dovodi do toga da u stanju s u mestu p postoje dva tokena. Kako je u pitanju sigurna WF mreža, prema definiciji 2.14, nije dozvoljeno da se u nekom mestu nađe više od jednog tokena, što znači da ne može biti $p_1 = p_2$, tako da je $p_1 \cap p_2 \cap \{p_1, p_2\} = \emptyset$, odnosno $a\bullet \cap b\bullet \cap \bullet t = \emptyset$, što je u suprotnosti sa polaznom pretpostavkom u dokazu da je $a\bullet \cap b\bullet \cap \bullet t \neq \emptyset$.

4) Pretpostavimo da $t \rightarrow_L a$, $t \rightarrow_L b$ i $a \|_L b$, i da je $\bullet a \cap \bullet b \cap t\bullet \neq \emptyset$. Treba dokazati da ovo dovodi do kontradikcije. Ako $t \rightarrow_L a$, teorema 3.1 nalaže da je $t\bullet \cap \bullet a \neq \emptyset$, što znači da postoji neko mesto $p_1 \in P$ takvo da $p_1 \in t\bullet$ i $p_1 \in \bullet a$, odnosno $p_1 = t\bullet \cap \bullet a$. Ako $t \rightarrow_L b$ teorema 3.1 nalaže da je $t\bullet \cap \bullet b \neq \emptyset$, što znači da postoji neko mesto $p_2 \in P$ takvo da $p_2 \in t\bullet$ i $p_2 \in \bullet b$, odnosno $p_2 = t\bullet \cap \bullet b$. Ako je $\bullet a \cap \bullet b \cap t\bullet \neq \emptyset$ onda je $p_1 \cap p_2 \cap \{p_1, p_2\} \neq \emptyset$, što znači da je $p_1 = p_2$. Neka je $p_1 = p_2 = p$ i neka je s stanje mreže N neposredno pre okidanja tranzicije t. Neka je s' stanje mreže posle izvršavanja t, odnosno $(N, s)[t\rangle\ (N, s')$. U stanju s' posle izvršenja t u mesto p smešten je jedan token. Kako je p ulazno mesto za a ($p = p_1 \in \bullet a$) a takođe i ulazno mesto za b ($p = p_2 \in \bullet b$) to su u stanju s' omogućeni i a i b. S obzirom na to da je $a \|_L b$ redosled izvršavanja a i b nije međusobno uslovljen. Ako se prvo izvrši $t \rightarrow_L a$, a će potrošiti jedini token iz p i neće vratiti token u p posle izvršenja s obzirom na to da je $a \|_L b$ a prema stavki 1 ove teoreme $a\bullet \cap \bullet b = \emptyset$, odnosno a ne proizvodi token za ulazno mesto za b, čime će b ostati onemogućeno i neće se izvršiti, što je u suprotnosti sa polaznom postavkom da je $a \|_L b$. Ako se prvo izvrši $t \rightarrow_L b$, b će potrošiti jedini token iz p i neće vratiti token u p posle izvršenja s obzirom da je $a \|_L b$ a prema stavki 1 ove teoreme $b\bullet \cap \bullet a = \emptyset$, odnosno b ne proizvodi token za ulazno mesto za a, čime će a ostati onemogućeno i neće se izvršiti, što je u suprotnosti sa polaznom postavkom da je $a \|_L b$. Da bi i a i b mogli da se izvrše posle t potrebno je da imaju zasebna ulazna mesta, odnosno da bude $p_1 \neq p_2$,

samim tim je i $p_1 \cap p_2 \cap \{p_1, p_2\} = \emptyset$, odnosno $\bullet a \cap \bullet b \cap t\bullet = \emptyset$, što je u suprotnosti sa polaznom pretpostavkom u dokazu da je $\bullet a \cap \bullet b \cap t\bullet \neq \emptyset$.

4.3.3 Problem visećih čvorova

Karakteristično za mreže dobijene iz slabo kompletnih dnevnika događaja je što često dolazi do pojave visećih čvorova u mreži, pri čemu neka aktivnost (čvor u P/T mreži) ostaje bez svog prethodnika i/ili sledbenika. Do pojave visećih čvorova u mreži dobijenoj na osnovu slabo kompletnog dnevnika događaja dolazi zbog velikog broja aktivnosti koje se mogu paralelno izvršavati i brzog otkrivanja paralelizma modifikovanom PM tehnikom. Osim toga, do pojave visećih čvorova u mreži dolazi i zbog svojstva slabo kompletnih dnevnika da se svi elementi relacije kauzalnosti ne mogu otkriti iz same evidencije zapisane u tragovima u dnevniku. Zbog toga se često dešava da se iz slabo kompletnog dnevnika događaja otkrije paralelizam između neke aktivnosti a i svih ostalih t_i, gde $a, t_i \in T$, sa kojima je bila u relaciji $a >_L t_i$ (ili $t_i >_L a$), a da se pri tom ne otkrije kauzalna relacija između aktivnosti a i bilo koje druge aktivnosti. U tom slučaju aktivnost a ostaje bez svog sledbenika (ili prethodnika) sa kojim bi trebalo da bude povezana u mreži.

Definicija 4.11. (*Viseći čvor*) Neka je $N = (P, T, F)$ ispravna SWF-mreža sa ulaznim mestom i i izlaznim mestom o, i neka je a aktivnost takva da $a \in T$ i $a \notin \{i, o\}$. Aktivnost $a \in T$ je *viseći čvor* u mreži akko ne postoji aktivnost $b \in T$ tako da $a\bullet \cap \bullet b \neq \emptyset$ i/ili $b\bullet \cap \bullet a \neq \emptyset$.

Kao direktna posledica definicije 4.11 proizilaze lema 4.4 i lema 4.5.

Lema 4.4. Neka je $N = (P, T, F)$ ispravna SWF-mreža, i neka je L dnevnik događaja od N. Ako za svako $a, b \in T$ gde $a >_L b$, postoji $b >_L a$ ili $b >>_L a$, tj. $a \parallel_L b$, onda a nema sledbenika, tj. $a\bullet \cap \bullet b = \emptyset$.

Dokaz: Prema teoremi 4.3 (stavka 1) ako je $a \parallel_L b$, onda je $a\bullet \cap \bullet b = \emptyset$.

Lema 4.5. Neka je $N = (P, T, F)$ ispravna SWF-mreža, i neka je L dnevnik događaja od N. Ako za svako $a, b \in T$ gde $a <_L b$, postoji $a >_L b$ ili $a >>_L b$, tj. $a \parallel_L b$, onda a nema prethodnika, tj. $b\bullet \cap \bullet a = \emptyset$.

Dokaz: Prema teoremi 4.3 (stavka 1) ako je $a \parallel_L b$, onda je $b\bullet \cap \bullet a = \emptyset$.

Definicijom 2.8 obuhvaćeno je svojstvo povezanosti WF-mreže koje podrazumeva povezanost svih čvorova u mreži, čime je eliminisana mogućnost postojanja visećih čvorova u WF-mreži. Za prevazilaženje problema postojanja visećih čvorova posmatraju se relacije u *otisku* dnevnika, i na osnovu tih relacija se mogu definisati pravila zaključivanja direktnih na osnovu indirektnih sledbenika i

prethodnika, čime se svakoj aktivnosti koja predstavlja viseći čvor može pronaći sledbenik i/ili prethodnik.

Pravilo 1. (*Zaključivanje direktnih na osnovu indirektnih sledbenika*) Neka je $N = (P, T, F)$ ispravna SWF mreža i neka je L_w slabo kompletan dnevnik događaja od N, i neka $a, b, c \in T$. Ako $a \Rightarrow_{Lw} c$, i ako postoji b tako da $b \rightarrow_{Lw} c$, pri čemu je $a \|_{Lw} b$, onda se umesto $a \Rightarrow_{Lw} c$ može zaključiti $a \rightarrow^{I}_{Lw} c$.

Dokaz: Neka je $a \Rightarrow_{Lw} c$, $b \rightarrow_{Lw} c$ i $a \|_{Lw} b$, treba dokazati da se umesto $a \Rightarrow_{Lw} c$ može zaključiti da $a \rightarrow^{I}_{Lw} c$. Neka je s stanje mreže neposredno pre okidanja a i b, tj. $(N, [i])[\sigma\rangle\ (N, s)$ gde je $\sigma = \langle t_1, \dots, t_{i-1}\rangle$, $i \in \{1,..., n - 2\}$ okidajuća sekvenca i $\sigma \in L_w$. Kako je $a \|_L b$, u stanju s su omogućeni i a i b, i mogu se izvršiti u međusobno nezavisnom redosledu. Neka u stanju s okine b i neka je s' stanje mreže posle okidanja b u stanju s, tj. $(N, s)[b\rangle\ (N, s')$. S obzirom na to da $b \rightarrow_{Lw} c$, onda prema teoremi 3.1 $b\bullet \cap \bullet c \neq \emptyset$, tj. postoji neko mesto $p_1 \in P$ takvo da $p_1 \in b\bullet$ i $p_1 \in \bullet c$, odnosno $p_1 = b\bullet \cap \bullet c$. S obzirom na to da je $a \|_{Lw} b$ to prema teoremi 4.3 (stavka 2) znači da $\bullet a \cap \bullet b = \emptyset$, odnosno a i b nemaju isto ulazno mesto. To znači da okidanje b u stanju s ne troši token iz ulaznog mesta za a, a kako je a bila omogućena i pre stanja s' i nije okinula ona je omogućena i u s'. Iako ima token u p_1, c se ne može izvršiti u stanju s' dok se ne izvrši a zbog postojanja relacije $a \Rightarrow_{Lw} c$. S obzirom na to da je a omogućena u s', posle okidanja a može se neposredno obaviti c, odnosno $a >_{Lw} c$. Kako ne postoji $c >_{Lw} a$, ni $c >>_{Lw} a$ (jer bi u tom slučaju prema definiciji 4.1 bilo $a \|_{Lw} c$), onda $a >_{Lw} c$ zapravo znači $a \rightarrow^{I}_{Lw} c$, što prema teoremi 3.1 nalaže da $a\bullet \cap \bullet c \neq \emptyset$, tj. postoji neko mesto $p_2 \in P$ takvo da $p_2 \in a\bullet$ i $p_2 \in \bullet c$, odnosno $p_2 = a\bullet \cap \bullet c$. Na taj način, umesto $a \Rightarrow_{Lw} c$ evidentirano iz dnevnika događaja L_w može se zaključiti $a \rightarrow^{I}_{Lw} c$. Kako je $a \|_{Lw} b$ to prema teoremi 4.3 (stavka 2) $a\bullet \cap b\bullet = \emptyset$, tj. $p_2 \cap p_1 = \emptyset$, odnosno $p_2 \neq p_1$. Zbog toga se u mrežu dodaje mesto $p_2 = a\bullet \cap \bullet c$. Pri tome je ispunjen i uslov iz teoreme 4.3 (stavka 3): ako $a \rightarrow_L c$, $b \rightarrow_L c$ i $a \|_L b$ onda $a\bullet \cap b\bullet \cap \bullet c = \emptyset$, tj. $p_2 \cap p_1 \cap \{p_1, p_2\} = \emptyset$.

Pravilo 2. (*Zaključivanje direktnih na osnovu indirektnih prethodnika*) Neka je $N = (P, T, F)$ ispravna SWF mreža i neka je L_w slabo kompletan dnevnik događaja od N, i neka $a, b, c \in T$. Ako $c \Leftarrow_{Lw} a$ i ako postoji b tako da $b \leftarrow_{Lw} a$, pri čemu je $c \|_{Lw} b$, onda se umesto $c \Leftarrow_{Lw} a$ može zaključiti $c \leftarrow^{I}_{Lw} a$.

Dokaz: Neka je $c \Leftarrow_{Lw} a$, $b \leftarrow_{Lw} a$ i $c \|_{Lw} b$, treba dokazati da se umesto $c \Leftarrow_{Lw} a$ može zaključiti da $c \leftarrow^{I}_{Lw} a$. Neka je s stanje mreže neposredno pre okidanja a, tj. $(N, [i])[\sigma\rangle\ (N, s)$ gde je $\sigma = \langle t_1, \dots, t_{i-1}\rangle$, $i \in \{1,..., n - 3\}$ okidajuća sekvenca i $\sigma \in L$. Neka je s' stanje mreže posle okidanja a u stanju s, tj. $(N, s)[a\rangle\ (N, s')$. Kako $b \leftarrow_{Lw} a$ (odnosno $a \rightarrow_{Lw} b$) to prema teoremi 3.1 $a \rightarrow_{Lw} b$ nalaže da je $a\bullet \cap \bullet b \neq \emptyset$, tj. postoji neko mesto $p_1 \in P$ takvo da $p_1 \in a\bullet$ i $p_1 \in \bullet b$, odnosno $p_1 = a\bullet \cap \bullet b$. U stanju s' p_1 je markirano, čime je b omogućeno. Kako $c \Leftarrow_{Lw} a$ (odnosno $a \Rightarrow_{Lw} c$) c se može izvršiti posle a, a kako je $c \|_{Lw} b$ to znači da se c i b mogu izvršavati u međusobno nezavisnom redosledu, što

znači da je i c u s' omogućeno. S obzirom da je $c \|_{Lw} b$, prema teoremi 4.3 (stavka 2) $c \|_{Lw} b$ nalaže da $\bullet c \cap \bullet b = \emptyset$, to znači da $p_1 \in \bullet b$ nije ulazno mesto za c, tj. $p_1 \notin \bullet c$. Ako se u stanju s' prvo izvrši c, to okidanje c u stanju s' ne troši token iz p_1. To znači da i posle okidanja c u stanju s' u p_1 ostaje token, čime je b omogućeno jer $p_1 \in \bullet b$, tako da se iza c može obaviti b. Kako se u stanju s' iza a može obaviti c pre b s obzirom da je $c \|_{Lw} b$, to znači da se može obaviti $a >_{Lw} c$. Kako ne postoji $c >_{Lw} a$, ni $c >>_{Lw} a$ (jer bi u tom slučaju prema definiciji 4.1 bilo $a \|_{Lw} c$) onda $a >_{Lw} c$ zapravo znači $a \rightarrow^{I}_{Lw} c$, odnosno $c \leftarrow^{I}_{Lw} a$. Na taj način umesto $c \Leftarrow_{Lw} a$ evidentirano iz dnevnika događaja L_w može se zaključiti $c \leftarrow^{I}_{Lw} a$. Prema teoremi 3.1 postojanje $a \rightarrow_{Lw} c$ nalaže da $a\bullet \cap \bullet c \neq \emptyset$, tj. postoji neko mesto p_2 takvo da $p_2 \in a\bullet$ i $p_2 \in \bullet c$, odnosno $p_2 = a\bullet \cap \bullet c$. Kako je $c \|_{Lw} b$ to prema teoremi 4.3 (stavka 2) $\bullet c \cap \bullet b = \emptyset$, tj. $p_2 \cap p_1 = \emptyset$, odnosno $p_2 \neq p_1$. Zbog toga se u mrežu dodaje mesto $p_2 = a\bullet \cap \bullet c$. Pri tome je ispunjen i uslov iz teoreme 4.3 (stavka 4): ako $a \rightarrow_L b$, $a \rightarrow_L c$ i $b \|_L c$ onda je $\bullet b \cap \bullet c \cap a\bullet = \emptyset$, tj. $p_1 \cap p_2 \cap \{p_1, p_2\} = \emptyset$.

Na osnovu Pravila 1 (Pravila 2) mogu se na osnovu otiska dnevnika zaključiti elementi kauzalne relacije $a \rightarrow^{I}_{Lw} c$ ($c \leftarrow^{I}_{Lw} a$) iz relacije $a \Rightarrow_{Lw} c$ ($c \Leftarrow_{Lw} a$) slabo kompletnog dnevnika događaja L_w, što doprinosi otkrivanju i onih kauzalnih relacija koje se ne mogu otkriti iz same evidencije u dnevniku. Kauzalna relacija koja se unosi u konačan izgled otiska dnevnika i nad kojom se primenjuje $\alpha^{\|}$-algoritam postaje: $\rightarrow_{Lf} = \rightarrow_{Lw} \cup \rightarrow^{I}_{Lw}$. Na ovaj način $\rightarrow_{Lf}$ postaje jednaka bazičnoj kauzalnoj relaciji, odnosno $\rightarrow_{Lf} = \rightarrow^{B}_{N}$, što omogućava otkrivanje originalne mreže iz slabo kompletnih dnevnika događaja. Što je veći broj elemenata $\rightarrow^{I}_{Lw}$ u relaciji $\rightarrow_{Lf}$, to je dnevnik događaja na osnovu kojeg se može konstruisati model oskudniji, odnosno sa manjim brojem zapisanih tragova.

Zbog toga, dnevnik događaja iz kojeg može da se dobije originalna mreža može biti nekompletan (s obzirom na uslov kompletnosti postavljen u definiciji 3.3), pa i kauzalno nekompletan s (obzirom na uslov kauzalne kompletnosti postavljen u definiciji 4.6), ali mora biti slabo kompletan (L_w) na način kako je to dato definicijom 4.8, i može biti veoma oskudan.

4.3.4 Primer otkrivanja originalne mreže iz slabo kompletnog dnevnika događaja

Posmatra se paralelan proces čiji je blok-strukturirani model prikazan na Slici 4.1 i dnevnik događaja L_2 sa zapisima dobijenim posle nekoliko izvršavanja procesa. I ovde su u dnevniku prikazani samo različiti tragovi, bez naznake broja njihovog pojavljivanja, s obzirom da učestalost njihovog pojavljivanja nije relavantna za ovo istraživanje.

$L_2 = [< a, b, c, d, e, f, g, h >, < a, f, g, c, e, d, b, h >]$

Bazična kauzalna relacija za mrežu na slici 4.1 jeste:

$\rightarrow^{B}_{N} = \{(a, b), (a, c), (a, f), (b, h), (c, d), (c, e), (d, h), (e, h), (f, g), (g, h)\}$

Otisak dnevnika događaja L_2 je dat u Tabeli 4.6.

	a	b	c	d	e	f	g	h
a	#	$\rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\rightarrow$	$\Rightarrow$	$\Rightarrow$
b	$\leftarrow$	#	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\rightarrow$
c	$\Leftarrow$	$\|\|$	#	$\rightarrow$	$\rightarrow$	$\|\|$	$\|\|$	$\Rightarrow$
d	$\Leftarrow$	$\|\|$	$\leftarrow$	#	$\|\|$	$\|\|$	$\|\|$	$\Rightarrow$
e	$\Leftarrow$	$\|\|$	$\leftarrow$	$\|\|$	#	$\|\|$	$\|\|$	$\Rightarrow$
f	$\leftarrow$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	#	$\rightarrow$	$\Rightarrow$
g	$\Leftarrow$	$\|\|$	$\|\|$	$\|\|$	$\|\|$	$\leftarrow$	#	$\rightarrow$
h	$\Leftarrow$	$\leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\Leftarrow$	$\leftarrow$	#

Tabela 4.6: *Otisak* dnevnika događaja L_2.

Iz otiska prikazanog u Tabeli 4.6 može se videti da je bazična kauzalna relacija dnevnika događaja L_2:

$\rightarrow_{L2} = \{(a, b), (a, f), (b, h), (c, d), (c, e), (f, g), (g, h)\}$

Može se primetiti da kauzalna relacija dnevnika događaja L_2 nije jednaka bazičnoj kauzalnoj relaciji za ovaj primer mreže, tj. $\rightarrow_{L2} \neq \rightarrow^{B}_{N}$, već važi: $\rightarrow^{B}_{N} \subset (\rightarrow_{L2} \cup \Rightarrow_{L2})$, $\rightarrow_{L2} \subset \rightarrow^{B}_{N}$, što L_2 čini slabo kompletnim logom.

Iz otiska se takođe može zaključiti da aktivnost c nema svog direktnog prethodnika (red koji se odnosi na c ne sadrži $\leftarrow$), a aktivnosti d i e nemaju direktne sledbenike (redovi koj se odnose na d i e ne sadrže $\rightarrow$), što zapravo znači da u mreži postoje viseći čvorovi.

S obzirom na pravila zaključivanja direktnih na osnovu indirektnih sledbenika (Pravilo 1) i prethodnika (Pravilo 2) u mreži sa visećim čvorovima, dobija se:

$c \Leftarrow_{L2} a, b \leftarrow_{L2} a, c \|_{L2} b$ onda $c \leftarrow^{I}_{L2} a$, ili $c \Leftarrow_{L2} a, f \leftarrow_{L2} a, c \|_{L2} f$ onda $c \leftarrow^{I}_{L2} a$ t.j. $a \rightarrow^{I}_{L2} c$
$d \Rightarrow_{L2} h, b \rightarrow_{L2} h, d \|_{L2} b$ onda $d \rightarrow^{I}_{L2} h$, ili $d \Rightarrow_{L2} h, g \rightarrow_{L2} h, d \|_{L2} g$ onda $d \rightarrow^{I}_{L2} h$
$e \Rightarrow_{L2} h, b \rightarrow_{L2} h, e \|_{L2} b$ onda $e \rightarrow^{I}_{L2} h$, ili $e \Rightarrow_{L2} h,\ g \rightarrow_{L2} h, e \|_{L2} g$ onda $e \rightarrow^{I}_{L2} h$

To znači da je: $\rightarrow^{I}_{L2} = \{(a, c), (d, h), (e, h)\}$, odnosno:
$\rightarrow_{L2f} = \rightarrow_{L2} \cup \rightarrow^{I}_{L2} = \{(a, b), (a, c), (a, f), (b, h), (c, d), (c, e), (d, h), (e, h), (f, g), (g, h)\}$

Može se primetiti da je sada: $\rightarrow_{L2f} = \rightarrow^{B}_{N}$.

Kao što je pokazano u prethodnom primeru prikazanom u sekciji 4.2.2, primenom $\alpha^{\|}$-algoritma na dnevnik događaja čija je relacija kauzalnosti jednaka bazičnoj kauzalnoj relaciji ($\rightarrow_{Lf} = \rightarrow^{B}_{N}$) dobijena mreža će biti jednaka originalnoj mreži.

Otkrivanje originalne mreže iz slabo kompletnog dnevnika događaja pomoću alata ProM

Na slici 4.14 dat je izgled slabo kompletnog dnevnika događaja L_2 iz razmatranog primera u .xes formatu.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file has been generated with the OpenXES library. It conforms -->
<!-- to the XML serialization of the XES standard for log storage and -->
<!-- management. -->
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<!-- OpenXES is available from http://www.openxes.org/ -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
  <extension name="Lifecycle" prefix="lifecycle" uri="http://www.xes-standard.org/lifecycle.xesext"/>
  <extension name="Organizational" prefix="org" uri="http://www.xes-standard.org/org.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/semantic.xesext"/>
  <global scope="trace">
    <string key="concept:name" value="__INVALID__"/>
  </global>
  <global scope="event">
    <string key="concept:name" value="__INVALID__"/>
    <string key="lifecycle:transition" value="complete"/>
  </global>
  <classifier name="MXML Legacy Classifier" keys="concept:name lifecycle:transition"/>
  <classifier name="Event Name" keys="concept:name"/>
  <classifier name="Resource" keys="org:resource"/>
  <string key="source" value="Rapid Synthesizer"/>
  <string key="concept:name" value="Slabo_kompletan_dnevnik_dogadjaja.xes"/>
  <string key="lifecycle:model" value="standard"/>
 <trace>
    <string key="concept:name" value="Case2.0"/>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
```

```
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
  </trace>
  <trace>
    <string key="concept:name" value="Case1.0"/>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:20:01.527+01:00"/>
      <string key="concept:name" value="A"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:21:01.527+01:00"/>
      <string key="concept:name" value="F"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
```

```
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:22:01.527+01:00"/>
      <string key="concept:name" value="G"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:01.527+01:00"/>
      <string key="concept:name" value="C"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:02.527+01:00"/>
      <string key="concept:name" value="E"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:03.527+01:00"/>
      <string key="concept:name" value="D"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:04.527+01:00"/>
      <string key="concept:name" value="B"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
    <event>
      <string key="org:resource" value="UNDEFINED"/>
      <date key="time:timestamp" value="2008-12-09T08:23:05.527+01:00"/>
      <string key="concept:name" value="H"/>
      <string key="lifecycle:transition" value="complete"/>
    </event>
  </trace>
</log>
```

Slika 4.14: Izgled dnevnika događaja L_2 u .xes formatu.

Postupak otkrivanja originalne mreže modifikovanom PM tehnikom i $\alpha^{\|}$-algoritmom iz slabo kompletnog dnevnika događaja alatom ProM sličan je postupku otkrivanja originalne mreže na osnovu kauzalno kompletnih dnevnika, opisanom u sekciji 4.2.2. Dakle, prvo se nad uvezenim kompletnim dnevnikom događaja startuje priključak: *Modified Alpha algorithm helper plug-in-find basic causality relation*, kao što je prikazano na slici 4.2, posle čega se na ekranu pojavljuje poruka: „*Basic causal relation successfully written to the file*“ (slika 4.3) ukoliko je bazična kauzalna relacija uspešno izdvojena.

Po dobijanju pomenute poruke na ekranu, iz uvezenog slabo kompletnog dnevnika događaja se može pronaći originalna mreža startovanjem priključka: *Mine for a Petri net using modified Alpha-*

algorithm, kao što je prikazano na slici 4.15, gde je uvezeni dnevnik L_2 dat pod nazivom *Slabo_kompletan_dnevnik_dogadjaja*.

U postupku otkrivanja originalnih mreža iz slabo kompletnih dnevnika događaja potrebno je izvršiti dodatne operacije, po pravilima zaključivanja direktnih na osnovu indirektnih sledbenika (Pravilo 1) i prethodnika (Pravilo 2). Za svaku indirektnu relaciju proverava se da li u skupu događaja postoji takav događaj koji zadovoljava neki od uslova iz pomenutih pravila. Ako se takav događaj pronađe, kolekciji *inferred_causal_relations* se dodaje upravo otkrivena kauzalna relacija.

Slika 4.15: Prikaz ProM okruženja sa uvezenim slabo kompletnim dnevnikom i aktivnim priključkom: *Mine for a Petri net using modified Alpha-algorithm*.

Zaključivanje direktnih na osnovu indirektnih sledbenika implementirano je u funkciji *weakly_completed_logs_find_causal_from_indirect_successor*, kao što je prikazano na slici 4.16

```
public void weakly_completed_logs_find_causal_from_indirect_successor(XLogInfo summary) {

    XEventClasses classes = summary.getEventClasses();
    XTrace trace = summary.getLog().get(0);

    Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
    while(it.hasNext()) {
        Pair<XEventClass, XEventClass> pair = it.next();
        for (int i=0; i<trace.size(); i++) {
            XEventClass e = classes.getClassOf(trace.get(i));
            if (!nodes_without_successor.contains(pair.getFirst())) continue;
            if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(e, pair.getSecond()))
                &&
                parallel_relationes.contains(new Pair<XEventClass, XEventClass>(e, pair.getFirst()))) {
                    inferred_causal_relations.add(pair);
                    System.out.println("Inferred causal successor " + pair.toString());
                    break;
            }
        }
    }
}
```

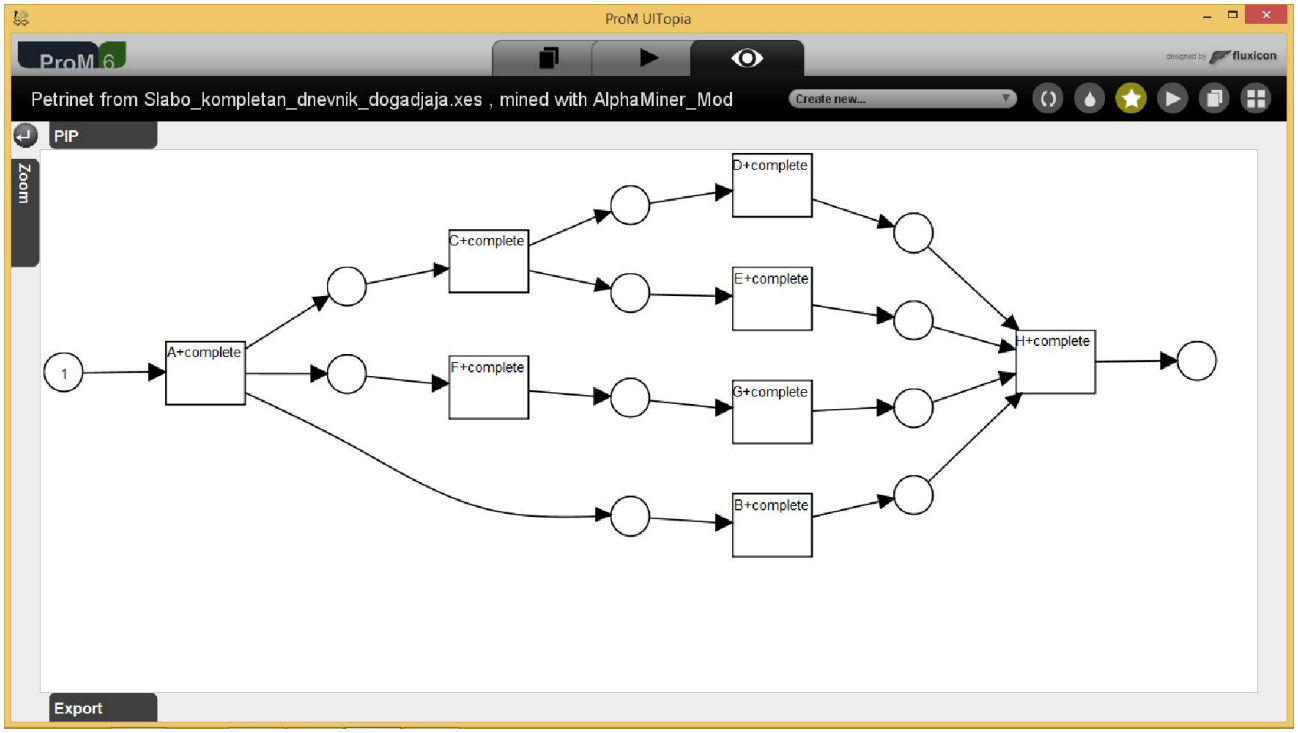
Slika 4.16: Zaključivanje direktnih na osnovu indirektnih sledbenika.

Zaključivanje direktnih na osnovu indirektnih prethodnika implementirano je u funkciji *weakly_completed_logs_find_causal_from_indirect_predecessor*, kao što je prikazano na slici 4.17.

```
public void weakly_completed_logs_find_causal_from_indirect_predecessor(XLogInfo summary) {

    XEventClasses classes = summary.getEventClasses();
    XTrace trace = summary.getLog().get(0);

    Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
    while(it.hasNext()) {
        Pair<XEventClass, XEventClass> pair = it.next();
        for (int i=0; i<trace.size(); i++) {
            XEventClass e = classes.getClassOf(trace.get(i));
            if (!nodes_without_predecessor.contains(pair.getSecond())) continue;
            if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getFirst(), e))
                &&
                parallel_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getSecond(), e))) {
                    inferred_causal_relations.add(pair);
                    System.out.println("Inferred causal predecessor " + pair.toString());
                    break;
            }
        }
    }
}
```

Slika 4.17: Zaključivanje direktnih na osnovu indirektnih prethodnika.

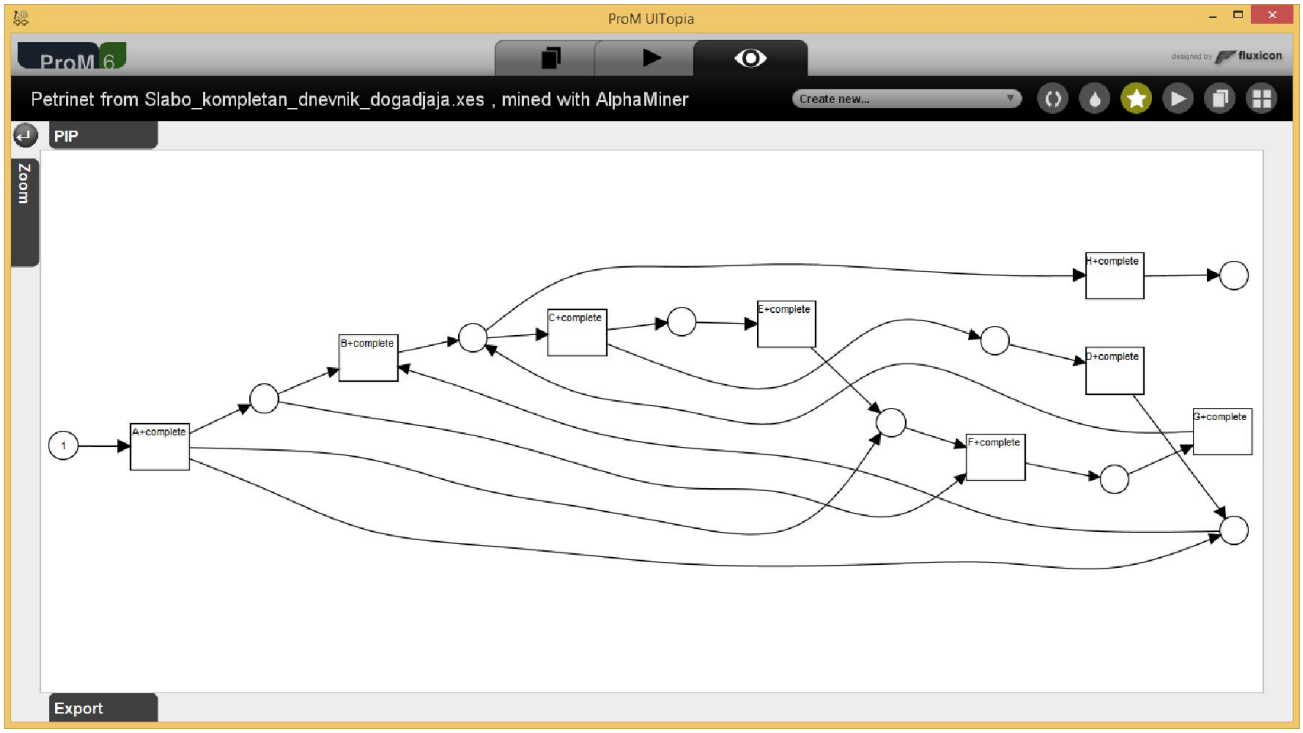
Kada se nad dnevnikom događaja L_2 startuje aktivan priključak *Mine for a Petri net using modified Alpha-algorithm*, dobija se model procesa prikazan u obliku Petri mreže (slika 4.18), koji je jednak mreži razmatranog primera procesa prikazanoj na slici 4.1.



Slika 4.18: Mreža dobijena na osnovu dnevnika L_2, primenom $\alpha^{\|}$-algoritma, tj. $N_2^{\|} = \alpha^{\|}(L_2)$.

Log L_2 nije kompletan, jer u relaciji $>_{L2}$ nedostaju elementi: $a > c$, $b > d$, $b > e$, $b > g$, $c > b$, $c > f$, $c > g$, $d > b$, $d > f$, $d > g$, $d > h$, $e > b$, $e > g$, $e > h$, $f > b$, $f > c$, $f > d$, $f > e$, $g > d$ i $g > e$, što bi potencijalno moglo da se obavi na osnovu modela procesa datog na slici 4.1 i dobijene WF-mreže $N_2^{||} = \alpha^{||}(L_2)$. Log L_2 je slabo kompletan, jer su ispunjeni uslovi iz definicije 4.8, tj.: $\rightarrow^B_N \subset (\rightarrow_{L2} \cup \Rightarrow_{L2})$ i $\rightarrow_{L2} \subset \rightarrow^B_N$.

Ukoliko se nad ovim dnevnikom događaja startuje aktivan priključak *Mine for a Petri net using Alpha-algorithm* (prikazan na slici 3.5), koji za pronalaženje originalne mreže koristi osnovni α-algoritam, dobija se model prikazan na slici 4.19. Može se primetiti da mreža $N_2 = \alpha(L_2)$ prikazana na slici 4.19 nije jednaka originalnoj mreži razmatranog primera prikazanoj na slici 4.1.



Slika 4.19: Mreža dobijena na osnovu dnevnika L_2, primenom α-algoritma, tj. $N_2 = \alpha^{||}(L_2)$.

4.3.5 Eksperimentalna analiza slabo kompletnih dnevnika događaja

Eksperimentalna analiza slabo kompletnih dnevnika događaja je takođe sprovedena na uzorku od 100 realnih primera paralelnih poslovnih procesa datih u Prilogu D, čije su karakteristike koje oslikavaju strukturu mreža date u Tabelama 4.2 i 4.3. Zadatak eksperimentalne analize u ovom delu istraživanja jeste da se ispita da li su slabo kompletni dnevnici događaja manji od kompletnih i kauzalno kompletnih dnevnika, i kakav je odnos njihovih veličina. U analizi su upoređivane veličine

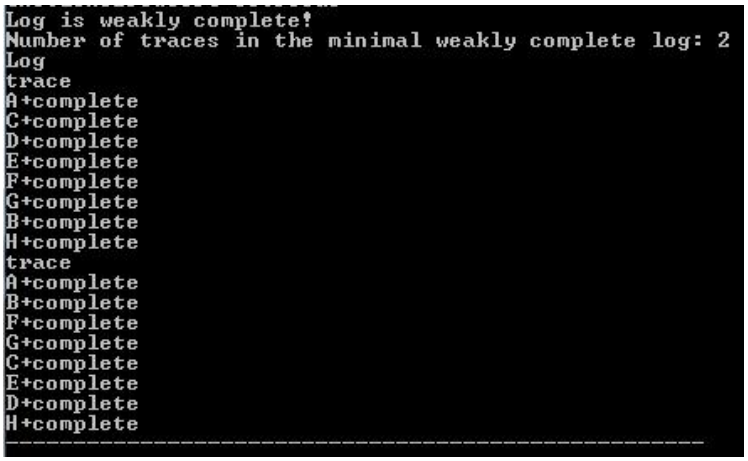
minimalnih slabo kompletnih dnevnika događaja sa veličinama minimalnih kompletnih i minimalnih kauzalno kompletnih dnevnika događaja, potrebnih za otkrivanje originalne mreže paralelnih procesa.

U tu svrhu takođe je korišćen priključak pod nazivom: *Modified Alpha-algorithm - Minimal Logs*, kao što je prikazano na slici 4.8. Pomoću ovog priključka, iz uvezenog kompletnog dnevnika događaja se izdvajaju kompletni, kauzalno kompletni i slabo kompletni dnevnici događaja sa najmanjim mogućim brojem tragova, i upoređuju se njihove međusobne veličine.

Izdvajanje minimalnih slabo kompletnih dnevnika događaja

Postupak izdvajanja minimalnih kompletnih, kauzalno kompletnih i slabo kompletnih dnevnika događaja iz uvezenog kompletnog dnevnika jeste zapravo postupak opisan u sekciji 4.2.3, gde se kao rezultat izvršavanja tog postupka na ekranu prikazuju veličine i izgledi dobijenih dnevnika događaja.

Posmatra se kompletan dnevnik događaja *L* prikazan u sekciji 4.2.3. Kada se na uvezeni dnevnik *L* startuje aktivan priključak: *Modified Alpha-algorithm - Minimal Logs* (slika 4.8), kao rezultat se, osim veličina i izgleda kauzalno kompletnog (slika 4.9) i kompletnog (slika 4.10) dnevnika, dobija i veličina i izgled slabo kompletnog dnevnika događaja (slika 4.20).



Slika 4.20: Rezultat izdvajanja minimalnog slabo kompletnog dnevnika događaja iz dnevnika *L*.

Iz dnevnika *L* i sa slika 4.9, 4.10 i 4.20 može se zaključiti sledeće: kompletan dnevnik događaja *L* ima 14 tragova, minimalni kompletan dnevnik izdvojen iz dnevnika *L* ima 6 tragova, minimalni kauzalno kompletan dnevnik izdvojen iz dnevnika *L* ima 4 traga, a minimalni slabo kompletan dnevnik izdvojen iz dnevnika *L* ima 2 traga.

Rezultati eksperimentalne analize

Opisani postupak izdvajanja minimalnih veličina kompletnog, kauzalno kompletnog i slabo kompletnog dnevnika (prikazan na primeru dnevnika *L*) i njihovo upoređivanje, u okviru

eksperimentalne analize, sprovedeni su na uzorku od 100 realnih primera paralelnih poslovnih procesa prikazanih u Prilogu D. U Tabeli 4.7 su prikazani rezultati izvršene uporedne analize minimalnih veličina kompletnih, kauzalno kompletnih i slabo kompletnih dnevnika događaja, potrebnih za otkrivanje originalnih mreža razmatranih primera.

Za lakše razumevanje tabela i slika koje slede koriste se navedene oznake, od kojih je većina data u sekciji 4.2.3:

- N_{mcl} označava broj tragova u minimalnom kompletnom dnevniku događaja
- N_{mccl} označava broj tragova u minimalnom kauzalno kompletnom dnevniku događaja
- N_{mwcl} označava broj tragova u minimalnom slabo kompletnom dnevniku događaja
- N_a označava ukupan broj aktivnosti u paralelnim granama
- N_b označava broj paralelnih grana u mreži

Broj primera	Broj tragova u dnevnicima događaja			N_{mwcl} manji od N_{mcl}	N_{mwcl} manji od N_{mccl}
	N_{mcl}	N_{mccl}	N_{mwcl}		
1	2	2	2	0,00%	0,00%
12	3	2	2	33,33%	0,00%
13	4	2	2	50,00%	0,00%
39	4	3	2	50,00%	33,33%
1	4	3	3	25,00%	0,00%
5	5	2	2	60,00%	0,00%
6	5	3	2	60,00%	33,33%
3	5	4	2	60,00%	50,00%
4	6	2	2	66,67%	0,00%
3	6	3	2	66,67%	33,33%
1	6	5	2	66,67%	60,00%
3	7	3	2	71,43%	33,33%
3	8	4	2	75,00%	50,00%
1	9	4	2	77,78%	50,00%
1	9	4	3	66,67%	25,00%
2	10	3	3	70,00%	0,00%
1	10	5	3	70,00%	40,00%
1	11	3	2	81,82%	33,33%
Ukupno				U proseku manji za	
100				52,74%	22,08%

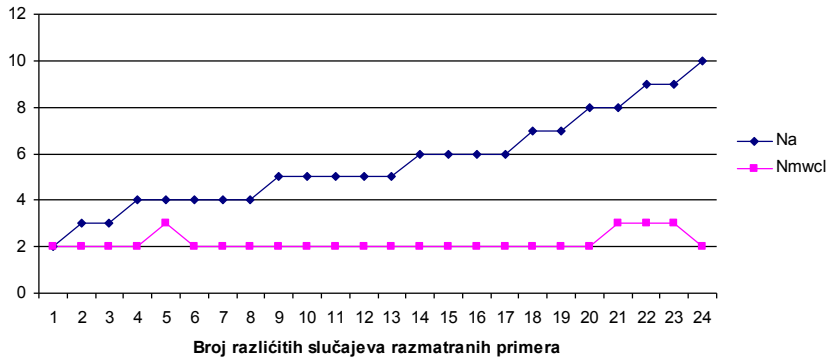
Tabela 4.7: Rezultati uporedne analize minimalnih veličina kompletnih, kauzalno kompletnih i slabo kompletnih dnevnika događaja.

Izvršena eksperimentalna analiza je pokazala da su veličine slabo kompletnih dnevnika događaja, iz kojih se mogu otkriti originalne mreže posmatranih paralelnih poslovnih procesa, manji ili (retko) jednaki veličini kompletnih dnevnika i kauzalno kompletnih dnevnika.

Iz Tabele 4.7 se može videti da su kod 99 primera (od razmatranih 100 primera) minimalni slabo kompletni dnevnici događaja manji od minimalnih kompletnih dnevnika u proseku za 52,74%, a kod jednog primera njihove vrednosti su jednake. Ni u jednom od posmatranih primera broj tragova u minimalnom kompletnom dnevniku događaja nije manji od broja tragova u minimalnom slabo kompletnom dnevniku.

Pored toga, iz Tabele 4.7 se može videti da su minimalni slabo kompletni dnevnici događaja manji od minimalnih kauzalno kompletnih dnevnika u proseku za 22,08%, posmatrano na uzorku od 100 primera. Pri tome su kod 38 primera (od razmatranih 100) njihove vrednosti jednake. Takođe, ni u jednom od posmatranih primera broj tragova u minimalnom kauzalnom kompletnom dnevniku događaja nije manji od broja tragova u minimalnom slabo kompletnom dnevniku.

Kao što je već rečeno u sekciji 4.2.3, rezultati eksperimentalne analize su pokazali da veličine kompletnih i kauzalno kompletnih dnevnika događaja iz kojih se mogu otkriti originalne mreže zavise od ukupnog broja aktivnosti u međusobno paralelnim granama (slika 4.11), kao i od broja paralelnih grana u mreži (slika 4.12), respektivno. Međutim, iz Tabele 4.8 i sa slike 4.21 se može videti da veličina slabo kompletnog dnevnika događaja ne zavisi od ukupnog broja aktivnosti u međusobno paralelnim granama.

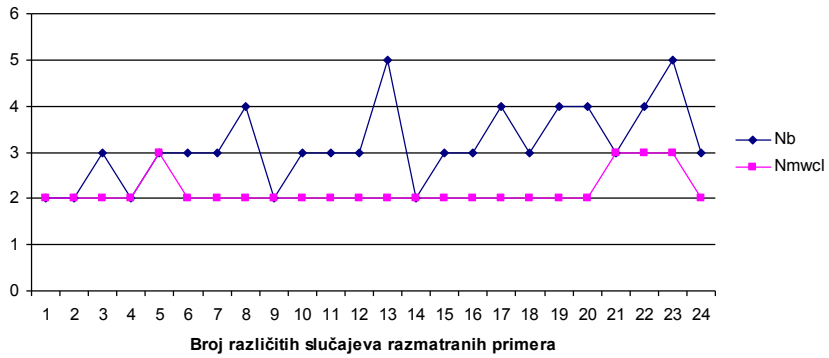


Slika 4.21: Odnos između ukupnog broja aktivnosti u paralelnim granama i broja tragova u minimalnom slabo kompletnom dnevniku događaja.

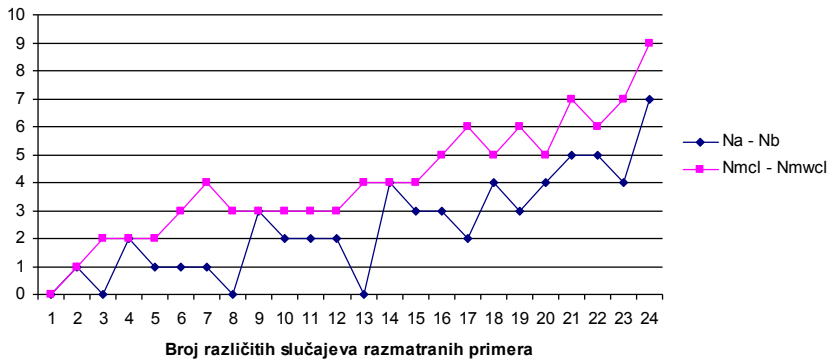
Iz Tabele 4.8 i sa slike 4.22 se može videti da veličina slabo kompletnog dnevnika događaja takođe ne zavisi od broja paralelnih grana u mreži. Razlika u veličinama minimalnih kompletnih i minimalnih slabo kompletnih dnevnika događaja, izražena u broju tragova, srazmerna je razlici između ukupnog broja aktivnosti u paralelnim granama i broju paralelnih grana u mreži (slika 4.23). Dok razlika između ukupnog broja aktivnosti u paralelnim granama i broja paralelnih grana u mreži ne utiče na odnos između veličina minimalnih kauzalno kompletnih i minimalnih slabo kompletnih dnevnika događaja (slika 4.24).

Broj primera	N_a	N_b	$N_a - N_b$	N_{mcl}	N_{mccl}	N_{mwcl}	$N_{mcl} - N_{mwcl}$	$N_{mccl} - N_{mwcl}$
1	2	2	0	2	2	2	0	0
12	3	2	1	3	2	2	1	0
39	3	3	0	4	3	2	2	1
13	4	2	2	4	2	2	2	0
1	4	3	1	4	3	2	2	1
3	4	3	1	5	3	2	3	1
1	4	3	1	6	3	2	4	1
2	4	4	0	5	4	2	3	2
4	5	2	3	5	2	2	3	0
1	5	3	2	5	2	2	3	0
3	5	3	2	5	3	2	3	1
1	5	3	2	5	4	2	3	2
1	5	5	0	6	5	2	4	3
4	6	2	4	6	2	2	4	0
2	6	3	3	6	3	2	4	1
1	6	3	3	7	3	2	5	1
1	6	4	2	8	4	2	6	2
2	7	3	4	7	3	2	5	1
2	7	4	3	8	4	2	6	2
1	8	4	4	9	4	2	7	2
2	8	3	5	10	3	3	7	0
1	8	4	4	9	4	3	6	1
1	9	4	5	9	4	3	6	1
1	9	5	4	10	5	3	7	2
1	10	3	7	11	3	2	9	1

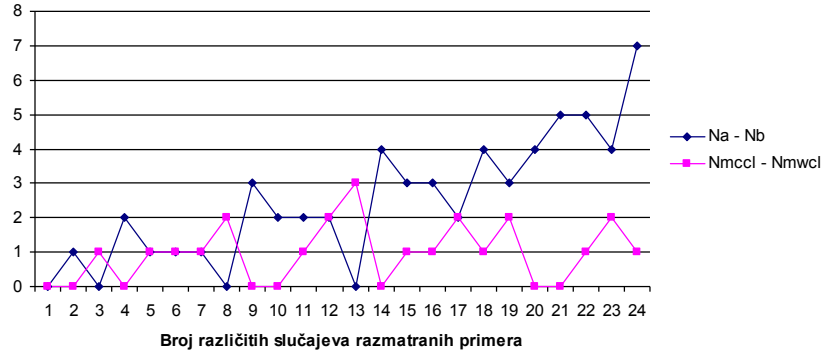
Tabela 4.8: Uticaj broja paralelnih grana u mreži i ukupnog broja aktivnosti u međusobno paralelnim granama na veličine kompletnih, kauzalno kompletnih i slabo kompletnih dnevnika događaja.



Slika 4.22: Odnos između broja paralelnih grana i broja tragova u minimalnom slabo kompletnom dnevnika događaja.



Slika 4.23: Odnos između razlike N_a - N_b i razlike N_{mcl} - N_{mwcl}.



Slika 4.24: Odnos između razlike N_a - N_b i razlike N_{mccl} - N_{mwcl}.

Test rangova (Wilcoxon-Mann-Whitney test)

Da bi se statistički potvrdile hipoteze da su slabo kompletni dnevnici događaja manji od kompletnih i kauzalno kompletnih dnevnika događaja, na rezultate prikazane u Tabeli 4.7 primeniće se test rangova, odnosno *Wilcoxon-Mann-Whitney test* [82].

Dokaz hipoteze da su minimalni slabo kompletni dnevnici manji od minimalnih kompletnih dnevnika događaja

Neka su: X = minimalni slabo kompletni dnevnici događaja, i Y = minimalni kompletni dnevnici događaja; treba testirati nultu hipotezu da su raspodele dva obeležja jednake, tj. H_0: $F_X = F_Y$, protiv alternativne hipoteze H_1: ”Minimalni slabo kompletni dnevnici X su manji od minimalnih kompletnih dnevnika Y ”.

Odgovarajuća kritična oblast C u tom slučaju je:

H_0	H_1	C
$F_X = F_Y$	X je manje od Y	$z_0 \leq -z_{0,5-\alpha}$

U tu svrhu, dobijeni rezultati eksperimentalne analize prikazani u Tabeli 4.7, predstavljaju se u obliku:

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4

X	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	3	3	3	2
Y	5	5	5	5	6	6	6	6	6	6	6	7	7	7	8	8	8	9	9	9	10	10	10	11

Sređivanjem rezultata u jedinstven neopadajući niz i dodeljivanjem 0 svakom elementu X, a 1 svakom elementu Y, dobija se niz:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1.

Na osnovu ovog niza testira se hipoteza H_0 protiv alternativne H_1, pri čemu se dobija:

$n_1 = 100, \quad n_2 = 100, \quad n = n_1 + n_2 = 200, \quad V = 0$

$E(V) = n_1 n_2 / 2 = 4900{,}5$

$D(V) = E(V)\,(\mathrm{n} + 1)/6 = 162533{,}25$

$z_0 = (V - E(V)) / [D(V)]^{1/2} = -12{,}155$

Za nivo značajnosti $\alpha = 0{,}05$ kritična oblast ovog testa je $C = (-\infty, -1{,}645]$. S obzirom na to da realizovana vrednost test statistike z_0 pripada kritičnoj oblasti C, to se nulta hipoteza H_0 odbacuje u korist alternativne H_1. Drugim rečima, na nivou značajnosti $\alpha = 0{,}05$ može se zaključiti da su minimalni slabo kompletni dnevnici događaja manji od minimalnih kompletnih dnevnika događaja.

Dokaz hipoteze da su minimalni slabo kompletni dnevnici manji od minimalnih kauzalno kompletnih dnevnika događaja

Neka su: X = minimalni slabo kompletni dnevnici događaja, i Y = minimalni kauzalno kompletni dnevnici događaja, treba testirati nultu hipotezu da su raspodele dva obeležja jednake, tj. H_0: $F_X = F_Y$, protiv alternativne hipoteze H_1: "Minimalni slabo kompletni dnevnici X su manji od minimalnih kauzalno kompletnih dnevnika Y ".

Odgovarajuća kritična oblast C u tom slučaju je:

H_0	H_1	C
$F_X = F_Y$	X je manje od Y	$z_0 \leq - z_{0,5-\alpha}$

U tu svrhu, dobijeni rezultati eksperimentalne analize prikazani u Tabeli 4.7, predstavljaju se u obliku:

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

X	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Y	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2
Y	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	3	3	3

X	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	2
Y	3	3	4	4	4	2	2	2	2	3	3	3	5	3	3	3	4	4	4	4	4	3	3	5	3

Sređivanjem rezultata u jedinstven neopadajući niz i dodeljivanjem 0 svakom elementu X, a 1 svakom elementu Y, dobija se niz:

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1.

Na osnovu ovog niza testira se hipoteza H_0 protiv alternativne H_1, pri čemu se dobija:

$n_1 = 100, \quad n_2 = 100, \quad n = n_1 + n_2 = 200$
$V = 35 + 35 + 35 + 35 + 35 = 175$
$E(V) = n_1 n_2/2 = 5000$
$D(V) = E(V)\,(n + 1)/6 = 167500$
$z_0 = (V - E(V))/\,[D(V)]^{1/2} = -11{,}789$

Za nivo značajnosti $\alpha = 0{,}05$, kritična oblast ovog testa je $C = (-\infty, -1{,}645]$. S obzirom na to da realizovana vrednost test statistike z_0 pripada kritičnoj oblasti C, to se nulta hipoteza H_0 odbacuje u korist alternativne H_1. Drugim rečima, na nivou značajnosti $\alpha = 0{,}05$ može se zaključiti da su minimalni slabo kompletni dnevnici događaja manji od minimalnih kompletnih dnevnika događaja.

V Praktična primena

5.1 Interaktivno kreiranje blok-strukturiranih modela paralelnih poslovnih procesa

Osnovna ideja primene paradigme PBD na domen razvoja informacionih sistema za upravljanje poslovnim procesima jeste da se na osnovu redosleda (scenarija) odigravanja aktivnosti poslovnog procesa dobije model procesa. Za ostvarivanje te ideje kod paralelnih poslovnih procesa, u ovom radu su korišćeni modifikovana PM tehnika i $\alpha^{\|}$ algoritam, primenjeni na slabo kompletne logove. U tu svrhu napravljen je sopstveni demonstracioni grafički korisnički interfejs, kojem se može pristupiti na adresi http://www.expertaya.com/ftn/primer1/graf.html.

Za predstavljanje načina rada algoritma na web stranici, u okviru grafičkog korisničkog interfejsa, korišćene su sledeće biblioteke:

- *Twitter bootstrap* (http://getbootstrap.com/) - stilovi i osnovni javascript kodovi za uobičajene grafičke elemente na formi i web stranici. *Bootstrap* je najpopularnije HTML, CSS i JS okruženje (engl. *framework*) za razvoj odgovarajućih, pre svega mobilnih, projekata na Internetu.

- *d3.js* (http://d3js.org/) - javascript biblioteka koja sadrži različite algoritme za vizuelizaciju na webu koji su implementirani korišćenjem HTML, SVG i CSS-a. D3.js koristi digitalne podatke da pokrene stvaranje i kontrolu dinamičkih i interaktivnih grafičkih oblika koji se pokreću u pretraživaču weba. U okviru kreiranog grafičkog korisničkog interfejsa, ova biblioteka je korišćena za grafički prikaz čvorova grafa aktivnosti, a za rasporedjivanje čvorova je korišćen *force layout* koji dolazi uz ovu biblioteku.

- *force layout* (http://bl.ocks.org/sathomas/11550728) - klasa algoritama za crtanje grafova sa puno estetike. Njihova svrha je postavljanje čvorova grafa u dvodimenzionalni ili trodimenzionalni prostor, tako da su sve strelice približno jednake dužine, dodeljivanjem „snage“ između skupa strelica i skupa čvorova zasnovane na njihovoj relativnoj poziciji.

Sama aplikacija je pisana u programskom jeziku JavaScript uz pomoć poznate biblioteke *jQuery* radi lakše manipulacije DOM elementima na stranici. Listing aplikacije se nalazi na kraju u Prilogu E (app.js).

U postupku interaktivnog kreiranja modela procesa, korisnik putem grafičkog korisničkog interfejsa demonstrira odigravanje različitih scenarija izvršavanja aktivnosti nekog poslovnog procesa, a sistem na osnovu toga kreira model procesa. Posle svakog odigranog scenarija, dobija se *kandidat-*

model koji predstavlja model svih odigranih scenarija izvršavanja aktivnosti, ali koji ne mora biti i model svih mogućih scenarija izvršavanja aktivnosti posmatranog procesa. Ideja je takođe i da sistem što pre, tj. na osnovu što je moguće manje odigranih scenarija, kreira model koji odgovara i svim mogućim scenarijima izvršavanja aktivnosti posmatranog procesa, koji je nazvan *konačnim modelom*. Da bi se to ostvarilo, sistem "navodi" korisnika da odigra scenario koji bi sistemu pružio najviše korisnih informacija za generalizaciju specifikacije modela, što je u opisu postupka demonstracije (koji sledi) detaljno objašnjeno.

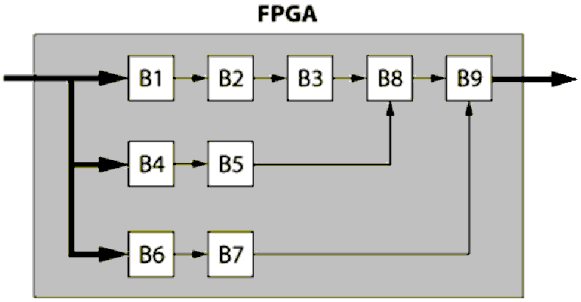
Pored zapisivanja relacija koje su demonstrirane od strane korisnika, korisnički interfejs neke relacije "zaključuje" kako bi ih generalizovao koristeći heuristiku i određena pravila, iako te relacije nisu „odigrane" u postupku demonstracije. Zbog toga kreirani korisnički interfejs pripada kategoriji inteligentnih interfejsa [83], [84], [85], kojoj pripadaju interfejsi koji poseduju neku komponentu veštačke inteligencije.

5.2 Primer kreiranja blok-strukturiranih modela paralelnih procesa pomoću demonstracije

5.2.1 Opis primera

Postupak demonstracije i kreiranja modela paralelnog poslovnog procesa biće prikazan na primeru procesa prikazanom na slici 5.1, koji predstavlja demonstrativni primer u ovom delu rada. Na slici 5.1 su prikazani FPGA procesori (engl. *Field-programmable gate array processors*) [86] koji omogućavaju paralelizam operacija, zbog čega su naročito pogodni za aplikacije sa zahtevima za obradom velike brzine. Tehnički gledano, FPGA procesori mogu da se koriste za rešavanje svakog problema koji se proračunava. Prednost korišćenja FPGA procesora leži u tome što su ponekad znatno brži za neke aplikacije zbog paralelne prirode i optimalnosti u pogledu broja kapija (engl. *gate*) koje se koriste za određeni proces. Konkretne oblasti primene FPGA procesora su: digitalna obrada signala, softverski definisani radio, ASIC (Application-Specific Integrated Circuit) izrada prototipova, obrada slika u medicini, kompjuterska vizija, prepoznavanje govora, kriptografija, bioinformatika, radio astronomija, detekcija metala, a u porastu je korišćenje i u drugim oblastima.

Do ovog primera procesa se došlo pretragom Interneta, i on se nalazi na adresi datoj pod [87]. Primer na slici 5.1 je jedan od primera paralelnih poslovnih procesa datih u Prilogu D [81], koji su korišćeni u eksperimentalnim analizama izloženim u sekciji 4.2.3 i sekciji 4.3.5.

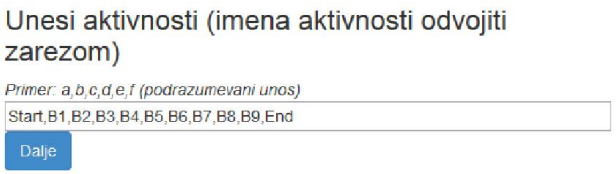


Slika 5.2: Ubrzavanje obrade podataka paralelizmom.

S obzirom na definisanje blok-strukturiranih WF-mreža i osobina paralelnih procesa, mreži na slici 5.1 odgovara stablo procesa: $\rightarrow (\wedge (\rightarrow (\wedge (\rightarrow (B1, B2, B3), \rightarrow (B4, B5)), B8), \rightarrow (B6, B7)), B9)$.

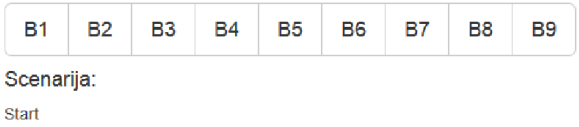
5.2.2 Postupak interaktivnog kreiranja modela

Startovanjem grafičkog korisničkog interfejsa na ekranu se dobija prostor za unos aktivnosti poslovnog procesa čiji se model kreira, kao što je prikazano na slici 5.2. Aktivnosti koje se izvršavaju kao prva i poslednja u toku izvršavanja procesa unose se na prvom i poslednjem mestu, respektivno, a sve ostale aktivnosti procesa se mogu uneti u proizvoljnom redosledu.



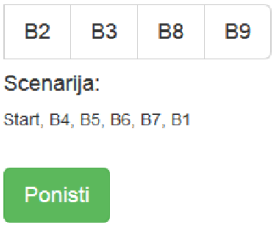
Slika 5.3: Izgled početnog prikaza grafičkog korisničkog interfejsa.

Pritiskom na tipku *Dalje* na ekranu se pojavljuje redosled aktivnosti predložen od strane sistema za sledeće odigravanje (s leva na desno), koji je jednak redosledu unosa aktivnosti (slika 5.3). Kako kod blok-strukturiranih paralelnih procesa postoji jedan ulaz i jedan izlaz, sistem će sam u redosled odigravanja uvek uvrstiti prvu i poslednju aktivnost procesa prema tome kako su one unete prvi put.



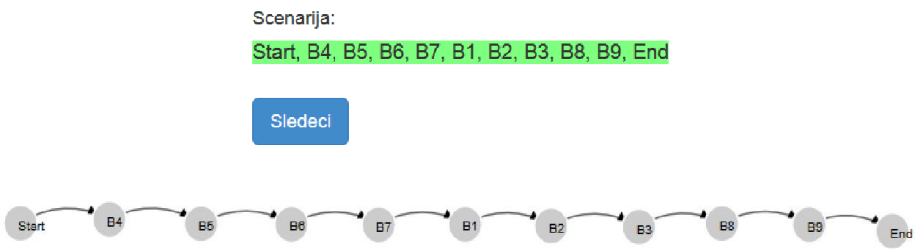
Slika 5.4: Ponuđeni redosled aktivnosti za odigravanje prvog scenarija.

Kod odigravanja prvog scenarija izvršavanja procesa, odabir redosleda preostalih aktivnosti (osim prve i poslednje) od strane korisnika je sasvim slobodan, i može biti nezavisan od redosleda koji je ponuđen (slika 5.4). Pritiskom na tipku *Poništi* može da se eliminiše prethodni izbor aktivnosti.



Slika 5.5: Odigravanje prvog scenarija.

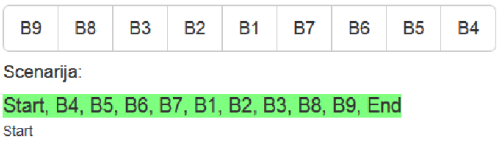
Po završetku odigravanja prvog scenarija na ekranu se dobija njegov izgled i model procesa dobijen na osnovu odigranog scenarija, koji predstavlja *kandidat-model* procesa (slika 5.5). Zelena boja kojom je markiran scenario znači da je odigravanje tog scenarija dovelo do promene u modelu, što je i očekivano s obzirom na to da je ovo prvi odigrani scenario i prvi kandidat-model.



Slika 5.6: Kandidat-model procesa koji odgovara prvom odigranom scenariju.

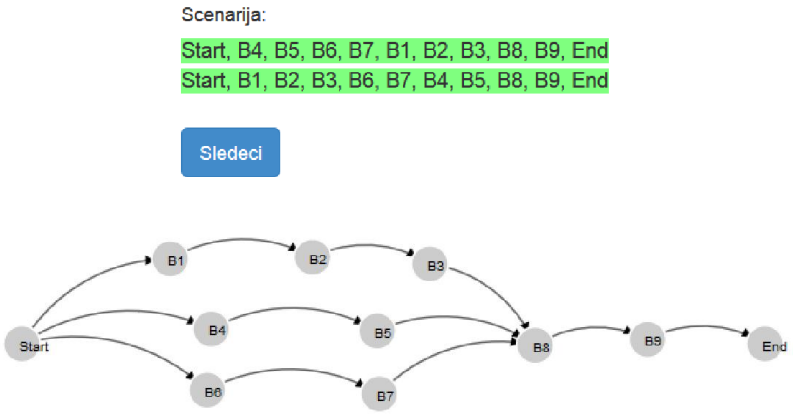
Pritiskom na tipku sa oznakom *Sledeći* sa slike 5.5 na ekranu se dobija redosled aktivnosti, predložen od strane sistema, za odigravanje u sledećem scenariju (slika 5.6). Zadatak korisnika nadalje je da se pridržava sledećeg pravila odabira aktivnosti iz redosleda predloženog od strane sistema:

Iz predloženog redosleda aktivnosti odabira se uvek prva aktivnost (počev s leva na desno) čije je izvršavanje moguće, i tako redom sve do poslednje aktivnosti.



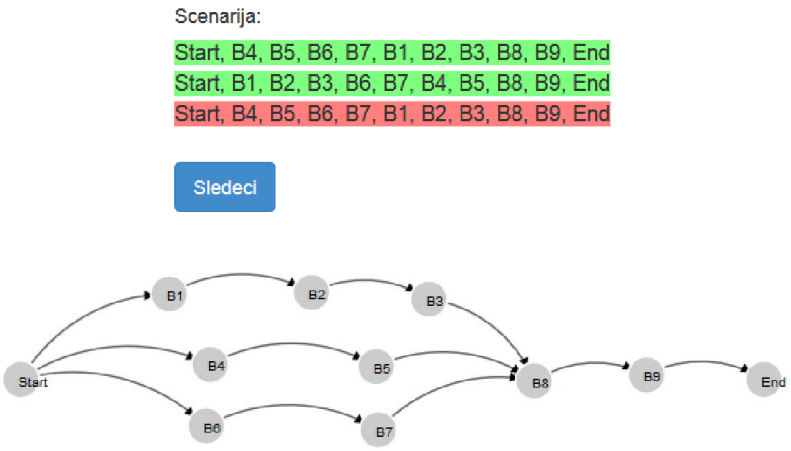
Slika 5.7: Predloženi redosled aktivnosti za odigravanje sledećeg scenarija.

Ukoliko je odigravanje novog scenarija dovelo do promena u modelu (kao što je to ovde slučaj), scenario se markira zelelnom bojom i dobija se novi kandidat-model procesa (slika 5.7) koji podržava oba odigrana scenarija.



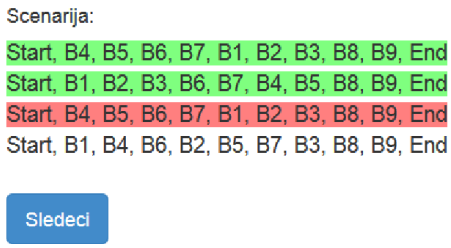
Slika 5.8: Izgled kandidat-modela procesa koji odgovara odigranim scenarijima.

Postupak se nastavlja pritiskom na tipku *Sledeći*, posle čega korisnik odigrava naredni scenario pridržavajući se pomenutog pravila odabira redosleda odigravanja aktivnosti iz redosleda predloženog od strane sistema. Ukoliko se prilikom odigravanja ponovi neki od već odigranih scenarija, ponovljeni scenario će biti markiran crvenom bojom (slika 5.8). S obzirom na to da kandidat-model odgovara svim odigranim scenarijima, ponovljeni scenario ne dovodi ni do kakvih promena u modelu.



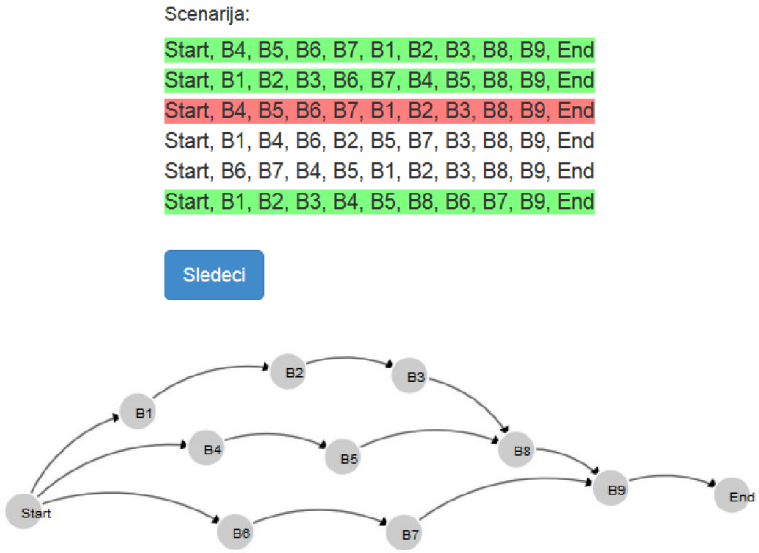
Slika 5.9: Ponovljeni scenario ne dovodi do promena u modelu.

Ponavljanje scenarija znači da će se, poštovanjem pomenutog pravila odabira aktivnosti iz predloženog redosleda, ponoviti i ostali scenariji, što može dovesti do sprečavanja odigravanja nekih drugačijih scenarija. Samim tim, to bi dovelo i do nemogućnosti eventualnog otkrivanja nekih drugih kandidat-modela koji im odgovaraju. Zbog toga je pojavljivanje ponovljenog scenarija indikacija korisniku da u sledećem odigravanju ne poštuje pomenuto pravilo odabira aktivnosti iz predloženog redosleda, nego da odigra scenario po sopstvenom slobodnom izboru (ako je moguće različit od onih koje je već odigrao), kao što je prikazano na slici 5.9.



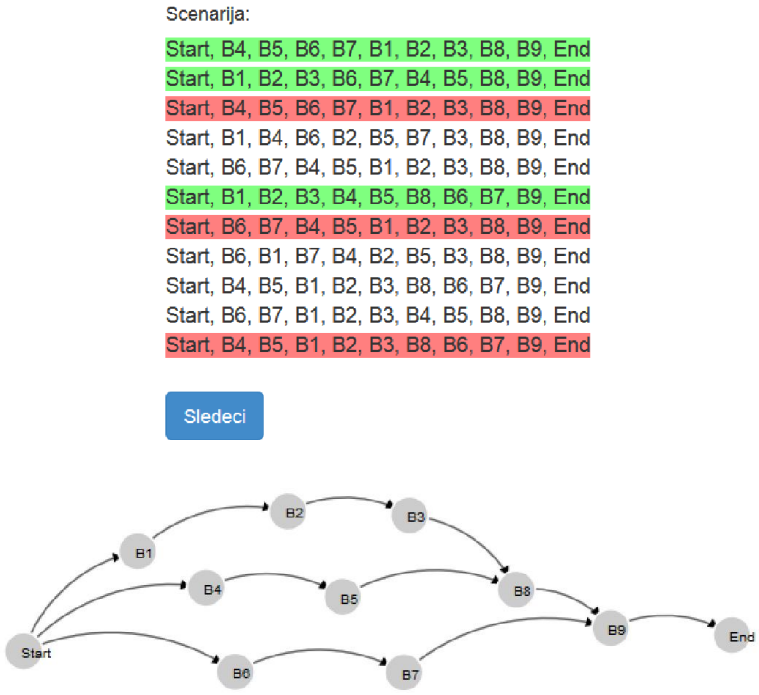
Slika 5.10: Posle ponovljenog scenarija, korisnik odigrava scenario po sopstvenom izboru, ne poštujući pravilo odabira aktivnosti iz predloženog redosleda.

Nadalje se postupak nastavlja uz poštovanje pravila odabira aktivnosti iz predloženog redosleda, sve do nove pojave ponovljenog scenarija (markiranog crvenom bojom), kada se odstupa od tog pravila i nastavlja na prethodno pomenuti način. Ukoliko neki od odigranih scenarija u tom postupku dovede do promene modela biće markiran zelenom bojom, i dobiće se novi kandidat-model, kao što je prikazano na slici 5.10.



Slika 5.11: Poslednji odigrani scenario doveo je do otkrivanja novog kandidat-modela.

Opisanim postupkom demonstracije korisniku se sugeriše da odigra što je moguće više različitih scenarija, kako bi se došlo do kreiranja *konačnog modela* procesa. Međutim, mnoštvo odigranih scenarija, bilo da su različiti ili ponovljeni, ne znači da će sigurno dovesti do nekakvih promena u modelu (slika 5.11). Sprovođenjem opisanog postupka demonstracije, u najvećem broju razmatranih slučajeva, prva pojava ponovljenog scenarija znači da odigravanje svih narednih scenarija neće dovesti do promena u modelu, odnosno da je poslednji dobijeni kandidat-model zapravo i konačan model procesa, o čemu će više reči biti kasnije.



Slika 5. 12: Konačan model razmatranog primera procesa.

5.2.3 Prikaz relacija modifikovane PM tehnike u postupku interaktivnog kreiranja modela

U toku izvršavanja demonstracije, posle svakog odigranog scenarija na ekranu se pojavljuje izgled svih relacija definisanih u modifikovanoj PM tehnici (slika 5.12), i *otisak* dnevnika događaja iz kojeg su relacije ustanovljene (slika 5.13), dobijeni na osnovu svih odigranih scenarija.

Relations:
>: (Start,B4),(B4,B5),(B5,B6),(B6,B7),(B7,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B1,B2),(B2,B3),(B3,B6),(B6,B7),(B7,B4),(B4,B5),(B5,B8),(B8,B9),(B9,End)
>>: (Start,B5),(Start,B6),(Start,B7),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B4,B6),(B4,B7),(B4,B1),(B4,B2),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B7),(B5,B1),(B5,B2),(B5,B3),(B5,B9),(B5,End),(B6,B1),(B6,B2),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B2),(B7,B3),(B7,B8),(B7,B9),(B7,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End),(Start,B2),(Start,B3),(Start,B6),(Start,B7),(Start,B5),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B6),(B1,B7),(B1,B4),(B1,B5),(B1,B8),(B1,B9),(B1,End),(B2,B6),(B2,B7),(B2,B4),(B2,B5),(B2,B8),(B2,B9),(B2,End),(B3,B7),(B3,B4),(B3,B5),(B3,B9),(B3,End),(B6,B4),(B6,B5),(B6,B8),(B6,B9),(B6,End),(B7,B5),(B7,B8),(B7,B9),(B7,End),(B4,B8),(B4,B9),(B4,End),(B5,B9),(B5,End),(B8,End)
<: (B4,Start),(B5,B4),(B6,B5),(B7,B6),(B1,B7),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B2,B1),(B3,B2),(B6,B3),(B7,B6),(B4,B7),(B5,B4),(B8,B5),(B9,B8),(End,B9)
<<: (B5,Start),(B6,Start),(B7,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B6,B4),(B7,B4),(B1,B4),(B2,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B7,B5),(B1,B5),(B2,B5),(B3,B5),(B9,B5),(End,B5),(B1,B6),(B2,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B2,B7),(B3,B7),(B8,B7),(B9,B7),(End,B7),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B6,Start),(B7,Start),(B5,Start),(B8,Start),(B9,Start),(End,Start),(B3,B1),(B6,B1),(B7,B1),(B4,B1),(B5,B1),(B8,B1),(B9,B1),(End,B1),(B6,B2),(B7,B2),(B4,B2),(B5,B2),(B8,B2),(B9,B2),(End,B2),(B7,B3),(B4,B3),(B5,B3),(B9,B3),(End,B3),(B4,B6),(B5,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B8,B7),(B9,B7),(End,B7),(B8,B4),(B9,B4),(End,B4),(B9,B5),(End,B5),(End,B8)
||: (B5,B6),(B6,B5),(B7,B1),(B1,B7),(B3,B6),(B6,B3),(B7,B4),(B4,B7),(B4,B6),(B6,B4),(B4,B1),(B1,B4),(B4,B2),(B2,B4),(B4,B3),(B3,B4),(B5,B7),(B7,B5),(B5,B1),(B1,B5),(B5,B2),(B2,B5),(B5,B3),(B3,B5),(B6,B1),(B1,B6),(B6,B2),(B2,B6),(B7,B2),(B2,B7),(B7,B3),(B3,B7)
→:(Start,B4),(B4,B5),(B6,B7),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B5,B8)
⇒:(Start,B5),(Start,B6),(Start,B7),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B4,B8),(B4,B9),(B4,End),(B5,B9),(B5,End),(B6,B8),(B6,B9),(B6,End),(B7,B8),(B7,B9),(B7,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End)
←:(B4,Start),(B5,B4),(B7,B6),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B8,B5)
⇐: (B5,Start),(B6,Start),(B7,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B8,B4),(B9,B4),(End,B4),(B9,B5),(End,B5),(B8,B6),(B9,B6),(End,B6),(B8,B7),(B9,B7),(End,B7),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8)
#: (Start,Start),(B1,B1),(B2,B2),(B3,B3),(B4,B4),(B5,B5),(B6,B6),(B7,B7),(B8,B8),(B9,B9),(End,End)
INFERRED →*i*: (B7,B8),(Start,B6)
INFERRED ←*i*: (B6,Start),(B8,B7)
NO_DIRECT →:B7,End
NO_DIRECT ←:Start,B6

Slika 5.13: Izgled relacija posle dva odigrana scenarija prikazana na slici 5.7.

Na slici 5.12 prikazan je izgled relacija koje su dobijene nakon dva odigrana scenarija posmatranog procesa, prikazana na slici 5.7. Osim relacija definisanih modifikovanom PM tehnikom, u prikazu je dato:

NO_DIRECT $\rightarrow$ - aktivnosti koje nemaju direktnog sledbenika,

NO_DIRECT $\leftarrow$ - aktivnosti koje nemaju direktnog prethodnika,

INFERRED $\rightarrow i$, INFERRED $\leftarrow i$ - relacije kauzalnosti koje nisu otkrivene na osnovu odigranih scenarija, već su naknadno zaključene iz otiska dnevnika događaja primenom Pravila 1 i/ili Pravila 2 za aktivnosti koje nemaju direktnog sledbenika i/ili prethodnika, respektivno.

Sa slike 5.12 se može videti da aktivnosti *B7* i *End* nemaju direktne sledbenike, a aktivnosti *B6* i *Start* nemaju direktne prethodnike. Kako su aktivnosti *Start* i *End* prva odnosno poslednja aktivnost u mreži, one će i ostati bez prethodnika odnosno sledbenika, respektivno. Međutim, iz relacija INFERRED sa slike se može videti da je na osnovu Pravila 1 i Pravila 2 dobijeno da je aktivnosti *B7* direktan sledbenik aktivnost *B8*, a aktivnosti *B6* direktan prethodnik je aktivnost *Start*.

Na slici 5.13 dat je izgled *otiska* dnevnika događaja nakon dva odigrana scenarija posmatranog procesa, koji su prikazani na slici 5.7. Pored ostalih relacija modifikovane metode dobijenih na osnovu odigranih scenarija, u otisku dnevnika su prikazane kauzalne relacije koje su dobijene ili na osnovu odigranih scenarija, ili su zaključene primenom Pravila 1 i Pravila 2, odnosno kauzalne relacije date u INFERRED, koje su označene sa $\rightarrow i$ i $\leftarrow i$.

	Start	B1	B2	B3	B4	B5	B6	B7	B8	B9	End
Start	#	→	⇒	⇒	→	⇒	→i	⇒	⇒	⇒	⇒
B1	←	#	→	⇒	||	||	||	||	⇒	⇒	⇒
B2	⇐	←	#	→	||	||	||	||	⇒	⇒	⇒
B3	⇐	⇐	←	#	||	||	||	||	→	⇒	⇒
B4	←	||	||	||	#	→	||	||	⇒	⇒	⇒
B5	⇐	||	||	||	←	#	||	||	→	⇒	⇒
B6	←i	||	||	||	||	||	#	→	⇒	⇒	⇒
B7	⇐	||	||	||	||	||	←	#	→i	⇒	⇒
B8	⇐	⇐	⇐	←	⇐	←	⇐	←i	#	→	⇒
B9	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	←	#	→
End	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	←	#

Slika 5.14: Otisak dnevnika događaja koga čine prva dva odigrana scenarija prikazana na slici 5.7.

Treba napomenuti da svaki kandidat-model, a samim tim i konačan model, ima različit izgled relacija modifikovane PM tehnike i različit otisak dnevnika događaja na osnovu kojeg je dobijen. Ipak, odigravanja scenarija koja ne dovode do promene modela mogu dovesti do promena u uzgledu relacija i otisku dnevnika, čak i po dobijanju konačnog modela. To se dešava u slučaju kada u izgledu relacija i u otisku dnevnika postoje INFERRED relacije koje učestvuju u formiranju bazične kauzalne relacije $\rightarrow^{B}_{N}$. Ukoliko se posle odigravanja nekog scenarija iz dobijenog traga zapisanog u dnevnku događaja mogu ustanoviti samo nove kauzalne relacije koje su prethodno dobijene kao INFERRED, a sve ostale relacije ostanu iste, to neće dovesti do promene u modelu jer će bazična kauzalna relacija $\rightarrow^{B}_{N}$ ostati ista. Međutim, izgledi relacija i otiska dnevnika događaja će se i u tom slučaju promeniti, jer će dobijene kauzalne relacije preći iz $\rightarrow i$ u $\rightarrow$.

5.3 Rezultati eksperimentalne analize kreiranja modela procesa pomoću demonstracije

Eksperimentalna analiza interaktivnog kreiranja modela procesa je takođe sprovedena na uzorku od 100 realnih primera paralelnih poslovnih procesa datih u Prilogu D [81], čije su karakteristike koje oslikavaju strukturu mreža date u Tabelama 4.2 i 4.3.

Eksperimentalna analiza se sastojala u višestrukom različitom odigravanju scenarija izvršavanja procesa datih u Prilogu D, praćenju rezultata dobijenih tim odigravanjima, i donošenju zaključaka na osnovu dobijenih rezultata. Prilikom višestrukog različitog odigravanja razmatranih paralelnih poslovnih procesa, primećeno je da se posle svakog odigranog scenarija dobija kandidat-model koji podržava sve odigrane scenarije. Takođe je primećeno i da se posle odigravanja niza scenarija kandidat-model više ne menja, odnosno da poslednji dobijeni kandidat-model podržava sve scenarije koji su odigrani kao i one koji bi se eventualno mogli odigrati posmatranim procesom, čime on postaje konačan model procesa.

Od posebnog interesa je bilo da se u procesu izvršavanja eksperimentalne analize razmotri posle koliko odigranih scenarija se može doći do konačnog modela i od čega to zavisi. Dobijeni rezultati su pokazali da broj odigranih scenarija neophodnih za dobijanje konačnog modela zavisi od redosleda odigravanja aktivnosti u prvom scenariju, s obzirom na strukturu mreže. U tom smislu razlikuju se 2 slučaja:

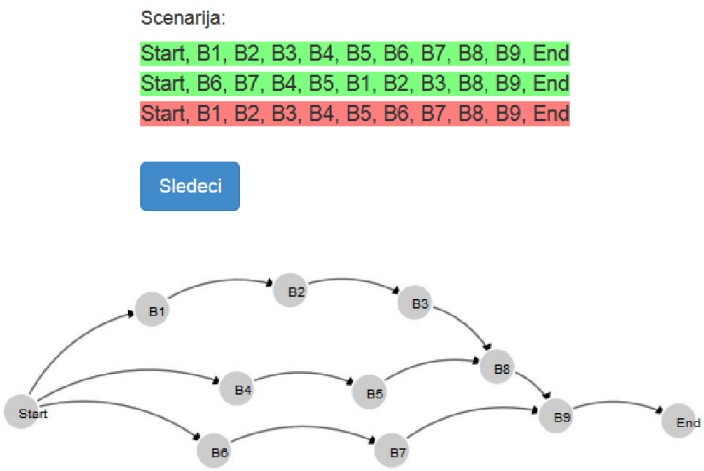
Slučaj 1. U prvom odigranom scenariju redosled izvršavanja aktivnosti je takav da se prvo odigravaju sve aktivnosti jedne grane, pa zatim sve aktivnosti druge grane paralelne sa prethodnom, i tako redom dok se ne izvrše sve aktivnosti iz svih međusobno paralelnih grana.

Slučaj 2. U prvom odigranom scenariju redosled izvršavanja aktivnosti je takav da se naizmenično izvršavaju aktivnosti iz različitih paralelnih grana.

Slučaj 1

U slučaju 1, dobijeni rezultati sprovedene eksperimentalne analize su pokazali da se kod 99 primera (od ukupnih 100primera) konačan model uvek dobija posle samo 2 različita odigrana scenarija (markirana zelenom bojom). To znači da je prva pojava ponovljenog scenarija (markiranog crvenom bojom) bila indikacija da je dobijeni kandidat-model zapravo i konačan model. Samo kod jednog primera je bilo neophodno da se posle prve pojave ponovljenog scenarija nastavi sa postupkom, datog u opisu toka demonstracije, da bi se došlo do konačnog modela.

Demonstrativni primer prikazan na slici 5.1 takođe može da se dobije posle samo dva različita odigrana scenarija, ako bi se kod odigravanja prvog scenarija pridržavali preporuka datih u slučaju 1, kao što je prikazano na slici 5.14. Sa slike 5.14 se vidi da je pojava prvog ponovljenog scenarija (markiranog crvenom bojom) indikacija da je otkriven konačan model razmatranog procesa.



Slika 5.15: Konačan model demonstrativnog primera procesa dobijen odigravanjem prema Slučaju 1.

Sa slike 5.14 se takođe može videti da je izgled konačnog modela razmatranog procesa isti sa modelom prikazanim na slici 5.11, iako je za njihovo dobijanje bilo potrebno odigrati različit broj scenarija.

Slučaj 2

U Slučaju 2 dobijeni rezultati sprovedene eksperimentalne analize su pokazali da se kod 88 primera (od ukupnih 100 primera) konačan model procesa uvek dobija posle samo 3 različita uzastopno odigrana scenarija (markirana zelenom bojom). To znači da je prva pojava ponovljenog scenarija (markiran crvenom bojom) bila indikacija da je kandidat-model zapravo i konačan model procesa. Kod 12 primera je bilo neophodno da se posle prve pojave ponovljenog scenarija nastavi sa postupkom, datog u opisu toka demonstracije, da bi se došlo do konačnog modela procesa. Ukupan broj odigranih scenarija neophodnih za dobijanje konačnog modela kod ovih 12 primera procesa zavisi od proizvoljnog odigravanja scenarija od strane korisnika u opisanom postupku, i kreće se od 5 scenarija pa naviše.

Kao što se iz prikazanog postupka demonstracije može videti, kada se demonstrativni primer procesa prikazan na slici 5.1 odigrava poštujući preporuke date u Slučaju 2, pojava prvog ponovljenog scenarija nije bila indikacija da je došlo do otkrivanja konačnog modela. Tek posle šestog odigranog scenarija, pojava drugog ponovljenog scenarija značila je da je došlo do otkrivanja konačnog modela razmatranog procesa. To znači da demonstrativni primer procesa sa slike 5.1 pripada grupi od 12 primera procesa iz eksperimentalne analize po preporukama Slučaja 2, kod kojih je za dobijanje konačnog modela procesa bilo neophodno odigrati još različitih scenarija posle dobijanja prvog ponovljenog scenarija.

Poređenja radi, na slikama 5.15 i 5.16 su dati izgledi relacija dobijenih na osnovu dnevnika događaja prikazanih na slikama 5.11 (dobijen prema preporukama iz Slučaja 1) i 5.14 (dobijen prema preporukama iz Slučaja 2), respektivno.

Relations:

>: (Start,B4),(B4,B5),(B5,B6),(B6,B7),(B7,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B1,B2),(B2,B3),(B3,B6),(B6,B7),(B7,B4),(B4,B5),(B5,B8),(B8,B9),(B9,End),(Start,B4),(B4,B5),(B5,B6),(B6,B7),(B7,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B1,B4),(B4,B6),(B6,B2),(B2,B5),(B5,B7),(B7,B3),(B3,B8),(B8,B9),(B9,End),(Start,B6),(B6,B7),(B7,B4),(B4,B5),(B5,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B1,B2),(B2,B3),(B3,B4),(B4,B5),(B5,B8),(B8,B6),(B6,B7),(B7,B9),(B9,End),(Start,B6),(B6,B7),(B7,B4),(B4,B5),(B5,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B6),(B6,B1),(B1,B7),(B7,B4),(B4,B2),(B2,B5),(B5,B3),(B3,B8),(B8,B9),(B9,End),(Start,B4),(B4,B5),(B5,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B6),(B6,B7),(B7,B9),(B9,End),(Start,B6),(B6,B7),(B7,B1),(B1,B2),(B2,B3),(B3,B4),(B4,B5),(B5,B8),(B8,B9),(B9,End),(Start,B4),(B4,B5),(B5,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B6),(B6,B7),(B7,B9),(B9,End)

>>: (Start,B5),(Start,B7),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B4,B7),(B4,B1),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B2),(B7,B8),(B7,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End),(Start,B2),(Start,B3),(Start,B7),(Start,B5),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B6),(B1,B5),(B1,B8),(B1,B9),(B1,End),(B2,B6),(B2,B7),(B2,B4),(B2,B8),(B2,B9),(B2,End),(B3,B7),(B3,B5),(B3,B9),(B3,End),(B6,B4),(B6,B5),(B6,B8),(B6,B9),(B6,End),(B7,B5),(B7,B8),(B7,End),(B4,B8),(B4,B9),(B4,End),(B5,B9),(B5,End),(B8,End),(Start,B5),(Start,B7),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B2),(B7,B8),(B7,End),(B4,B1),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B8,End),(Start,B2),(Start,B3),(Start,B5),(Start,B8),(Start,B7),(Start,B9),(Start,End),(B1,B3),(B1,B5),(B1,B8),(B1,B6),(B1,B9),(B1,End),(B2,B4),(B2,B8),(B2,B6),(B2,B7),(B2,B9),(B2,End),(B3,B5),(B3,B7),(B3,B9),(B3,End),(B4,B8),(B4,B7),(B4,B9),(B4,End),(B5,B9),(B5,End),(B8,B7),(B8,End),(B6,B9),(B6,End),(B7,End),(Start,B7),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B6,B4),(B6,B5),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B5),(B7,B2),(B7,B8),(B7,End),(B4,B1),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B6,B4),(B6,B5),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B1,B5),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B7,B2),(B7,B5),(B7,B8),(B7,End),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B2,B8),(B2,B9),(B2,End),(B5,B9),(B5,End),(B3,B9),(B3,End),(B8,End),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B7),(Start,B9),(Start,End),(B4,B1),(B4,B3),(B4,B8),(B4,B7),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B1,B3),(B1,B8),(B1,B6),(B1,B9),(B1,End),(B2,B8),(B2,B6),(B2,B7),(B2,B9),(B2,End),(B3,B7),(B3,B9),(B3,End),(B8,B7),(B8,End),(B6,B9),(B6,End),(B7,End),(Start,B7),(Start,B2),(Start,B3),(Start,B5),(Start,B8),(Start,B9),(Start,End),(B6,B3),(B6,B4),(B6,B5),(B6,B8),(B6,B9),(B6,End),(B7,B2),(B7,B5),(B7,B8),(B7,End),(B4,B1),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B8,End),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B7),(Start,B9),(Start,End),(B6,B4),(B6,B5),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B5),(B7,B2),(B7,B8),(B7,End),(B1,B3),(B1,B5),(B1,B8),(B1,B9),(B1,End),(B2,B4),(B2,B8),(B2,B6),(B2,B7),(B2,B9),(B2,End),(B3,B5),(B3,B7),(B3,B9),(B3,End),(B4,B8),(B4,B7),(B4,B9),(B4,End),(B5,B9),(B5,End),(B8,B7),(B8,End),(B6,B9),(B6,End),(B7,End),(Start,B7),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B5),(B1,B8),(B1,B6),(B1,B9),(B1,End),(B2,B4),(B2,B8),(B2,B6),(B2,B7),(B2,B9),(B2,End),(B3,B5),(B3,B7),(B3,B9),(B3,End),(B4,B8),(B4,B9),(B4,End),(B5,B9),(B5,End),(B8,End),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B7),(Start,B9),(Start,End),(B4,B1),(B4,B3),(B4,B8),(B4,B7),(B4,B9),(B4,End),(B5,B2),(B5,B9),(B5,End),(B1,B3),(B1,B8),(B1,B6),(B1,B9),(B1,End),(B2,B8),(B2,B6),(B2,B7),(B2,B9),(B2,End),(B3,B7),(B3,B9),(B3,End),(B8,B7),(B8,End),(B6,B9),(B6,End),(B7,End)

<: (B4,Start),(B5,B4),(B6,B5),(B7,B6),(B1,B7),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B2,B1),(B3,B2),(B6,B3),(B7,B6),(B4,B7),(B5,B4),(B8,B5),(B9,B8),(End,B9),(B4,Start),(B5,B4),(B6,B5),(B7,B6),(B1,B7),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B4,B1),(B6,B4),(B2,B6),(B5,B2),(B7,B5),(B3,B7),(B8,B3),(B9,B8),(End,B9),(B6,Start),(B7,B6),(B4,B7),(B5,B4),(B1,B5),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B2,B1),(B3,B2),(B4,B3),(B5,B4),(B8,B5),(B6,B8),(B7,B6),(B9,B7),(End,B9),(B6,Start),(B7,B6),(B4,B7),(B5,B4),(B1,B5),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B6,Start),(B1,B6),(B7,B1),(B4,B7),(B2,B4),(B5,B2),(B3,B5),(B8,B3),(B9,B8),(End,B9),(B4,Start),(B5,B4),(B1,B5),(B2,B1),(B3,B2),(B8,B3),(B6,B8),(B7,B6),(B9,B7),(End,B9),(B6,Start),(B7,B6),(B1,B7),(B2,B1),(B3,B2),(B4,B3),(B5,B4),(B8,B5),(B9,B8),(End,B9),(B4,Start),(B5,B4),(B1,B5),(B2,B1),(B3,B2),(B8,B3),(B6,B8),(B7,B6),(B9,B7),(End,B9)

<<: (B5,Start),(B7,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B7,B4),(B1,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B2,B7),(B8,B7),(End,B7),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B7,Start),(B5,Start),(B8,Start),(B9,Start),(End,Start),(B3,B1),(B6,B1),(B5,B1),(B8,B1),(B9,B1),(End,B1),(B6,B2),(B7,B2),(B4,B2),(B8,B2),(B9,B2),(End,B2),(B7,B3),(B5,B3),(B9,B3),(End,B3),(B4,B6),(B5,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B8,B7),(End,B7),(B8,B4),(B9,B4),(End,B4),(B5,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B7,B2),(B8,B2),(B9,B2),(End,B2),(B9,B5),(End,B5),(B8,B7),(End,B7),(B9,B3),(End,B3),(End,B8),(B7,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B4,B6),(B5,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B2,B7),(B8,B7),(End,B7),(B1,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B5,Start),(B8,Start),(B7,Start),(B9,Start),(End,Start),(B4,B6),(B5,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B2,B7),(B8,B7),(End,B7),(B1,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B5,Start),(B8,Start),(B7,Start),(B9,Start),(End,Start),(B3,B1),(B5,B1),(B8,B1),(B6,B1),(B9,B1),(End,B1),(B4,B2),(B8,B2),(B6,B2),(B7,B2),(B9,B2),(End,B2),(B5,B3),(B7,B3),(B9,B3),(End,B3),(B8,B4),(B7,B4),(B9,B4),(End,B4),(B9,B5),(End,B5),(B7,B6),(End,B8),(B9,B6),(End,B6),(End,B7),(B7,Start),(B5,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B4,B6),(B5,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B2,B7),(B8,B7),(End,B7),(B1,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B7,Start),(B2,Start),(B5,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B4,B6),(B5,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B5,B1),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B2,B7),(B5,B7),(B8,B7),(End,B7),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B8,B2),(B9,B2),(End,B2),(B9,B5),(End,B5),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B8,Start),(B7,Start),(B9,Start),(End,Start),(B1,B4),(B3,B4),(B8,B4),(B7,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B6,B1),(B9,B1),(End,B1),(B8,B2),(B6,B2),(B7,B2),(B9,B2),(End,B2),(B7,B3),(B9,B3),(End,B3),(B7,B8),(End,B8),(B9,B6),(End,B6),(End,B7),(B7,Start),(B2,Start),(B3,Start),(B5,Start),(B8,Start),(B9,Start),(End,Start),(B6,B3),(B4,B6),(B5,B6),(B8,B6),(B9,B6),(End,B6),(B2,B7),(B5,B7),(B8,B7),(End,B7),(B3,B1),(B5,B1),(B8,B1),(B9,B1),(End,B1),(B4,B2),(B8,B2),(B9,B2),(End,B2),(B5,B3),(B9,B3),(End,B3),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B6,B1),(B9,B1),(End,B1),(B8,B2),(B6,B2),(B7,B2),(B9,B2),(End,B2),(B7,B3),(B9,B3),(End,B3),(B7,B8),(End,B8),(B9,B6),(End,B6),(End,B7)

||: (B5,B6),(B6,B5),(B7,B1),(B1,B7),(B3,B6),(B6,B3),(B7,B4),(B4,B7),(B1,B4),(B4,B1),(B4,B6),(B6,B4),(B6,B2),(B2,B6),(B2,B5),(B5,B2),(B5,B7),(B7,B5),(B7,B3),(B3,B7),(B5,B1),(B1,B5),(B3,B4),(B4,B3),(B8,B6),(B6,B8),(B6,B1),(B1,B6),(B4,B2),(B2,B4),(B5,B3),(B3,B5),(B7,B2),(B2,B7),(B7,B8),(B8,B7)

→:(Start,B4),(B4,B5),(B6,B7),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B5,B8),(Start,B6),(B7,B9)

⇒:(Start,B5),(Start,B7),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B4,B8),(B4,B9),(B4,End),(B5,B9),(B5,End),(B6,B9),(B6,End),(B7,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End)

←:(B4,Start),(B5,B4),(B7,B6),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B8,B5),(B6,Start),(B9,B7)

⇐: (B5,Start),(B7,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B8,B4),(B9,B4),(End,B4),(B9,B5),(End,B5),(B9,B6),(End,B6),(End,B7),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8)

#: (Start,Start),(B4,B4),(B5,B5),(B6,B6),(B7,B7),(B1,B1),(B2,B2),(B3,B3),(B8,B8),(B9,B9),(End,End)

INFERRED →*i*: EMPTY

INFERRED ←*i*: EMPTY

NO_DIRECT →:End

NO_DIRECT ←:Start

Slika 5.15: Izgled relacija dobijenih na osnovu dnevnika događaja sa slike 5.11.

Relations:
>: (Start,B1),(B1,B2),(B2,B3),(B3,B4),(B4,B5),(B5,B6),(B6,B7),(B7,B8),(B8,B9),(B9,End),(Start,B6),(B6,B7),(B7,B4),(B4,B5),(B5,B1),(B1,B2),(B2,B3),(B3,B8),(B8,B9),(B9,End),(Start,B1),(B1,B2),(B2,B3),(B3,B4),(B4,B5),(B5,B6),(B6,B7),(B7,B8),(B8,B9),(B9,End)
>>: (Start,B2),(Start,B3),(Start,B4),(Start,B5),(Start,B7),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B4),(B1,B5),(B1,B6),(B1,B7),(B1,B8),(B1,B9),(B1,End),(B2,B4),(B2,B5),(B2,B6),(B2,B7),(B2,B8),(B2,B9),(B2,End),(B3,B5),(B3,B6),(B3,B7),(B3,B9),(B3,End),(B4,B6),(B4,B7),(B4,B8),(B4,B9),(B4,End),(B5,B7),(B5,B8),(B5,B9),(B5,End),(B6,B8),(B6,B9),(B6,End),(B7,B9),(B7,End),(B8,End),(Start,B7),(Start,B4),(Start,B5),(Start,B2),(Start,B3),(Start,B8),(Start,B9),(Start,End),(B6,B4),(B6,B5),(B6,B1),(B6,B2),(B6,B3),(B6,B8),(B6,B9),(B6,End),(B7,B5),(B7,B1),(B7,B2),(B7,B3),(B7,B9),(B7,End),(B4,B1),(B4,B2),(B4,B3),(B4,B8),(B4,B9),(B4,End),(B5,B2),(B5,B3),(B5,B8),(B5,B9),(B5,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B8,End),(Start,B2),(Start,B3),(Start,B4),(Start,B5),(Start,B7),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B4),(B1,B5),(B1,B6),(B1,B7),(B1,B8),(B1,B9),(B1,End),(B2,B4),(B2,B5),(B2,B6),(B2,B7),(B2,B8),(B2,B9),(B2,End),(B3,B5),(B3,B6),(B3,B7),(B3,B9),(B3,End),(B4,B6),(B4,B7),(B4,B8),(B4,B9),(B4,End),(B5,B7),(B5,B8),(B5,B9),(B5,End),(B6,B8),(B6,B9),(B6,End),(B7,B9),(B7,End),(B8,End)
<: (B1,Start),(B2,B1),(B3,B2),(B4,B3),(B5,B4),(B6,B5),(B7,B6),(B8,B7),(B9,B8),(End,B9),(B6,Start),(B7,B6),(B4,B7),(B5,B4),(B1,B5),(B2,B1),(B3,B2),(B8,B3),(B9,B8),(End,B9),(B1,Start),(B2,B1),(B3,B2),(B4,B3),(B5,B4),(B6,B5),(B7,B6),(B8,B7),(B9,B8),(End,B9)
<<: (B2,Start),(B3,Start),(B4,Start),(B5,Start),(B7,Start),(B8,Start),(B9,Start),(End,Start),(B3,B1),(B4,B1),(B5,B1),(B6,B1),(B7,B1),(B8,B1),(B9,B1),(End,B1),(B4,B2),(B5,B2),(B6,B2),(B7,B2),(B8,B2),(B9,B2),(End,B2),(B5,B3),(B6,B3),(B7,B3),(B9,B3),(End,B3),(B6,B4),(B7,B4),(B8,B4),(B9,B4),(End,B4),(B7,B5),(B8,B5),(B9,B5),(End,B5),(B8,B6),(B9,B6),(End,B6),(B9,B7),(End,B7),(End,B8),(B7,Start),(B4,Start),(B5,Start),(B2,Start),(B3,Start),(B8,Start),(B9,Start),(End,Start),(B4,B6),(B5,B6),(B1,B6),(B2,B6),(B3,B6),(B8,B6),(B9,B6),(End,B6),(B5,B7),(B1,B7),(B2,B7),(B3,B7),(B9,B7),(End,B7),(B1,B4),(B2,B4),(B3,B4),(B8,B4),(B9,B4),(End,B4),(B2,B5),(B3,B5),(B8,B5),(B9,B5),(End,B5),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(End,B8),(B2,Start),(B3,Start),(B4,Start),(B5,Start),(B7,Start),(B8,Start),(B9,Start),(End,Start),(B3,B1),(B4,B1),(B5,B1),(B6,B1),(B7,B1),(B8,B1),(B9,B1),(End,B1),(B4,B2),(B5,B2),(B6,B2),(B7,B2),(B8,B2),(B9,B2),(End,B2),(B5,B3),(B6,B3),(B7,B3),(B9,B3),(End,B3),(B6,B4),(B7,B4),(B8,B4),(B9,B4),(End,B4),(B7,B5),(B8,B5),(B9,B5),(End,B5),(B8,B6),(B9,B6),(End,B6),(B9,B7),(End,B7),(End,B8)
||: (B3,B4),(B4,B3),(B5,B6),(B6,B5),(B7,B4),(B4,B7),(B5,B1),(B1,B5),(B1,B4),(B4,B1),(B1,B6),(B6,B1),(B1,B7),(B7,B1),(B2,B4),(B4,B2),(B2,B5),(B5,B2),(B2,B6),(B6,B2),(B2,B7),(B7,B2),(B3,B5),(B5,B3),(B3,B6),(B6,B3),(B3,B7),(B7,B3),(B4,B6),(B6,B4),(B5,B7),(B7,B5)
→:(Start,B1),(B1,B2),(B2,B3),(B4,B5),(B6,B7),(B7,B8),(B8,B9),(B9,End),(Start,B6),(B3,B8)
⇒:(Start,B2),(Start,B3),(Start,B4),(Start,B5),(Start,B7),(Start,B8),(Start,B9),(Start,End),(B1,B3),(B1,B8),(B1,B9),(B1,End),(B2,B8),(B2,B9),(B2,End),(B3,B9),(B3,End),(B4,B8),(B4,B9),(B4,End),(B5,B8),(B5,B9),(B5,End),(B6,B8),(B6,B9),(B6,End),(B7,B9),(B7,End),(B8,End)
←:(B1,Start),(B2,B1),(B3,B2),(B5,B4),(B7,B6),(B8,B7),(B9,B8),(End,B9),(B6,Start),(B8,B3)
⇐: (B2,Start),(B3,Start),(B4,Start),(B5,Start),(B7,Start),(B8,Start),(B9,Start),(End,Start),(B3,B1),(B8,B1),(B9,B1),(End,B1),(B8,B2),(B9,B2),(End,B2),(B9,B3),(End,B3),(B8,B4),(B9,B4),(End,B4),(B8,B5),(B9,B5),(End,B5),(B8,B6),(B9,B6),(End,B6),(B9,B7),(End,B7),(End,B8)
#: (Start,Start),(B1,B1),(B2,B2),(B3,B3),(B4,B4),(B5,B5),(B6,B6),(B7,B7),(B8,B8),(B9,B9),(End,End)
INFERRED →/: (B5,B8),(Start,B4)
INFERRED ←/: (B4,Start),(B8,B5)
NO_DIRECT →:B5,End
NO_DIRECT ←:Start,B4

Slika 5.16: Izgled relacija dobijenih na osnovu dnevnika događaja sa slike 5.14.

Takođe poređenja radi, na slikama 5.17 i 5.18 su dati izgledi otisaka dnevnika događaja prikazanih na slikama 5.11 (dobijen prema preporukama iz Slučaja 1) i 5.14 (dobijen prema preporukama iz Slučaja 2), respektivno.

	Start	B4	B5	B6	B7	B1	B2	B3	B8	B9	End
Start	#	→	⇒	→	⇒	→	⇒	⇒	⇒	⇒	⇒
B4	←	#	→	||	||	||	||	||	⇒	⇒	⇒
B5	⇐	←	#	||	||	||	||	||	→	⇒	⇒
B6	←	||	||	#	→	||	||	||	||	⇒	⇒
B7	⇐	||	||	←	#	||	||	||	||	→	⇒
B1	←	||	||	||	||	#	→	⇒	⇒	⇒	⇒
B2	⇐	||	||	||	||	←	#	→	⇒	⇒	⇒
B3	⇐	||	||	||	||	⇐	←	#	→	⇒	⇒
B8	⇐	⇐	←	||	||	⇐	⇐	←	#	→	⇒
B9	⇐	⇐	⇐	⇐	←	⇐	⇐	⇐	←	#	→
End	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	←	#

Slika 5.17: Izgled otiska dnevnika događaja sa slike 5.11.

	Start	B1	B2	B3	B4	B5	B6	B7	B8	B9	End
Start	#	→	⇒	⇒	→*i*	⇒	→	⇒	⇒	⇒	⇒
B1	←	#	→	⇒	||	||	||	||	⇒	⇒	⇒
B2	⇐	←	#	→	||	||	||	||	⇒	⇒	⇒
B3	⇐	⇐	←	#	||	||	||	||	→	⇒	⇒
B4	←*i*	||	||	||	#	→	||	||	⇒	⇒	⇒
B5	⇐	||	||	||	←	#	||	||	→*i*	⇒	⇒
B6	←	||	||	||	||	||	#	→	⇒	⇒	⇒
B7	⇐	||	||	||	||	||	←	#	→	⇒	⇒
B8	⇐	⇐	⇐	←	⇐	←*i*	⇐	←	#	→	⇒
B9	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	←	#	→
End	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	⇐	←	#

Slika 5.18: Izgled otiska dnevnika događaja sa slike 5.14.

Upoređivanjem izgleda relacija sa slika 5.15 i 5.16, i izgleda otisaka dnevnika sa slika 5.17 i 5.18, može se primetiti da su oni međusobno različiti, iako su modeli koji se dobijaju na osnovu njih jednaki. Sa slika se može zaključiti da kada se konačan model dobija iz manjeg broja scenarija (slika 5.14) onda su neke kauzalne relacije koje se javljaju u otisku dnevnika zaključene na osnovu Pravila1 i/ili Pravila 2 (INFERRED: $\rightarrow i$ i $\leftarrow i$), a kada je konačan model dobijen iz većeg broja scenarija (slika 5.11) onda je velika verovatnoća da se sve kauzalne relacije mogu dobiti iz samih tragova zapisanih u dnevniku događaja.

Dakle, rezultati eksperimentalne analize sprovedene na 100 primera paralelnih poslovnih procesa su pokazali:

- u Slučaju 1, kod 99% razmatranih primera konačan model se dobija do pojave prvog ponovljenog scenarija posle 2 odigrana scenarija,
- u Slučaju 2, kod 88% razmatranih primera konačan model se dobija do pojave prvog ponovljenog scenarija posle 3 odigrana scenarija.

Iz ovoga se može zaključiti da se do konačnog modela paralelnog poslovnog procesa čije se izvršavanje demonstrira može doći posle vrlo malog broja odigranih scenarija (najčešće 2-3 scenarija). Osim toga, dobijeni model u bilo kom trenutku (kandidat-model) verno oslikava strukturu i ponašanje procesa iskazano svim prethodno demonstriranim scenarijima.

VI Zaključak

U okviru istraživanja prikazanog ovim radom izučavana je primenljivost paradigme *programiranja pomoću demonstracije* na domen razvoja informacionih sistema za upravljanje poslovnim procesima. Jedan od glavnih ciljeva ove disertacije jeste pronalaženje eventualnog načina za interaktivnu konstrukciju modela poslovnih procesa korišćenjem demonstracije, na osnovu odigravanja različitih scenarija izvršavanja poslovnog procesa od strane korisnika.

Ručno rađen model sistema često nije u potpunosti u skladu sa njegovim stvarnim ponašanjem, već pruža samo idealizovan ili poželjan pogled na poslovne procese koji se obavljaju u sistemu. To dovodi do problema u efikasnom korišćenju sistema, posebno u pogledu ponovnog korišćenja, modifikacije i iznalaženja načina za poboljšanje i unapređenje sistema. Problem pronalaženja modela poslovnih procesa koji stvarno oslikava strukturu i ponašanje sistema, detaljno je preciziran na početku rada. Date su definicije nekih osnovnih pojmova koji čine polaznu osnovu tehnike predložene u radu, čijom se primenom rešava pomenuti problem pronalaženja validnog modela paralelnog poslovnog procesa.

Zatim je dat pregled i izvršena je analiza postojećih rešenja dostupnih u literaturi. Postojeća rešenja su ukratko opisana, sa izuzetkom dva pristupa koji su od izuzetnog značaja za ovo istraživanje: *Play-in/Play-out* i *Pocess Mining*. Tehnike *play-in* i *play-out*, koje se primenjuju u razvoju reaktivnih sistema, poslužile su kao polazna osnova u ovom istraživanju, ali primenjene na sasvim drugi domen - razvoj informacionih sistema za upravljanje poslovnim procesima. Čitav pristup i tehnika kojom je omogućeno ostvarivanje interaktivne konstrukcije blok-struktuiranih modela paralelnih poslovnih procesa predstavljenih u radu, zasnovani su na konceptima i idejama discipline *process mining*, kod koje se otkrivanje modela procesa zasniva na evidenciji zapisanoj u dnevnicima događaja tokom izvršavanja procesa. Zbog toga su pomenuti pristupi detaljnije predstavljeni i analizirani u svetlu potreba i interesovanja vezanih za predstavljeno istraživanje.

U radu je potom detaljno opisano predloženo rešenje. U sklopu rešenja predložen je postupak modifikacije postojeće PM tehnike otkrivanja modela procesa, uvođenjem nove relacije između aktivnosti procesa - *indirektne relacije*. Uvedena relacija indirekcije dovela je do modifikacije postojećeg α-algoritma, što je rezultovalo dobijanjem modifikovanog $\alpha^{\|}$ - algoritma primenljivog na posebnu vrstu procesa - na paralelne procese.

Pokazalo se da primenom modifikovanog $\alpha^{\|}$-algoritma kod paralelnih poslovnih procesa, dnevnici događaja koji čuvaju zapise o izvršavanju procesa mogu biti znatno oskudniji u pogledu broja tragova neophodnih za otkrivanje modela, nego što to zahteva α-algoritam. Zbog toga su u daljem radu predložene i definisane dve nove vrste dnevnika događaja: *kauzalno kompletni* i *slabo kompletni* dnevnici, i precizno su definisani uslovi koje moraju ispuniti kako bi posedovali svojstvo kauzalne

odnosno slabe kompletnosti. Time je kod paralelnih procesa prevaziđen jedan od osnovnih problema koji se javlja kod primene α-algoritma, a to je problem kompletnosti dnevnika događaja, jer se α-algoritmom do originalne mreže može doći samo na osnovu dnevnika događaja koji ispunjava uslov kompletnosti, koji je rigorozniji od uslova kauzalne i slabe kompletnosti dnevnika. Primena $\alpha^{\|}$-algoritma na kauzalno kompletne i slabo kompletne dnevnike događaja, u cilju otkrivanja originalne mreže, detaljno je predstavljena na konkretnom demonstrativnom primeru u radu.

Ipak, da bi se dobijeni rezultati verifikovali na većem broju primera procesa, i to primera korišćenih u praksi, napravljena je baza od 100 realnih primera paralelnih poslovnih procesa, koji su poslužili za eksperimentalnu analizu, a koji su dati u Prilogu D. Do pomenutih primera procesa se došlo pretragom Interneta i odabirom javno dostupnih modela poslovnih procesa.

Kako bi eksperimentalna analiza mogla biti sprovedena, kreirani su potrebni priključci postojećem ProM alatu, koji zapravo predstavlja okruženje koje pruža podršku za različite tehnike otkrivanja modela procesa. U radu su predstavljena tri priključka koja su u tu svrhu kreirana, a njihovi programski kodovi se nalaze u zasebnim paketima i priloženi su u Prilozima A, B i C. Za otkrivanje originalne mreže primenom modifikovanog $\alpha^{\|}$-algoritma na kauzalno kompletan ili slabo kompletan dnevnik događaja, kreiran je priključak: *Mine for a Petri net using modified Alpha-algorithm*. U postupku otkrivanja modela procesa, najpre je potrebno odrediti *bazičnu kauzalnu relaciju*, definisanu u okviru modifikovane PM tehnike, što je omogućeno priključkom: *Modified Alpha algorithm helper plug-in-find basic causality relation*.

Osim otkrivanja modela realnih primera paralelnih poslovnih procesa iz baze, zadatak izvršene eksperimentalne analize je bio da se uporede veličine kompletnih, kauzalno kompletnih i slabo kompletnih dnevnika događaja u pogledu broja tragova, kako bi se evidentirala prednost korišćenja novih vrsta dnevnika događaja predloženih u radu. Da bi se to ostvarilo, u okviru postojećeg ProM alata napravljen je priključak pod nazivom: *Modified Alpha-algorithm - Minimal Logs*, pomoću kojeg se iz datog kompletnog dnevnika događaja izdvajaju kompletni, kauzalno kompletni i slabo kompletni dnevnici događaja sa najmanjim mogućim brojem tragova, i upoređuju se njihove međusobne veličine.

Rezultati eksperimentalne analize, sprovedene na uzorku od 100 realnih primera paralelnih procesa, pokazali su da su minimalni kauzalno kompletni dnevnici manji od minimalnih kompletnih dnevnika događaja u proseku za 37,55%. Takođe, rezultati su pokazali da su minimalni slabo kompletni dnevnici manji od minimalnih kauzalno kompletnih dnevnika u proseku za 22,08%, a od minimalnih kompletnih dnevnika manji su u proseku za 52,74%. Rezultati analize, da su kauzalno kompletni i slabo kompletni dnevnici događaja manji od kompletnih dnevnika, su i statistički potvrđeni primenom testa rangova, odnosno *Wilcoxon-Mann-Whitney* testa na dobijene rezultate.

Osim što otkrivanje modela procesa iz dnevnika događaja sa manjim brojem tragova dovodi do bržeg dobijanja modela, vrednost dobijenih rezultata istraživanja predstavljenog u radu je i mogućnost otkrivanja modela iz dnevnika događaja koji nisu kompletni, što se primenom α-algoritma ne može ostvariti. Upravo zbog toga, α-algoritam se nije mogao upotrebiti za interaktivnu konstrukciju modela procesa. Svojstvo $\alpha^{\|}$-algoritma da se njegovom primenom na slabo kompletnim dnevnicima događaja može doći do otkrivanja modela, uspešno je iskorišćeno za kreiranje modela paralelnih poslovnih procesa pomoću demonstracije, čime je ostvaren krajnji cilj ovog istraživanja.

U tu svrhu kreiran je sopstveni grafički korisnički interfejs koji omogućava korisniku odigravanje različitih scenarija izvršavanja aktivnosti procesa korišćenjem direktne manipulacije. Kreirani demonstracioni korisnički interfejs pripada grupi inteligentnih korisničkih interfejsa, jer sadrži komponente veštačke inteligencije koje se ogledaju u tome što sistem sam predlaže redosled odigravanja aktivnosti procesa (kako bi što je pre moguće otkrio model procesa) i zaključuje neke relacije koje nisu odigrane.

U radu je detaljno opisan postupak interaktivnog kreiranja modela na primeru procesa ubrzavanja obrade podataka paralelizmom pomoću FPGA procesora. Kako je u samom postupku demonstracije opisano, posle svakog odigravanja scenarija izvršavanja procesa od strane korisnika, sistem kreira *kandidat-model* koji podržava sve prethodno odigrane scenarije. Kada se kreirani kandidat-model ne menja bez obzira na sva odigravanja scenarija, onda je poslednji dobijeni kandidat-model zapravo *konačan model*.

I u ovom delu rada izvršena je eksperimentalna analiza, čiji je cilj bio da se istraži da li može da se ustanovi minimalan broj odigravanja scenarija neophodan za dobijanje konačnog modela, i od čega to zavisi. Eksperimentalna analiza je takođe sprovedena na uzorku od 100 realnih primera paralenih poslovnih procesa iz baze primera datih u Prilogu D. Rezultati izvršene eksperimentalne analize su pokazali da je odigravanje prvog scenarija od strane korisnika od ključnog značaja, jer na osnovu toga sistem nadalje predlaže redosled odigravanja aktivnosti procesa. Na kraju je prikazana detaljna analiza rezultata, ali uopšteno govoreći, do konačnog modela se može doći posle vrlo malog broja odigranih scenarija, najčešće posle 2-3 odigrana različita scenarija.

Interaktivnim kreiranjem modela procesa prevazilazi se problem neusklađenosti između modela i stvarnog ponašanja sistema, koji dolazi do izražaja kod ručno rađenih modela. Posebna prednost prikazanog načina kreiranja modela pomoću demonstracije jeste što poslednji kreirani model uvek podržava sve prethodno odigrane scenarije, tako da je on uvek validan model za prikazano ponašanje sistema. To daje velike mogućnosti za modifikaciju i proširenje sistema, jer se model uvek prilagođava svakom novom odigranom scenariju. Generalno govoreći, ako dopunjavanje ponašanja sistema odigravanjem novih scenarija posmatramo kao beskonačan proces, onda se do konačnog modela ne može i ne mora ni doći. U tom slučaju ulogu konačnog modela preuzima poslednji kreirani kandidat-model, sve do odigravanja novog scenarija koji će dovesti do promena u modelu, odnosno do novog kandidat-modela.

Osnovni nedostatak prikazanog istraživanja je što su dobijeni rezultati ograničeni na određenu vrstu procesa, a to su *paralelni procesi*, što umanjuje upotrebljivost dobijenih rezultata u praksi. Sa druge strane, to ostavlja mnogo prostora za nastavak istraživanja. Budući istraživački poduhvat bi mogao da se odnosi na ispitivanje primenljivosti ovde prikazanih i ostvarenih ideja i rezultata na mešovite procese, i eventualno otkrivanje nove tehnike i algoritma kojim bi se te ideje mogle ostvariti kod mešovitih procesa. U sklopu tog istraživanja mogao bi se rešavati i problem kompletnosti dnevnika događaja kod mešovitih procesa, koji postoji u PM tehnici otkrivanja modela procesa. Ostvarivanje ideje interaktivnog kreiranja modela mešovitih procesa predstavljalo bi krupan korak u razvoju informacionih sistema za upravljanje poslovnim procesima.

VII Literatura

1. http://www.omg.org - Documentation of the Unified Modeling Language (UML), available from the Object Management Group (OMG).

2. http://www.omg.org/spec/BPMN/2.0

3. W.M.P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

4. W.M.P. van der Aalst, A.J.M.M. Weijters, L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs". *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128-1142, 2004.

5. W.M.P. van der Aalst, A.J.M.M. Weijters, L. Maruster, "Workflow Mining: Which Processes can be Rediscovered?" BETA Working Paper Series, WP 74, Eindhoven Univ. of Technology, Eindhoven, 2002.

6. D. Harel and R. Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer-Verlag, 2003.

7. D. Harel, "From Play-In Scenarios To Code: An Achievable Dream," *Computer* 34:1 (January 2001), IEEE Press, 53-60.

8. R. Bergenthum, J. Desel, R. Lorenz, S. Mauser, "Process Mining Based on Regions of Languages". In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management* (BPM 2007), volume 4714 of Lecture Notes in Computer Science, pages 375-383. Springer, Berlin, 2007.

9. R. Agrawal, D. Gunopulos, F. Leymann, "Mining Process Models from Workflow Logs," *Proc. Sixth Int'l Conf. Extending Database Technology*, pp. 469-483, 1998.

10. J.E. Cook, A.L. Wolf, "Discovering Models of Software Processes from Event-Based Data," *ACM Trans. Software Eng. and Methodology*, vol. 7, no. 3, pp. 215-249, 1998.

11. W.M.P. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, C. Guenther, "Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting," *Software and Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

12. J. Carmona, J. Cortadella, M. Kishinevsky, "A Region-Based Algorithm for Discovering Petri Nets from Event Logs," in *Business Process Management* (BPM2008), 2008, pp. 358–373.

13. J. Carmona, J. Cortadella, "Process Mining Meets Abstract Interpretation," in *ECML/PKDD 210*, ser. Lecture Notes in Artificial Intelligence, J. Balcazar, Ed., vol. 6321. Springer-Verlag, Berlin, 2010, pp. 184–199.

14. S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, "Robust Process Discovery with Artificial Negative Events," *Journal of Machine Learning Research*, vol. 10, pp. 1305–1340, 2009.

15. A. Medeiros, A. Weijters, W.M.P. van der Aalst, "Genetic Process Mining: An Experimental Evaluation," *Data Mining and Knowledge Discovery*, vol. 14, no. 2, pp. 245–304, 2007.

16. M. Sole, J. Carmona, "Process Mining from a Basis of Regions," in *Applications and Theory of Petri Nets 2010*, ser. Lecture Notes in Computer Science, J. Lilius and W. Penczek, Eds., vol. 6128. Springer-Verlag, Berlin, 2010, pp. 226–245.

17. A. Weijters, W. van der Aalst, "Rediscovering Workflow Models from Event-Based Data using Little Thumb," *Integrated Computer-Aided Engineering*, vol. 10, no. 2, pp. 151–162, 2003.

18. J. van der Werf, B. van Dongen, C. Hurkens, A. Serebrenik, "Process Discovery using Integer Linear Programming," *Fundamenta Informaticae*, vol. 94, pp. 387–412, 2010.

19. C. W. Guenther, W. M. P. van der Aalst, "Fuzzy Mining: Adaptive Process Simpli_cation Based on Multi-perspective Metrics," in: *BPM 2007*, Vol. 4714 of LNCS, Springer, 2007, pp. 328–343.

20. A.J.M.M. Weijters, J.T.S. Ribeiro, "Flexible Heuristics Miner (FHM)," *BETA Working Paper Series*, WP 334, Eindhoven University of Technology, Eindhoven, 2010.

21. J. Carmona, J. Cortadella, M. Kishinevsky, "A Region-Based Algorithm for Discovering Petri Nets from Event Logs," in *Business Process Management* (BPM2008), 2008, pp. 358–373.

22. M. Sole, J. Carmona, "Process Mining from a Basis of Regions," in *Applications and Theory of Petri Nets 2010*, ser. Lecture Notes in Computer Science, J. Lilius and W. Penczek, Eds., vol. 6128. Springer-Verlag, Berlin, 2010, pp. 226–245.

23. W.M.P. van der Aalst, C. Stahl, *Modeling Business Processes: A Petri Net Oriented Approach*. MIT press, Cambridge, MA, 2011.

24. W.M.P. van der Aalst, "The Application of Petri Nets to Workflow Management," *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

25. W.M.P. van der Aalst, "Verification of Workflow Nets," *Application and Theory of Petri Nets*, P. Azema and G. Balbo, eds., pp. 407- 426, Berlin: Springer-Verlag, 1997.

26. W. Damm, D. Harel, "LSCs: Breathing Life into Message Sequence Charts," *Formal Methods in System Design* 19:1 (2001), 45-80.

27. D. Harel, R. Marelly, "Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach," *Software and System Modeling* (SoSyM) 2 (2003), 82-107.

28. D. Harel, H. Kugler, R. Marelly, A. Pnueli, "Smart Play-Out of Behavioral Requirements," *Proc. 4th Int. Conf. on Formal Methods in Computer-Aided Design* (FMCAD 2002), November 2002, pp. 378-398.

29. D. Harel, H. Kugler, R. Marelly, "The Play-in/Play-out Approach and Tool: Specifying and Executing Behavioral Requirements," *Proc. Israeli Workshop on Programming Languages & Development Environments* (PLE'02), July 2002.

30. D. Harel, H. Kugler, A. Pnueli, "Smart Play-Out Extended: Time and Forbidden Elements", *Proc. 4th Int. Conf. on Quality Software* (QSIC'04), IEEE Computer Society Press, 2004, pp. 2-10.

31. D. Harel, A. Kantor, S. Maoz, "On the Power of Play-Out for Scenario-Based Programs," In *Concurrency, Compositionality and Correctness: Essays in Honor of Willem-Paul de Roever* (Dams, Hanneman and Steffen, eds.), Lecture notes in Computer Science, Vol. 5930 Springer, 2010, pp 207-220.

32. http://en.wikipedia.org/wiki/Programming_by_demonstration

33. B. A. Myers, "Visual Programming, Programming by Example, and Program Visualization: A Taxonomy," Proc. SIGCHI'86: *Human Factors in Computing Systems*, ACM Press, April 1986, pp. 59-66.

34. D. C. Smith. *Pygmalion: A Computer Program to Model and Stimulate Creative Thought.* Ph.D. dissertation, Stanford University, 1975. Also published by Birkhauser Verlag, Basel and Stuttgart, 1977.

35. G. A. Curry. *Programming by Abstract Demonstration.* Ph.D. dissertation, University of Washington, Seattle. Technical Report No. 78-03-02. March 1978.

36. B. Shneiderman, "Direct manipulation: a step beyond programming languages," *Computer* 16(8): 57-69 (August 1983).

37. D. C. Halbert. *Programming by Example*. PhD Thesis. Computer Science Division, Dept. of EE&CS, University of California, Berkeley. 1984. Also: Xerox Office Systems Division, Systems Development Department, TR OSD-T8402, December, 1984.

38. B. Myers, "Creating user interfaces using programming by example, visual programming, and constraints," *ACM Transact. Program. Lang. Syst.* 12, 2 (Apr. 1990), 143–177.

39. B. A. Myers, "Peridot: Creating User Interfaces by Demonstration," *Watch What I Do: Programming by Demonstration*, Allen Cypher, et. al., eds. Cambridge, MA: The MIT Press, 1993. pp. 125-153.

40. B. A. Myers, "Garnet: Uses of Demonstrational Techniques," *Watch What I Do: Programming by Demonstration*, Allen Cypher, et. al., eds. Cambridge, MA: The MIT Press, 1993. pp. 219-236.

41. B. A. Myers, "Ideas from Garnet for Future User Interface Programming Languages," *Languages for Developing User Interfaces*. Boston: Jones and Bartlett, 1992. pp. 147-157.

42. B. A. Myers, A. S. Ferrency, B. D. Kyle, "The Amulet Environment: New Models for Effective User Interface Software Development, " *IEEE Transactions on software engineering*, VOL. 23, NO. 6, JUNE 1997.

43. Getting More Out Of Programming-By-Demonstration Richard G. McDaniel and Brad A. Myers HCI Institute, School of Computer Science Carnegie Mellon UniversityCHI 99 15-20 MAY 1999.

44. A. Cypher, "Watch What I Do: Programming by Demonstration," *MIT Press*, *Cambridge*, *MA*, 1993. 652 pages.

45. H. Lieberman, R. S. Amant, R. Potter, Luke Zettlemoyer, "Visual Generalization in Programming by Example," *Communications of the ACM*, March 2000. Also in Henry Lieberman, ed. Your Wish is My Command, Morgan Kaufmann, 2001.

46. H. Lieberman, "Tinker: A Programming by Demonstration System for Beginning Programmers," in *Watch What I Do: Programming by Demonstration*, Allen Cypher, ed., MIT Press, 1993.

47. H. Lieberman, "A User Interface for Knowledge Acquisition from Video*," Conference of the American Association for Artificial Intelligence*, Seattle, August 1994.

48. Henry. Lieberman, "Computer-Aided Design of User Interfaces by Example," (Keynote presentation), *Conference on Computer-Aided Design of User Interfaces*, Valenciennes, France, May 2002.

49. H. Friedrich, R. Dillmann, "Robot programming based on a single demonstration and user intentions," in: *3rd European Workshop on Learning Robots at ECML'95*, 1995.

50. J. Chen, A. Zelinsky, "Programing by demonstration: Coping with suboptimal teaching actions," *The International Journal of Robotics Research* 22 (5) (2003) 299–319.

51. A. Billard, S. Callinon, R. Dillmann, S. Schaal, "Robot programming by demonstration," in: B. Siciliano, O. Khatib (Eds.), *Handbook of Robotics*, Springer, New York, NY, USA, 2008 (Chapter 59).

52. C. G. Atkeson, S. Schaal, "Robot learning from demonstration," in: *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML'97, 1997.

53. S. Chernova, M. Veloso, "Learning equivalent action choices from demonstration," in: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'08, 2008.

54. B. D. Argall, et al., "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, Volume 57, Issue 5, 2009, pages 469–483.

55. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "*Object-Oriented Modeling and Design*", Prentice-Hall, 1991.

56. D. Milicev, *"Model-Driven Development with Executable UML"*, Wiley/Wrox, July 2009, ISBN 9780470481639.

57. J.E. Cook and A.L. Wolf, "Event-Based Detection of Concurrency," *Proc. Sixth Int'l Symp. the Foundations of Software Eng*. (FSE-6), pp. 35-45, 1998.

58. J. Herbst, "A Machine Learning Approach to Workflow Management," *Proc. 11th European Conf. Machine Learning*, pp. 183-194, 2000.

59. J. Herbst, "Dealing with Concurrency in Workflow Induction," *Proc. European Concurrent Eng. Conf.*, U. Baake, R. Zobel, and M. Al-Akaidi, eds., 2000.

60. J. Herbst, D. Karagiannis, "Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models," *Int'l J. Intelligent Systems in Accounting, Finance, and Management*, vol. 9, pp. 67-92, 2000.

61. G. Schimm, "Process Mining," http://www.processmining.de/, 2004.

62. G. Schimm, "Process Miner–A Tool for Mining Process Schemes from Event-Based Data," *Proc. Eighth European Conf. Artificial Intelligence* (JELIA), S. Flesca and G. Ianni, eds., pp. 525-528, 2002.

63. A. Rozinat, W.M.P. van der Aalst. "Conformance Checking of Processes Based on Monitoring Real Behavior." *Information Systems*, 33(1):64–95, 2008.

64. W. M. P. van der Aalst. Process Mining: "Discovering and Improving Spaghetti and Lasagna Processes". In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 13-20, Paris, France, April 2011. IEEE.

65. L. Maruster, W.M.P. van der Aalst, A.J.M.M. Weijters, A. van den Bosch, W. Daelemans, "Automated Discovery of Workflow Models from Hospital Data," *Proc. 13th Belgium-Netherlands Conf. Artificial Intelligence* (BNAIC 2001), B. Kro¨se, M. de Rijke, G. Schreiber, and M. van Someren, eds., pp. 183-190, 2001.

66. L. Maruster, A.J.M.M. Weijters, W.M.P. van der Aalst, A. van den Bosch, "Process Mining: Discovering Direct Successors in Process Logs," *Proc. Fifth Int'l Conf. Discovery Science* (Discovery Science 2002), pp. 364-373, 2002.

67. A.J.M.M. Weijters and W.M.P. van der Aalst, "Workflow Mining: Discovering Workflow Models from Event-Based Data," *Proc. ECAI Workshop Knowledge Discovery and Spatial Data*, C. Dousson, F. Ho¨ppner, and R. Quiniou, eds., pp. 78-84, 2002.

68. L. Wen, W. M. P. van der Aalst, J. Wang, J. Sun, "Mining Process Models with Non-Free-Choice Constructs," *Data Mining and Knowledge Discovery* 15 (2) (2007) 145–180.

69. L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, J. Sun, "Mining Process Models with Prime Invisible Tasks," *Data and Knowledge Engineering* 69 (10) (2010) 999–1021.

70. IEEE Task Force on Process Mining, "Process Mining Manifesto," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar, Eds., vol. 99. Springer-Verlag, Berlin, 2012, pp. 169–194.

71. A.K.A de Medeiros, "*Genetic Process Mining*" PhD Thesis, Eindhoven University of Technology, 2006.

72. W. M. P. van der Aalst, A. K. Alves de Medeiros, A. J. M. M. Weijters, "Genetic Process Mining," in: G. Ciardo, P. Darondeau (Eds.), *Applications and Theory of Petri Nets* 2005, Vol. 3536 of LNCS, Springer, 2005, pp. 48–69.

73. W. M. P. van der Aalst, A. Adriansyah, B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *WIREs Data Min. Knowl. Discov.* 2(2), 182–192 (2012).

74. H.M.W. Verbeek, B.F. van Dongen, J. Mendling, W.M.P. van der Aalst, "Interoperability in the ProM Framework," In T. Latour and M. Petit, editors, *Proceedings of the EMOI-INTEROP Workshop at the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, pages 619-630. Namur University Press, 2006.

75. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, A.J.M.M. Weijters, "ProM 4.0: Comprehensive Support for Real Process Analysis," In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484-494. Springer-Verlag, Berlin, 2007.

76. B. F. van Dongen, A. K. Alves de Medeiros, L. Wen, "Process Mining: Overview and Outlook of Petri Net Discovery Algorithms," *ToPNOC 2* (2009) 225–242.

77. A.K.A. de Medeiros, W.M.P. van der Aalst, A.J.M.M. Weijters, "Workflow Mining: Current Status and Future Directions," In R. Meersman, Z. Tari, and D.C. Schmidt, editors, On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer, Berlin, 2003.

78. J.J. Leemans, D. Fahland, W.M.P. van der Aalst, "Discovering Block-Structured Process Models from Incomplete Event Logs," Sander Eindhoven University of Technology, the Netherlands.

79. https://svn.win.tue.nl/trac/prom/wiki/setup/HowToBecomeAProMDeveloper

80. https://svn.win.tue.nl/repos/prom/Framework/trunk/

81. http://фтнкм.срб/fakultet/zaposleni/julijana-lekic/Examples-Block-Structured-Parallel-Process.

82. B. Č. Popović: *Matematička statistika*, Prirodno-matematički fakultet, Univerzitet u Nišu, 2009.

83. Brad A. Myers, Richard McDaniel, David Wolber, "Programming by example: intelligence in demonstrational interfaces", *Communications of the ACM*, Vol. 43 No. 3, Pages 82-89

84. Brad A. Myers, "Demonstrational Interfaces: A Step Beyond Direct Manipulation," *People and Computers VI*. Dan Diaper and Nick Hammond, eds. Cambridge, England: Cambridge University Press, 1991. pp. 11-30.

85. Brad A. Myers, "Using AI Techniques to Create User Interfaces by Example," in Joseph W. Sullivan, ed, *Intelligent User Interfaces*. Reading, MA: Addison-Wesley/ACM Press, 1991. pp. 385-401.

86. https://en.wikipedia.org/wiki/Field-programmable_gate_array

87. https://www.google.rs/search?q=parallel++process&biw=1366&bih=672&tbm=isch&tbo=u&source=univ&sa=X&ei=b7xVVc-

88. J. Lekić, D. Milićev, "Modifikacija alfa algoritma za otkrivanje modela poslovnih procesa iz nekompletnih dnecnika događaja" Zbornik 57. konferencije ETTRAN, Zlatibor, 3-6. juna 2013, str. RT6.2.1-6.

89. J. Lekic, D. Milicev, " Discovering Models of Parallel Workflow Processes from Incomplete Event Logs," In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 477-482.

VIII PRILOZI

Prilog A

Programski kod priključka *Modified Alpha algorithm helper plug-in-find basic causality relation*

```
package alphaminer_mod_basic_relation;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.concurrent.CancellationException;
import java.util.concurrent.ExecutionException;

import org.deckfour.xes.classification.XEventClass;
import org.deckfour.xes.info.XLogInfo;
import org.deckfour.xes.info.XLogInfoFactory;
import org.deckfour.xes.model.XLog;
import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
import org.processmining.framework.connections.ConnectionCannotBeObtained;
import org.processmining.framework.plugin.PluginContext;
import org.processmining.framework.plugin.PluginExecutionResult;
import org.processmining.framework.plugin.PluginParameterBinding;
import org.processmining.framework.plugin.Progress;
import org.processmining.framework.plugin.annotations.Plugin;
import org.processmining.framework.plugin.annotations.PluginVariant;
import org.processmining.framework.plugin.events.Logger.MessageLevel;
import org.processmining.framework.util.Pair;
import org.processmining.plugins.log.logabstraction.LogAbstractionConnection;
import org.processmining.plugins.log.logabstraction.LogRelations;

@Plugin(name = "Alpha Miner Basic Relation", parameterLabels = { "Log", "Summary", "Log Relations" }, returnLabels =
{ "Message" }, returnTypes = {String.class})
public class AlphaMiner_Mod_BasicRelation {

	private LogRelations relations;
	private List<XEventClass> eventClasses;

	//**************** 3 Variants without nets **********************//
	@UITopiaVariant(uiLabel = "Modified Alpha-algorithm helper plug-in - find basic causal relation", affiliation =
UITopiaVariant.EHV, author = "Lena Parezanovic", email = "pl123283m@student.etf.rs", pack =
"AlphaMiner_Mod_Basic_Relation")
	@PluginVariant(variantLabel = "Default event classes", requiredParameterLabels = { 0 })
	public String doMining(PluginContext context, XLog log) throws CancellationException, InterruptedException,
			ExecutionException {
		XLogInfo info = XLogInfoFactory.createLogInfo(log);
		return doAlphaMiningPrivate(context, log, info);
	}
	@PluginVariant(variantLabel = "User-defined event classes", requiredParameterLabels = { 0, 1 })
	public String doMining(PluginContext context, XLog log, XLogInfo summary) throws CancellationException,
			InterruptedException, ExecutionException {
		return doAlphaMiningPrivate(context, log, summary);
```

```
	}
	@PluginVariant(variantLabel = "User-defined event classes and relations", requiredParameterLabels = { 1, 2 })
	public String doMining(PluginContext context, XLogInfo summary, LogRelations relations)
			throws InterruptedException, ExecutionException {
		return doAlphaMiningPrivateWithRelations(context, summary, relations);
	}
	//*************************************************************//
	private String doAlphaMiningPrivate(PluginContext context, XLog log, XLogInfo summary)
			throws CancellationException, InterruptedException, ExecutionException {

		// First check if a connection providing log ralations exists.
		try {
			LogAbstractionConnection logAbstractionConnection =
context.getConnectionManager().getFirstConnection(LogAbstractionConnection.class, context, log);
			if (logAbstractionConnection != null) {
				LogRelations logRelations = logAbstractionConnection.getRelations();
				if (logRelations != null)
					return doAlphaMiningPrivateWithRelations(context, summary, logRelations);
			}
		} catch (ConnectionCannotBeObtained e) {
			// Ignore, we try another way later
		}
		// No log relations are specified, so find a plugin that can construct them.
		// This is done, by asking the plugin manager for a plugin, that:
		// 1) It's a plugin (i.e. the annotation is Plugin.class),
		// 2) It returns LogRelations (i.e. one of the return types is LogRelations.class or any subclass thereof),
		// 3) It can be executed in a child of this context, which is of type context.getPluginContextType(),
		// 4) It can be executed on the given input (i.e. no extra input is needed, and all input is used),
		// 5) It accepts the input in any given order (i.e. not in the specified order),
		// 6) It does not have to be user visible
		// 7) It can use objects given by log and summary (i.e. types log.getClass() and summary.getClass()).
		Collection<Pair<Integer, PluginParameterBinding>> plugins =
context.getPluginManager().find(Plugin.class,
			LogRelations.class, context.getPluginContextType(), true, false, false, XLog.class,
summary.getClass());

		if (plugins.isEmpty()) {
			context.log("No plugin found to create log relations, please specify relations manually",
				MessageLevel.ERROR);
			return null;
		}
		// Let's just take the first available plugin for the job of constructing log abstractions
		Pair<Integer, PluginParameterBinding> plugin = plugins.iterator().next();

		// Now, the binding can be executed on the log and the summary
		// FIrst, we instantiate a new context for this plugin, which is a child context of the current context.
		PluginContext c2 = context.createChildContext("Log Relation Constructor");

		// Let's notify our lifecyclelisteners about the fact that we created a new context. this is
		// optional, but if this is not done, then the user interface doesn't show it (if there is a UI).
		context.getPluginLifeCycleEventListeners().firePluginCreated(c2);

		// At this point, we execute the binding to get the LogRelations. For this, we call the invoke method
		// on the PluginParameterBinding stored in the plugin variable. The return type is LogRelations.class and
		// as input we give the new context c2, the log and the summary. Note that the plugin might return
mulitple
		// objects, hence we extract the object with number x, where x is stored as the first element of the plugin
		// variable.
```

```
                PluginExecutionResult pluginResult = plugin.getSecond().invoke(c2, log, summary);
                pluginResult.synchronize();
                LogRelations relations = pluginResult.<LogRelations>getResult(plugin.getFirst());

                // Now we have the relations and we can continue with the mining.
                return doAlphaMiningPrivateWithRelations(context, relations.getSummary(), relations);
        }
        private String doAlphaMiningPrivateWithRelations(PluginContext context, XLogInfo summary, LogRelations
relations)
                        throws InterruptedException, ExecutionException {
                this.relations = relations;
                eventClasses = new ArrayList<XEventClass>(summary.getEventClasses().size());
                eventClasses.addAll(summary.getEventClasses().getClasses());
                eventClasses.removeAll(relations.getLengthOneLoops().keySet());
                final Progress progress = context.getProgress();

                progress.setMinimum(0);
                progress.setMaximum(5);
                progress.setIndeterminate(false);

                PrintWriter writer;
                try {
                        writer = new PrintWriter("basic_causal_relation.txt", "UTF-8");
                        for (Pair<XEventClass, XEventClass> causal : relations.getCausalDependencies().keySet()) {
                                writer.println(causal.getFirst());
                                writer.println(causal.getSecond());
                        }
                        writer.close();
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return "Error while writing to the file.";
                } catch (UnsupportedEncodingException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return "Error while writing to the file.";
                }
                return "Basic causal relation successfully written to the file.";
        }
}
```

Prilog B

Programski kod priključka *Mine for a Petri net using modified Alpha-algorithm*

```
package alphaminer_mod;

import java.util.HashSet;
import java.util.Set;

import org.deckfour.xes.classification.XEventClass;

class Tuple_Mod {

	public Set<XEventClass> leftPart = new HashSet<XEventClass>();

	public Set<XEventClass> rightPart = new HashSet<XEventClass>();

	public int maxRightIndex = 0;
	public int maxLeftIndex = 0;

	public Tuple_Mod() {
	}
	public boolean isSmallerThan(Tuple_Mod tuple) {
		return tuple.leftPart.containsAll(leftPart) && tuple.rightPart.containsAll(rightPart);
	}
	public Tuple_Mod clone() {
		Tuple_Mod clone = new Tuple_Mod();
		clone.leftPart.addAll(leftPart);
		clone.rightPart.addAll(rightPart);
		clone.maxRightIndex = maxRightIndex;
		clone.maxLeftIndex = maxLeftIndex;
		return clone;
	}
	public int hashCode() {
		return leftPart.hashCode() + 37 * rightPart.hashCode() + maxRightIndex + maxLeftIndex;
	}
	public boolean equals(Object o) {
		if (o instanceof Tuple_Mod) {
			Tuple_Mod t = (Tuple_Mod) o;
			return (t.maxRightIndex == maxRightIndex) && (t.maxLeftIndex == maxLeftIndex)
					&& t.leftPart.equals(leftPart) && t.rightPart.equals(rightPart);
		}
		return false;
	}
	public String toString() {
		return "{" + leftPart.toString() + "} --> {" + rightPart.toString() + "}";
	}
}
package alphaminer_mod;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Stack;
import java.util.concurrent.CancellationException;
import java.util.concurrent.ExecutionException;

import org.deckfour.xes.classification.XEventClass;
import org.deckfour.xes.classification.XEventClasses;
import org.deckfour.xes.extension.std.XConceptExtension;
import org.deckfour.xes.info.XLogInfo;
import org.deckfour.xes.info.XLogInfoFactory;
import org.deckfour.xes.model.XLog;
import org.deckfour.xes.model.XTrace;
import org.processmining.connections.logmodel.LogPetrinetConnectionImpl;
import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
import org.processmining.framework.connections.ConnectionCannotBeObtained;
import org.processmining.framework.plugin.PluginContext;
import org.processmining.framework.plugin.PluginExecutionResult;
import org.processmining.framework.plugin.PluginParameterBinding;
import org.processmining.framework.plugin.Progress;
import org.processmining.framework.plugin.annotations.Plugin;
import org.processmining.framework.plugin.annotations.PluginVariant;
import org.processmining.framework.plugin.events.Logger.MessageLevel;
import org.processmining.framework.util.Pair;
import org.processmining.framework.util.search.MultiThreadedSearcher;
import org.processmining.framework.util.search.NodeExpander;
import org.processmining.models.connections.petrinets.behavioral.InitialMarkingConnection;
import org.processmining.models.graphbased.directed.petrinet.Petrinet;
import org.processmining.models.graphbased.directed.petrinet.elements.Place;
import org.processmining.models.graphbased.directed.petrinet.elements.Transition;
import org.processmining.models.graphbased.directed.petrinet.impl.PetrinetFactory;
import org.processmining.models.semantics.petrinet.Marking;
import org.processmining.plugins.log.logabstraction.LogAbstractionConnection;
import org.processmining.plugins.log.logabstraction.LogRelations;

@Plugin(name = "Alpha Miner Modification", parameterLabels = { "Log", "Summary", "Log Relations" }, returnLabels =
{ "Petrinet",
                "Marking", "Message" }, returnTypes = { Petrinet.class, Marking.class, String.class })
public class AlphaMiner_Mod implements NodeExpander<Tuple_Mod> {

        private LogRelations relations;
        private List<XEventClass> eventClasses;

        // Return string, massage set appropriately bellow, this initial value is returned only in the case of some error.
        String ret = "Error occured while discovering the net.";

        enum LogType {
                CAUSALLY_COMPLETE_LOG,
                WEAKLY_COMPLETE_LOG
        }
        private LogType logType;
```

```
	public HashSet<Pair<XEventClass,XEventClass>> temp;
	public HashSet<Pair<XEventClass, XEventClass>> indirect_causal_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> causal_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> parallel_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> direct_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> indirect_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> inferred_causal_relations;
	public HashSet<Pair<XEventClass, XEventClass>> basic_relation;
	public HashSet<XEventClass> nodes_without_successor;
	public HashSet<XEventClass> nodes_without_predecessor;

	public void find_direct (XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();

		for (int i=0; i<summary.getNumberOfTraces(); i++) {
			XTrace trace = summary.getLog().get(i);
			for (int j=0; j<trace.size()-1; j++) {
				XEventClass e1 = classes.getClassOf(trace.get(j));
				XEventClass e2 = classes.getClassOf(trace.get(j+1));
				Pair<XEventClass, XEventClass> pair = new Pair<XEventClass, XEventClass>(e1,
e2);

				if(!direct_relationes.contains(pair)) {
					System.out.println("Moja klasa direct " + pair.toString());
					direct_relationes.add(pair);
				}
			}
		}
	}
	public void find_indirect (XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();

		for (int i=0; i<summary.getNumberOfTraces(); i++) {
			XTrace trace = summary.getLog().get(i);
			for (int j=0; j<trace.size()-2; j++) {
				for (int k=j+2; k<trace.size(); k++) {
					XEventClass e1 = classes.getClassOf(trace.get(j));
					XEventClass e2 = classes.getClassOf(trace.get(k));
					Pair<XEventClass, XEventClass> pair = new Pair<XEventClass,
XEventClass>(e1, e2);
					if(!indirect_relationes.contains(pair) && !direct_relationes.contains(pair)) {
						System.out.println("Moja klasa indirect " + pair.toString());
						indirect_relationes.add(pair);
					}
				}
			}
		}
	}
	public void find_parallel (XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();

		for ( XEventClass e1 : classes.getClasses()) {
			for ( XEventClass e2 : classes.getClasses()) {
				Pair<XEventClass, XEventClass> pair1 = new Pair<XEventClass, XEventClass>(e1,
e2);
```

```
                                Pair<XEventClass, XEventClass> pair2 = new Pair<XEventClass, XEventClass>(e2,
e1);
                                if ((direct_relationes.contains(pair1) && direct_relationes.contains(pair2))
                                        || (direct_relationes.contains(pair1) && indirect_relationes.contains(pair2))
                                        || (direct_relationes.contains(pair2) && indirect_relationes.contains(pair1))
                                        || (indirect_relationes.contains(pair1) && indirect_relationes.contains(pair2))
                                        ) {

                                        if (!parallel_relationes.contains(pair1) && !parallel_relationes.contains(pair2))
                                        {
                                                parallel_relationes.add(pair1);
                                                parallel_relationes.add(pair2);
                                                System.out.println("Moja klasa parallel = " + pair1.toString());
                                                System.out.println("Moja klasa parallel = " + pair2.toString());
                                        }
                                }
                        }
                }
        }
        public void find_causal(XLogInfo summmary) {
                causal_relationes.addAll(direct_relationes);
                if(causal_relationes.isEmpty())
                        System.out.println("PRAZNA");
                Iterator<Pair<XEventClass, XEventClass>> it = causal_relationes.iterator();
                while(it.hasNext())
                {
                        System.out.println("Moja klasa causal 1 " + it.next().toString());
                }
                causal_relationes.removeAll(parallel_relationes);
                it = causal_relationes.iterator();
                while(it.hasNext())
                {
                        System.out.println("Moja klasa causal 2 " + it.next().toString());
                }
        }
        public void find_indirect_causal(XLogInfo summmary) {
                indirect_causal_relationes.addAll(indirect_relationes);
                if(indirect_causal_relationes.isEmpty())
                        System.out.println("PRAZNA");
                Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
                while(it.hasNext())
                {
                        Pair<XEventClass, XEventClass> pair = it.next();
                        if(direct_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getSecond(),
pair.getFirst()))
                                ||
                                indirect_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getSecond(),
pair.getFirst()))
                                ) {
                                it.remove();
                        }
                }
                it = indirect_causal_relationes.iterator();
                while(it.hasNext())
                {
                        System.out.println("Moja klasa inderect causal 2 " + it.next().toString());
                }
```

```
}
public boolean isCausal(XEventClass from, XEventClass to)
{
	Pair<XEventClass, XEventClass> e1 = new Pair<XEventClass, XEventClass>(from, to);
	Pair<XEventClass, XEventClass> e2 = new Pair<XEventClass, XEventClass>(to, from);

	if (causal_relationes.contains(e1)){
		return true;
	}
	return false;
}
public boolean isParallel(XEventClass from, XEventClass to)
{
	if (parallel_relationes.contains(new Pair<XEventClass, XEventClass>(from, to))) {
		return true;
	}
	return false;
}
public void find_nodes_without_successor(XLogInfo summary) {
	XEventClasses classes = summary.getEventClasses();
	XTrace trace = summary.getLog().get(0);
	boolean hasSuccessor = false;
	for (int i=0; i<trace.size(); i++) {
		hasSuccessor = false;
		XEventClass e = classes.getClassOf(trace.get(i));
		Iterator<Pair<XEventClass, XEventClass>> it = causal_relationes.iterator();
		while(it.hasNext()) {
			Pair<XEventClass, XEventClass> pair = it.next();
			System.out.println("Find node no succ " + pair.toString() + " " + e.toString());
			if(pair.getFirst().equals(e)) {
				hasSuccessor = true;
				break;
			}
		}
		if (hasSuccessor == false) {
			System.out.println("Node without successor " + e.toString());
			nodes_without_successor.add(e);
		}
	}
}
public void find_nodes_without_predecessor(XLogInfo summary) {
	XEventClasses classes = summary.getEventClasses();
	XTrace trace = summary.getLog().get(0);
	boolean hasPredecessor = false;
	for (int i=0; i<trace.size(); i++) {
		hasPredecessor = false;
		XEventClass e = classes.getClassOf(trace.get(i));
		Iterator<Pair<XEventClass, XEventClass>> it = causal_relationes.iterator();
		while(it.hasNext()) {
			Pair<XEventClass, XEventClass> pair = it.next();
			if(pair.getSecond().equals(e)) {
				hasPredecessor = true;
				break;
			}
		}
		if (hasPredecessor == false) {
			System.out.println("Node without predecessor " + e.toString());
			nodes_without_predecessor.add(e);
```

```
                        }
                }
        }
        public void weakly_completed_logs_find_causal_from_indirect_successor(XLogInfo summary) {
        XEventClasses classes = summary.getEventClasses();
                XTrace trace = summary.getLog().get(0);

                Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
                while(it.hasNext()) {
                        Pair<XEventClass, XEventClass> pair = it.next();
                        for (int i=0; i<trace.size(); i++) {
                                XEventClass e = classes.getClassOf(trace.get(i));
                                if (!nodes_without_successor.contains(pair.getFirst())) continue;
                                if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(e,
pair.getSecond()))
                                        &&
                                        parallel_relationes.contains(new Pair<XEventClass, XEventClass>(e,
pair.getFirst()))) {
                                                inferred_causal_relations.add(pair);
                                                System.out.println("Inferred causal successor " + pair.toString());
                                                break;
                                }
                        }
                }
        }
        public void weakly_completed_logs_find_causal_from_indirect_predecessor(XLogInfo summary) {

                XEventClasses classes = summary.getEventClasses();
                XTrace trace = summary.getLog().get(0);

                Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
                while(it.hasNext()) {
                        Pair<XEventClass, XEventClass> pair = it.next();
                        for (int i=0; i<trace.size(); i++) {
                                XEventClass e = classes.getClassOf(trace.get(i));
                                if (!nodes_without_predecessor.contains(pair.getSecond())) continue;
                                if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getFirst(), e))
                                        &&
                                        parallel_relationes.contains(new Pair<XEventClass,
XEventClass>(pair.getSecond(), e))) {
                                                inferred_causal_relations.add(pair);
                                                System.out.println("Inferred causal predecessor " + pair.toString());
                                                break;
                                }
                        }
                }
        }
        //**************** 3 Variants without nets **********************//
        @UITopiaVariant(uiLabel = "Mine for a Petri Net using modified Alpha-algorithm", affiliation =
UITopiaVariant.EHV, author = "Lena Parezanovic", email = "pl123283m@student.etf.rs", pack = "AlphaMiner_Mod")
        @PluginVariant(variantLabel = "Default event classes", requiredParameterLabels = { 0 })
        public Object[] doMining(PluginContext context, XLog log) throws CancellationException, InterruptedException,
                        ExecutionException {
                XLogInfo info = XLogInfoFactory.createLogInfo(log);
                return doAlphaMiningPrivate(context, log, info);
        }
```

```
	@PluginVariant(variantLabel = "User-defined event classes", requiredParameterLabels = { 0, 1 })
	public Object[] doMining(PluginContext context, XLog log, XLogInfo summary) throws CancellationException,
			InterruptedException, ExecutionException {
		return doAlphaMiningPrivate(context, log, summary);
	}
	@PluginVariant(variantLabel = "User-defined event classes and relations", requiredParameterLabels = { 1, 2 })
	public Object[] doMining(PluginContext context, XLogInfo summary, LogRelations relations)
			throws InterruptedException, ExecutionException {
		return doAlphaMiningPrivateWithRelations(context, summary, relations);
	}
	//*************************************************************//

	private Object[] doAlphaMiningPrivate(PluginContext context, XLog log, XLogInfo summary)
			throws CancellationException, InterruptedException, ExecutionException {

		// First check if a connection providing log ralations exists.
		try {
			LogAbstractionConnection logAbstractionConnection =
context.getConnectionManager().getFirstConnection(LogAbstractionConnection.class, context, log);
			if (logAbstractionConnection != null) {
				LogRelations logRelations = logAbstractionConnection.getRelations();
				if (logRelations != null)
					return doAlphaMiningPrivateWithRelations(context, summary, logRelations);
			}
		} catch (ConnectionCannotBeObtained e) {
			// Ignore, we try another way later
		}
		// No log relations are specified, so find a plugin that can construct them.
		// This is done, by asking the plugin manager for a plugin, that:
		// 1) It's a plugin (i.e. the annotation is Plugin.class),
		// 2) It returns LogRelations (i.e. one of the return types is LogRelations.class or any subclass thereof),
		// 3) It can be executed in a child of this context, which is of type context.getPluginContextType(),
		// 4) It can be executed on the given input (i.e. no extra input is needed, and all input is used),
		// 5) It accepts the input in any given order (i.e. not in the specified order),
		// 6) It does not have to be user visible
		// 7) It can use objects given by log and summary (i.e. types log.getClass() and summary.getClass()).
		Collection<Pair<Integer, PluginParameterBinding>> plugins =
context.getPluginManager().find(Plugin.class,
				LogRelations.class, context.getPluginContextType(), true, false, false, XLog.class,
summary.getClass());

		if (plugins.isEmpty()) {
			context.log("No plugin found to create log relations, please specify relations manually",
					MessageLevel.ERROR);
			return null;
		}
		// Let's just take the first available plugin for the job of constructing log abstractions
		Pair<Integer, PluginParameterBinding> plugin = plugins.iterator().next();

		// Now, the binding can be executed on the log and the summary
		// FIrst, we instantiate a new context for this plugin, which is a child context of the current context.
		PluginContext c2 = context.createChildContext("Log Relation Constructor");

		// Let's notify our lifecyclelisteners about the fact that we created a new context. this is
		// optional, but if this is not done, then the user interface doesn't show it (if there is a UI).
		context.getPluginLifeCycleEventListeners().firePluginCreated(c2);

		// At this point, we execute the binding to get the LogRelations. For this, we call the invoke method
```

```
		// on the PluginParameterBinding stored in the plugin variable. The return type is LogRelations.class and
		// as input we give the new context c2, the log and the summary. Note that the plugin might return
mulitple
		// objects, hence we extract the object with number x, where x is stored as the first element of the plugin
		// variable.

		PluginExecutionResult pluginResult = plugin.getSecond().invoke(c2, log, summary);
		pluginResult.synchronize();
		LogRelations relations = pluginResult.<LogRelations>getResult(plugin.getFirst());

		// Now we have the relations and we can continue with the mining.
		return doAlphaMiningPrivateWithRelations(context, relations.getSummary(), relations);
	}
	private Object[] doAlphaMiningPrivateWithRelations(PluginContext context, XLogInfo summary, LogRelations
relations)
			throws InterruptedException, ExecutionException {
		this.relations = relations;
		eventClasses = new ArrayList<XEventClass>(summary.getEventClasses().size());
		eventClasses.addAll(summary.getEventClasses().getClasses());
		eventClasses.removeAll(relations.getLengthOneLoops().keySet());
		final Progress progress = context.getProgress();

		for(int i=0; i<summary.getEventClasses().size(); i++)
		{
			System.out.println(summary.getEventClasses().getByIndex(i));
		}
			////////////////////////////////// Reading basic causal relation from file ////////////////////////////////

		basic_relation = new HashSet<Pair<XEventClass,XEventClass>>();

		try {
			BufferedReader reader = new BufferedReader(new FileReader("basic_causal_relation.txt"));
			int i = 0;
			String s;
			try {
				while((s = reader.readLine()) != null) {
					XEventClass e1 = new XEventClass(s, i);
					i++;
					if((s = reader.readLine()) != null) {
						XEventClass e2 = new XEventClass(s, i);
						Pair<XEventClass, XEventClass> pair = new Pair<XEventClass,
XEventClass>(e1, e2);
						basic_relation.add(pair);
						System.out.println("basic relation " + pair.toString());
						i++;
					}
				}
				reader.close();
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
		progress.setMinimum(0);
		progress.setMaximum(5);
		progress.setIndeterminate(false);

		temp = new HashSet<Pair<XEventClass,XEventClass>>();

		final Stack<Tuple_Mod> stack = new Stack<Tuple_Mod>();

		//////////////////////////// Find relations ////////////////////////////

		direct_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
		indirect_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
		parallel_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
		causal_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
		indirect_causal_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
		inferred_causal_relations = new HashSet<Pair<XEventClass,XEventClass>>();
		temp = new HashSet<Pair<XEventClass,XEventClass>>();
		nodes_without_successor = new HashSet<XEventClass>();
		nodes_without_predecessor = new HashSet<XEventClass>();

		find_direct(summary);
		find_indirect(summary);
		find_parallel(summary);
		find_causal(summary);
		find_indirect_causal(summary);

		////////////////////////////////////////////////////////////////////////


		////////////////// Check the type of log ////////////////////////

		if (causal_relationes.equals(basic_relation)) {
			logType = LogType.CAUSALLY_COMPLETE_LOG;
			System.out.println("Log je kauzalno kompletan");
		}
		else {
			temp.addAll(causal_relationes);
			temp.addAll(indirect_causal_relationes);
			if(temp.containsAll(basic_relation)) {
				if (basic_relation.containsAll(causal_relationes)) {
					System.out.println("Log je slabo kompletan");
					find_nodes_without_successor(summary);
					find_nodes_without_predecessor(summary);
					weakly_completed_logs_find_causal_from_indirect_successor(summary);
					weakly_completed_logs_find_causal_from_indirect_predecessor(summary);
					causal_relationes.addAll(inferred_causal_relations);
					logType = LogType.WEAKLY_COMPLETE_LOG;
				}
				else {
					System.out.println("Log nije ni kauzalno ni slabo kompletan, ne moze se
otkriti originalna mreza.");
					context.getFutureResult(0).cancel(true);
					ret = "The log is not causal nor weakly completed, the net could not be
disovered.";
					return new Object[] { null, null, ret };
				}
			}
		}
```

```
		//////////////////////////////////////////////////////////////////

		// Initialize the tuples to the causal depencencies in the log
		for (Pair<XEventClass, XEventClass> causal : causal_relationes) {
			if (!isCausal(causal.getFirst(), causal.getSecond())) {
				continue;
			}
			if (progress.isCancelled()) {
				context.getFutureResult(0).cancel(true);
				return new Object[] { null, null, null };
			}
			if (!eventClasses.contains(causal.getFirst()) || !eventClasses.contains(causal.getSecond())) {
				continue;
			}
			Tuple_Mod tuple = new Tuple_Mod();
			tuple.leftPart.add(causal.getFirst());
			tuple.rightPart.add(causal.getSecond());
			tuple.maxRightIndex = eventClasses.indexOf(causal.getSecond());
			tuple.maxLeftIndex = eventClasses.indexOf(causal.getFirst());
			stack.push(tuple);
		}
		progress.inc();

		// Expand the tuples
		final List<Tuple_Mod> result = new ArrayList<Tuple_Mod>();

		MultiThreadedSearcher<Tuple_Mod> searcher = new MultiThreadedSearcher<Tuple_Mod>(this,
				MultiThreadedSearcher.BREADTHFIRST);

		searcher.addInitialNodes(stack);
		searcher.startSearch(context.getExecutor(), progress, result);

		if (progress.isCancelled()) {
			context.getFutureResult(0).cancel(true);
			return new Object[] { null, null, null };
		}
		// Add transitions
		Map<XEventClass, Transition> class2transition = new HashMap<XEventClass, Transition>();

		Petrinet net = PetrinetFactory.newPetrinet("Petrinet from "
				+ XConceptExtension.instance().extractName(relations.getLog())+" , mined with
AlphaMiner_Mod");

		context.getFutureResult(0).setLabel(net.getLabel());
		context.getFutureResult(1).setLabel("Initial Marking of " + net.getLabel());

		for (XEventClass eventClass : summary.getEventClasses().getClasses()) {
			Transition transition = net.addTransition(eventClass.toString());
			class2transition.put(eventClass, transition);
		}
		progress.inc();

		Map<Tuple_Mod, Place> tuple2place = new HashMap<Tuple_Mod, Place>();
		// Add places for each tuple
		for (Tuple_Mod tuple : result) {
			Place p = net.addPlace(tuple.toString());
			for (XEventClass eventClass : tuple.leftPart) {
```

```
                                net.addArc(class2transition.get(eventClass), p);
                        }
                        for (XEventClass eventClass : tuple.rightPart) {
                                net.addArc(p, class2transition.get(eventClass));
                        }
                        tuple2place.put(tuple, p);
                }
                progress.inc();

                Marking m = new Marking();

                // Add initial and final place
                Place pstart = net.addPlace("Start");
                for (XEventClass eventClass : relations.getStartTraceInfo().keySet()) {
                        net.addArc(pstart, class2transition.get(eventClass));
                }
                m.add(pstart);

                Place pend = net.addPlace("End");
                for (XEventClass eventClass : relations.getEndTraceInfo().keySet()) {
                        net.addArc(class2transition.get(eventClass), pend);
                }
                progress.inc();

                context.addConnection(new InitialMarkingConnection(net, m));
                context.addConnection(new LogPetrinetConnectionImpl(summary.getLog(), summary.getEventClasses(),
net, reverse(class2transition)));
                if (logType == LogType.CAUSALLY_COMPLETE_LOG) {
                        ret = "The log is causally complete, the net is successfully discovered.";
                }
                else if (logType == LogType.WEAKLY_COMPLETE_LOG) {
                        ret = "The log is weakly complete, the net is successfully discovered.";
                }
                return new Object[] { net, m, ret };
        }
        /**
         * Flip the mapping given around (so values map to keys); handles multiple values with same key.
         * @param class2transition
         * @return
         */
        protected Collection<Pair<Transition, XEventClass>> reverse(Map<XEventClass, Transition> class2transition) {
                List<Pair<Transition, XEventClass>> result = new
ArrayList<Pair<Transition,XEventClass>>(class2transition.size());
                for (Entry<XEventClass, Transition> entry : class2transition.entrySet()) {
                        result.add(new Pair<Transition, XEventClass>(entry.getValue(), entry.getKey()));
                }
                return result;
        }
        private boolean canExpandLeft(Tuple_Mod toExpand, XEventClass toAdd) {
                // Check if the event class in toAdd has a causal depencendy
                // with all elements of the rightPart of the tuple.
                for (XEventClass right : toExpand.rightPart) {
                        if (!hasCausalRelation(toAdd, right)) {
                                return false;
                        }
                }
                // Check if the event class in toAdd does not have a relation
                // with any of the elements of the leftPart of the tuple.
```

```
			for (XEventClass left : toExpand.leftPart) {
				if (hasRelation(toAdd, left)) {
					return false;
				}
			}

			return true;
		}
		private boolean canExpandRight(Tuple_Mod toExpand, XEventClass toAdd) {
			// Check if the event class in toAdd has a causal depencendy
			// from all elements of the leftPart of the tuple.
			for (XEventClass left : toExpand.leftPart) {
				if (!hasCausalRelation(left, toAdd)) {
					return false;
				}
			}
			// Check if the event class in toAdd does not have a relation
			// with any of the elements of the rightPart of the tuple.
			for (XEventClass right : toExpand.rightPart) {
				if (hasRelation(right, toAdd)) {
					return false;
				}
			}

			return true;
		}
		private boolean hasRelation(XEventClass from, XEventClass to) {
			if (!from.equals(to)) {
				if (isCausal(from, to)) {
					return true;
				}
				if (isCausal(to, from)) {
					return true;
				}
			}
			if (isParallel(from, to)) {
				return true;
			}
			return false;
		}
		private boolean hasCausalRelation(XEventClass from, XEventClass to) {

			if (isCausal(from, to)) {
				return true;
			}
			return false;
		}
		public Collection<Tuple_Mod> expandNode(Tuple_Mod toExpand, Progress progress, Collection<Tuple_Mod>
resultsSoFar) {
			Collection<Tuple_Mod> tuples = new HashSet<Tuple_Mod>();
			int startIndex = toExpand.maxLeftIndex + 1;
			for (int i = startIndex; i < eventClasses.size(); i++) {
				if (progress.isCancelled()) {
					return tuples;
				}
				XEventClass toAdd = eventClasses.get(i);
```

```
                        if (canExpandLeft(toExpand, toAdd)) {
                                Tuple_Mod newTuple = toExpand.clone();
                                newTuple.leftPart.add(toAdd);
                                newTuple.maxLeftIndex = i;
                                tuples.add(newTuple);
                        }
                }
                startIndex = toExpand.maxRightIndex + 1;
                for (int i = startIndex; i < eventClasses.size(); i++) {
                        if (progress.isCancelled()) {
                                return tuples;
                        }
                        XEventClass toAdd = eventClasses.get(i);

                        if (canExpandRight(toExpand, toAdd)) {
                                Tuple_Mod newTuple = toExpand.clone();
                                newTuple.rightPart.add(toAdd);
                                newTuple.maxRightIndex = i;
                                tuples.add(newTuple);
                        }
                }
                return tuples;
        }
        public void processLeaf(Tuple_Mod toAdd, Progress progress, Collection<Tuple_Mod> resultCollection) {
                synchronized (resultCollection) {
                        Iterator<Tuple_Mod> it = resultCollection.iterator();
                        boolean largerFound = false;
                        while (!largerFound && it.hasNext()) {
                                Tuple_Mod t = it.next();
                                if (t.isSmallerThan(toAdd)) {
                                        it.remove();
                                        continue;
                                }
                                largerFound = toAdd.isSmallerThan(t);
                        }
                        if (!largerFound) {
                                resultCollection.add(toAdd);
                        }
                }
        }
}
```

Prilog C

Programski kod priključka *Modified Alpha-algorithm - Minimal Logs*

```
package alphaminer_mod_find_minimal_logs_from_any_log;

import java.util.HashSet;
import java.util.Set;

import org.deckfour.xes.classification.XEventClass;

class Tuple_Mod {

	public Set<XEventClass> leftPart = new HashSet<XEventClass>();

	public Set<XEventClass> rightPart = new HashSet<XEventClass>();

	public int maxRightIndex = 0;
	public int maxLeftIndex = 0;

	public Tuple_Mod() {
	}
	public boolean isSmallerThan(Tuple_Mod tuple) {
		return tuple.leftPart.containsAll(leftPart) && tuple.rightPart.containsAll(rightPart);
	}
	public Tuple_Mod clone() {
		Tuple_Mod clone = new Tuple_Mod();
		clone.leftPart.addAll(leftPart);
		clone.rightPart.addAll(rightPart);
		clone.maxRightIndex = maxRightIndex;
		clone.maxLeftIndex = maxLeftIndex;
		return clone;
	}
	public int hashCode() {
		return leftPart.hashCode() + 37 * rightPart.hashCode() + maxRightIndex + maxLeftIndex;
	}
	public boolean equals(Object o) {
		if (o instanceof Tuple_Mod) {
			Tuple_Mod t = (Tuple_Mod) o;
			return (t.maxRightIndex == maxRightIndex) && (t.maxLeftIndex == maxLeftIndex)
					&& t.leftPart.equals(leftPart) && t.rightPart.equals(rightPart);
		}
		return false;
	}
	public String toString() {
		return "{" + leftPart.toString() + "} --> {" + rightPart.toString() + "}";
	}
}
package alphaminer_mod_find_minimal_logs_from_complete_log;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
```

```
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.Stack;
import java.util.concurrent.CancellationException;
import java.util.concurrent.ExecutionException;

import org.deckfour.xes.classification.XEventClass;
import org.deckfour.xes.classification.XEventClasses;
import org.deckfour.xes.extension.std.XConceptExtension;
import org.deckfour.xes.info.XLogInfo;
import org.deckfour.xes.info.XLogInfoFactory;
import org.deckfour.xes.model.XLog;
import org.deckfour.xes.model.XTrace;
import org.processmining.connections.logmodel.LogPetrinetConnectionImpl;
import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
import org.processmining.framework.connections.ConnectionCannotBeObtained;
import org.processmining.framework.plugin.PluginContext;
import org.processmining.framework.plugin.PluginExecutionResult;
import org.processmining.framework.plugin.PluginParameterBinding;
import org.processmining.framework.plugin.Progress;
import org.processmining.framework.plugin.annotations.Plugin;
import org.processmining.framework.plugin.annotations.PluginVariant;
import org.processmining.framework.plugin.events.Logger.MessageLevel;
import org.processmining.framework.util.Pair;
import org.processmining.framework.util.search.MultiThreadedSearcher;
import org.processmining.framework.util.search.NodeExpander;
import org.processmining.models.connections.petrinets.behavioral.InitialMarkingConnection;
import org.processmining.models.graphbased.directed.petrinet.Petrinet;
import org.processmining.models.graphbased.directed.petrinet.elements.Place;
import org.processmining.models.graphbased.directed.petrinet.elements.Transition;
import org.processmining.models.graphbased.directed.petrinet.impl.PetrinetFactory;
import org.processmining.models.semantics.petrinet.Marking;
import org.processmining.plugins.log.logabstraction.LogAbstractionConnection;
import org.processmining.plugins.log.logabstraction.LogRelations;

@Plugin(name = "Alpha Miner", parameterLabels = { "Log", "Summary", "Log Relations" }, returnLabels = { "Petrinet",
                "Marking" }, returnTypes = { Petrinet.class, Marking.class })
public class AlphaMiner implements NodeExpander<Tuple_Mod> {

        private LogRelations relations;
        private List<XEventClass> eventClasses;

        enum LogType {
                COMPLETE_LOG,
                CAUSALLY_COMPLETE_LOG,
                WEAKLY_COMPLETE_LOG
        }
                private LogType logType;

        // Return string, massage set appropriately bellow, this initial value is returned only in the case of some error.
        String ret = "Error occured while discovering the net.";

        //*************** 3 Variants without nets **********************//
        @UITopiaVariant(uiLabel = "Modified Alpha-algorithm - Minimal Logs", affiliation = UITopiaVariant.EHV,
author = "Lena Parezanovic", email = "pl123283m@student.etf.rs", pack = "AlphaMiner_Mod Find Minimal Logs from
Complete Log")
```

```
	@PluginVariant(variantLabel = "Default event classes", requiredParameterLabels = { 0 })
	public Object[] doMining(PluginContext context, XLog log) throws CancellationException, InterruptedException,
			ExecutionException {
		XLogInfo info = XLogInfoFactory.createLogInfo(log);
		return doAlphaMiningPrivate(context, log, info);
	}

	@PluginVariant(variantLabel = "User-defined event classes", requiredParameterLabels = { 0, 1 })
	public Object[] doMining(PluginContext context, XLog log, XLogInfo summary) throws CancellationException,
			InterruptedException, ExecutionException {
		return doAlphaMiningPrivate(context, log, summary);
	}

	@PluginVariant(variantLabel = "User-defined event classes and relations", requiredParameterLabels = { 1, 2 })
	public Object[] doMining(PluginContext context, XLogInfo summary, LogRelations relations)
			throws InterruptedException, ExecutionException {
		return doAlphaMiningPrivateWithRelations(context, summary, relations);
	}
	//************************************************************//

	private Object[] doAlphaMiningPrivate(PluginContext context, XLog log, XLogInfo summary)
			throws CancellationException, InterruptedException, ExecutionException {

		// First check if a connection providing log ralations exists.
		try {
			LogAbstractionConnection logAbstractionConnection =
context.getConnectionManager().getFirstConnection(LogAbstractionConnection.class, context, log);
			if (logAbstractionConnection != null) {
				LogRelations logRelations = logAbstractionConnection.getRelations();
				if (logRelations != null)
					return doAlphaMiningPrivateWithRelations(context, summary, logRelations);
			}
		} catch (ConnectionCannotBeObtained e) {
			// Ignore, we try another way later
		}
		// No log relations are specified, so find a plugin that can construct them.
		// This is done, by asking the plugin manager for a plugin, that:
		// 1) It's a plugin (i.e. the annotation is Plugin.class),
		// 2) It returns LogRelations (i.e. one of the return types is LogRelations.class or any subclass thereof),
		// 3) It can be executed in a child of this context, which is of type context.getPluginContextType(),
		// 4) It can be executed on the given input (i.e. no extra input is needed, and all input is used),
		// 5) It accepts the input in any given order (i.e. not in the specified order),
		// 6) It does not have to be user visible
		// 7) It can use objects given by log and summary (i.e. types log.getClass() and summary.getClass()).
		Collection<Pair<Integer, PluginParameterBinding>> plugins =
context.getPluginManager().find(Plugin.class,
				LogRelations.class, context.getPluginContextType(), true, false, false, XLog.class,
summary.getClass());

		if (plugins.isEmpty()) {
			context.log("No plugin found to create log relations, please specify relations manually",
					MessageLevel.ERROR);
			return null;
		}
		// Let's just take the first available plugin for the job of constructing log abstractions
		Pair<Integer, PluginParameterBinding> plugin = plugins.iterator().next();
```

```
		// Now, the binding can be executed on the log and the summary
		// FIrst, we instantiate a new context for this plugin, which is a child context of the current context.
		PluginContext c2 = context.createChildContext("Log Relation Constructor");

		// Let's notify our lifecyclelisteners about the fact that we created a new context. this is
		// optional, but if this is not done, then the user interface doesn't show it (if there is a UI).
		context.getPluginLifeCycleEventListeners().firePluginCreated(c2);

		// At this point, we execute the binding to get the LogRelations. For this, we call the invoke method
		// on the PluginParameterBinding stored in the plugin variable. The return type is LogRelations.class and
		// as input we give the new context c2, the log and the summary. Note that the plugin might return
mulitple
		// objects, hence we extract the object with number x, where x is stored as the first element of the plugin
		// variable.

		PluginExecutionResult pluginResult = plugin.getSecond().invoke(c2, log, summary);
		pluginResult.synchronize();
		LogRelations relations = pluginResult.<LogRelations>getResult(plugin.getFirst());

		// Now we have the relations and we can continue with the mining.
		return doAlphaMiningPrivateWithRelations(context, relations.getSummary(), relations);
	}
	public HashSet<XTrace> traces;
	public HashSet<Pair<XEventClass, XEventClass>> parallel_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> parallel_relationes_causal_log;
	public HashSet<Pair<XEventClass, XEventClass>> direct_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> causal_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> causal_relationes_causal_log;
	public HashSet<Pair<XEventClass, XEventClass>> indirect_relationes;
	public HashSet<Pair<XEventClass, XEventClass>> inferred_causal_relations;
	public HashSet<Pair<XEventClass, XEventClass>> basic_relation;
	public HashSet<XEventClass> nodes_without_successor;
	public HashSet<XEventClass> nodes_without_predecessor;
	public HashSet<Pair<XEventClass, XEventClass>> indirect_causal_relationes;
	public HashSet<Pair<XEventClass,XEventClass>> temp;
	public HashSet<Pair<XEventClass, XEventClass>> parallel_relationes_compl_log;
	public HashSet<Pair<XEventClass, XEventClass>> causal_relationes_compl_log;

	public void find_direct (Set<XTrace> set, XLogInfo summary) {
		XEventClasses classes = summary.getEventClasses();
		direct_relationes.clear();

		for (XTrace trace : set) {
			for (int j=0; j<trace.size()-1; j++) {
				XEventClass e1 = classes.getClassOf(trace.get(j));
				XEventClass e2 = classes.getClassOf(trace.get(j+1));
				Pair<XEventClass, XEventClass> pair = new Pair<XEventClass, XEventClass>(e1,
e2);

				if(!direct_relationes.contains(pair)) {
					direct_relationes.add(pair);
				}
			}
		}
	}
	public void find_indirect (Set<XTrace> set, XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();
```

```
		indirect_relationes.clear();

		for (XTrace trace : set) {
			for (int j=0; j<trace.size()-2; j++) {
				for (int k=j+2; k<trace.size(); k++) {
					XEventClass e1 = classes.getClassOf(trace.get(j));
					XEventClass e2 = classes.getClassOf(trace.get(k));
					Pair<XEventClass, XEventClass> pair = new Pair<XEventClass,
XEventClass>(e1, e2);
					if(!indirect_relationes.contains(pair) && !direct_relationes.contains(pair)) {
						indirect_relationes.add(pair);
					}
				}
			}
		}
	}
	public void find_parallel (XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();
		parallel_relationes.clear();
		for ( XEventClass e1 : classes.getClasses()) {
			for ( XEventClass e2 : classes.getClasses()) {
				Pair<XEventClass, XEventClass> pair1 = new Pair<XEventClass, XEventClass>(e1,
e2);
				Pair<XEventClass, XEventClass> pair2 = new Pair<XEventClass, XEventClass>(e2,
e1);
				if ((direct_relationes.contains(pair1) && direct_relationes.contains(pair2))
					|| (direct_relationes.contains(pair1) && indirect_relationes.contains(pair2))
					|| (direct_relationes.contains(pair2) && indirect_relationes.contains(pair1))
					|| (indirect_relationes.contains(pair1) && indirect_relationes.contains(pair2))
					) {

					if (!parallel_relationes.contains(pair1) && !parallel_relationes.contains(pair2))
					{
						parallel_relationes.add(pair1);
						parallel_relationes.add(pair2);
					}
				}

			}
		}
	}
	public void find_parallel_compl_log (Set<XTrace> set, XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();
		parallel_relationes_compl_log.clear();
		for ( XEventClass e1 : classes.getClasses()) {
			for ( XEventClass e2 : classes.getClasses()) {
				Pair<XEventClass, XEventClass> pair1 = new Pair<XEventClass, XEventClass>(e1,
e2);
				Pair<XEventClass, XEventClass> pair2 = new Pair<XEventClass, XEventClass>(e2,
e1);
				if ((direct_relationes.contains(pair1) && direct_relationes.contains(pair2))
					) {

					if (!parallel_relationes_compl_log.contains(pair1) &&
!parallel_relationes_compl_log.contains(pair2))
```

```
                                        {
                                                parallel_relationes_compl_log.add(pair1);
                                                parallel_relationes_compl_log.add(pair2);
                                        }
                                }
                        }
                }
        }
        public void find_indirect_causal(XLogInfo summmary) {
                indirect_causal_relationes.clear();
                indirect_causal_relationes.addAll(indirect_relationes);
                Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
                while(it.hasNext())
                {
                        Pair<XEventClass, XEventClass> pair = it.next();
                        if(direct_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getSecond(),
pair.getFirst()))
                                ||
                                indirect_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getSecond(),
pair.getFirst()))
                                ) {
                                it.remove();
                        }
                }
        }
        public void find_causal(XLogInfo summmary) {
                causal_relationes.clear();
                causal_relationes_compl_log.clear();
                causal_relationes.addAll(direct_relationes);
                causal_relationes_compl_log.addAll(direct_relationes);

                causal_relationes.removeAll(parallel_relationes);
                causal_relationes_compl_log.removeAll(parallel_relationes_compl_log);
        }
        public void find_nodes_without_successor(XLogInfo summary) {
                XEventClasses classes = summary.getEventClasses();
                nodes_without_successor.clear();
                XTrace trace = summary.getLog().get(0);
                boolean hasSuccessor = false;
                for (int i=0; i<trace.size(); i++) {
                        hasSuccessor = false;
                        XEventClass e = classes.getClassOf(trace.get(i));
                        Iterator<Pair<XEventClass, XEventClass>> it = causal_relationes.iterator();
                        while(it.hasNext()) {
                                Pair<XEventClass, XEventClass> pair = it.next();
                                if(pair.getFirst().equals(e)) {
                                        hasSuccessor = true;
                                        break;
                                }
                        }
                        if (hasSuccessor == false) {
                                nodes_without_successor.add(e);
                        }
                }
        }
        public void find_nodes_without_predecessor(XLogInfo summary) {
                XEventClasses classes = summary.getEventClasses();
                nodes_without_predecessor.clear();
```

```
		XTrace trace = summary.getLog().get(0);
		boolean hasPredecessor = false;
		for (int i=0; i<trace.size(); i++) {
			hasPredecessor = false;
			XEventClass e = classes.getClassOf(trace.get(i));
			Iterator<Pair<XEventClass, XEventClass>> it = causal_relationes.iterator();
			while(it.hasNext()) {
				Pair<XEventClass, XEventClass> pair = it.next();
				if(pair.getSecond().equals(e)) {
					hasPredecessor = true;
					break;
				}
			}
			if (hasPredecessor == false) {
				nodes_without_predecessor.add(e);
			}
		}
	}
	public void weakly_completed_logs_find_causal_from_indirect_successor(XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();
		XTrace trace = summary.getLog().get(0);
		inferred_causal_relations.clear();
		Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
		while(it.hasNext()) {
			Pair<XEventClass, XEventClass> pair = it.next();
			for (int i=0; i<trace.size(); i++) {
				XEventClass e = classes.getClassOf(trace.get(i));
				if (!nodes_without_successor.contains(pair.getFirst())) continue;
				if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(e,
pair.getSecond()))
					&&
					parallel_relationes.contains(new Pair<XEventClass, XEventClass>(e,
pair.getFirst()))) {
						inferred_causal_relations.add(pair);
						break;
				}
			}
		}
	}
	public void weakly_completed_logs_find_causal_from_indirect_predecessor(XLogInfo summary) {

		XEventClasses classes = summary.getEventClasses();
		XTrace trace = summary.getLog().get(0);
		inferred_causal_relations.clear();
		Iterator<Pair<XEventClass, XEventClass>> it = indirect_causal_relationes.iterator();
		while(it.hasNext()) {
			Pair<XEventClass, XEventClass> pair = it.next();
			for (int i=0; i<trace.size(); i++) {
				XEventClass e = classes.getClassOf(trace.get(i));
				if (!nodes_without_predecessor.contains(pair.getSecond())) continue;
				if (causal_relationes.contains(new Pair<XEventClass, XEventClass>(pair.getFirst(), e))
					&&
					parallel_relationes.contains(new Pair<XEventClass,
XEventClass>(pair.getSecond(), e))) {
						inferred_causal_relations.add(pair);
						break;
```

```
                                }
                        }
                }
        }

        public static <XTrace> Set<Set<XTrace>> powerSet(Set<XTrace> originalSet) {
           Set<Set<XTrace>> sets = new HashSet<Set<XTrace>>();
           if (originalSet.isEmpty()) {
                sets.add(new HashSet<XTrace>());
                return sets;
           }
           List<XTrace> list = new ArrayList<XTrace>(originalSet);
           XTrace head = list.get(0);
           Set<XTrace> rest = new HashSet<XTrace>(list.subList(1, list.size()));
           for (Set<XTrace> set : powerSet(rest)) {
                Set<XTrace> newSet = new HashSet<XTrace>();
                newSet.add(head);
                newSet.addAll(set);
                sets.add(newSet);
                sets.add(set);
           }
           return sets;
        }
        private Object[] doAlphaMiningPrivateWithRelations(PluginContext context, XLogInfo summary, LogRelations
relations)
                        throws InterruptedException, ExecutionException {
                this.relations = relations;
                eventClasses = new ArrayList<XEventClass>(summary.getEventClasses().size());
                eventClasses.addAll(summary.getEventClasses().getClasses());
                eventClasses.removeAll(relations.getLengthOneLoops().keySet());
                final Progress progress = context.getProgress();

                progress.setMinimum(0);
                progress.setMaximum(5);
                progress.setIndeterminate(false);

                ////////////////////////////////// Reading causal relation from complete log //////////////////////////////////

                basic_relation = new HashSet<Pair<XEventClass,XEventClass>>();
                Iterator it = relations.getCausalDependencies().keySet().iterator();
                while(it.hasNext()) {
                        basic_relation.add((Pair<XEventClass, XEventClass>) it.next());
                }
                ////////////////////////////////////////////////////////////////////////////////////////////////////////////

                temp = new HashSet<Pair<XEventClass,XEventClass>>();
                direct_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
                parallel_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
                causal_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
                indirect_relationes = new HashSet<Pair<XEventClass,XEventClass>>();
                inferred_causal_relations = new HashSet<Pair<XEventClass,XEventClass>>();
                nodes_without_successor = new HashSet<XEventClass>();
                nodes_without_predecessor = new HashSet<XEventClass>();
                indirect_causal_relationes = new HashSet<Pair<XEventClass, XEventClass>>();
                parallel_relationes_compl_log = new HashSet<Pair<XEventClass, XEventClass>>();
                causal_relationes_compl_log = new HashSet<Pair<XEventClass, XEventClass>>();

                Set<XTrace> originalSet = new HashSet<XTrace>();
```

```
		for (int i=0; i<summary.getNumberOfTraces(); i++) {
			XTrace trace = summary.getLog().get(i);
			originalSet.add(trace);
		}
		Set<Set<XTrace>> sets = powerSet(originalSet);
		ArrayList<Set<XTrace>> list = new ArrayList<Set<XTrace>>(sets);
		for (int i=0; i<list.size()-1; i++) {
			for (int j=i+1; j<list.size(); j++) {
				if (list.get(i).size() > list.get(j).size()) {
					Set<XTrace> t = list.get(i);
					list.set(i, list.get(j));
					list.set(j, t);
				}
			}
		}
		boolean minimal_weak = false;
		boolean minimal_causal = false;
		boolean minimal_compl = false;
		int traces_compl_log_num = 0;
		int traces_causal_log_num = 0;
		int traces_weak_log_num = 0;

	for(int i=0; i<list.size() && !(minimal_causal && minimal_weak && minimal_compl); i++) {

		Set<XTrace> set = list.get(i);
			find_direct(set, summary);
			find_indirect(set, summary);
			find_parallel(summary);
			find_parallel_compl_log(set, summary);
			find_causal(summary);
			find_indirect_causal(summary);

			////////////////// Check the type of log and if it is minimal ////////////////////////

			if (causal_relationes_compl_log.containsAll(relations.getCausalDependencies().keySet())
					&&
relations.getCausalDependencies().keySet().containsAll(causal_relationes_compl_log)
					&& !minimal_compl) {
				if(parallel_relationes_compl_log.containsAll(relations.getParallelRelations().keySet())
						&&
relations.getParallelRelations().keySet().containsAll(parallel_relationes_compl_log)) {
					System.out.println();
					System.out.println("MINIMAL COMPLETE LOG!!!!!");
					XEventClasses classes = summary.getEventClasses();
					for(XTrace t : set) {
						System.out.println("trace");
						for(int j=0; j<t.size(); j++) {
							System.out.println(classes.getClassOf(t.get(j)));
						}
					}
					traces_compl_log_num = set.size();
					System.out.println("Number of traces in the minimal complete log: " +
traces_compl_log_num);
					System.out.println("-------------------------------------------------------");
					minimal_compl = true;
				}
			}
```

```
                        if (causal_relationes.containsAll(basic_relation) &&
basic_relation.containsAll(causal_relationes) && !minimal_causal) {
                                logType = LogType.CAUSALLY_COMPLETE_LOG;
                                System.out.println();
                                System.out.println("MINIMAL CAUSAL LOG!!!!!");
                                XEventClasses classes = summary.getEventClasses();
                                for(XTrace t : set) {
                                        System.out.println("trace");
                                        for(int j=0; j<t.size(); j++) {
                                                System.out.println(classes.getClassOf(t.get(j)));
                                        }
                                }
                                traces_causal_log_num = set.size();
                                System.out.println("Number of traces in the minimal causal log: " +
traces_causal_log_num);
                                System.out.println("------------------------------------------------------------");
                                minimal_causal = true;
                        }
                        temp.clear();
                        temp.addAll(causal_relationes);
                        temp.addAll(indirect_causal_relationes);
                        if(temp.containsAll(relations.getCausalDependencies().keySet())
                                && relations.getCausalDependencies().keySet().containsAll(causal_relationes)
                                && !minimal_weak) {
                                        find_nodes_without_successor(summary);
                                        find_nodes_without_predecessor(summary);
                                        weakly_completed_logs_find_causal_from_indirect_successor(summary);
                                        weakly_completed_logs_find_causal_from_indirect_predecessor(summary);
                                        causal_relationes.addAll(inferred_causal_relations);
                                        logType = LogType.WEAKLY_COMPLETE_LOG;

                                        System.out.println();
                                        System.out.println("MINIMAL WEAK LOG!!!!!");
                                        XEventClasses classes = summary.getEventClasses();
                                        for(XTrace t : set) {
                                                System.out.println("trace");
                                                for(int j=0; j<t.size(); j++) {
                                                        System.out.println(classes.getClassOf(t.get(j)));
                                                }
                                        }
                                        traces_weak_log_num = set.size();
                                        System.out.println("Number of traces in the minimal weak log: " +
traces_weak_log_num);
                                        System.out.println("------------------------------------------------------------");
                                        minimal_weak = true;
                        }
                }
                final Stack<Tuple_Mod> stack = new Stack<Tuple_Mod>();

                // Initialize the tuples to the causal depencencies in the log
                for (Pair<XEventClass, XEventClass> causal : causal_relationes) {
                        if (!isCausal(causal.getFirst(), causal.getSecond())) {
                                continue;
                        }
                        if (progress.isCancelled()) {
                                context.getFutureResult(0).cancel(true);
                                return new Object[] { null, null, null };
                        }
```

```
                        if (!eventClasses.contains(causal.getFirst()) || !eventClasses.contains(causal.getSecond())) {
                                continue;
                        }
                        Tuple_Mod tuple = new Tuple_Mod();
                        tuple.leftPart.add(causal.getFirst());
                        tuple.rightPart.add(causal.getSecond());
                        tuple.maxRightIndex = eventClasses.indexOf(causal.getSecond());
                        tuple.maxLeftIndex = eventClasses.indexOf(causal.getFirst());
                        stack.push(tuple);
                }
                progress.inc();

                // Expand the tuples
                final List<Tuple_Mod> result = new ArrayList<Tuple_Mod>();

                MultiThreadedSearcher<Tuple_Mod> searcher = new MultiThreadedSearcher<Tuple_Mod>(this,
                                MultiThreadedSearcher.BREADTHFIRST);

                searcher.addInitialNodes(stack);
                searcher.startSearch(context.getExecutor(), progress, result);

                if (progress.isCancelled()) {
                        context.getFutureResult(0).cancel(true);
                        return new Object[] { null, null, null };
                }
                // Add transitions
                Map<XEventClass, Transition> class2transition = new HashMap<XEventClass, Transition>();

                Petrinet net = PetrinetFactory.newPetrinet("Petrinet from "
                                + XConceptExtension.instance().extractName(relations.getLog())+" , mined with
AlphaMiner_Mod");

                context.getFutureResult(0).setLabel(net.getLabel());
                context.getFutureResult(1).setLabel("Initial Marking of " + net.getLabel());

                for (XEventClass eventClass : summary.getEventClasses().getClasses()) {
                        Transition transition = net.addTransition(eventClass.toString());
                        class2transition.put(eventClass, transition);
                }
                progress.inc();

                Map<Tuple_Mod, Place> tuple2place = new HashMap<Tuple_Mod, Place>();
                // Add places for each tuple
                for (Tuple_Mod tuple : result) {
                        Place p = net.addPlace(tuple.toString());
                        for (XEventClass eventClass : tuple.leftPart) {
                                net.addArc(class2transition.get(eventClass), p);
                        }
                        for (XEventClass eventClass : tuple.rightPart) {
                                net.addArc(p, class2transition.get(eventClass));
                        }
                        tuple2place.put(tuple, p);
                }
                progress.inc();

                Marking m = new Marking();
```

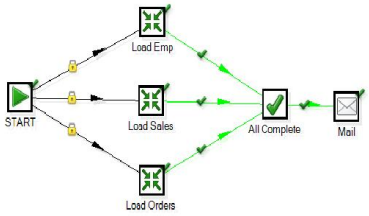
```
		// Add initial and final place
		Place pstart = net.addPlace("Start");
		for (XEventClass eventClass : relations.getStartTraceInfo().keySet()) {
			net.addArc(pstart, class2transition.get(eventClass));
		}
		m.add(pstart);

		Place pend = net.addPlace("End");
		for (XEventClass eventClass : relations.getEndTraceInfo().keySet()) {
			net.addArc(class2transition.get(eventClass), pend);
		}
		progress.inc();

		context.addConnection(new InitialMarkingConnection(net, m));
		context.addConnection(new LogPetrinetConnectionImpl(summary.getLog(), summary.getEventClasses(),
net, reverse(class2transition)));
		if (logType == LogType.CAUSALLY_COMPLETE_LOG) {
			ret = "The log is causally complete, the net is successfully discovered.";
		}
		else if (logType == LogType.WEAKLY_COMPLETE_LOG) {
			ret = "The log is weakly complete, the net is successfully discovered.";
		}
		return new Object[] { net, m, ret };
	}
	/**
	 * Flip the mapping given around (so values map to keys); handles multiple values with same key.
	 * @param class2transition
	 * @return
	 */
	protected Collection<Pair<Transition, XEventClass>> reverse(Map<XEventClass, Transition> class2transition) {
		List<Pair<Transition, XEventClass>> result = new
ArrayList<Pair<Transition,XEventClass>>(class2transition.size());
		for (Entry<XEventClass, Transition> entry : class2transition.entrySet()) {
			result.add(new Pair<Transition, XEventClass>(entry.getValue(), entry.getKey()));
		}
		return result;
	}
	private boolean canExpandLeft(Tuple_Mod toExpand, XEventClass toAdd) {
		// Check if the event class in toAdd has a causal depencendy
		// with all elements of the rightPart of the tuple.
		for (XEventClass right : toExpand.rightPart) {
			if (!hasCausalRelation(toAdd, right)) {
				return false;
			}
		}
		// Check if the event class in toAdd does not have a relation
		// with any of the elements of the leftPart of the tuple.
		for (XEventClass left : toExpand.leftPart) {
			if (hasRelation(toAdd, left)) {
				return false;
			}
		}
		return true;
	}
	private boolean canExpandRight(Tuple_Mod toExpand, XEventClass toAdd) {
		// Check if the event class in toAdd has a causal depencendy
		// from all elements of the leftPart of the tuple.
		for (XEventClass left : toExpand.leftPart) {
```

```
                        if (!hasCausalRelation(left, toAdd)) {
                                return false;
                        }
                }
                // Check if the event class in toAdd does not have a relation
                // with any of the elements of the rightPart of the tuple.
                for (XEventClass right : toExpand.rightPart) {
                        if (hasRelation(right, toAdd)) {
                                return false;
                        }
                }
                return true;
        }
        public boolean isCausal(XEventClass from, XEventClass to)
        {
                Pair<XEventClass, XEventClass> e1 = new Pair<XEventClass, XEventClass>(from, to);
                Pair<XEventClass, XEventClass> e2 = new Pair<XEventClass, XEventClass>(to, from);

                if (causal_relationes.contains(e1)){
                        return true;
                }
                return false;
        }
        public boolean isParallel(XEventClass from, XEventClass to)
        {
                if (parallel_relationes.contains(new Pair<XEventClass, XEventClass>(from, to))) {
                        return true;
                }
                return false;
        }
        private boolean hasRelation(XEventClass from, XEventClass to) {
                if (!from.equals(to)) {
                        if (isCausal(from, to)) {
                                return true;
                        }
                        if (isCausal(to, from)) {
                                return true;
                        }
                }
                if (isParallel(from, to)) {
                        return true;
                }
                return false;
        }
        private boolean hasCausalRelation(XEventClass from, XEventClass to) {

                if (isCausal(from, to)) {
                        return true;
                }
                return false;
        }
        public Collection<Tuple_Mod> expandNode(Tuple_Mod toExpand, Progress progress, Collection<Tuple_Mod>
resultsSoFar) {

                Collection<Tuple_Mod> tuples = new HashSet<Tuple_Mod>();
                int startIndex = toExpand.maxLeftIndex + 1;
                for (int i = startIndex; i < eventClasses.size(); i++) {
```

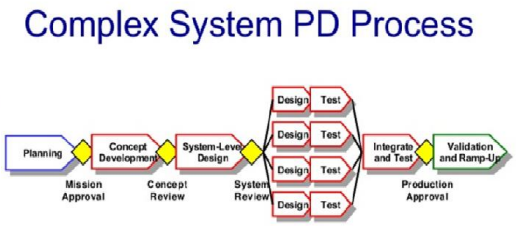
```
                        if (progress.isCancelled()) {
                                return tuples;
                        }
                        XEventClass toAdd = eventClasses.get(i);

                        if (canExpandLeft(toExpand, toAdd)) {
                                Tuple_Mod newTuple = toExpand.clone();
                                newTuple.leftPart.add(toAdd);
                                newTuple.maxLeftIndex = i;
                                tuples.add(newTuple);
                        }
                }
                startIndex = toExpand.maxRightIndex + 1;
                for (int i = startIndex; i < eventClasses.size(); i++) {
                        if (progress.isCancelled()) {
                                return tuples;
                        }
                        XEventClass toAdd = eventClasses.get(i);

                        if (canExpandRight(toExpand, toAdd)) {
                                Tuple_Mod newTuple = toExpand.clone();
                                newTuple.rightPart.add(toAdd);
                                newTuple.maxRightIndex = i;
                                tuples.add(newTuple);
                        }
                }
                return tuples;
        }
        public void processLeaf(Tuple_Mod toAdd, Progress progress, Collection<Tuple_Mod> resultCollection) {
                synchronized (resultCollection) {
                        Iterator<Tuple_Mod> it = resultCollection.iterator();
                        boolean largerFound = false;
                        while (!largerFound && it.hasNext()) {
                                Tuple_Mod t = it.next();
                                if (t.isSmallerThan(toAdd)) {
                                        it.remove();
                                        continue;
                                }
                                largerFound = toAdd.isSmallerThan(t);
                        }
                        if (!largerFound) {
                                resultCollection.add(toAdd);
                        }
                }
        }
}
```

Prilog D

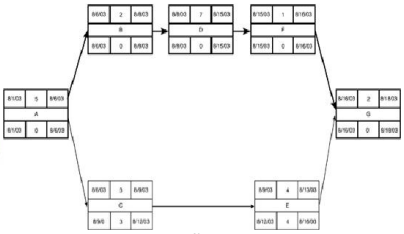
Primeri paralelnih poslovnih procesa



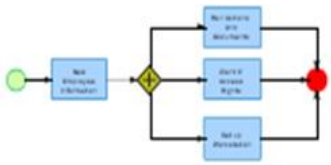
[UNSET]



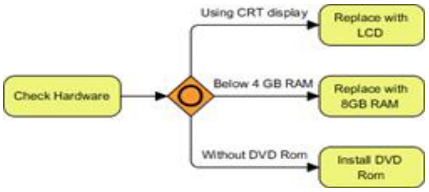
2-development-processes-and- organizations--14-638



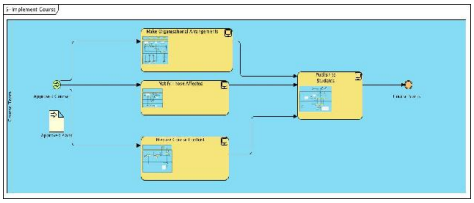
04fig06



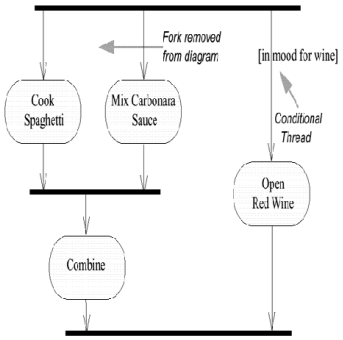
4fe199050cf2d4c6ff087f44



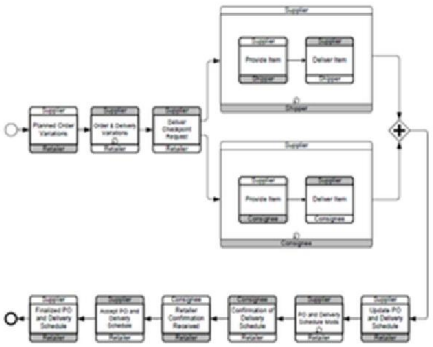
05_inclusive_gateway



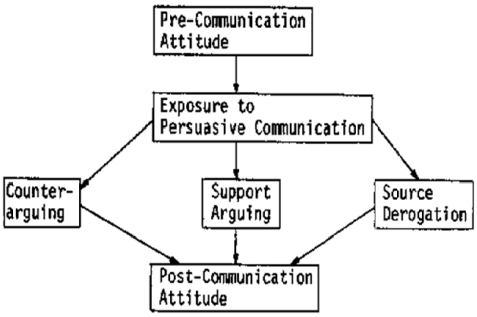
5-Implement-Course



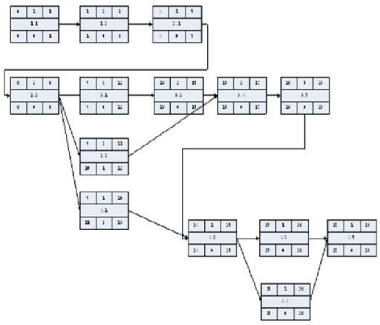
09fig02



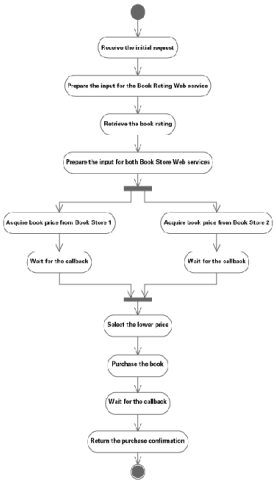
300px-ChoreographySpecExample2



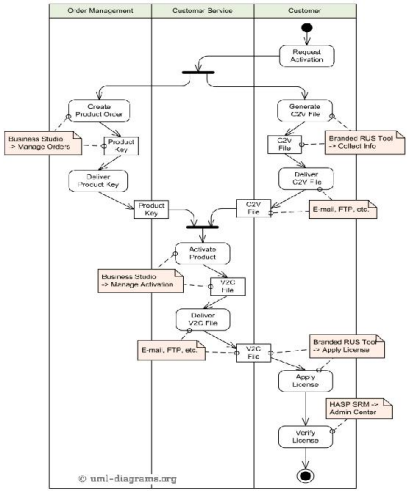
04367f04



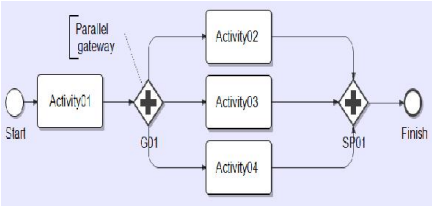
101308-1104-information5



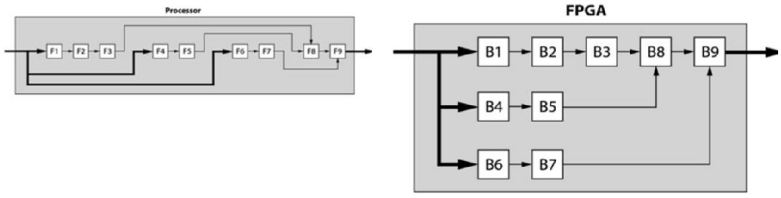
119258



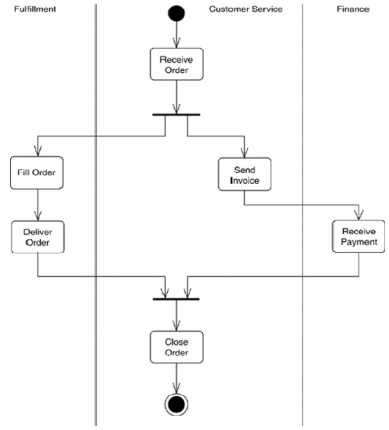
activity-examples-hasp-activation



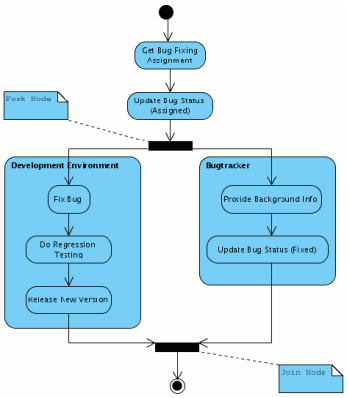
6012.image029



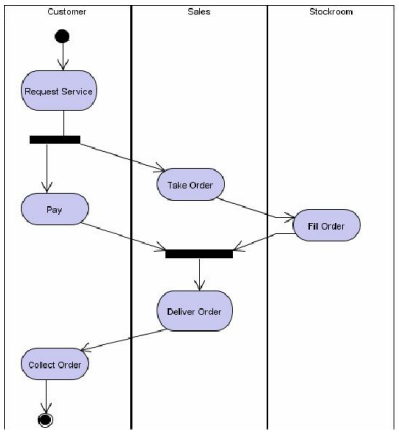
Accelerating data processing via parallelism



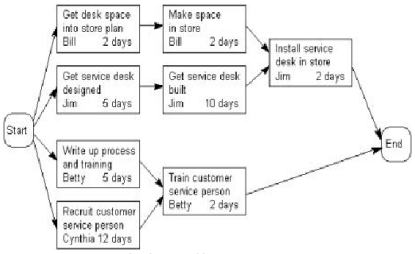
activity_swim_lanes



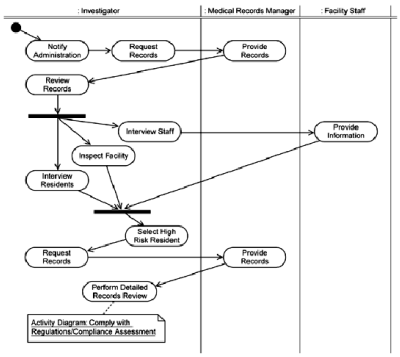
activityDiagram-forkJoin



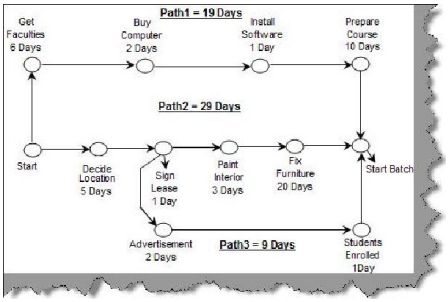
activity-example



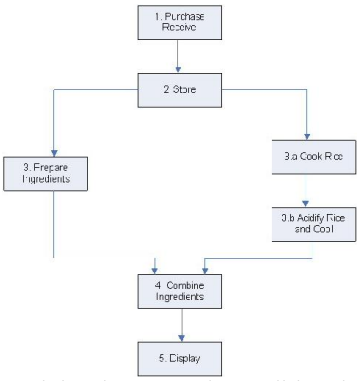
activity_diagram2



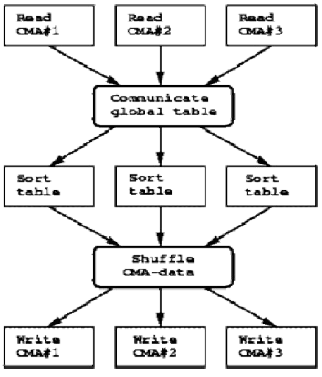
afig06



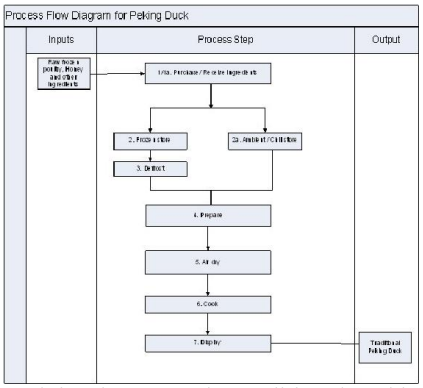
AONforComputerInstitute



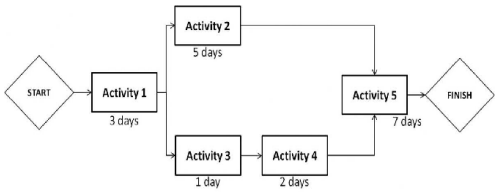
applying-haccp-to-the-traditional-sushi-chinese-style-duck-clip-ons-otp-fcp-1



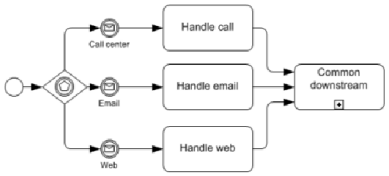
Assimilation-10-6-3



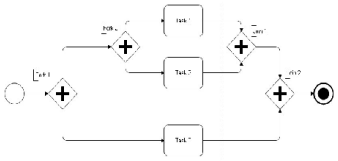
applying-haccp-to-the-traditional-sushi-chinese-style-duck-clip-ons-otp-fcp-2



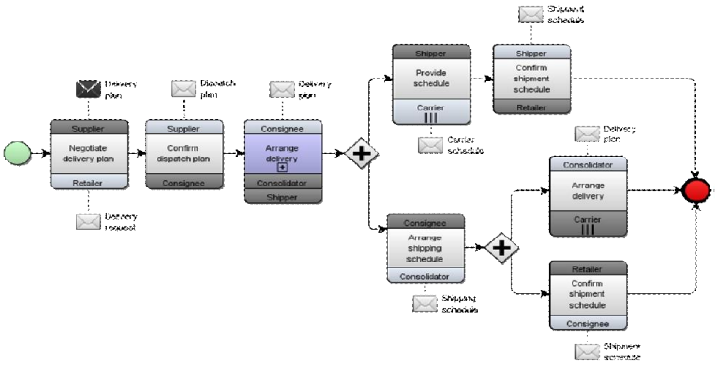
apmas-cpm-002



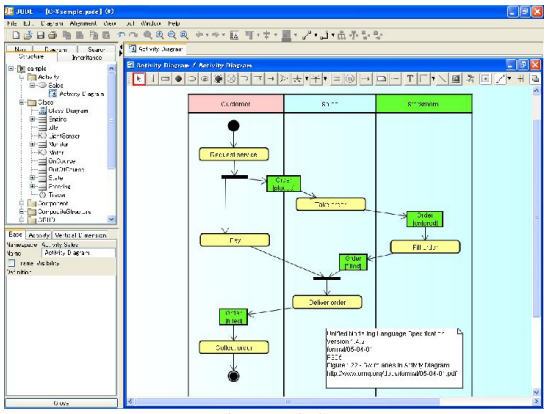
bootstrap



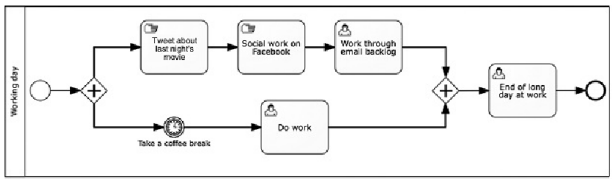
BPMN diagram



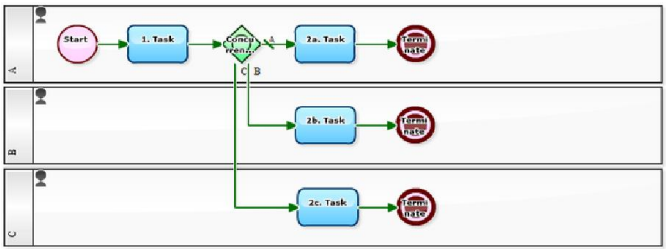
Choreography2



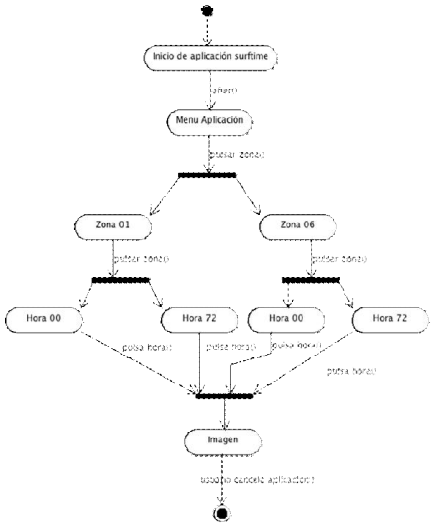
community_activity_en



BPM image_4



BPMN-terminate-end-event-2-en



Diagramadeactividad

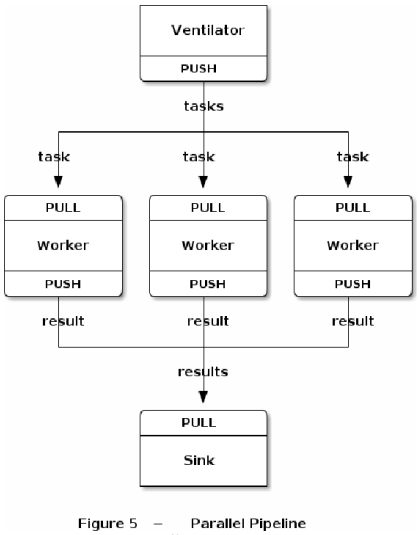
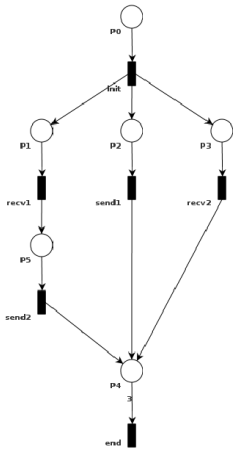
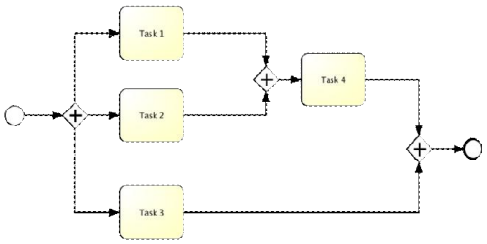


Figure 5 – Parallel Pipeline

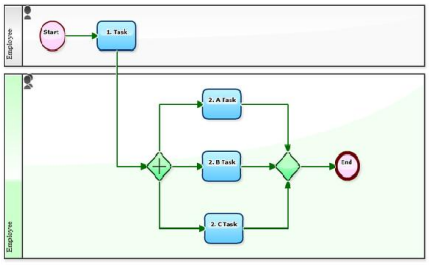
fig5



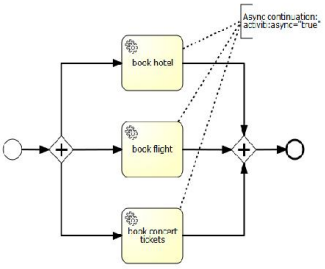
example-petri-net



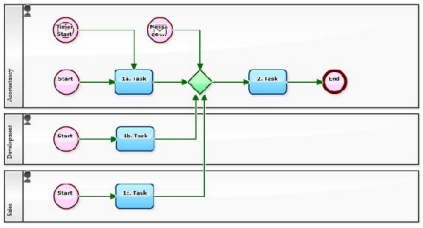
bpmn.unbalanced.parallel.gateway



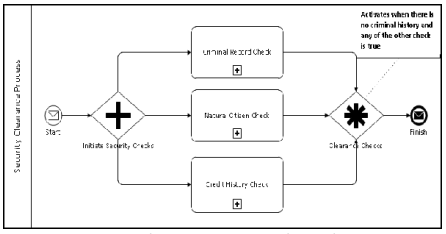
BPMN-ad-hoc-parallel-en



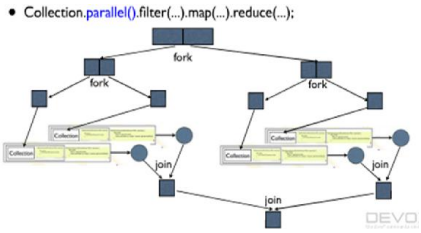
bpmn.why.exclusive.jobs



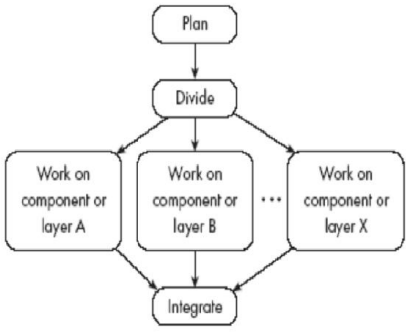
BPMN-multi-start-2-en



complexgateway_thumb



image_thumb26



Concurrent

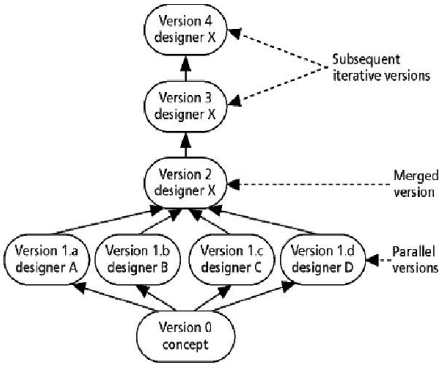
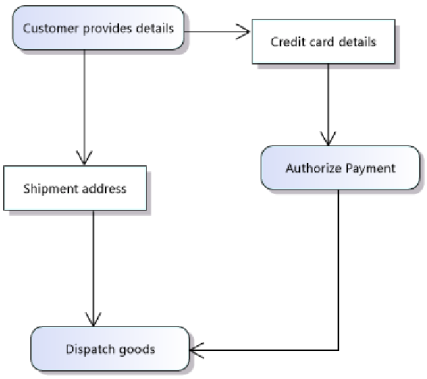
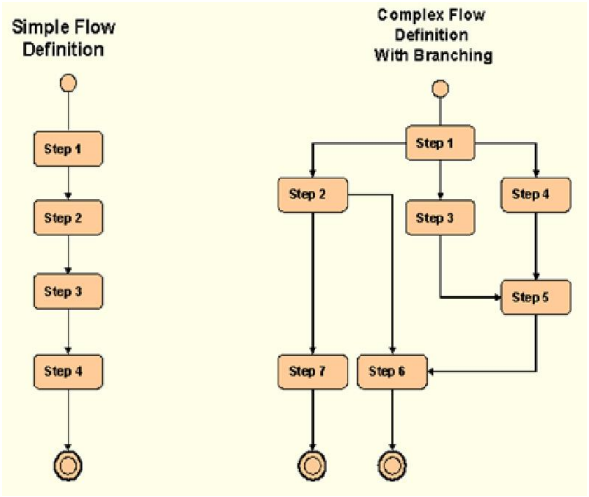


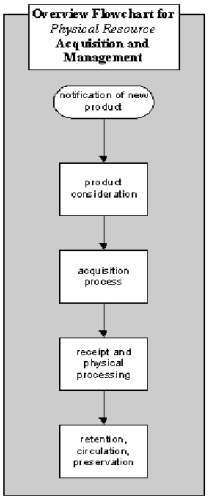
fig1

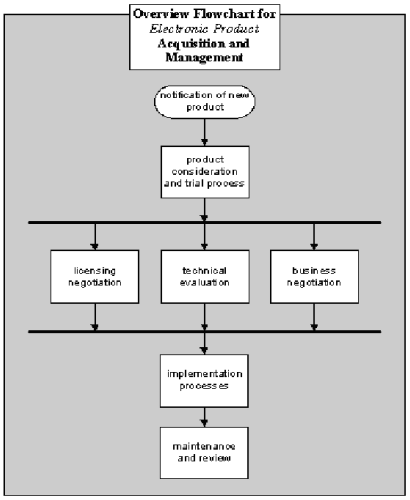


IC267861

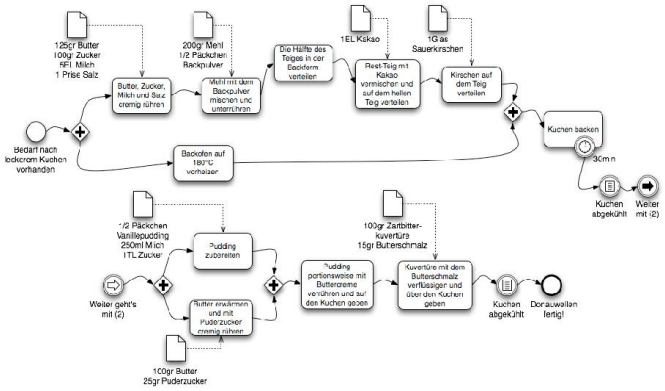


h-00100050000_image002





dlfermi0408appb01



Donauwellen

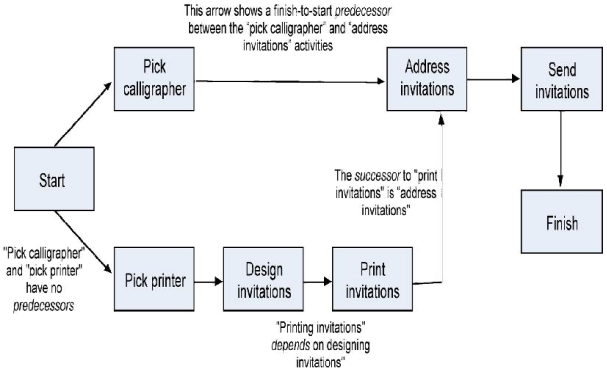
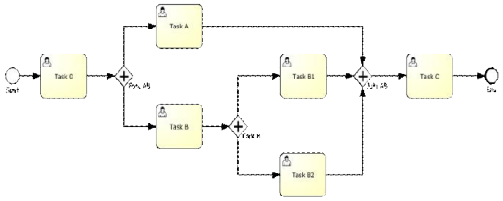
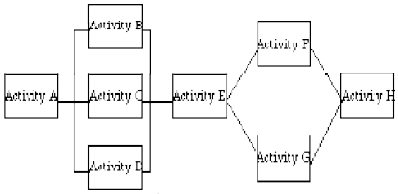


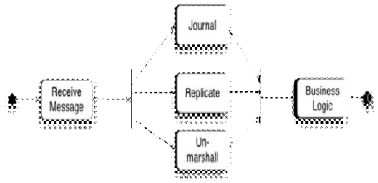
Fig30



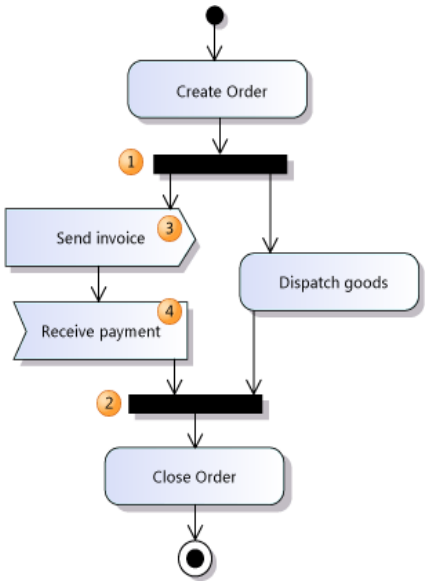
ForkJoin



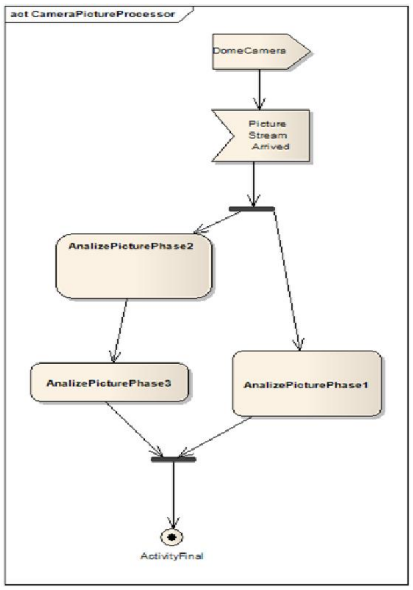
gloss5



input-activity



IC267860



image_thumb[9]

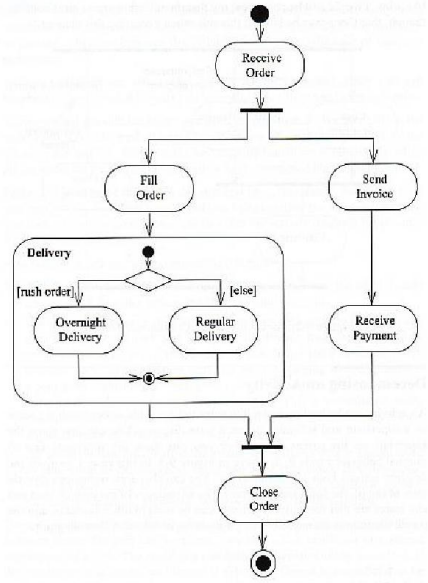
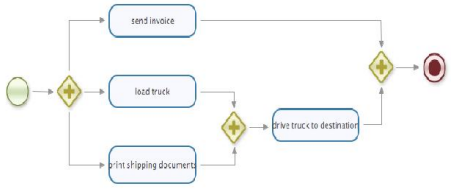
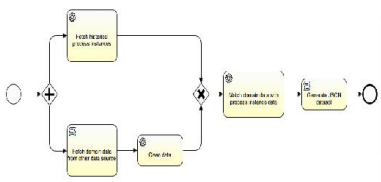


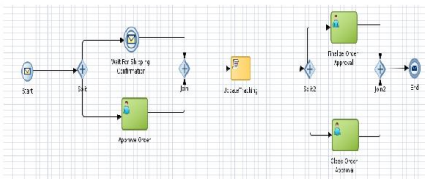
image013



process.concurrency2



Screen-Shot-2013-03-22-at-13.22.49



wftask_bpmn_1

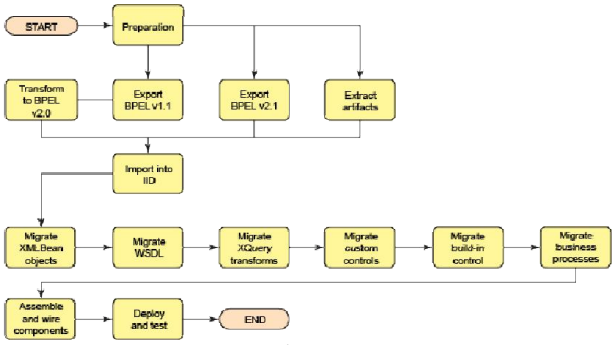


image001

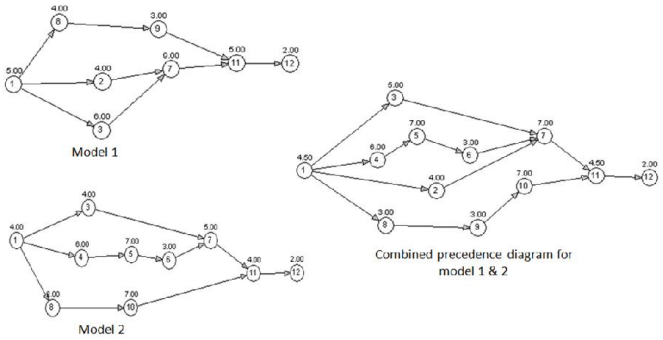


image34

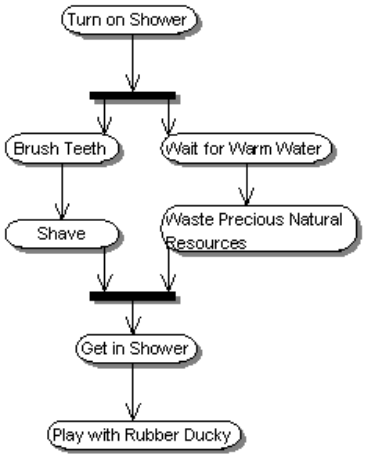
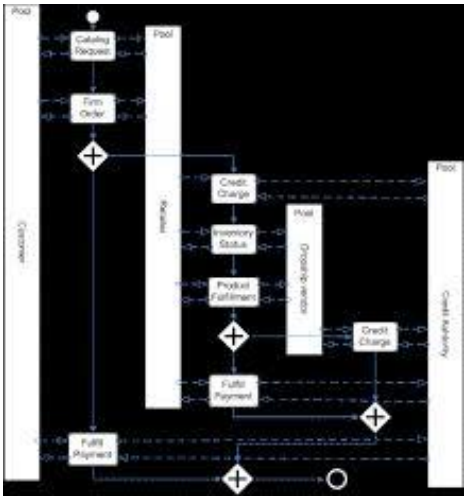
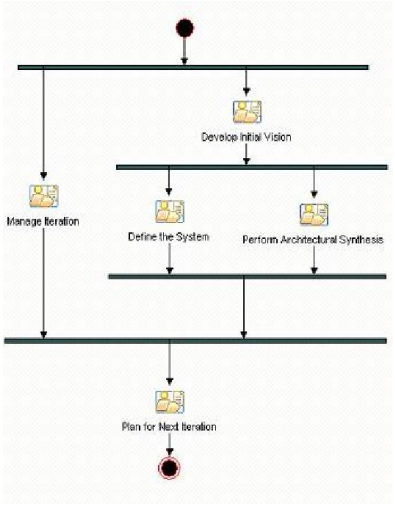


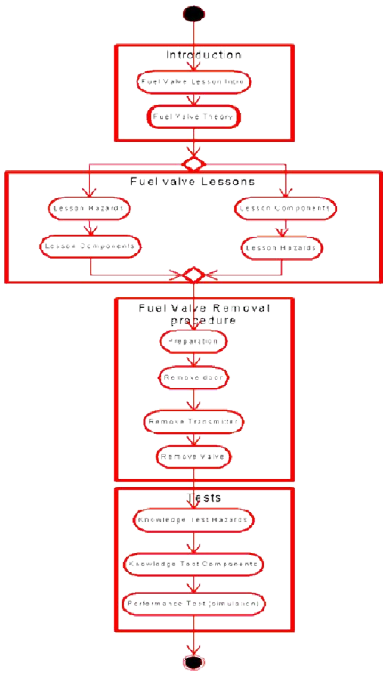
image0111



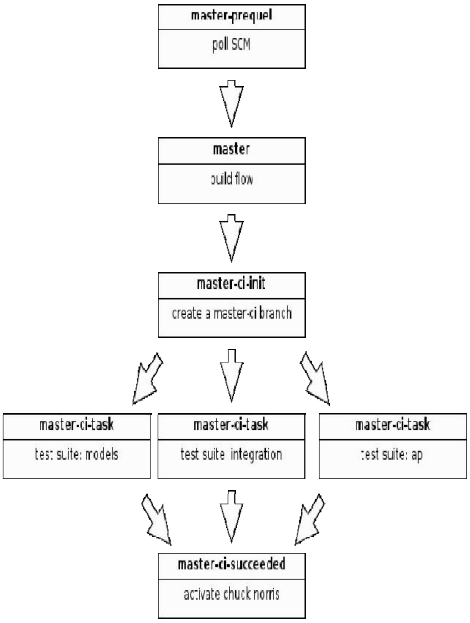
images



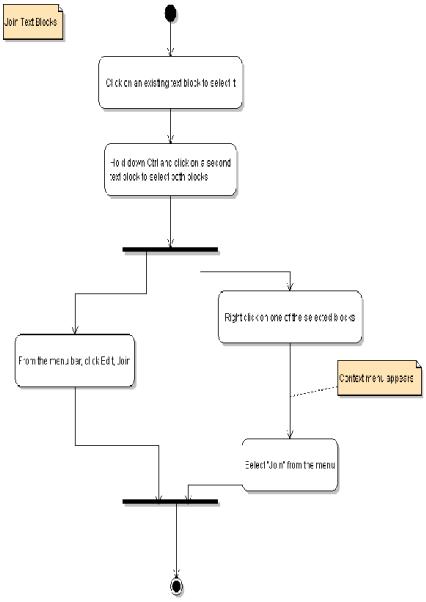
iteration_diagram



imsld_bestv1p03



master-ci



JoinTextBlocks-ActivityDiagram

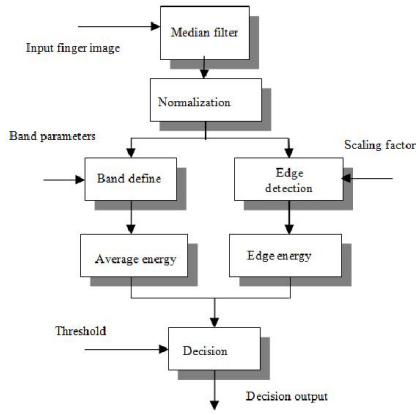
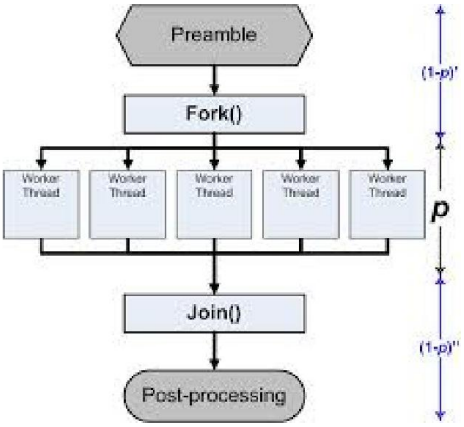
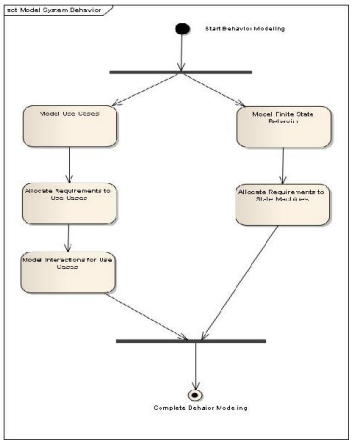


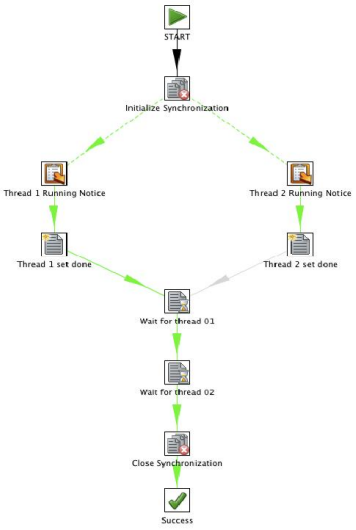
image28



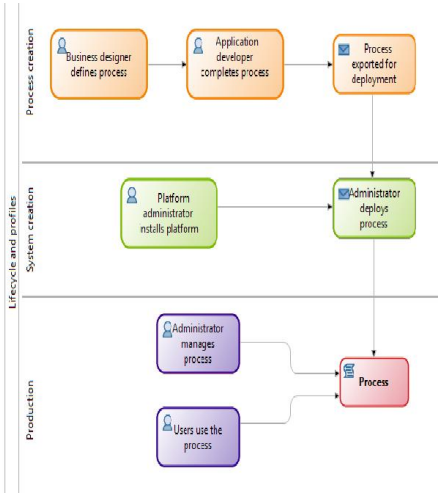
index



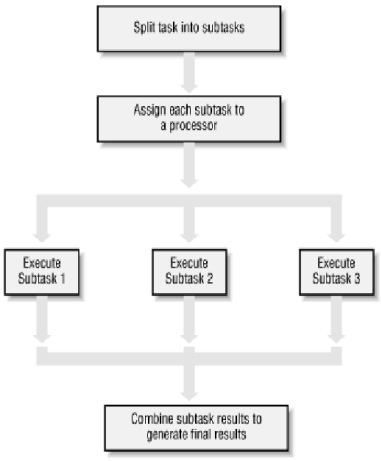
model-system-behavior



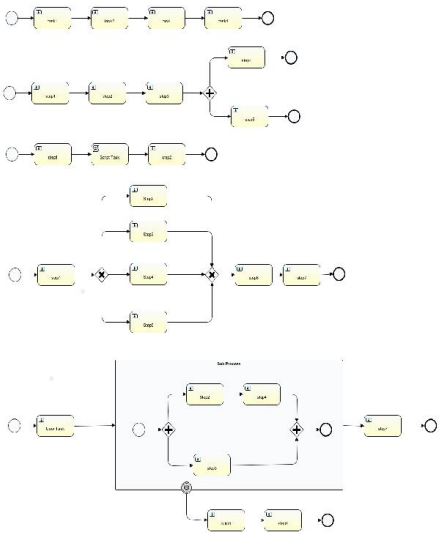
Screenshot



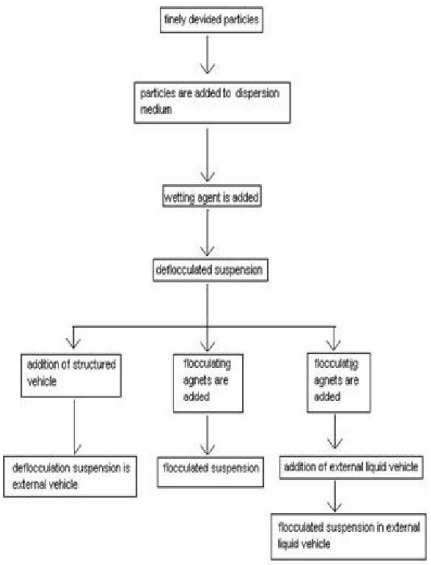
lifecycle_lanes



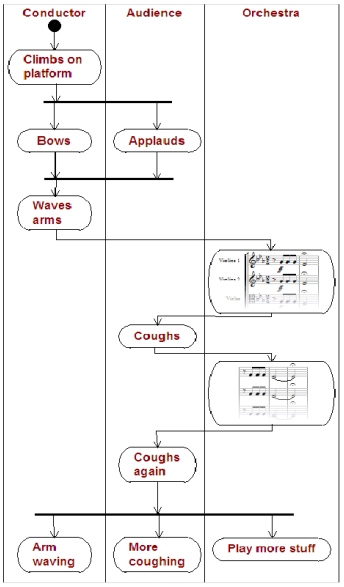
OPP.0108



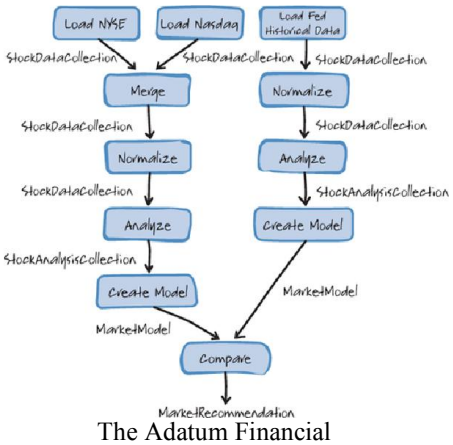
Processes



Pharmaceutical FLOW CHART FOR MANUFACTURING file19



uml_activity



The Adatum Financial Dashboard_1.jpg

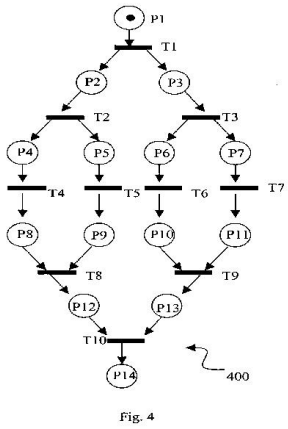
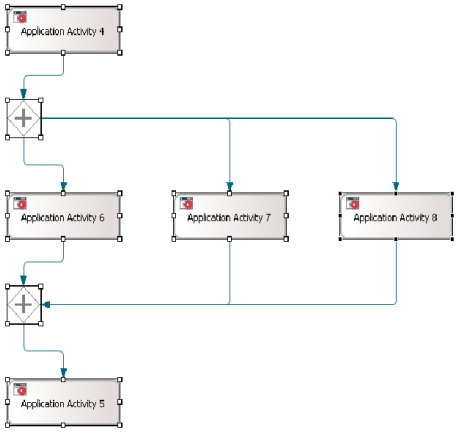
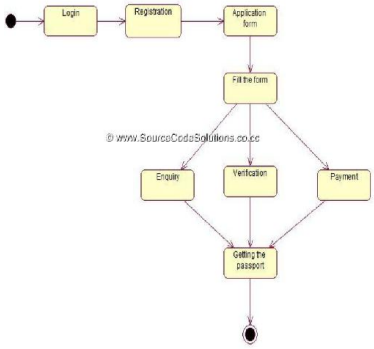


Fig. 4

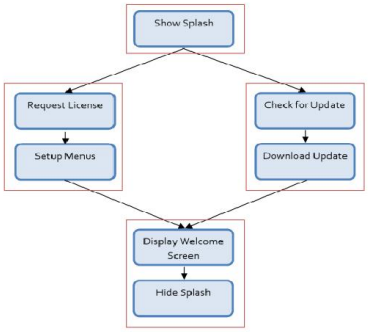
US20050234575A1-20051020-D00004



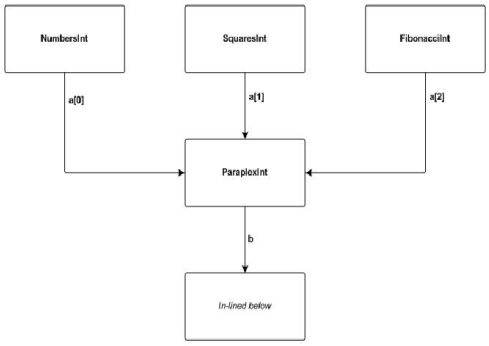
patterns-split-sychn-result



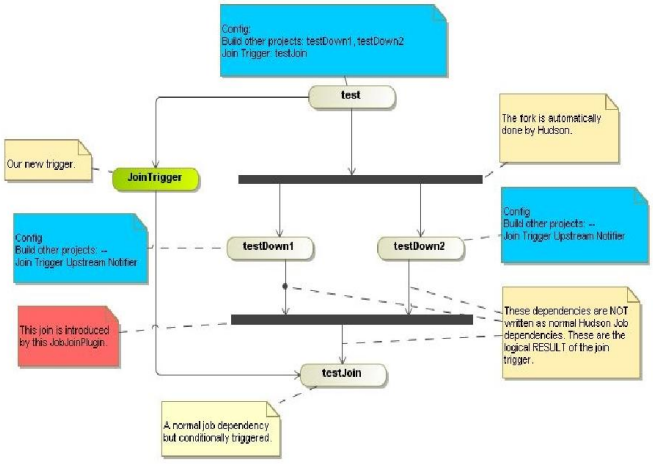
State Chart-Diagram-Register-for-Passport-Automation-System-...



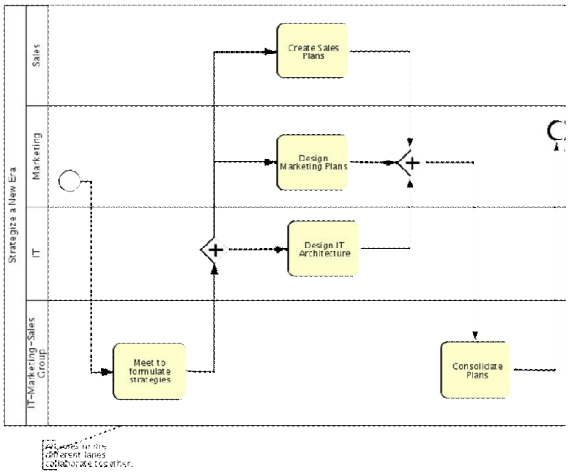
parallel_w_thumb



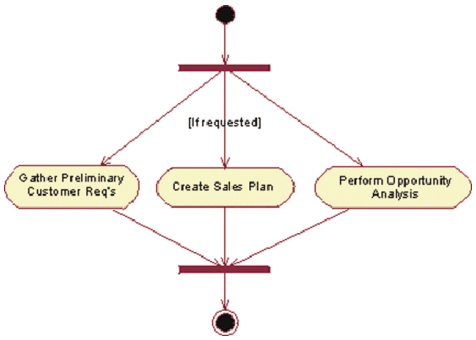
Parallel1



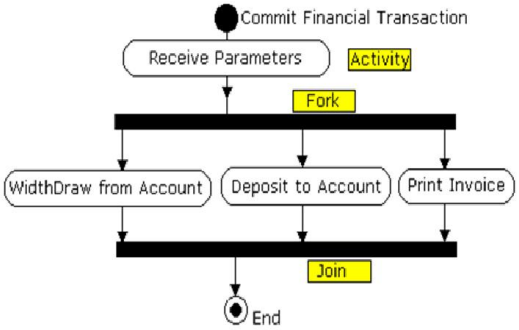
JobJoinPlugin-Config



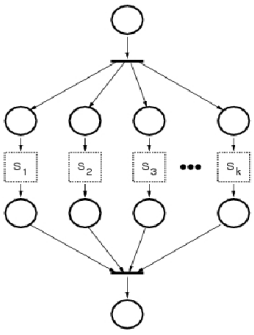
png



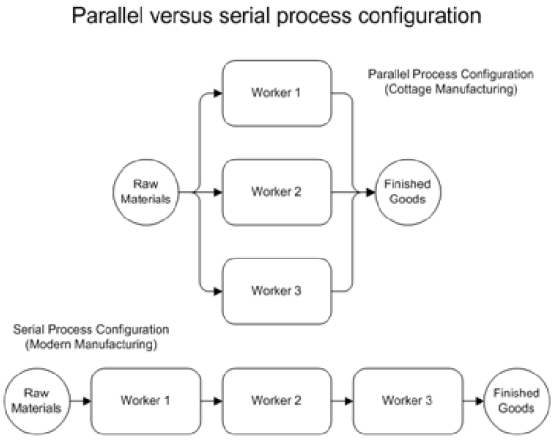
t_activityDiagrams_fig7



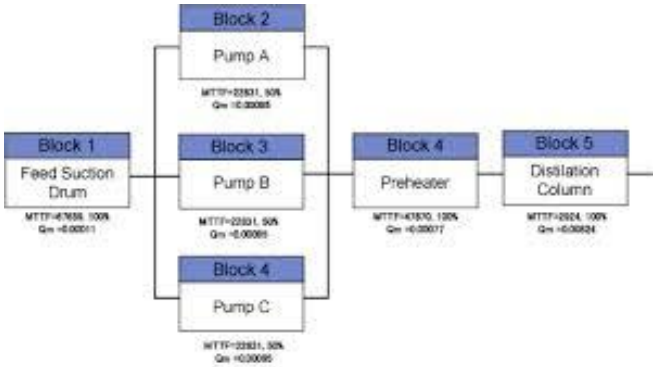
swp6_3c



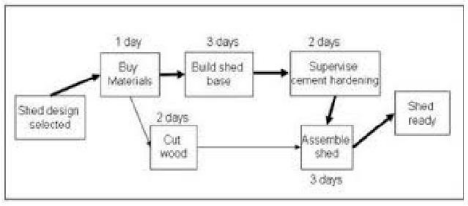
petri-img-4



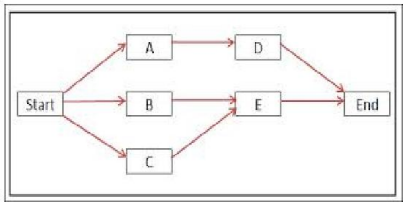
ParallelSerial



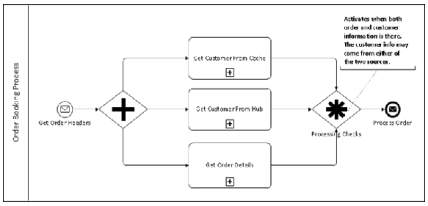
images8



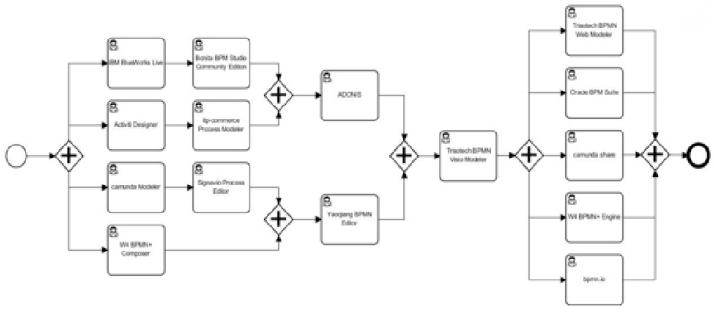
images9



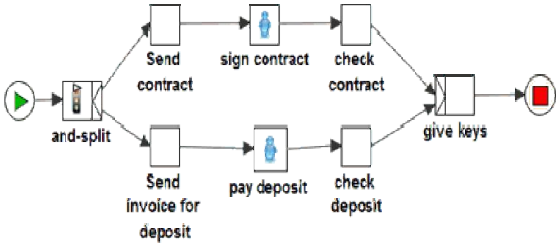
images4



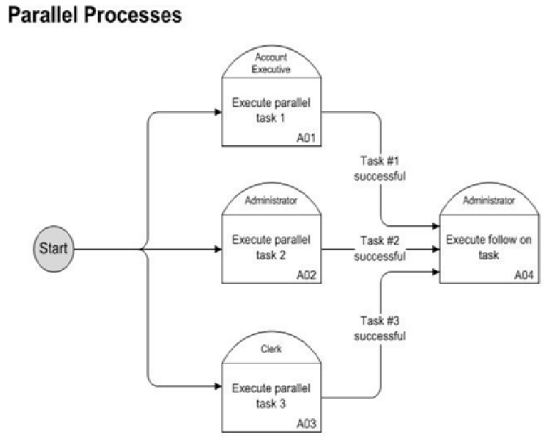
ordercomplexgateway_thumb



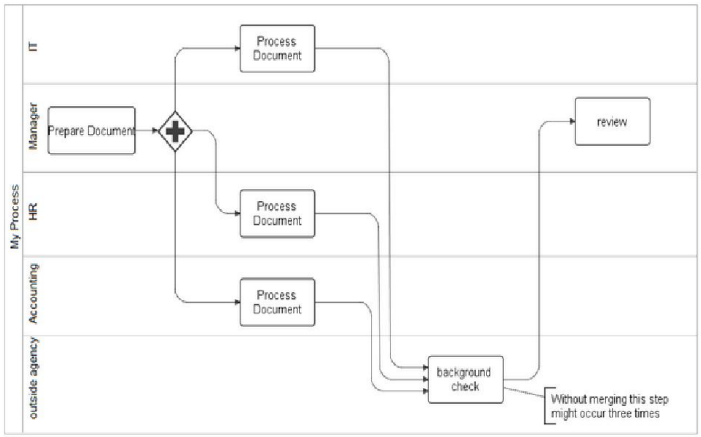
Making-BPMN_fig6



yawl-housing-agency3



Parallel-processes-more



parallel-nomerge

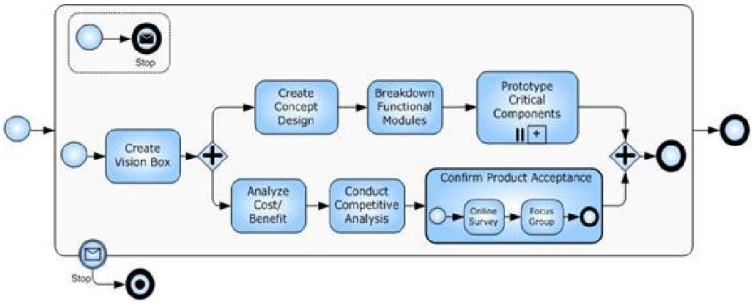
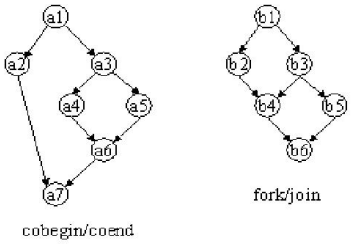
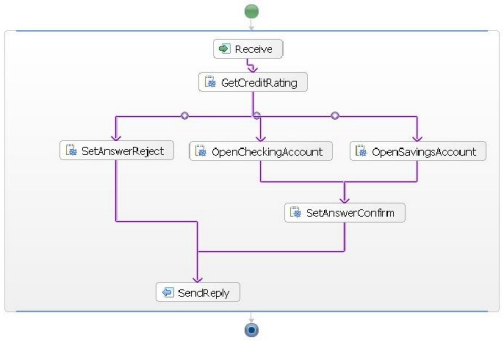


image015



co-versus-join



parallelp

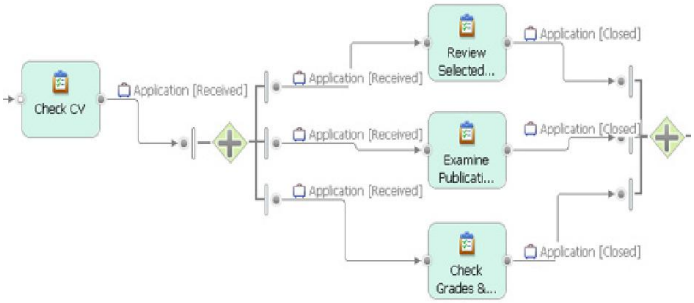
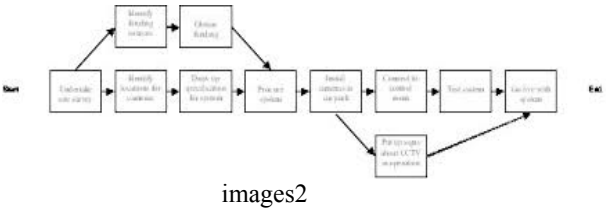


image025



images2

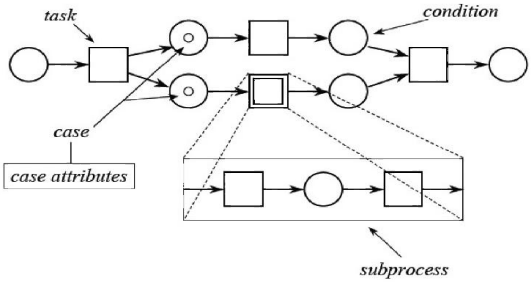


image17

Prilog E

Listing aplikacije grafičkog korisničkog interfejsa

```
var playedOrders = []
var activities = ['a', 'b', 'c', 'd', 'e', 'f'];
var startActivity = activities[0];
var endActivity = activities[activities.length - 1];
var previousModel, currentModel;
var currentOrder = [];
var Pair = function(x, y) {
  this.x = x;
  this.y = y;

  this.toString = function() {
    return "(" + this.x + ',' + this.y + ')';
  }
  this.reverse = function() {
    return new Pair(this.y, this.x);
  }
}

var PairList = function() {
  this.arr = []
  this.push = function(elem) {
    this.arr.push(elem);
  }
  this.last = function() {
    return this.arr[this.arr.length - 1];
  }
  this.lastIndex = function() {
    return this.arr.length;
  }
  this.concat = function(pairList) {
    var result = new PairList();
    result.arr = result.arr.concat(this.arr, pairList.arr);
    return result;
  }
  this.uniq = function() {
    var result = new PairList();
    for (var i = 0; i < this.arr.length; i++) {
      if (result.indexOf(this.arr[i]) == -1) {
        result.push(this.arr[i]);
      }
    }
    return result;
  }
  this.indexOf = function(pair) {
    for(var i = 0; i < this.arr.length; i++) {
      var currentPair = this.arr[i];
      if(currentPair.x == pair.x && currentPair.y == pair.y) {
        return i;
      }
    }
    return -1;
  }
```

```
  this.reversePairs = function() {
    var reversed = new PairList();
    for(var i = 0; i < this.arr.length; i++) {
      reversed.push(this.arr[i].reverse());
    }
    return reversed;
  }
  this.removeAll = function(list) {
    var result = new PairList();
    for(var i = 0; i < this.arr.length; i++) {
      var pair1 = this.arr[i];
      var found = false;
      for(var j = 0; j < list.arr.length; j++) {
        var pair2 = list.arr[j];
        if(pair1.x == pair2.x && pair1.y == pair2.y) {
          found = true;
          break;
        }
      }
      if(!found) {
        result.push(pair1);
      }
    }
    return result;
  }
  this.toString = function() {
    if(this.arr.length == 0) {
      return 'EMPTY'
    }
    var strPairs = [];
    for(var i = 0; i < this.arr.length; i++) {
      strPairs.push(this.arr[i].toString());
    }
    return strPairs.join(',');
  }
}

function gt() {
  var pairs = new PairList();
  for(var i = 0; i < playedOrders.length; i++) {
    var order = playedOrders[i];
    for(var j = 0; j < order.length - 1; j++) {
      pairs.push(new Pair(order[j], order[j+1]));
    }
  }

  return pairs;
}

function gtgt() {
  var pairs = new PairList();
  for(var i = 0; i < playedOrders.length; i++) {
    var order = playedOrders[i];
    for(var j = 0; j < order.length - 1; j++) {
      for(k = j + 2; k < order.length; k++) {
        pairs.push(new Pair(order[j], order[k]));
      }
    }
```

```
  }
  return pairs.removeAll(gt());
}

function lt() {
  return gt().reversePairs();
}

function ltlt() {
  return gtgt().reversePairs();
}

function ii() {
  var ii = new PairList();

  var candidates = gt().concat(gtgt()).uniq();
  for(var i = 0; i < candidates.arr.length - 1; i++) {
    for(var j = i + 1; j < candidates.arr.length; j++) {
      var pair1 = candidates.arr[i];
      var pair2 = candidates.arr[j];
      if(pair1.x == pair2.y && pair1.y == pair2.x) {
        ii.push(pair1);
        ii.push(pair2);
      }
    }
  }

  return ii;
}

function xxii() {
  var allPairs = new PairList();
  for(var i = 0; i < activities.length; i++) {
    for(var j = 0; j < activities.length; j++) {
      allPairs.push(new Pair(activities[i], activities[j]));
    }
  }
  return allPairs.removeAll(gt().concat(gtgt()).concat(lt()).concat(ltlt()));
}

function __D() {
  var gtList = gt();
  var checkList = lt().concat(ltlt()).uniq();
  result = gtList.removeAll(checkList).uniq();
  return result;
}

function noC__Activities() {
  var reverseDirectPairs = C__();
  var noC__s = activities.slice();
  for(var i = 0; i < reverseDirectPairs.arr.length; i++) {
    var pair = reverseDirectPairs.arr[i];
    var elementThatHasC__ = pair.x;
    var index = noC__s.indexOf(elementThatHasC__);
    if(index != -1) {
      noC__s.splice(index, 1);
    }
  }
```

```
  return noC__s;
}
function no__DActivities() {
  var directPairs = __D();
  var no__Ds = activities.slice();
  for(var i = 0; i < directPairs.arr.length; i++) {
    var pair = directPairs.arr[i];
    var elementThatHas__D = pair.x;
    var index = no__Ds.indexOf(elementThatHas__D);
    if(index != -1) {
      no__Ds.splice(index, 1);
    }
  }
  return no__Ds;
}

function additional__D() {
  var directPairs = __D();
  var indirectPairs = IID();
  var iiPairs = ii();
  var no__Ds = no__DActivities();
  var additionalPairs = new PairList();

  for(var i = 0; i < indirectPairs.arr.length; i++) {
    var pair = indirectPairs.arr[i];
    var a = pair.x;
    var c = pair.y;
    for(var j = 0; j < directPairs.arr.length; j++) {
      var directPair = directPairs.arr[j];
      if(directPair.y == c) {
        var b = directPair.x;
        // search for the a ||L b
        for(var k = 0; k < iiPairs.arr.length; k++) {
          var iiPair = iiPairs.arr[k];
          if(iiPair.y == b && iiPair.x == a) {
            if(no__Ds.indexOf(a) != -1) {
              additionalPairs.push(new Pair(a, c));
              break;
            }
          }
        }
      }
    }
  }
  return additionalPairs.uniq();
}
function allAdditional__D() {
  return additional__D().concat(additionalC__().reversePairs());
}

function additionalC__() {
  var directPairs = C__();
  var indirectPairs = CII();
  var iiPairs = ii();
  var noC__s = noC__Activities();
  var additionalPairs = new PairList();
  for(var i = 0; i < indirectPairs.arr.length; i++) {
    var pair = indirectPairs.arr[i];
```

```
    var c = pair.x;
    var a = pair.y;
    for(var j = 0; j < directPairs.arr.length; j++) {
      var directPair = directPairs.arr[j];
      if(directPair.y == a) {
        var b = directPair.x;
        // search for the c ||l b
        for(var k = 0; k < iiPairs.arr.length; k++) {
          var iiPair = iiPairs.arr[k];
          if(iiPair.x == c && iiPair.y == b) {
            if(noC__s.indexOf(c) != -1) {
              additionalPairs.push(new Pair(c, a));
              break;
            }
          }
        }
      }
    }
  }

  return additionalPairs.uniq();
}

function allAdditionalC__() {
  return additionalC__().concat(additional__D().reversePairs());
}

function IID() {
  var checkList = lt().concat(ltlt()).uniq();
  return gtgt().removeAll(checkList).uniq();
}

function C__() {
  return __D().reversePairs();
}

function CII() {
  return IID().reversePairs();
}

function createTable() {
  $('#table-placeholder').empty();
  var table = $('<table></table>').addClass('table');
  for(var i = 0; i < activities.length + 1; i++) {
    var row = $('<tr></tr>');
    if(i == 0) {
      row.append($('<td></td>'));
    } else {
      row.append($('<td>' + activities[i - 1] + '</td>').addClass('active'));
    }
    for(var j = 0; j < activities.length; j++) {
      if(i == 0) {
        row.append($('<td>' + activities[j] + '</td>').addClass('active'));
      } else {
        row.append($('<td>XX</td>'));
      }
    }
    table.append(row);
```

```
  }
  $('#table-placeholder').append(table);
}
function displayTable() {
  createTable();
  fillSymbol('&#x2192;', __D());
  fillSymbol('&#x21d2;', IID()); // =>
  fillSymbol('&#x2190;', C__()); // <-
  fillSymbol('&#x21d0;', CII());
  fillSymbol('II', ii());
  fillSymbol('#', xxii());
  fillSymbol('&#x2192;<i>i</i>', allAdditional__D());
  fillSymbol('&#x2190;<i>i</i>', allAdditionalC__());
  previousModel = currentModel;
  currentModel = __D().concat(allAdditional__D());
  // if model changed
  console.log("Current model: " + currentModel.toString());
  console.log("Previous model: " + previousModel.toString());

  var largerModel = currentModel.arr.length > previousModel.arr.length ? currentModel : previousModel;
  var smallerModel = currentModel.arr.length <= previousModel.arr.length ? currentModel : previousModel;

  if(largerModel.removeAll(smallerModel).arr.length > 0) {
    // mark order
    $('.order:last').addClass("green");
  }

  // check if playedOrders contains lastOrder
  var currentOrder = playedOrders[playedOrders.length - 1];
  for(var i = 0; i < playedOrders.length - 1; i++) {
    var oldOrder = playedOrders[i];
    if(oldOrder.join(",") == currentOrder.join(",")) {
      $('.order:last').addClass("red");
    }
  }
}
function fillSymbol(sym, list) {
  for(var i = 0; i < list.arr.length; i++) {
    var pair = list.arr[i];
    var row = activities.indexOf(pair.x) + 1;
    var col = activities.indexOf(pair.y) + 1;
    $('.table tr:eq(' + row + ') td:eq(' + col + ')').html(sym);
  }
}

function displayGraphModel() {
   var width = 1000,
    height = 500;

  $('#graph-placeholder').empty();
  var nodes = {};
  for(var i = 0; i < activities.length; i++) {
    var activity = activities[i];
    if(i == 0) {
      nodes[activity] = { name: activity, fixed: true, x: 20, y: (height / 2) };
    } else if(i == activities.length - 1) {
      nodes[activity] = { name: activity, fixed: true, x: (width - 20), y: (height / 2) };
    } else {
```

```
    nodes[activity] = { name: activity };
  }
}
var links = []

var pairs = __D().concat(allAdditional__D());
for(var i = 0; i < pairs.arr.length; i++) {
  var pair = pairs.arr[i];
  links.push({ source: nodes[pair.x], target: nodes[pair.y], value: 0.5 });
}

var force = d3.layout.force()
.nodes(d3.values(nodes))
.links(links)
.size([width, height])
.linkDistance(70)
.charge(-400)
.on("tick", tick)
.start();

var drag = force.drag()
  .on("dragstart", dragstart);

function dragstart(d) {
  d3.select(this).classed("fixed", d.fixed = true);
}
var container = $('#graph-placeholder')[0];
var svg = d3.select(container).append("svg")
.attr("width", width)
.attr("height", height);

// build the arrow.
svg.append("svg:defs").selectAll("marker")
.data(["end"])      // Different link/path types can be defined here
.enter().append("svg:marker")    // This section adds in the arrows
.attr("id", String)
.attr("viewBox", "0 -15 20 20")
.attr("refX", 25)
.attr("refY", -1.5)
.attr("markerWidth", 10)
.attr("markerHeight", 10)
.attr("orient", "auto")
.append("svg:path")
.attr("d", "M0,-5L10,0L0,5");

// add the links and the arrows
var path = svg.append("svg:g").selectAll("path")
.data(force.links())
.enter().append("svg:path")
//    .attr("class", function(d) { return "link " + d.type; })
.attr("class", "link")
.attr("marker-end", "url(#end)");

// define the nodes
var node = svg.selectAll(".node")
.data(force.nodes())
.enter().append("g")
.attr("class", "node")
```

```
  .call(force.drag);

  // add the nodes
  node.append("circle")
  .attr("r", 14);

  // add the text
  node.append("text")
  .attr("x", -2)
  .attr("dy", ".5em")
  .text(function(d) { return d.name; });

  // add the curvy lines
  function tick() {
    path.attr("d", function(d) {
      var dx = d.target.x - d.source.x,
      dy = d.target.y - d.source.y,
      dr = Math.sqrt(dx * dx + dy * dy);
      return "M" +
        d.source.x + "," +
        d.source.y + "A" +
        dr + "," + dr + " 0 0,1 " +
        d.target.x + "," +
        d.target.y;
    });

    node.attr("transform", function(d) {
      return "translate(" + d.x + "," + d.y + ")";
    });
  }
}

// shuffle array
function shuffle(o){
  for(var j, x, i = o.length; i; j = Math.floor(Math.random() * i), x = o[--i], o[i] = o[j], o[j] = x);
  return o;
};

// start and end activities are always the same
function shuffleActivities() {
  var newArray = []
  newArray.push(activities[0]);
  console.log(shuffle(activities.slice(1, activities.length - 1)));
  newArray = newArray.concat(shuffle(activities.slice(1, activities.length - 1)));
  newArray.push(activities[activities.length - 1]);

  return newArray;
}

// start and end activities are not reversed
function reverseActivities() {
  var newArray = [];
  newArray.push(activities[0]);
  var lastPlayed = playedOrders[playedOrders.length - 1];
  for(var j = lastPlayed.length - 2; j > 0; j--) {
    newArray.push(lastPlayed[j]);
  }
  newArray.push(lastPlayed[lastPlayed.length - 1]);
```

```
  return newArray;
}
function displayRelations() {
  var $relations = $('#relations .bg-success').empty();
  $relations.append('<strong> Relations: </strong><br/>');
  $relations.append('>: ' + gt().toString() + '<br/>');
  $relations.append('>>: ' + gtgt().toString() + '<br/>');
  $relations.append('<: ' + lt().toString() + '<br/>');
  $relations.append('<<: ' + ltlt().toString() + '<br/>');
  $relations.append('||: ' + ii().toString() + '<br/>');
  $relations.append('&#x2192;:' + __D().toString() + '<br/>');
  $relations.append('&#x21d2;:' + IID().toString() + '<br/>'); // =>
  $relations.append('&#x2190;:'  + C__().toString() + '<br/>'); // <-
  $relations.append('&#x21d0;: ' + CII().toString() + '<br/>');
  $relations.append('#: ' + xxii().toString() + '<br/>');
  $relations.append('INFERRED &#x2192;<i>i</i>: ' + allAdditional__D().toString() + '<br/>');
  $relations.append('INFERRED &#x2190;<i>i</i>: ' + allAdditionalC__().toString() + '<br/>');
  $relations.append('NO_DIRECT &#x2192;:' + no__DActivities() + '<br/>');
  $relations.append('NO_DIRECT &#x2190;:' + noC__Activities() + '<br/>');
  $relations.show();
}

function displayNextStep() {
  $('#next-step').hide();
  var shuffledActivities = reverseActivities();
  currentOrder = [];
  currentOrder.push(startActivity);
  $('#current-order').html(currentOrder.join(', '));
  for(var i = 1; i < shuffledActivities.length - 1; i++) {
    var btnAction = $('<button type="button" class="btn btn-default btn-lg">' + shuffledActivities[i] + '</button>');
    btnAction.click(function() {
      currentOrder.push($(this).html());
      $('#current-order').html(currentOrder.join(', '));
      // remove previous undo button
      $('.undo.hide').remove();
      $(this).removeClass('btn').addClass('undo hide');
      if($('.btn-group .btn').length == 0) {
        currentOrder.push(endActivity);
        $('#current-order').html(currentOrder.join(', '));
        playedOrders.push(currentOrder);
        $('#current-order').before('<span class="order">' + currentOrder.join(', ') + '</span><br/>');
        $('#current-order').empty();
        displayRelations();
        displayTable();
        displayGraphModel();
        $('#next-step').show();
        $('.undo.hide').remove();
        $('#undo').hide();
      } else {
        $('#undo').show();
      }
    });
    $(".btn-group").append(btnAction);
  }
}

function undoLastAction() {
  $('.undo.hide').removeClass('undo hide').addClass('btn');
```

```
  currentOrder.pop();
  $('#current-order').html(currentOrder.join(', '));
  $('#undo').hide();
}

function startWithExample(){
  console.log('app started');
  previousModel = new PairList();
  currentModel = new PairList();
  currentOrder.push(startActivity);
  $('#current-order').html(currentOrder.join(', '));
  for(var i = 1; i < activities.length - 1; i++) {
    var btnAction = $('<button type="button" class="btn btn-default btn-lg">' + activities[i] + '</button>');
    btnAction.click(function() {
      currentOrder.push($(this).html());
      $('#current-order').html(currentOrder.join(', '));
      // remove previous undo button
      $('.undo.hide').remove();
      $(this).removeClass('btn').addClass('undo hide');
      if($('.btn-group .btn').length == 0) {
        currentOrder.push(endActivity);
        $('#current-order').html(currentOrder.join(', '));
        playedOrders.push(currentOrder);
        $('#current-order').before('<span class="order">' + currentOrder.join(', ') + '</span><br/>');
        $('#current-order').empty();
        displayRelations();
        displayTable();
        displayGraphModel();
        $('#next-step').show();
        $('.undo.hide').remove();
        $('#undo').hide();
      } else {
        $('#undo').show();
      }
    });
    $(".btn-group").append(btnAction);
  }

  $('#next-step').click(displayNextStep);
  $('#undo').click(undoLastAction);
}

$(document).ready(function() {
  $('#activity-entry-done').click(function() {
    if($('#activity-input').val() == '') {
      alert("Unos prazan, bice iskorišćene aktivnosti: a, b, c, d, e, f");
    } else {
      activities = $('#activity-input').val().split(',');
      startActivity = activities[0];
      endActivity = activities[activities.length - 1];
    }
    $('#activity-entry').fadeOut();
    $('#example').fadeIn();
    startWithExample();
  });
});
```

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку да у Дигитални репозиторијум Универзитета у Приштини, са привременим седиштем у Косовској Митровици унесе моју докторску дисертацију под насловом:

Развој информационих система за управљање пословним процесима демонстрацијом

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Приштини са привременим седиштем у Косовској Митровици могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

(5) Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Косовској Митровици, 29.11.2023.