VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Dokumentų valdymo sistemų darbo sekų transformavimas iš BPMN į WF modelį

Transformation from BPMN to WF model of document management system workflows

Magistro baigiamasis darbas

Atliko:	Miroslav Kisly	(parašas)
Darbo vadovas:	Dr. Andrius Kurtinaitis	(parašas)
Darbo recenzentas:	Prof. Romas Baronas	(parašas)

Vilnius – 2008

Turinys

Anotacija

Darbe yra nagrinėjami du paplitę darbo sekų modeliai – BPMN ir Windows Workflow Foundation (WF), bei tiriama, kaip BPMN darbo sekas transformuoti į WF vykdymo modelį. BPMN ir WF yra iš esmės skirtingos kalbos ir pagrindinė transformavimo problema yra susijusi su nestruktūrizuotų ciklų eliminavimu, t.y. jų konvertavimu į proceso atžvilgiu ekvivalenčias struktūrizuotas konstrukcijas. Darbe pasirinktas žinomas algoritmas, skirtas panašioms ciklinėms konstrukcijoms transformuoti naudojant tęstinumo semantiką. Jis buvo adaptuotas BPMN modeliui. Algoritme naudojama tęstinumo semantika buvo praplėsta atsižvelgiant į BPMN ir WF modeliuose esančias konstrukcijas bei patobulinto algoritmo ypatumus. Buvo sukurti nauji algoritmai, skirti nagrinėjamai transformacijai atlikti. Transformavimo algoritmui patikrinti buvo sukurta programa, leidžianti tęstinumo semantika užrašytą BPMN darbo seką transformuoti į pavidalą, nesunkiai konvertuojamą į WF. Atliktos eksperimentinės darbo sekų transformacijos parodė, kad sukurtas algoritmas yra veiksmingas.

BPMN, WF, Windows Workflow Foundation, tęstinumo semantika, transformavimas, darbo seka, algoritmas.

Abstract

This paper analyzes the two most widely used workflow models – BPMN and Windows Workflow Foundation (WF). It also analyzes how a BPMN workflow can be transformed into a WF execution model. BPMN and WF represent two fundamentally different classes of languages and the basic problem is related to convertion of unstructured cycles to their structured equivalents. A known algorithm used for similar cyclic constructions transformations using continuation semantics was chosen. This algorithm was adopted for the BPMN model. The continuation semantics were extended in regards to BPMN and WF model constructions and the specifics of the improved algorithm. Finally, in order to solve the transformation, three new algorithms were created: algorithm for BPMN workflow specification with extended continuation semantics; algorithm for transforming the specification to structured model; algorithm for converting structured model specification to WF model. A specialized program, which transforms extended continuation semantics to a structured model, was written to test the transformation algorithm. Testing this program with exemplary transformations proved that the transformation algorithm was working correctly.

BPMN, WF, Windows Workflow Foundation, continuation semantics, transformation, workflow, algorithm.

Įvadas

Kiekvieną dieną susiduriame su milžiniškais informacijos srautais, supančiais mūsų rutininį gyvenimą. Informacijos kiekis vis didėja, o susigaudyti tokiuose srautuose tampa vis sunkiau. Iškilus tokiai problemai, rinkoje atsiranda vis daugiau dokumentų valdymo sistemų gundančių savo patogumu ir funkcionalumu, skirtu elektroninio formato dokumentų pildymui, įvairaus pobūdžio informacijai saugoti, redagavimui bei paieškai palengvinti. Tačiau besikuriančios informacinės visuomenės poreikiai auga labai sparčiai ir informacijos valdymo problema darosi vis aktualesnė. Besiplečiant dokumentų valdymo sistemoms, informacinė visuomenė pageidauja vis tobulesnių sistemų, kurios sugebėtų ne tik tai saugoti ir apdoroti informaciją, bet ir atitinkamai ją valdyti, laikantis iš anksto aprašytų arba laikui bėgant besikeičiančių įmonės verslo procesų, susietų su saugoma informacija. Tokie procesai, gyvuojantys dokumentų valdymo sistemose, vadinami darbo sekomis.

Per pastaruosius kelis metus darbo sekos užtikrintai įsitvirtino beveik visame IT pasaulyje. Tai tapo praktiškai tam tikros rūšies mada. Laikui bėgant pasaulinėje rinkoje atsirado nemažai produktų užsiimančių verslo procesų modeliavimu, valdymu, pavaizdavimu bei vykdymu. Tokiomis aplinkybėmis susiformavo naujos problemos, susijusios su pastarųjų produktų ir įvairių darbo sekų modelių suderinamumu. Kiekviena kompanija siekdama įsitvirtini IT rinkoje su savo unikalia programų sistema, stengiasi plačiai populiarinti savo sukurtą darbo sekų pavaizdavimo modelį, darbo sekų apibrėžimo formatą ir t.t. Todėl kuriant naujas programų sistemas tenka iš anksto prisirišti prie tam tikros sistemos, o galimi pakeitimai reikalaus labai daug papildomų investicijų. Siekiant išspręsti pastarąsias problemas atsirado įvairios organizacijos, kurios padėjo pirmuosius žingsnius link verslo procesų bei darbo sekų standartizavimo. Šios organizacijos bendradarbiauja su daugybe kitų organizacijų, mokslo institucijų vienaip ar kitaip suinteresuotų verslo procesų bei darbo sekų raida. To pasekoje yra išleidžiami atitinkami standartai, padedantys įvesti tam tikrą tvarką tarp produktų bei kitų standartų gausos. Šio darbo kontekste verta paminėti vieną įtakingiausių standartizavimo organizacijų BPMI – verslo procesų valdymo organizacija (angl. Business Process Management Initiative). Šios organizacijos dėka 2004 metais po daugiau negu dviejų metų intensyvių tyrimų bei kruopštaus darbo buvo išleistas BPMN 1.0 standartas (angl. Business Process Modeling Notation) skirtas verslo procesų modelių pavaizdavimo standartizavimui. Šis standartas tapo faktiškai *de-facto* standartu verslo procesų modeliavime. Šiuo metu yra sukurta apie 40 stambių produktų palaikančių šį standartą (2006 m. lapkričio 28 d. duomenimis, http://www.bpmn.org).

Standartizavimo organizacijos yra toli gražu ne vienintelės institucijos įtakojančios verslo procesų vystymą. Yra ir daugiau didelių kompanijų vienaip ar kitaip įtakojančių IT rinką bei joje

vyraujančias madas susijusias su verslo procesais. Vienas iš tokių pavyzdžių yra visiem žinoma Microsoft korporacija, pirmaujanti savo IT produktų mastais bei populiarumu visame pasaulyje. Nors Microsoft tiesiogiai neužsiiminėja tarptautinių standartų išleidimu, ji investuoja milžiniškus pinigus į naujų produktų bei technologijų kūrimą. Rezultate sukurtos naujovės tampa tam tikrais standartais naudojamais visame pasaulyje. Atsižvelgusi į verslo procesų valdymo poreikį ir aktualumą, 2006 metų pabaigoje Microsoft išleido .NET 3.0 platformą, kurios viena iš sudedamųjų dalių buvo WF (angl. Windows Workflow Foundation). WF yra platforma skirta verslo procesų valdymui. Į jos sudėtį įeina: darbo sekų vykdymo mašina, kurios architektūra leidžia lanksčiai priderinti ją prie kitos sistemos funkcionalumo; darbo sekų apibrėžimo bei stebėjimo įrankiai. Pasinaudojant WF platforma Microsoft žada padaryti tam tikrą perversmą taikomųjų programų kūrimo srityje teigdama, kad tai yra daug žadanti ateities technologija, padėsianti išspręsti daugybę dabartinių problemų susijusių su verslo procesų valdymu. Todėl yra labai naudinga ištirti pateiktą technologiją, įvertinti jos galimybės bei suderinamumą su jau naudojamais standartais [MSD07a, MSD07b, MSD07c].

Naudojant BPMN ir WF modelius pagrindinė problema yra ta, kad transformuoti BPMN sumodeliuotą procesą į WF modelį nėra paprasta, kadangi tai yra iš esmės skirtingų klasių modeliai. Su panašia problema buvo jau susidurta ir anksčiau sprendžiant BPMN į BPEL transformavimo problemas, susijusias su nestruktūrizuotų ciklų buvimu.

Pagrindinis šio darbo tikslas – sukurti dokumentų valdymo sistemų (toliau DVS) darbo sekų transformaciją, aprašančią, kaip transformuoti BPMN į WF sekų modelį. Tam tikslui pasiekti numatoma:

- išrinkti ir formalizuoti BPMN standarto žymėjimų poaibį skirtą DVS darbo sekoms aprašyti;
- pagal formalizuotus modelių aprašus sukurti transformaciją iš BPMN į WF;
- sukurti programą sugebančią pagal pateiktą algoritmą transformuoti BPMN darbo seką, aprašytą formaliąja kalba, į WF pavidalą.

Kuriant transformaciją iš BPMN į WF buvo remiamasi keliais straipsniais, kuriuose yra nagrinėjamos ir sprendžiamos problemos susijusios su nestruktūrizuotų ciklų prastinimu. Pirmame šio darbo skyriuje yra išanalizuotos dokumentų valdymo sistemos ir darbo sekos, BPMN ir WF modeliai, darbo sekų modelių transformavimo problemos bei jų sprendimai aprašyti įvairiuose straipsniuose, išrinktos BMPN struktūros pritaikomos DVS sistemose. Antrame skyriuje yra aprašytas sukurtas transformavimo algoritmas, skirtas darbo sekų transformavimui iš BPMN į WF.

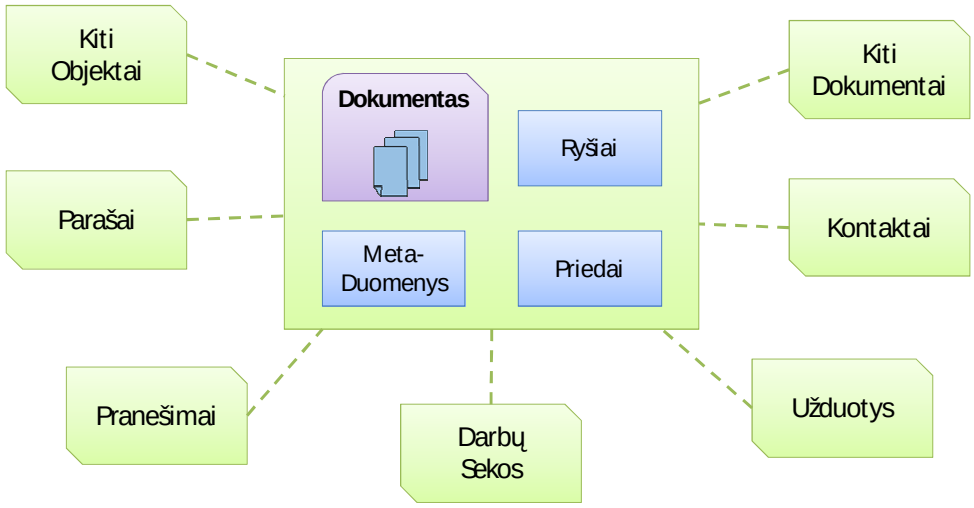
1. Problemos analizė

1.1. Dokumentų valdymo sistemos ir darbo sekos

1.1.1. Dokumentų valdymo sistemos sąvoka ir jos funkcijos

Dokumentų valdymo sistema vadinama automatizuota elektroninių dokumentų kontroliavimo sistema, apimanti visą dokumentų gyvavimo ciklą [Hol06]. Dokumentų valdymo sistema leidžia užtikrinti informacijos saugumą, kuris yra suprantamas kaip sistemos sugebėjimas saugoti sukauptą informaciją nuo galimo dingimo bei neteisėto priėjimo.

Dokumentas yra neatsiejama dokumentų valdymo sistemų sąvoka, t.y. saugomas informacijos vienetas, su kuriuo sistema turėtų leisti naudotojams atlikti įvairias operacijas. Pagal apibrėžimą, dokumentas yra įrašyta informacija, kuri gali būti traktuojama kaip vienetas [Hol95].



1 pav. Dokumento sandara ir jo ryšiai

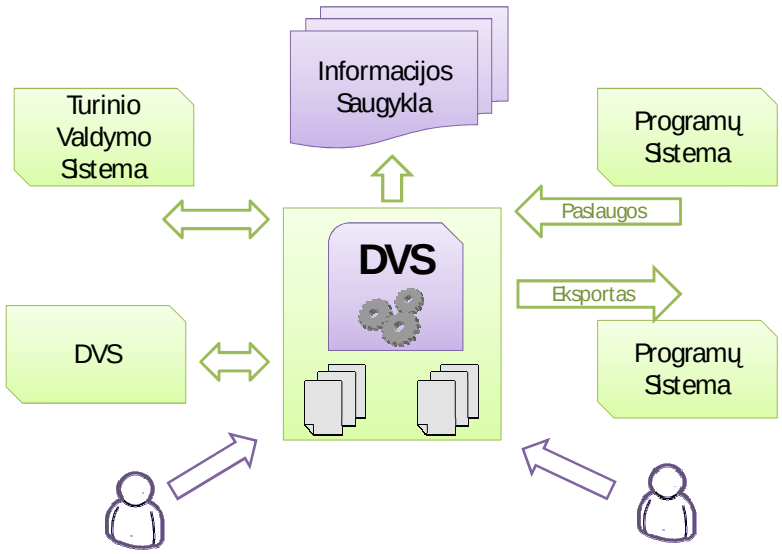
Dokumentų valdymo sistemoje dokumentas yra objektas susidedantis iš tam tikrų metaduomenų ir priedų. Dokumento priedai - tai bylos failinės sistemos požiūriu, o metaduomenys – dokumento formos atributai, kuriuose yra saugoma tipizuota informacija, pavyzdžiui, apibrėžto tipo paveiksliukas, data, laikas, tekstas, skaičius, valiuta, nuoroda į kitą sistemos objektą ir t.t. (1 pav.). Ypatingos rūšies atributai yra nuorodos. Nuorodomis yra sudaromi ryšiai tarp įvairių sistemos objektų.

Jeigu į dokumentų valdymo sistemą pažiūrėtume iš funkcinės pusės, tai galima būtų išgryninti tam tikras sekančias pagrindines funkcijas:

1. Dokumentų kūrimas – dokumentai gali būti kuriami įvairias būdais: rankiniu būdu, skanuojant dokumentą, gaunant informaciją per kitus išorinius interfeisus (faksas, el.

paštas). Kuriant dokumentus ne rankiniu būdu, svarbią rolę groja informacijos įkėlimo automatizavimas, pvz. sąskaitų faktūrų skanavimas, OCR atpažinimas, pramoninis dokumentų skanavimas, gaunamos korespondencijos fiksavimas.

2. Dokumentų klasifikavimas – galimybė skirstyti dokumentus į profilius, klasifikuojant juos į atitinkamus katalogus (klases) bei segtuvus – hierarchinė struktūra.
3. Dokumentų registravimo valdymas – galimybė „užšaldyti“ tam tikrą dokumento informacijos būseną – dokumento kopijavimas (su visomis versijomis) ir padarytos kopijos saugojimas nuo pakeitimų analogija.
4. Dokumentų versijų valdymas – galimybė kurti dokumentų versijas, užtikrinti dokumento vientisumą jo redagavimo metu (angl. Check-in / Check-out).
5. Duomenų paieška ir ataskaitų generavimas.
6. Elektroninis parašas – galimybė pasirašyti dokumentą pasinaudojant elektroninio parašo sertifikatais ir sertifikavimo centrais suteikiant dokumentui juridinę galią.
7. Saugos mechanizmai – galimybė saugoti dokumentus ir kitus sistemos objektus nuo nesankcionuoto priėjimo pasinaudojant tam tikrais saugos mechanizmais, pavyzdžiui, vienam prisijungusiam prie sistemos naudotojui suteikti galimybę tik matyti tam tikrą dokumentą, o kitam - registruoti ir tvirtinti.
8. Darbo sekų valdymas.



2 pav. Dokumentų valdymo sistema

Nepaisant daugybės dokumento valdymo sistemos funkcijų ji gali būti suprantama kaip tam tikra abstrakcija – informacijos saugykla, prie kurios galima prieiti turint atitinkamą saugos priėjimą (2 pav.). Tokiu būdu, prie saugomos informacijos gali prieiti ne tik registruoti sistemos naudotojai, bet ir kitos sistemos, kurioms saugoma informacija gali būti naudinga. Dokumentų valdymo sistemos dažnai turi tam tikrą priėjimo sąsają (angl. API) arba pateikia WEB paslaugas,

kuriomis pasinaudojant galima pasiekti saugomą informaciją bei atlikti su ja apibrėžtus veiksmus.

1.1.2. Darbo sekos

Įmonės diegia dokumentų valdymo sistemas dažnai ne tik tam, kad jos saugotų informaciją, bet ir ją atitinkamai valdytų. Kiekviena įmonė turi savo vidinius nusistovėjusius procesus. Tokie apibrėžti procesai turi savo dalyvius bei susijusią informaciją. Siekiant tobulinti tokių procesų vykdymą reikia juos optimizuoti. Kaip tai padaryti? Proceso optimizavimo metu proceso valdymas perduodamas sistemai. Tokiu atveju proceso dalyviams (žmonėms) lieka mažiau mechaninio darbo, nes dalį darbo atlieka sistema, o tam tikros proceso veiklos gali būti pilnai automatizuotos ir atliekamos sistemos. Be to, sistema gali kontroliuoti proceso vykdymą. Tai reiškia procesas negalės nukrypti nuo apibrėžtų taisyklių, o tam tikras proceso veiklas galės atlikti tik atitinkamas teises turintys žmonės. Taigi sėkmingai įdiegus procesą į sistemą sutaupomi įmonės pinigai. Vadinasi dokumentų valdymo sistema gali tapti ne tik informacijos saugykla, bet ir procesų valdymo sistema.

Kaip pavyzdį paimkime gamybos įmonės darbuotojų išleidimo į atostogas procesą, sudarytą iš tokių žingsnių:

1. Darbuotojas suranda prašymo pavyzdį. Pavyzdžio ieškojimas gali užtrukti.
2. Prieš rašydamas prašymą, darbuotojas turėtų keiptis pas administratorę, kad sužinotų, kiek dienų jis gali pasiimti nemokamų atostogų.
3. Parašytas prašymas nukeliauja pas tiesioginį vadovą. Vadovas jį patvirtina ar nepatvirtina. Nepatvirtinimo atveju darbuotojas turės derinti atostogas su vadovu.
4. Vadovo patvirtintas prašymas keliauja pas gamybos vadovą.
5. Vadovų patvirtintą prašymą administratorė nuneša pasirašyt direktoriui.
6. Prašymo kopija keliauja į finansų skyrių ir atsiduria archyvo segtuve.

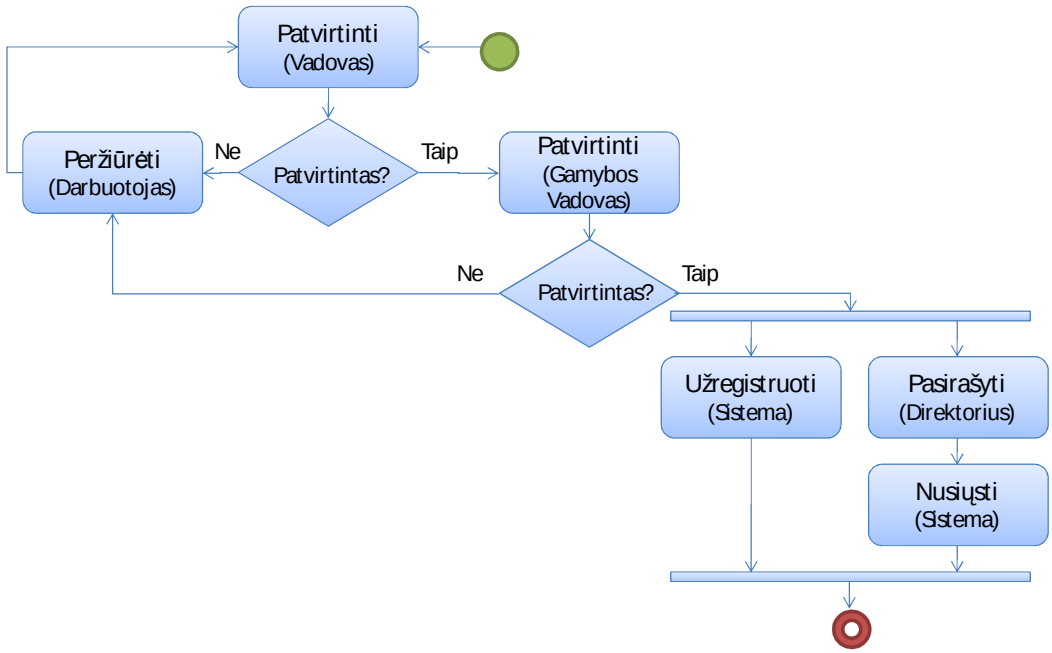
Nesunku pastebėti, kad toks paprastas procesas kaip darbuotojų išleidimas į atostogas nėra optimalus, o tai reiškia jį galima optimizuoti. 3-me paveikslėlyje yra pateikta UML veiklos diagrama vaizduojanti šio proceso variantą adaptuotą dokumentų valdymo sistemai.

Adaptuotas procesas susideda iš šešių veiklų ir dviejų sąlygų. Proceso valdymą atlieka sistema, darbuotojas turi tik inicijuoti procesą arba sukurti dokumentą kuris inicijuos procesą:

1. Darbuotojas prisijungia prie sistemos, sukuria atostogų prašymo dokumentą. Darbuotojas užpildo tik pateiktos formos laukus. Įvedamų laukų kiekis yra minimalus: atostogų forma (apmokamos/neapmokamos, eilės tvarka/skubos tvarka), atostogų data, komentaras (jei atostogos turi būti suteiktos skubos tvarka). Išsaugojus dokumentą sistema: automatiškai patikrina ar laukai yra korektiškai užpildyti, tikrina

ar norimas atostogų dienų skaičius neviršija leistiną, inicijuojamas išleidimo į atostogas procesas (3 pav.).

2. Darbuotojo vadovas gauna užduotį patvirtinti atostogų dokumentą. Vadovas patvirtina arba nepatvirtina dokumentą. Nepatvirtinimo atveju jis parašo, kodėl nepatvirtino.
3. Nepatvirtinimo atveju darbuotojas atsižvelgdamas į vadovo komentarą turi pakoreguoti atostogų dokumentą (gauna užduotį). Patvirtinimo atveju gamybos vadovas gauna dokumento patvirtinimo užduotį.
4. Nepatvirtinimo atveju situacija analogiška aukščiau išvardintajai.
5. Patvirtinimo atveju sistema automatiškai užregistruoja dokumentą. Direktorius gautą dokumentą pasirašo jį skaitmeniniu parašu.
6. Procesas užbaigiamas.



3 pav. Darbuotojų išleidimo į atostogas procesas

Adaptuotas procesas yra žymiai efektyvesnis už pirmąjį ir sutaupo daug žmogiškumų resursų bei įneša kontrolės žingsnių. Toks sistemoje atliekamas įmonės procesas ir yra vadinamas darbo seka. Apibendrinus galima išvardinti sekančius tokio darbo sekos privalumus:

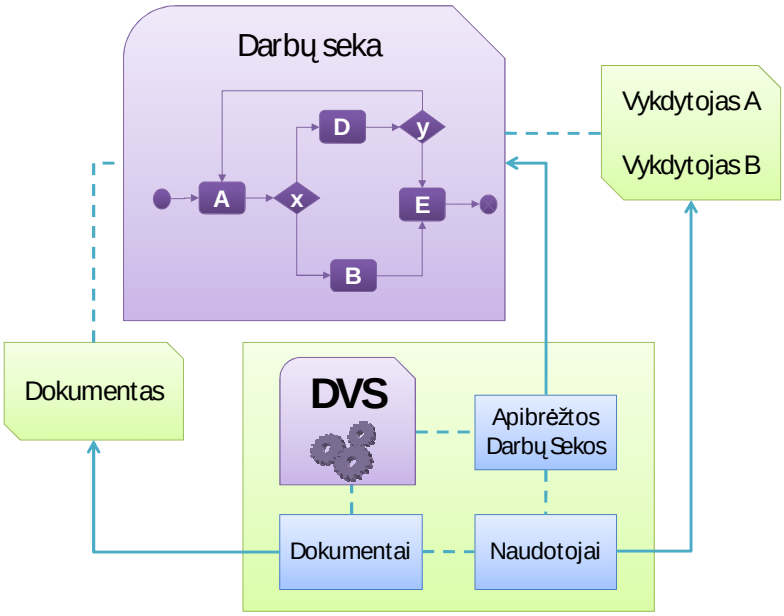
- darbuotojas pateikia tik būtiną informaciją, likusią turi sistema;
- darbuotojas bei administratorė sutaupo laiko;
- sistema automatiškai atlieka tam tikrus patikrinimus atsižvelgdama į turimą informaciją;
- sistema pati žino, kam kurti ir deleguoti užduotis (jei vadovas atostogauja, užduotis deleguojama kitam);

- taupomas popierius.

Formaliai, darbo seka – tai verslo proceso visiškas arba dalinis automatizavimas, kuriuo metu dokumentai, informacija ar užduotys yra nuo vieno veiklos dalyvio (vykdytojo) perduodami kitam, siekiant atlikti veiksmus atsižvelgiant į iš anksto aprašytas taisykles [Hol95]. Veiklos dalyviu arba tiesiog vykdytoju gali būti:

- sistemos naudotojas ar naudotojų grupė;
- sistemos naudotojų dinaminė grupė – grupė, kurios nariai gali būti išskaičiuojami pagal aprašytą algoritmą atsižvelgiant į dokumento atributus ar sistemos parametrus;
- pati sistema – sistema atlikinėdama užduotį gali ją deleguoti kitai sistemai.

Taigi dokumentų valdymo sistemoje gali būti išsaugotos apibrėžtos darbo sekos. Apibrėžta darbo seka iniciavimo metu gali būti surišta su pasirinktu dokumentu, o veiklų vykdytojais tampa arba pati sistema, arba sistemoje užregistruoti naudotojai (4 pav.).



4 pav. Darbo sekos ir dokumentų valdymo sistema

Darbo sekų palaikymas susideda iš tam tikro funkcionalumo, kuri galima paskirstyti į kelias sritis:

- galimybė apibrėžti darbo seką;
- apibrėžtos darbo sekos patikrinimas ir optimizavimas;
- apibrėžtos darbo sekos vykdymas;
- vykdomos darbo sekos stebėjimas bei protokolavimas.

1.2. Darbo sekų modeliai

Darbo sekos gyvavimas prasideda nuo jos apibrėžimo arba kitaip sakant užrašymo formaliąja kalba. Yra įvairių būdų tai padaryti:

- parašyti dokumentą (specifikaciją);
- aprašyti proceso darbo seką pagal apibrėžtą struktūrą (pvz. grįsta XML);
- nupaišyti darbo sekos diagramą (pavyzdžiui UML veiklos diagrama).

Svarbu yra suvokti, kad modeliuojant darbo sekas bet kokiu iš išvardintų būdu, rezultatas turi būti vienodas – tiksliai apibrėžtas procesas. Visais atvejais turi būti tenkinama pagrindinė sąlyga – apibrėžta darbo seka turi būti suprantama vienareikšmiškai, o tam pasiekti yra naudojami atitinkami standartai. Aukščiau pateikti trys procesų apibrėžimo būdai turi skirtingą pritaikomumą. Pavyzdžiui, parašytą specifikaciją gali būti nesunkiai interpretuojama žmogaus (priklauso nuo pasirinkto specifikavimo būdo). Darbo seka apibrėžta tam tikra struktūra turi privalumą, kad ją lengvai gali interpretuoti ir vykdyti kompiuterinė sistema. Nupaišyta diagrama gali būti ypač lengvai suprantama ir modifikuojama žmogaus, bet, kad ją galėtų interpretuoti sistema, ją teks transformuoti į tam tikro formato struktūrą. Toliau tam tikrais žymėjimais pavaizduota darbo seką vadinsime modeliu, o apibrėžimo procesą – modeliavimu.

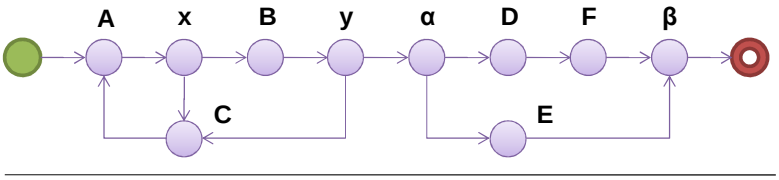
Nepriklausomai nuo to, kokį proceso modeliavimo būdą pasirinksime, visus modelius galima paskirstyti į du pagrindinius tipus [OAD06]:

- modelis grįstas grafais (angl. graph-oriented);
- modelis struktūrizuotas blokais (angl. block structured).

Kiekvienas iš išvardintų modelių pasižymi tam tikrais privalumais bei trūkumais, todėl kiekvienas iš tų modelių turi savo naudojimo nišas. Toliau panagrinėsime juos šiek tiek detaliau.

Jeigu panagrinėsime darbo sekos konstrukciją, tai išaiškinsime, kad bendriniu atveju ji susideda iš sekančių pagrindinių komponenčių:

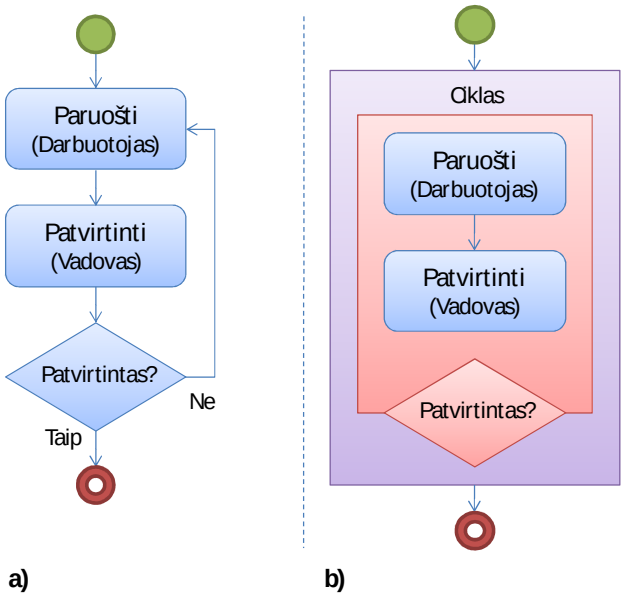
- pradžios taškas – mazgas, nuo kurio yra pradedamas darbo sekos vykdymas. Kiekviena darbo seka turi turėti vieną ir tik vieną pradžios tašką;
- pabaigos taškas – darbo sekos išeities taškas. Jų turi būti bent vienas;
- veiklos – pagrindiniai darbo sekos konstrukcijos vienetai, kuriuose atliekamas apibrėžtas darbas;
- sąlygos – mazgas, kuriame yra atliekamas nustatytos sąlygos patikrinimas;
- ryšiai – ryšiai tarp aukščiau išvardintų komponenčių. Ryšiai yra kryptingi. Jų paskirtis yra nurodyti, koks darbo sekos žingsnis arba žingsniai turi būti atliekami toliau.



A, B – patvirtinimo veikla
C – peržiūrėjimo veikla
D – pasirašymo veikla
E – registravimo veikla
F – nusiuntimo veikla
x, y – patvirtinimo sąlygos
α, β – lygiagretumo mazgai

5 pav. Darbo sekos pavaizdavimas grafu

Tokia darbo sekų konstrukcija yra validi tiek grafais grįstiems modeliams, tiek struktūrizuotiems blokais modeliams, bet kiekviename iš jų skirtingai suvokiama. Grafais grįstuose modeliuose sumodeliuotą diagramą galima tiesiogiai transformuoti į paprastą grafą (detaliau bus panagrinėta vėliau). Pavyzdžiui, anksčiau pateiktame 3-me paveikslėlyje darbo seką atitinkančią UML veiklos diagramą galime pavaizduoti paprastu grafu. Toks grafas yra pateiktas 5-me paveikslėlyje. Čia A, B, C, D E, F raidėmis yra pažymėtos veiklos; x, y – sąlygos, o α, β – lygiagretumo mazgai. Kaip matome visos anksčiau išvardintos darbo sekos komponentės išskyrus ryšius yra suprantamos kaip grafo viršūnės, o ryšiai – orientuoti grafo lankai (atsiranda papildomos lygiagretumo viršūnės). Tokiu būdu galima pavaizduoti kiekvieną grafais grįstą modelį orientuotu grafu.



a) **b)**

6 pav. **a) Grafais gristas modelis**
b) Struktūrizuotas blokais modelis

Modeliai struktūrizuoti blokais yra komponuojami blokais arba kitaip sakant komponentais. Visi darbo sekos konstrukcijos elementai įskaitant ir ryšius yra komponentai. Modeliai konstruojami įdedant vienus komponentus į kitus (kompozicijos principas). Priklausomai nuo konkretaus modelio įdėti komponentai gali būti interpretuojami įvairiai.

Dažniausiai, komponentai esantys kito komponento viduje yra interpretuojami kaip tam tikra seka.

6-me a) paveikslėlyje yra pavaizduotas labai paprastas grafais grįstas modelis susidedantis iš dviejų veiklų ir vienos sąlygos. To paveikslėlio b) variante yra pavaizduotas modelio struktūrizuoto blokais pavyzdys, kuris yra ekvivalentus a) modeliui. Jis susideda iš tų pačių dviejų veiklų ir ciklo komponento, kuris atstoja grafo viršūnės sąlygą ir ryšį (orientuotas grafo lankas). Komponento-ciklo viduje esantys komponentai yra interpretuojami kaip veiklų seką.

1.3. BPMN standartas

Kaip jau buvo minima anksčiau, kad užtinkinti darbo sekų pavaizdavimo vienareikšmiškumą reikia naudoti tam tikrus apibrėžtus standartus. BPMN būtent ir yra vienas iš tokių standartų. Šio standarto pirmoji versija buvo išleista dar 2004 metais BPMI.org organizacijos dėka. Dabar šio standarto tobulinimą perėmė kita organizacija – OMG, sukūrusi UML standarte veiklos diagramą [BPM07].

BPMN – tai sistema grafinių žymėjimų skirtų verslo procesų schemos, suprantamos žmogui, pavaizdavimui. Verslo procesų diagrama (BPD, Business Process Diagram), BPMN pagrindas, konstruojasi panašiais principais kaip ir tradicinės blokų schemos [BPM07], [Whi04a]. BPD buvo projektuojama taip, kad butų pasiekti du pagrindiniai tikslai [OR03]:

1. Būtų lengva naudoti ir suprati. Šios diagramos gali būti naudojamos norint greitai ir paprastai sumodeliuoti verslo procesą suprantamą ne tik techninį išsilavinimą turinčiam žmogui;
2. Būtų galimybė modeliuoti kompleksinius verslo procesus, kuriuos galima natūraliai transformuoti (angl. map) į verslo procesų vykdymo kalbas.

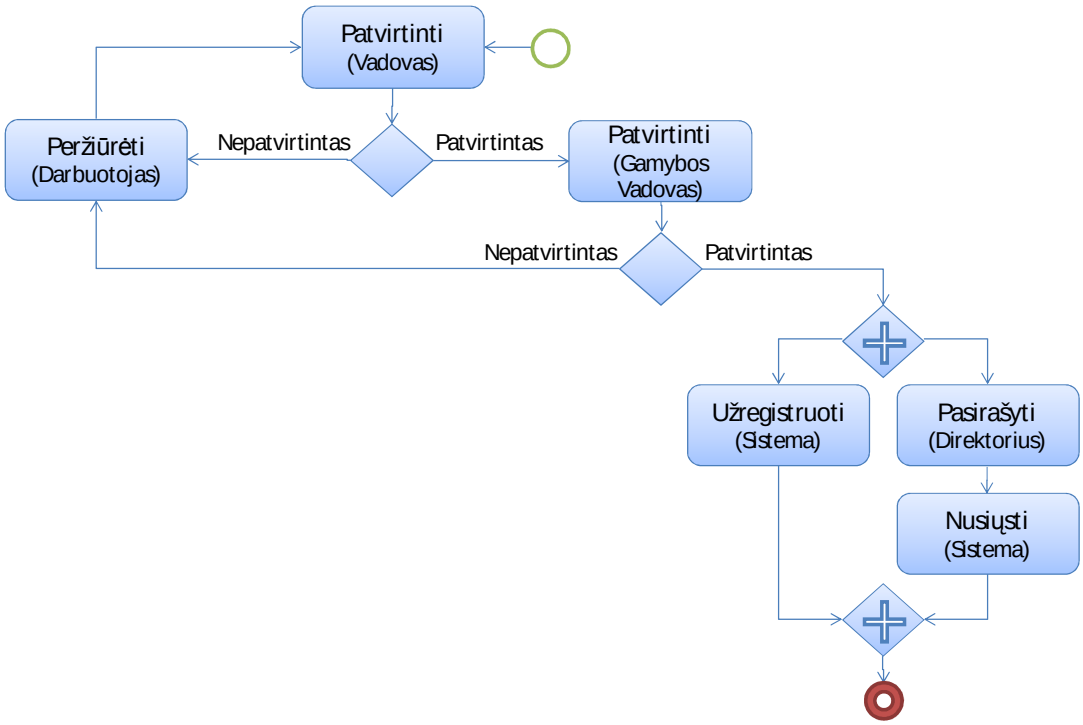
BPD susideda iš sekančių pagrindinių objektų grupių [Whi04a], [Whi04b]:

1. Srauto objektai (angl. Flow Objects):
    - įvykis – tai gali būti proceso pradžios ar pabaigos taškas arba įvykis iškylantis proceso vykdymo metu;
    - veikla – pagrindinė konstrukcinė proceso dalis – atliekamas konkretus darbas arba paprocesis;
    - vartai (angl. Gateway) – tai proceso eigos (kelio) susikirtimo arba išsišakojimo mazgas. Prie vartų taip pat priskiriama sąlyga (irgi turi įeiti ir išeitis). Yra įvairaus tipo vartų: AND, OR, XOR.
2. Sujungimo objektai (anlg. Connecting Objects) – priemonė sujungti kitus BPD objektus. Nurodo, kokia eiga turi būti vykdomas procesas. Yra išskiriamos trys sujungimo linijos:

- sekos – vaizduojama įprastinė proceso eiga tarp BPD tekėjimo objektų;
- pranešimo – vaizduojamas pranešimų siuntimas tarp veiklų;
- asociacijos – vaizduojami įvairaus pobūdžio asociacijos tarp diagramos objektų.

3. Takeliai (angl. Swimlanes) – tam tikri grupavimo konteineriai skirti proceso veikloms išrūšiuoti, pavyzdžiui, pagal roles (analogija UML veiklos diagramoje). Atskiruose takeliuose galima vaizduoti atskirus procesus.
4. Artefaktai (angl. Artifacts) – žymėjimai skirti pagalbinei, paaiškinamai informacijai pateikti. Yra trijų tipų artefaktai:
   - duomenų objektas – skirtas nurodyti, kokius duomenis veikla gamina arba panaudoja;
   - grupė – skirta sugrupuoti diagramos objektus. Neša tik informatyvią prasmę;
   - anotacija – papildoma informaciją apie objektą.

Išvardintos objektų grupės parodo tik apibendrintą BPMN standarto objektų struktūrą ir čia nėra išvardinti visi standarte aprašyti objektai. Jie taip pat nebus detaliai aprašomi ir nagrinėjami šiame darbe.



7 pav. Darbuotojų išleidimo į atostogas proceso BPD diagrama

BPD diagrama, taip kaip ir UML veiklos diagrama, vaizduoja grafais grįstus modelius. Taigi žiūrint formaliai BPD tai yra tam tikras grafas su įvestomis konstravimo taisyklėmis [Gao06]. 7-me paveikslėlyje yra pateikta BPD pavyzdinė diagrama, kurioje yra pavaizduotas anksčiau nagrinėtas darbuotojų išleidimo į atostogas procesas. Kaip matome, jis beveik niekuo nesiskiria nuo veiklos diagramos, pateiktos 3-me paveikslėlyje. Vienas pagrindinis skirtumas yra

lygiagretumo žymėjimas. BPD diagramoje yra panaudoti AND tipo vartai. AND tipo vartai reiškia, kad nuo išsišakojimo mazgo lygiagrečiai vykdomos visos išsišakojimo atšakos, o susijungimo mazge laukiama, kol visos įeinančios šakos bus įvykdytos.

1.4. Worfklow Foundation ir sekų modelis

2006 metų pabaigoje buvo išleista naujoji .NET 3.0 platformos versija, kurioje atsirado naujovė - WF (angl. Windows Workflow Foundation). WF leidžia išspręsti daugybę problemų susijusių su darbo sekų valdymu, pradedant nuo jų apibrėžimo ir baigiant vykdymu. Tai leidžia santykinai nedideliais kaštais pritaikyti šią platformą dokumentų valdymo sistemose [ACW05], [MSD07a]. Kita vertus WF pateikia savo įrankį leidžiantį sumodeliuoti tam tikrą darbo seką sekų modelio pavidalu (WF terminologija – Sequence Workflow), o šis modelis nėra suderinamas su anksčiau minėtu darbo sekų modeliavimo BPMN standartu. To pasekoje gali tekti integruoti kitą darbo sekų apibrėžimo įrankį suderinamą su BPMN standartu, o įrankiu apibrėžtą struktūrą automatiškai transformuoti į WF platformai suprantamą pavidalą.

WF pateikiamas sekų modelis, priešingai negu BPD, yra struktūrizuoto blokais modelio tipas. Taigi visi WF sekų modelio darbo sekos konstrukcijos elementai įskaitant ir ryšius yra komponentai, o modelis konstruojamas iš komponentų, kur vieni komponentai įdedami į kitus. Kaipgi nustatyti sumodeliuotoje darbo sekoje vykdymo eiliškumą? Atsakymas yra paprastas. Reikia remtis sekančiomis pagrindinėmis taisyklėmis:

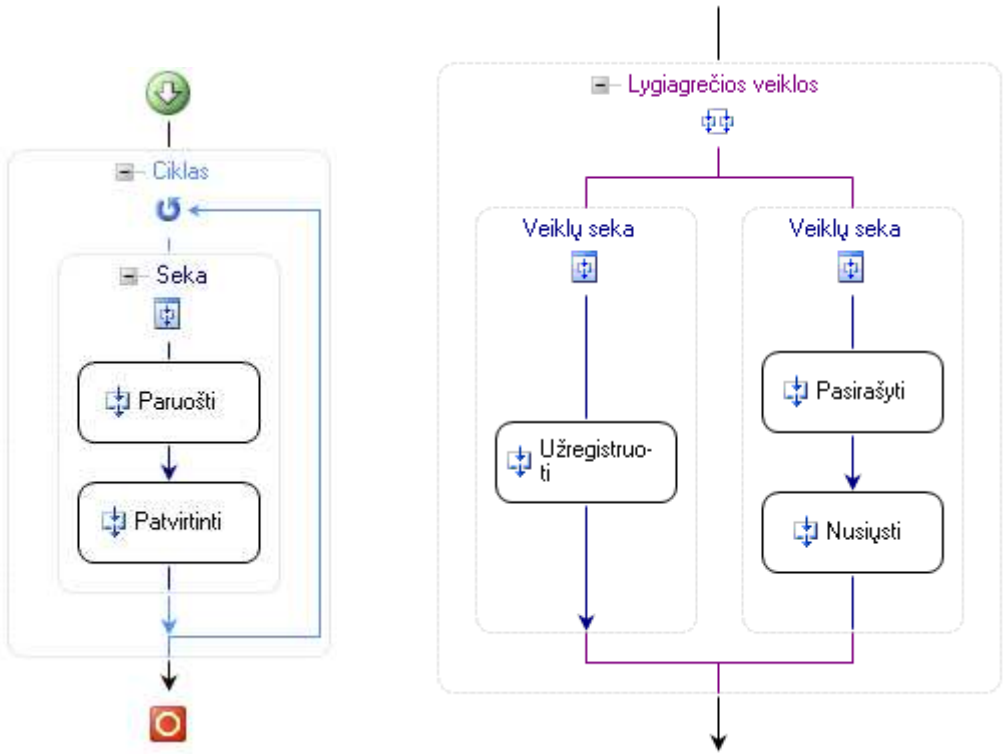
1. Visi komponentai-vaikai esantys kitame komponente-tėve yra seka, kurios elementai yra interpretuojami pagal komponento-tėvo taisykles. Pavyzdžiui, sekos komponente esančios veiklos vykdomos eilės tvarka (eilės tvarka nurodo seka), o lygiagretumo komponente esantys komponentai vykdomi lygiagrečiai.
2. Ciklo komponentas yra vienintelė „grįžimo atgal“ galimybė.

Suvokti ir naudotis išvardintomis taisyklėmis yra labai paprasta. Tokią struktūrą taip pat nesunkiai galima pavaizduoti grafu, bet atvirkštinė transformacija dažniausiai yra pakankamai komplikuota, o tam tikrais atvejais gal būt ir visai neįmanoma. Detaliau ši problema bus paanalizuota vėliau.

8-me paveikslėlyje yra pavaizduotos dvi paprastos WF sekų modelio diagramos sumodeliuotos WF karkaso modeliuotoju. 8 a) paveikslėlis atitinka anksčiau 6-me paveikslėlyje pavaizduotoms darbo sekoms, o 8 b) - 5-me paveiksliuko nagrinėtos darbo sekos grafo α, β dalį.

Svarbus faktas yra tas, kad .NET 3.0 pateikiamas WF sekų modelis nėra standartas, o aprašytas tam tikras konstrukcinis modelis (kodo pavidalu). Tokį modelį galima laisvai plėsti kuriant savo komponentus [Kit07]. Kaip pavyzdį paimkime egzistuojanti WF lygiagretumo komponentą. Jis atitinka BPD vartų AND sąvoka. WF sekų modelį galima papildyti nauju

lygiagretumo komponentu, kuris atitiktų tarkim BPD vartų OR sąvoką. Taigi WF sekų modelis tai yra tam tikras karkasas konstruoti darbo sekas pagal laisvai apibrėžtą komponentų pavaizdavimą. WF sukonstruotą modelį galima saugoti XAML formatu (XML gristas struktūrinis formatas), kuris irgi turi atitikti tas pačias konstravimo taisykles. XAML formatu aprašyta darbo seką galima sukompiliuoti į DLL biblioteką ir vykdyti, pavyzdžiui, dokumentų valdymo sistemoje. Taigi nagrinėjant WF sekų modelį nesvarbu ar nagrinėsime XAML formato modelį, ar tam tikrą vizualių žymėjimų rinkinį, nes abiem atvejais galioja tos pačios konstravimo koncepcijos [MSD07b] [MSD07c].



a) **b)**

8 pav. WF darbo sekos sekų modeliai

1.5. Darbo sekų modelių transformavimas

Kaip jau buvo minima, visus darbo sekų modelius galima paskirstyti į du tipus: grįstus grafais bei struktūrizuotus blokais. Grafais grįsti modeliai yra lengvai suvokiami žmogui bei juos nesudėtingai galima nagrinėti bei modeliuoti. Darbo sekų vykdymo platformose yra paplitusios blokais struktūrizuoti modeliai, pavyzdžiui, WF sekų modelis, BPEL. Todėl yra poreikis automatinio transformavimo iš vieno modelio tipo į kitą be žmogaus įsikišimo.

Anksčiau minėtas transformavimas struktūrizuoto blokais į grįstą grafais modelį yra tiesioginis, kadangi kiekvieną struktūrizuoto blokais modelio komponentą galima pakeisti grafo viršūne bei atitinkamais orientuotais lankais, kur visi lankai išskyrus ciklo komponentą yra vienakrypčiai.

Atvirkštinis transformavimas, deja, nėra toks paprastas ir jį visiškai automatizuoti gali nepavykti. Tą faktą patvirtina straipsniai, kuriuose panaši problema yra nagrinėjama [OAD06], [Gao06], [ODB05], [KH04]. Pavyzdžiui, [KH04] straipsnyje yra pateiktas sprendimas, kaip spręsti problemas kylančias transformuojant nestruktūrizuotus verslo procesų modelius į ekvivalenčius verslo procesus paremtus struktūrizuotais ciklais. Žiūrint iš mūsų nagrinėjamos problemos perspektyvos, šiame straipsnyje yra nagrinėjami irgi du anksčiau šiame darbe minimi modelių tipai ir konkrečiai yra sprendžiama ciklų transformavimo problema.

Straipsnyje [ODB05] buvo sprendžiama ta pati ciklų transformavimo problema. Šis darbas yra ypatingas tuo, kad jame pateiktas transformavimo algoritmas yra adaptuotas į BPMN/UML - BPEL transformavimą. BPEL yra plačiai paplitusi verslo procesų apibrėžimo kalba skirta WEB paslaugoms realizuoti ir vykdyti verslo procesus. Ši kalba, taip pat kaip ir WF sekų modelis, yra struktūrizuota blokais.

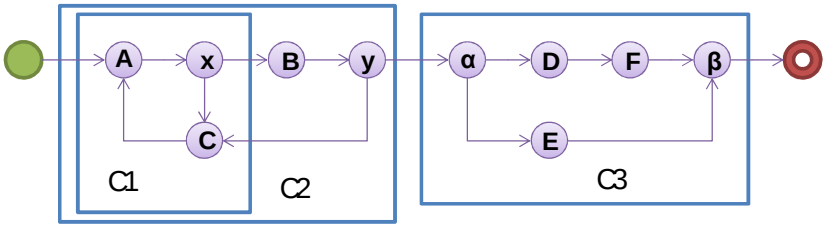
Anksčiau minėtų straipsnių dėka atsirado dar vienas darbas [OAD06]. Šiame moksliniame straipsnyje yra pateikta transformacija iš BPMN standarto BPD diagramų į BPEL struktūras. Čia vertėtų paminėti tai, kad tokio tipo transformacija yra įdomi tuo, kad BPMN pateikia modelį skirtą verslo procesų pavaizdavimui, o BPEL savo ruoštu yra modelis skirtas XML verslo procesų apibrėžimui. Šis straipsnis parodo, kad tokių modelių transformavimas yra visai įmanomas ir nėra skirtumo ar modelis reprezentuoja vaizdinę notaciją ar tam tikras aprašytas struktūras teksto pavidalu (šiuo atveju XML). Straipsnyje taip pat buvo pastebėta, kad [KH04], [ODB05] darbuose pateikti transformavimo algoritmai turi tam tikrus apribojimus. Darbe [OAD06] yra pateikta transformacija, kuri atsižvelgia į papildomas problemas atsirandančias transformuojant AND ir XOR tipo mazgus arba šnekant BPMN terminais – susikirtimo vartus (žiūrėkite BPMN standarto trumpą aprašymą pateiktą anksčiau).

Svarbu yra pastebėti, kad anksčiau minėtuose straipsniuose yra pateiktos tam tikros transformacijos leidžiančios transformuoti nustatytus verslo procesus, bet nėra išpildomas transformavimo pilnumas. Jeigu rišimės prie darbo sekų naudojamų dokumentų valdymo sistemose, tai galima būtų pastebėti, kad tam tikrais atvejais pateiktos transformacijos nėra optimalios ir nevisai tinkamos DVS.

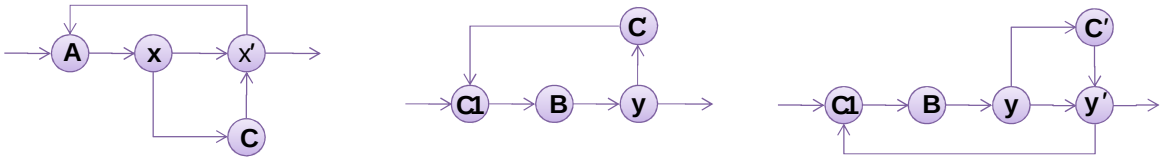
Kadangi WF atsirado visai neseniai, tai iki šiol dar neatsirado nei vieno straipsnio, kuriame butų nagrinėjama transformacija iš BPMN į WF sekų modelį. Be to, minėtuose straipsniuose dažniausiai yra nagrinėjami bendriniai verslo procesai. Taigi atsiranda poreikis BPMN-WF sekų modelio transformacijai. Be to, šiame darbe bus nagrinėjama ne visa aibė verslo procesų, o tik darbo sekos pritaikomos konkrečiai dokumentų valdymo sistemoms.

Kad geriau suvokti tam tikrus transformavimo subtilumus, dar kartą grįžkime prie 1.1.2 skyriuje (3 pav.) nagrinėto darbuotojų išleidimo į atostogas proceso ir pabandykime jį

transformuoti į WF sekų modelį. Paprastumo dėlei, nagrinėkime šio proceso grafo pavaizdavimą pateiktą 5-me paveikslėlyje. Pirmiausiai dekomponuokime nagrinėjamą grafą į dalis, kurias galima nagrinėti atskirai (9 pav.). Ši technika yra naudojama ir [OAD06] straipsnio pateiktame transformavimo algoritme. Dabar nagrinėkime kiekvieną iš dekomponavimo metu gautų dalių atskirai. Pradekime nuo C1 dalies. Šitą dalį reikia transformuoti į izomorfinį grafą, kurį galima būtų pakeisti komponentais su ciklu. Toks grafas yra pavaizduotas 10-me a) paveikslėlyje. Kaip matome, prie grafo buvo pridėta nauja sąlygos viršūnė x′ , kur x′ = x. Be to, naujai sukurtas grafas yra izomorfinis pradiniam tada ir tiktai tada, kai C neįtakoja x (x′) sąlygos. Jeigu C įtakotų x sąlygą, tai įvykdžius veiklą C ir praėjus pro sąlyga x′ galėtų neįvykti grįžimas prie veiklos A, kas turėtų įvykti, vadinasi, C1 nebūtų izomorfinis transformuotam C1.



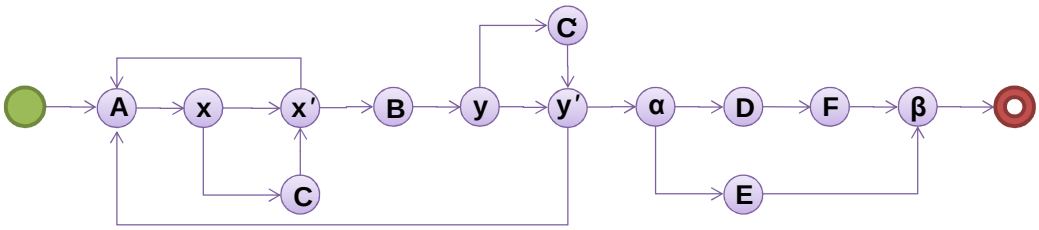
9 pav. Grafo dekomponavimas



a) **b)** **c)**

10 pav. Grafo transformuotos dalys

Sekančiu darbu bus grafo C2 transformavimas. Pradžiai klonuojame viršūnę C ir pažymime ją C′ (10-as b pav.). Veikla C yra ekvivalenti C′ veiklai. Gautą grafą transformuojame tokiu pačiu principu kaip ir C1 įvedant sąlygą y′, kur y = y′ bei C′ neįtakoja y (y′). Tokiu būdu gauname grafą pavaizduota 10-me c) paveikslėlyje, kuris yra izomorfinis C2 grafui, kai y = y′, x = x′, C neįtakoja x bei C′ neįtakoja y. Grafo C3 transformuoti nėra poreikio. Taigi belieka sujungti grafus į vieną ir gauname transformuotą grafą, kuris yra izomorfinis pradiniam (11 pav.). Tokį grafą nesunkiai galime transformuoti į WF sekų modelį, kuris yra pavaizduotas šio darbo priede. Tame pačiame priede galima taip pat pamatyti transformuoto grafo pavaizdavimą BPD diagramos pavidalu.



11 pav. Transformuotas grafas

Taigi nagrinėjamo nesudėtingo proceso transformavimas parodė, kad ta procedūra nėra labai triviali ir reikalauja tam tikrų pastangų. Netgi šiam atvejui prireikė pritaikyti du skirtingus grafo dalių transformavimo metodus. Neabejotina, kad egzistuoja dar nemažai įvairių situacijų, kai transformacija būtų pakankamai netriviali ir reikalautų tam tikrų transformavimo metodų taikymo bei algoritmo aprašymo.

1.6. BMPN žymėjimų išrinkimas

1.6.1. Pagrindimas

Transformuojant bet kokį objektą X į Y, visų pirma reikia aprašyti, kokioms aibėms priklauso X ir Y objektai. Kad aprašymas būtų vienareikšmis ir santykinai lengvai tikrinamas ne tik žmogaus bet ir mašinos, jis turi būti specifikuotas pasirinkta formalia specifikavimo kalba. Mūsų atveju transformuosime objektus X ir Y, kur $X \in Mx \subset BPMN$, $Y \in My \subset WF$. Taigi mums reikia formaliai aprašyti žymėjimų ir struktūrų aibes Mx ir My, kurios yra atitinkamai BPMN standarto ir WF sekų modelio poaibiai.

1.6.2. Žymėjimų išrinkimas

BPMN standarte yra apibrėžta daugybė žymėjimų, kuriomis galima aprašyti bet kokį grafais grįstą verslo procesą. Dalį žymėjimų nagrinėti nėra būtina. Kita dalis gali būti nepritaikoma dokumentų valdymų sistemose. Atsižvelgiant į šiuos ir dar kai kuriuos aspektus, šiame darbe buvo nuspręsta nenagrinėti sekančių žymėjimų:

1. Žymėjimai leidžiantys aprašyti daugialypius procesus.

   Pagrindimas. Nagrinėsime DVS aplinkas, kuriose procesai vykdomi su dokumentais, kur skirtingi procesai neįtakoja vienį kitų veiklos. Taigi šiame darbe nenagrinėsime daugialypių procesų, tarp kurių įmanomi kokie nors sinchroniniai ar asinchroniniai tarp-procesiniai ryšiai.

2. Žymėjimai nešantys papildomą informacinę prasmę (artefaktai).

   Pagrindimas. Šio tipo žymėjimai visiškai neįtakoja proceso vykdymo. Todėl jų nagrinėjimas transformacijose yra bevertis.

3. Žymėjimai grįsti įvykiais bei pranešimais.

   Pagrindimas. Šiame darbe nuspręsta nenagrinėti šių žymėjimų (įvedame apribojimą). Tokio tipo procesai būna tipiškai sunkiau suvokiami ir retai naudojami DVS.

4. Žymėjimai leidžiantys skirtingiems procesams dalintis tarpusavyje informacija.

   Pagrindimas. Darbe nenagrinėsime daugialypių procesų, todėl žymėjimai leidžiantys apibrėžti ryšius tarp tokių procesų yra nereikalingi.

5. Žymėjimai dubliuojantys vieni kitus.

   Pagrindimas. BMPN standarte yra daugybė žymėjimų, kurie yra ekvivalentūs procesų vykdymo prasme, bet skirtingi vaizdavimo prasme. Atliekant modelių transformacijas tokie žymėjimai interpretuojami kaip ekvivalentūs. Todėl norint išvengti painiavos tokius žymėjimus eliminuosime.

Taigi atsižvelgiant į aukščiau aprašytus niuansus buvo atrinkti sekantys BPMN žymėjimai:

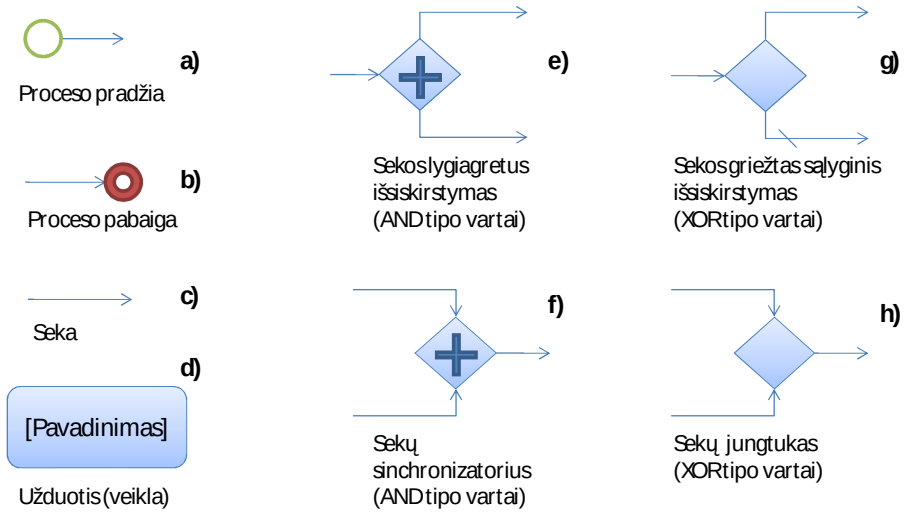
- proceso pradžios taškas;
- proceso pabaigos taškas;
- užduotis (veikla);
- sekos tėkmė – elementų sujungimo linija apibrėžianti nuoseklųjį proceso vykdymą (angl. Sequence flow);
- pralaidumo vartai (mazgai):
  - išsišakojimo vartai:
    - AND tipo vartai – lygiagretus sekos išsišakojimas;
    - XOR tipo vartai – griežtas sąlyginis sekos išsišakojimas su negalimu lygiagretumu;
  - sujungimo vartai:
    - AND tipo vartai – sinchronizavimas visų įėjimų;
    - XOR tipo vartai – praleidimas tik vieno įėjimo.

Toliau nagrinėsime tik aukščiau išvardintus žymėjimus ir vadinsime juos baziniais BPMN žymėjimais. Išvardinti žymėjimai yra pavaizduoti 9-me paveikslėlyje.

Apibrėžimas 1 (Bazinis žymėjimų rinkinis)

Baziniu BMPN žymėjimų rinkiniu vadinsime *BPN* = (S, F, A, G, L), kur:

- S – proceso įėjimo taškas;
- F – proceso išėjimo taškas;
- A – veiklų rinkinys;
- G = (G_{XOR}, G_{AND}, G'_{XOR}, G'_{AND}) – AND ir XOR vartų rinkinys (su apostrofomis pažymėti sinchronizaciniai vartai)
- L – nuosekli tėkmė;



9 pav. Bazinis BPMN žymėjimų rinkinys

1-je lentelėje yra pateiktos taisyklės, pagal kurias turi būti jungiami *BPN* elementai. Kairiajame lentelės stulpelyje ir viršutinėje eilutėje yra įrašyti elementai S, A, G, F priklausantys *BPN*, tarp kurių galimi atitinkami ryšiai. Simboliu **X** yra pažymėti atvejai, kai jungti objektų negalima, √ – kai sujungimas galimas. Kai sujungimas yra galimas, laužtinėse skliausteliuose [**a**, **b**] yra nurodyta, koks galimas kiekybinis sujungimas, kur **a**, **b** ∈ {1, *}, o **a** žymi iš kiek objektų į vieną galimas sujungimas, **b** – į kiek objektų iš vieno (iš daug į daug nėra galimas). Pavyzdžiui, [1, *] žymėjimas reiškia, kad iš daug į vieną sujungimas negalimas, o iš vieno į daug – galimas; [*, *] – galimas sujungimas tiek iš daug į vieną, tiek iš vieno į daug. BPMN standartas leidžia daugiau jungimo galimybių negu yra pateikta šioje lentelėje, bet tų jungimų aibė yra pilnai padengiama kitais BPN žymėjimais.

Toliau nagrinėsime tik tokius *BPN* elementais aprašytus modelius, kuriuose taip elementų galimi lentelėje aprašyti sujungimo atvejai.

Apibrėžimas 2 (Bazinė BPD diagrama)

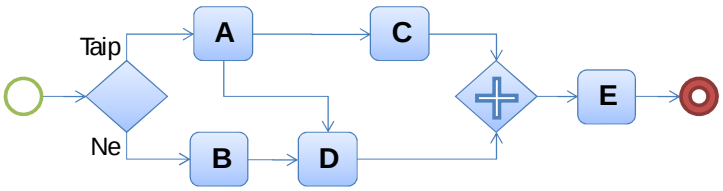
Bazine BPD diagrama vadinsime verslo proceso modelį (diagramą) aprašytą **tik** *BPN* žymėjimais, kur žymėjimų objektai turi būti sujungti pagal taisykles aprašytas 1-je lentelėje.

1 lentelė . BPN objektų sujungimo taisyklės

Iš \ Į	S	A	G	F
S	X	√ , [1, 1]	√ , [1, 1]	√, [1, 1]
A	X	√ , [1, 1]	√ , [1, 1]	√, [1, 1]
G	X	√ , [1, *]	√ , [*, *]	√ , [1, 1]
F	X	X	X	X

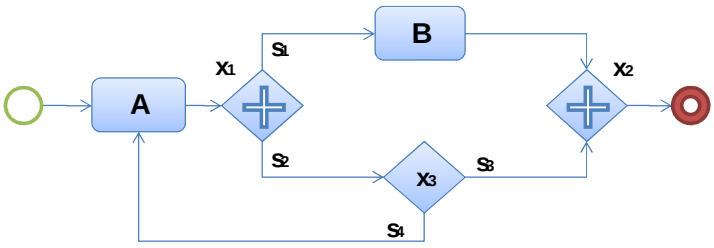
BPMN standartas apibrėžia ne tik procesų modeliavimo žymėjimus, bet ir tam tikrus apribojimus. Tų apribojimų nevisada užtenka, kad sumodeliuotas procesas būtų visiškai aiškus. Kartais netgi paprastame modelyje galima iš pirmo žvilgsnio nepastebėti tam tikrų nepageidaujamų situacijų, kuriuos gali iškilti vykdant procesą. Kadangi BPMN standartas leidžia sumodeliuoti tokio tipo modelius, modeliavimo programoms (modeliuotojams) arba žmonėms modeliuojantiems procesus reikia patiems rūpintis, kad jų kūrinys būtų prognozuojamas, t. y. būtų vykdomas taip, kaip tikisi žmogus.

Pavyzdžiui, 10-me paveikslėlyje yra pateiktas sumodeliuotas modelis su potencialia aklaviete [BPM04a]. Pavaizduotame procese, susidursime su aklaviete tuo atveju, kai sąlyginiame išsišakojime bus patenkinta sąlyga „Ne“. Toliau bus vykdomos nuosekliai veiklos B ir D ir procesas sustos, kadangi sinchronizavimo mazgas lauks įėjimo iš kito išsišakojimo irgi (veiklos C užbaigimo).



10 pav. Verslo procesas su potencialia aklaviete

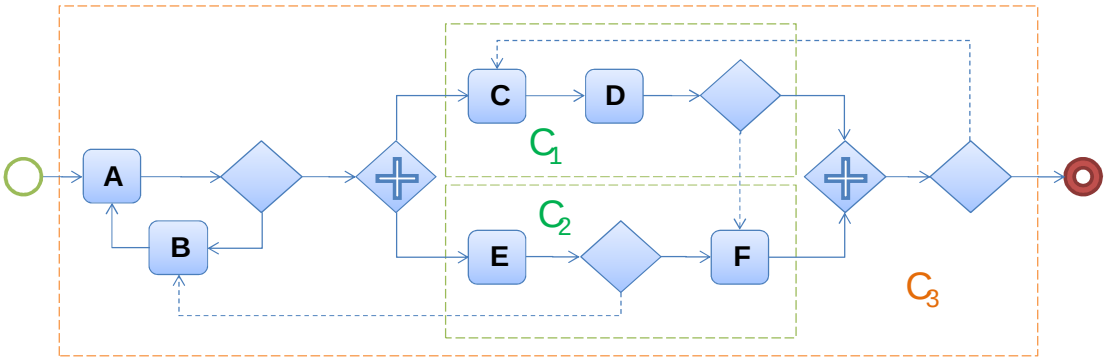
Kitas pavyzdys, pateiktas 11-me paveikslėlyje, demonstruoja irgi savotišką nepageidaujamą situaciją. Šiame procese sinchronizatorius x_2 lauks įėjimų iš dviejų atšakų. Taigi šioje situacijoje prie sinchronizatoriaus x_2 kaupsis laukiantieji įėjimai iš s_1-x_2 atšakos tol, kol nebus patenkinta sąlyga s_3, o veikla B bus vykdoma tiek kartų, kiek bus patenkinta sąlyga s_4.



11 pav. Verslo procesas su nepageidaujama būsena

Situacijų išvardintų dviejuose paskutiniuose pavyzdžiuose ir daugybės kitų galima išvengti pasinaudojant procesų modeliavimo gerosiomis praktikomis. Mes pasinaudosime šiomis taisyklėmis:

1. Užtikrinti, kad galimas lygiagretus išsišakojimas būtų visada sinchronizuojamas.
   - *Paaiškinimas*. Šis reikalavimas patenkinamas, kai kiekvienas galimas lygiagretus išsišakojimo mazgas naudojamas kartu su atitinkamu sinchronizuojančiu mazgu. Kadangi BPD lygiagretus išsišakojimas galimas tik nagrinėjant AND (OR tipo vartai nėra nagrinėjami, kadangi jie nesunkiai eliminuojami pasinaudojant AND ir XOR tipo vartais) tipo vartus, būtina, kad AND tipo išsišakojimo vartai BPD diagramoje turėtų atitinkamus AND tipo sujungimo vartus.
2. Nejungti tarpusavyje objektų, kurie randasi skirtinguose vykdymo srityse.
   - *Paaiškinimas*. Proceso vykdymo sritis, tai proceso dalis, kuri vykdoma nuosekliai. Lygiagretus išsišakojimas (proceso vykdymas) atsirandantis naudojant AND tipo vartus sudaro kitą vykdymo sritį. Pavyzdžiui, 12-me paveikslėlyje yra pavaizduotas procesas su trimis skirtingomis vykdymo sritimis C_1, C_2, C_3, o brūkšnine linija yra pavaizduoti ryšiai, kurie neatitinka šio reikalavimo, kadangi yra sujungti objektai esantys skirtingose vykdymo srityse.



12 pav. Proceso vykdymo sritys

Išvardintus tris reikalavimus (praktikas) modeliavimui vadinsime *validumo reikalavimais* ir žymėsime jas $V = (v_1, v_2, v_3)$, kur v_1, v_2, v_3 – aprašytos praktikos.

Apibrėžimas 3 (Validi bazinė diagrama BPD)

Bazinę diagramą BPD vadinsime *validžia*, jei ji tenkins visus $v_i \in V$ *validumo reikalavimus*.

Toliau nagrinėsime tik validžias BPD.

2. Transformavimas

2.1. BPMN žymėjimų specifikavimas

2.1.1. *Tęstinumo* semantikos taikymas

Prieš atliekant atitinkamas transformacijas su PBD diagramomis, jas reikia užrašyti formalia specifikavimo kalba. Tam tikslui naudosime tęstinumo (angl. Continuation) semantiką. Tęstinumo semantika - tai žymėjimo forma, sukurta programų su nestruktūrizuotais perėjimais specifikavimui [Rey98]. Straipsnyje [KH04] buvo pateiktas metodas, skirtas aprašyti abstrakčius verslo procesus, sudarytus iš nestruktūrizuotų ciklų pasinaudojant šia semantika. Šiame darbe pateiktas patobulintas pastarosios metodikos [KH04] variantas pritaikytas BPD procesams aprašyti.

2.1.2. Specifikavimas

Šiame skyrelyje pateiksime metodą, leidžianti užrašyti nagrinėjamą BPD darbo seką tęstinumo semantika. Šia semantika užrašytas darbo sekas vadinsime lygčių sistemomis. Kiekviena sistema gali būti sudaryta iš pagrindinės sistemos ir kelių lygčių posistemių. Lygčių sistema ar posistemė yra sudaryta iš lygčių. Lygtis yra lygybė, kurios kairėje pusėje yra kintamasis x_i, dešinėje – atitinkamas reiškinys, ekvivalentus kintamajam x_i. BPD darbo seką reikia dekomponuoti į paprocesius, laikantis taisyklės, kad kiekvienas XOR arba AND išsišakojimas, turintis atitinkamą sinchronizatorių yra paprocesis C_i priklausantis procesui arba kitam paprocesiui. Kiekvienam procesui ir paprocesiui reikia sudaryti atskirą lygčių sistemą pagal sekančias taisykles:

1. Kiekvienam proceso ir paprocesio žymėjimų elementui $e_i \in (S, F, A) \in$ BPN (žiūrėti 1-mą apibrėžimą) priskiriame atitinkamą kintamąjį x_i. Kiekvienas paprocesis yra interpretuojamas kaip atskiras procesas turintis įeitį ir išeitį, kuriems reikia priskirti atitinkamą kintamąjį. Paprocesių įeitis ir išeitis gali atstoti vartų mazgus.
2. Kiekvienam kintamajam x_i sukuriame x_i = <reiškinys> formos lygtį. Reiškinį formuojame remiantis šiomis taisyklėmis:
   - užrašome Invoke $< e_i >$; , kur $e_i \in (S, F, A)$ yra elementas pažymėtas kintamuoju $x_{i,}$. Paskui užrašome (atskirdami kabliataškiu) tai, kas turi būti vykdoma darbo sekoje po elemento e_i kvietimo:
     - jei tai yra $e_j \in (F, A)$, užrašome kintamąjį x_j, kuriuo yra pažymėtas e_j elementas;

- jei tai yra $e_j \in G_{XOR}$ neturintis atitinkamo sinchronizatoriaus $G'_{XOR,}$ užrašome sąlygas atitinkančias vartų išsišakojimo sąlygas pavidalu:

  If (<Sąlyga>) Begin
  <x_m>;
  [EndIf;]
  ...
  [ElseIf (<Sąlyga>)] Begin <x_n>; EndIf;]

  kur x_m ir x_n yra kintamieji, kuriais pažymėti elementai e_m ir e_n kviečiami patenkinus atitinkamas išsišakojimo sąlygas;

- jei tai yra $e_j \in G_{XOR}$ turintis atitinkamą sinchronizatorių G'_{XOR}, tada užrašome:

  Xor
  If (<Sąlyga>) Begin <C_m>; EndIf;
  ...
  [If (<Sąlyga>) Begin <C_n>; EndIf;]
  EndXor;
  x_k;

  kur C_m ir C_n yra paprocesių sisteminiai kintamieji. Po *Xor-EndXor* sekcijos užrašome kintamąjį x_k, kuriuo pažymėtas elementas e_k einantis po $e'_j \in G'_{XOR}$ (po sinchronizatoriaus);

- jei tai yra $e_j \in G_{AND}$ turintis atitinkamą sinchronizatorių G'_{AND}, (validi BPD turi turėti sinchronizatorių), tada užrašome:

  Parallel
  Branch <C_m>; EndBranch; [..; Branch <C_n>; EndBranch;]
  EndParallel;
  x_k;

  kur C_m ir C_n yra paprocesų sisteminiai kintamieji. Po *Parallel-EndParallel* sekcijos užrašome kintamąjį x_k, kuriuo pažymėtas elementas e_k einantis po $e'_j \in G'_{AND}$ (po sinchronizatoriaus);

- jei procese nuosekliai yra vykdomi elementai e_i, ... e_j, tokiu atveju kelias išraiškas (lygtis) galima sujungti į vieną išraišką (lygtį):

  <x_i> = Invoke <e_i >; <x_{i+1}>;
  ....
  <x_{j-1}> = Invoke <e_{j-1} >; <x_j>;
  ⇓ ⇓ ⇓
  <x_i> = Invoke <e_i>; [Invoke <e_{i+1}>; ..; Invoke <e_{j-1}>;] <x_j>;

3. Lygčių tvarka sistemoje nėra svarbi.

2.2. Specifikacijos transformavimas

Grafais grįstas proceso modelis turi tam tikras struktūras, kuriuose galimi nestruktūrizuoti cikliniai perėjimai. [KH04] darbe buvo pateiktas transformavimo metodas skirtas tokioms nestruktūrizuotoms struktūroms paversti į struktūrizuotas. Šis metodas grindžiamas kitu metodu, kuris buvo taikomas dar kompiliavimo teorijoje sprendžiant *GO-TO eliminavimo* problemą [Amm92]. Jau net prieš kelis dešimtmečius buvo įrodyta, kad tokius nestruktūrizuotus perėjimus galima transformuoti į funkciškai ekvivalenčią programą panaudojant ciklų bei sąlygų struktūras [BJ66]. Pasinaudojant [KH04] pateiktu metodu šiame darbe pateiksime jo pagrindu sukurtą patobulintą metodą skirtą BPD diagramoms užrašytomis *tęstinumo* semantika transformuoti į tokį pavidalą, kuris yra nesunkiai konvertuojamas į WF sekų modelį. Šis metodas buvo papildytas dvejomis taisyklėmis bei įvesti pradiniai reikalavimai, leidžiantys užtikrinti ar transformuota struktūra yra ekvivalenti konkrečioje darbo sekų vykdymo aplinkoje.

2.2.1. Transformavimo taisyklės

BPD specifikacijos transformavimui naudosime šias pagrindines taisykles:

- sukeitimo taisyklė;
- sąlygų perskirstymo taisyklė;
- sąlygų išskirstymo taisyklė;
- sąlygų apjungimo taisyklė;
- faktorizavimo taisyklė;
- derekursyvacijos taisyklė.

Toliau šios transformavimo taisyklės bus aprašytos detaliau.

Sukeitimo taisyklė

Sukeitimo taisyklė sumažina kintamųjų skaičių lygčių sistemoje. Paprasčiausiu atveju, kintamieji sukeičiami tokiose vietose, kur yra proceso dalis einanti sekos pavidalu (nuosekliai, be išsišakojimų). Taisyklė nusako, kad lygties dešinėje pusėje esantį kintamąjį x_i galima pakeisti į kitos lygties dešinę išraišką, kuri yra ekvivalenti duotajam kintamajam x_i:

x_0 = Invoke A; $\mathbf{x_1}$; $\Rightarrow$ x_0 = Invoke A;
$\mathbf{x_1}$ = Invoke B; Invoke B;

Sąlygų perskirstymo taisyklė

Sąlygų perskirstymo taisyklė yra skirta sąlyginių išsišakojimų eiliškumo tvarkai pakeisti. Ši taisyklė gali būti įvairių formų, tačiau viena iš dažniausiai naudojamų perskirstymo formų yra ši:

x_0 = If (**c_1**) Begin
    x_1;
ElseIf (**c_2**) Begin
    x_2;
EndIf;

$\Rightarrow$

x_0 = If ($\neg$**c_1** $\wedge$ **c_2**) Begin x_2; EndIf;
If (c_1) Begin x_1; EndIf;

Naudoti šią taisyklę reikia labai atsargiai. Iš pirmo žvilgsnio logiškai atrodantis perskirstymas gali būti teisingas ne visais atvejais, kadangi tam tikras sąlygas gali įtakoti kitos veiklos. Pavyzdžiui, pateiktoje paskutinėje perskirstymo formoje, jei sekos grandinėje einančioje nuo x_2 iki x_0 (neimtinai) yra įvykdomos veiklos, kurios įtakoja sąlygą c_1, perskirstymas tampa neteisingas. Tokioms situacijoms valdyti įvesime naują sąvoką – *transformavimo pradinis reikalavimas*. Transformavimo pradinius reikalavimus aprašinėsime [x_i: $x_j \not\rightarrow s_k$] pavidalu; čia x_i – kintamasis nusakantis vietą, kur turi būti tikrinamas pradinis reikalavimas (lygties kairės pusės kintamasis), o $x_j \not\rightarrow s_k$ užrašas reiškia, kad x_j neįtakoja s_k sąlygos. Kaip pasinaudoti šiais reikalavimais bus aprašyta vėliau.

Sąlygų išskirstymo ir apjungimo taisyklės

Šios taisyklės yra naudojamos *IfElse* struktūrai eliminuoti bei kelioms sąlygoms sujungti į vieną. Bendriniu pavidalu toks perskirstymas atrodo taip:

x_0 = If (**c_1**) Begin
    If (**c_2**) Begin Invoke A; EndIf;
ElseIf (**c_3**) Begin
    If (**c_4**) Begin Invoke **x_1**; EndIf;
EndIf;

$\Rightarrow$

x_0 = If (**c_1** $\wedge$ **c_2**) Begin Invoke A; EndIf;
If ($c_3 \wedge c_4$) Begin x_1; EndIf;

Faktorizavimo taisyklė

Faktorizavimo taisyklė sumažina lygtyje vienodų kintamųjų skaičių. Ši taisyklė yra pritaikoma tokiais atvejais, kai yra sąlyginis išsišakojimas *If-Else*, kur kiekvienoje iš sąlyginių išsišakojimų yra tas pats kintamasis x_i. Kintamasis x_i eliminuojamas iš pirmojo sąlyginio išsišakojimo, o antrame išsišakojime įvedamas naujas sąlygos predikatas. Tokiu būdu sąlygos tampa mažiau priklausomos:

x_0 = invoke D;
    If (c) Begin
        Invoke A; **x_1**;
    ElseIf (d) Begin
        Invoke B; **x_2**;
    EndIf;

$\Rightarrow$

x_0 = invoke D;
    If (c) Begin Invoke A; EndIf;
    If ($\neg$c $\wedge$ d) Begin **B**; EndIf;
    If (c $\vee$ ($\neg$c $\wedge$ d)) Begin **x_1**; EndIf;

Šioje taisyklėje kaip ir sąlygų perskirstymo taisyklėje gali atsirasti transformavimo pradiniai reikalavimai. Pavyzdžiui, pateiktoje formoje turi būti išpildomas [x_0: B $\not\rightarrow$ c] reikalavimas.

Derekursyvacijos taisyklė

Derekursyvacijos taisyklė yra pagrindinė taisyklė skirta rekursyvių kreipinių eliminavimui. Taisyklė yra taikoma tokiais atvejais, kai lygties dešinėje pusėje figūruoja kintamasis x_i sutampantis su kintamuoju esančiu lygties kairėje pusėje. Tokiu atveju lygties dešinėje pusėje esančios išraiškos pakeičiamos į struktūrizuotą ciklo struktūrą:

$\mathbf{x_0}$ =	Invoke A;		$\mathbf{x_0}$ =	Repeat
	If (c) Begin $\mathbf{x_0}$; EndIf;	$\Rightarrow$		Invoke A;
	Invoke B;			While (c);
				Invoke B;

Šios taisyklės negalima taikyti, jei prieš minėtąjį kintamąjį x_i yra bent vienas kintamasis x_j. Tokiu atveju prieš taikant šią taisyklę reikia pasinaudoti kitomis auksčiau aprašytomis taisyklėmis.

2.2.2. Transformavimas

Pasinaudojant aukščiau aprašytomis taisyklėmis transformacijai atlikti naudosime tokį algoritmą:

1. Išskaidome procesą į paprocesius ir pažymime juos sisteminiais kintamaisiais Ci.
2. Kiekvienam procesui ir paprocesiui sudarome lygčių sistemas ir sprendžiame jas nepriklausomai.
3. Pasinaudodami anksčiau aprašytomis transformavimo taisyklėmis išsprendžiame sudarytas lygčių sistemas ir posistemes:
    a. Pasirenkame lygčių sistemai pritaikomą taisyklę ir pritaikome ją. Jei nerandame pritaikomos taisyklės, grįžtame vieną žingsnį atgal ir bandome pritaikyti kitokią taisyklę negu buvo pritaikyta anksčiau.
    b. Žingsnį (a.) kartojame tol, kol sistema/posistemė netampa išspręsta. Lygčių sistema būna išspręsta, kai lieka tik viena lygtis x_i = <*išraiška*>. <*Išraiška*> yra sistemos sprendinys.
4. Pagrindinio proceso gautoje lygtyje vietoj C_i paprocesių sisteminių kintamųjų įrašome atitinkamas išspręstų sistemų sprendinius. Tokiu būdu iš visų pradinių lygčių sistemų gauname tik vieną x_i = <*išraiška*> pavidalo lygtį, kur lygties dešinėje išraiškoje nėra nei vieno kintamojo x_j bei sisteminio kintamojo C_j. Gautos lygties dešinę išraišką vadinsime sistemos galutiniu sprendiniu.

Transformuojant procesą M naudojant pateiktą algoritmą galima gauti skirtingus ekvivalenčius procesus N_i, kadangi algoritmo žingsnyje 3.**a** pasirenkamų pritaikomų taisyklių tvarka nėra reglamentuojama. Savaime aišku, kad egzistuoja tokia pritaikomų taisyklių kombinacija, kad transformuotas procesas N_j būna optimaliausias.

2.2.3. Sprendimo optimizavimas

Transformavimo metu gautą sprendinį prieš verčiant į WF modelį galima irgi optimizuoti. Optimizavimo metu sumažinamas išraiškoje esančių elementų skaičius bei prastinamos sąlyginės išraiškos.

Galimi įvairaus pobūdžio loginių išraiškų optimizavimai paremti loginėmis išvadomis. Pagrindines iš jų galima apibendrinti tokiomis situacijomis:

- jei loginėje išraiškoje turime $s_1 \vee s_2 \equiv$ true arba $s_1 \vee \neg s_1$, tokiu atveju sąlyga eliminuojama, pavyzdžiui:
  - If (AB $\vee$ AC) Begin Invoke D; EndIf; galima užrašyti kaip Invoke D;, jei iš elemento A į D egzistuoja tik vieninteliai AB ir AC išsišakojimai;
  - If ((AB $\vee \neg$AB) $\wedge$ DC) Begin Invoke C; EndIf; galima užrašyti kaip If (DC) Begin Invoke C; EndIf;
- jei loginėje išraiškoje turime $s_1 \wedge s_2 \equiv$ false arba $s_1 \wedge \neg s_1$, tokiu atveju sąlyga eliminuojama kartu su jos išraiška. Pavyzdžiui:
  - If (AB $\wedge$ AC) Begin Invoke D; EndIf; galima praleisti, jei iš elemento A į D egzistuoja tik vieninteliai AB ir AC išsišakojimai;
- išraiškos pradžioje galima praleisti Invoke Begin;, o pabaigoje – Invoke End;
- lygties pabaigoje galima praleisti If (s_1) Begin Invoke End;EndIf; , kadangi logiškai šis sąlyginis patikrinimas neturi jokios prasmės, kadangi pasibaigus sakiniams procesas užbaigiamas bet kokiu atveju.

2.3. Sprendimo konvertavimas į WF

Išsprendus pagrindinę lygčių sistemą užrašytą *tęstinumo* semantika ir suoptimizavus galutinį sprendinį, galima prieiti prie paskutinio transformavimo žingsnio, kuriuo metu gauta išraiška būtų pakeista į WF sekų modelį. Tam tikslui kiekvieną išraiškos struktūros elementą reikia pakeisti į atitinkamą WF sekų modelio XAML pavidalą, pasinaudojant sekančiomis taisyklėmis:

- darbo seką apgaubiame į *SequentialWorkflowActivity* konteinerį:
  <SequentialWorkflowActivity> ... </SequentialWorkflowActivity>
- elementai, kurie yra vykdomi nuosekliai apgaubiame į *SequenceActivity* konteinerį:
  <SequenceActivity> ...</SequenceActivity>
- veiklą aprašome kaip atitinkamą <Activity> komponentą:
  <Activity name = “pavadinimas“/>

Pastaba. Veikla gali būti pakeista į kito tipo struktūrą pagal realizaciją.

- *Repeat* struktūrą aprašome kaip *RepeatActivity* konteinerį:

  <RepeatActivity condition="sąlyga"> ... </RepeatActivity>
- *Xor* struktūrą arba sąlyginį išsišakojimą aprašome kaip *SwitchActivity* ir *CaseActivity* arba *If-Else* komponentus:

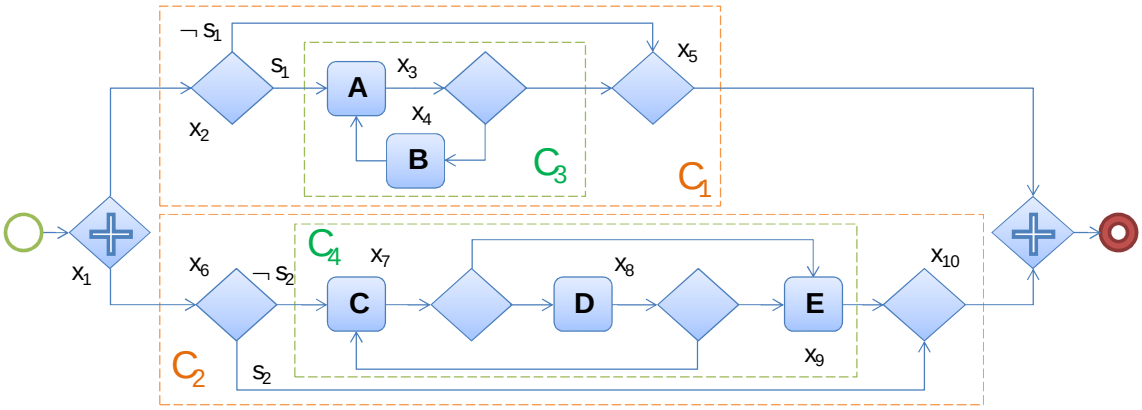
  <SwitchActivity>
  <CaseActivity condition="condition"> ... </CaseActivity>
  ....
  <CaseActivity condition="condition"> ... </CaseActivity>
  </SwitchActivity>

arba

  <IfElseActivity>
  <IfElseBranchActivity condition="condition"> .... </IfElseBranchActivity>
  <IfElseBranchActivity> ... </IfElseBranchActivity>
  </IfElseActivity>

- *Parallel* struktūrą pakeičiama į *ParallelActivity* konteinerį:

  <ParallelActivity> ... </ParallelActivity>



13 pav . Pavyzdinis procesas

2.4. Pavyzdinis transformavimas

Šiame skyrelyje pagal anksčiau pateiktą transformavimo algoritmą transformuosime pavyzdinį procesą pavaizduotą 13-me paveikslėlyje. Duotasis procesas yra bazinė validi BPD. Komponentus išdėstytus šioje diagramoje reikia išskaidyti į paprocesius. Pagrindinį procesą išskaidome į du paprocesius C_1, C_2, kadangi turime AND tipo vartus. Kiekvienoje AND srityje yra po viena XOR sritys, turinti po du išsišakojimus. Sąlyginius išsišakojimus neturinčius jokių komponentų atskirai nenagrinėsime. Tokiu būdu gauname dar du paprocesius C_3, C_4 (13-as paveikslėlis).

Kiekvieną elementą $e_i \in (A, S, F)$ ir $e_j \in (G_{XOR}, G'_{XOR})$ pažymime kintamaisiais (pastaruosius pagal poreikį). e_j elementų kintamieji atstos paprocesių pradžios ir pabaigos kintamuosius. Toliau pereiname prie procesų ir paprocesių atitinkamų lygčių sistemų sudarymo:

1. Sudarome lygčių sistemą pagrindiniam procesui C:

(0) x_1 = Invoke Begin;
Parallel
C_1; C_2;
EndParallel;
Invoke End;

2. Sudarome lygčių sistemą paprocesiui C_1:

(1) x_2 = Invoke Begin;
XorBegin
If (s_1) Begin C_3; EndIf;
EndXor;
Invoke End;

3. Sudarome lygčių sistemą paprocesiui C_2:

(2) x_6 = Invoke Begin;
XorBegin
If (s_2) Begin C_4; EndIf;
EndXor;
Invoke End;

4. Sudarome lygčių sistemą paprocesiui C_3:

(3) x_2 = Invoke Begin; x_3;
(4) x_3 = Invoke A;
If (AB) Begin x_4 ElseIf Begin x_5 EndIf;
(5) x_4 = Invoke B; x_3;
(6) x_5 = Invoke End;

Pastaba: AB yra dirbtinis sąlygų pavadinimas. AB pažymime sąlygą einančią iš mazgo A į B;

5. Sudarome lygčių sistemą paprocesiui C_4:

(7) x_6 = Invoke Begin; x_7;
(8) x_7 = Invoke C;
If (CD) Begin x_8; ElseIf Begin x_9; EndIf;
(9) x_8 = Invoke D;
If (DE) Begin x_9; ElseIf Begin x_7; EndIf;
(10) x_9 = Invoke E; x_{10};
(11) x_{10} = Invoke End;

Pastaba: CD ir DE yra dirbtiniai sąlygų pavadinimai.

Dabar sprendžiame sudarytas sistemas/posistemes:

1. Posistemių C_1 ir C_2 nesprendžiame, nes jos yra sdarytos iš vienos lygties.
2. Sprendžiame posistemę C_3.

Pasinaudojame *sukeitimo* taisykle ir į lygtį (4) vietoje kintamųjų x_4 ir x_5 atitinkamai įstatome lygtis (5) ir (6):

(12) x_3 = Invoke A;

If (AB) Begin Invoke B; x_3;
ElseIf Begin Invoke End; EndIf;

Pasinaudojame *faktorizavimo* taisykle ir (12) lygtyje išskaidome sąlygas:

(13) x_3 = Invoke A;
If (AB) Begin Invoke B; EndIf;
If (AB) Begin x_3; EndIf;
If (¬AB) Begin Invoke End; EndIf;

(13) lygtyje pasinaudojame *derekursyvacijos* taisykle ir eliminuojame kintamąjį x_3. Į (3) lygtį vietoje x_3 įstatome gautą lygtį. Tokiu būdu gauname lygtį:

(14) x_2 = Invoke Begin;
Repeat
Invoke A;
If (AB) Begin Invoke B; EndIf;
While (AB);
If (¬AB) Begin Invoke End; EndIf;

Posistemė išspręsta.

3. Sprendžiame posistemę C_4

Į (8) lygtį vietoje kintamojo x_8 įstatome (9) lygtį pasinaudodami *sukeitimo* taisykle. (10) lygtyje pasinaudodami ta pačia taisykle vietoje kintamojo x_{10} įstatome (11) lygtį:

(15) x_7 = Invoke C;
If (CD) Begin
Invoke D;
If (DE) Begin x_9; ElseIf Begin x_7; EndIf;
Else Begin x_9; EndIf;
(16) x_9 = Invoke E; Invoke End;

Pasinaudojame *sąlygų išskirstymo ir apjungimo* taisyklėmis ir perskirstome sąlygas (15) lygtyje:

(17) x_7 = Invoke C;
If (CD) Begin Invoke D; EndIf;
If (CD ∧ DE) Begin x_9; EndIf;
If (CD ∧ ¬DE) Begin x_7; EndIf;
If (¬CD) Begin x_9; EndIf;

Pasinaudojame *perskirstymo* taisykle (17) lygtyje, bei optimizuojame lygtį apjungdami (CD ∧ DE) ir (¬CD) sąlygas:

(18) x_7 = Invoke C;
If (CD) Begin Invoke D; EndIf;
If (CD ∧ ¬DE) Begin x_7; EndIf;
If ((CD ∧ DE) ∨ ¬CD) Begin x_9; EndIf;

(18) lygtyje pasinaudojame *derekursyvacijos* taisykle ir eliminuojame kintamąjį x_7. Vietoje x_9 ir x_{10} įstatome (16) ir (11) lygtis pasinaudodami *sukeitimo* taisykle. Pasinaudojus ta pačia taisykle į (7) lygtį įstatome gautą lygtį. Tokiu būdu gauname galutinę lygtį:

(19) x_6 = Invoke Begin;
Repeat
Invoke C;
If (CD) Begin Invoke D; EndIf;
While (CD $\wedge$ $\neg$DE);
If ((CD $\wedge$ DE) $\vee$ $\neg$CD) Begin Invoke E; EndIf;
Invoke End;

4. Išsprendus lygčių sistemas, gautose lygtyse praleidžiame *Invoke Begin;* bei *Invoke End;* .
5. Vietoje sisteminių kintamųjų C_1 , C_2 , C_3 ir C_4 įrašome gautas lygtis. Tokiu būdu gauname sprendinį:

```
Parallel
    Branch
        XorBegin
            If (s1) Begin
                Repeat
                    Invoke A;
                    If (AB) Begin Invoke B; EndIf;
                While (AB);
            EndIf;
        EndXor;
    EndBranch;
    Branch
        XorBegin
            If (s2) Begin
                Repeat
                    Invoke C;
                    If (CD) Begin Invoke D; EndIf;
                While (CD ∧ ¬DE);
                If ((CD ∧ DE) ∨ ¬CD) Begin Invoke E; EndIf;
            EndIf;
        EndXor;
    EndBranch;
EndParallel;
```

Gautą sprendinį konvertuojame į XAML pavidalą:

```
<SequentialWorkflowActivity>
    <ParallelActivity>
        <IfElseActivity>
            <IfElseBranchActivity condition="Sal">
                <RepeatActivity condition="AB">
                    <Activity name="A">
                    <IfElseActivity>
                        <IfElseBranchActivity condition="AB">
                            <Activity name="B">
                        </IfElseBranchActivity>
                    </IfElseActivity>
                </RepeatActivity>
            </IfElseBranchActivity>
        </IfElseActivity>
        <IfElseActivity>
            <IfElseBranchActivity condition="Sal">
                <RepeatActivity condition="CD ∧ ¬DE">
                    <Activity name="C">
                    <IfElseActivity>
                        <IfElseBranchActivity condition="CD">
                            <Activity name="D">
                        <IfElseBranchActivity>
                    </IfElseActivity>
                </RepeatActivity>
                <IfElseActivity>
                    <IfElseBranchActivity condition="(CD ∧ DE) ∨ ¬CD">
                        <Activity name="E">
                    </IfElseBranchActivity>
                </IfElseActivity>
            </IfElseBranchActivity>
        </IfElseActivity>
    </ParallelActivity>
</SequentialWorkflowActivity>
```

2.5. Transformavimo programa

2.5.1. Programos aprašymas

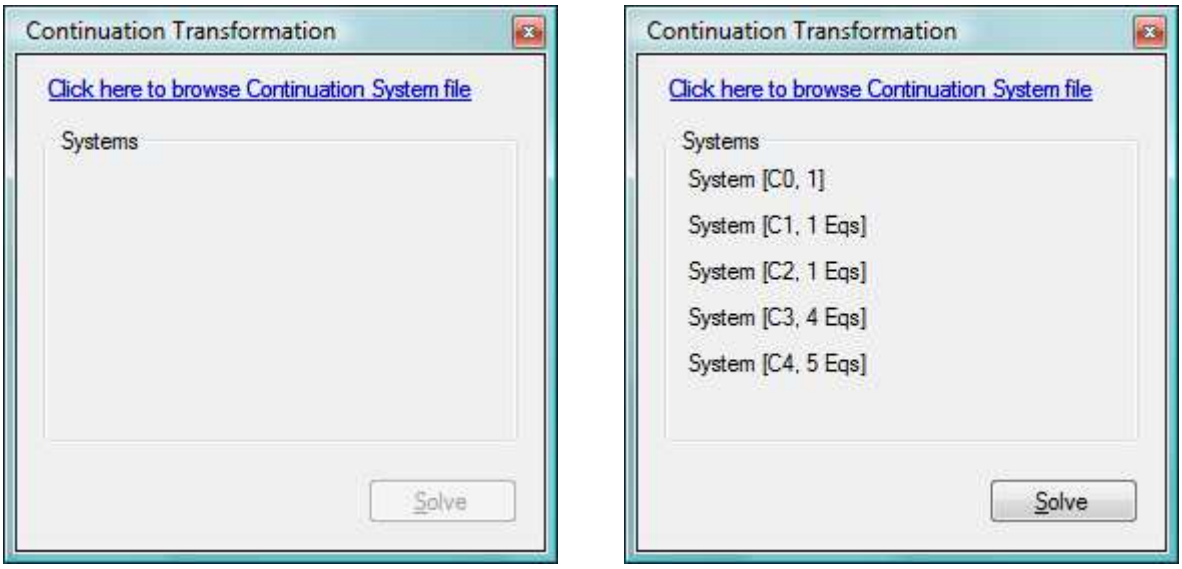
Aprašytam transformavimo algoritmui patikrinti bei eksperimentinėms darbo sekų transformacijoms atlikti buvo sukurta speciali programa. Programos įvestis – darbo sekos sistema aprašyta išplėstine tęstinumo semantika; rezultatas – sistemos nuoseklus sprendimas bei pilnai išspręsta sistema, kurios sprendinys yra lengvai transformuojamas į WF pavidalą.

Darbe aprašytas algoritmas skirtas aprašytai darbo sekai išspręsti nereglamentuoja taisyklių pasirinkimo tvarkos. Kuriant programa buvo eksperimentuojama su įvairiais taisyklių tvarkos pasirinkimo algoritmais. Galiausiai programoje buvo pasirinktas paprastas taisyklių pasirinkimo algoritmas:

- taikyti *pakeitimo* taisyklę tol, kol ji yra pritaikoma;
- taikyti *faktorizavimo* taisyklę tol, kol ji yra pritaikoma;
- taikyti *sąlygų perskirstymo* taisyklę tol, kol ji yra pritaikoma;
- taikyti *derekursyvacijos* taisyklę tol, kol ji yra pritaikoma;
- taikyti *sąlygų išskirstymo* ir *apjungimo* taisykles;
- grįžti prie pirmo žingsnio, jei sistemoje yra daugiau negu viena lygtis ir buvo sėkmingai pritaikyta bent viena taisyklė. Jei nei viena taisyklė nebuvo pritaikyta ir sistemoje yra daugiau negu viena lygtis – sistema nėra išsprendžiama.

2.5.2. Programos naudojimas

Programa buvo sukurta su C# programavimo kalba, todėl jos vykdymui yra būtina MS Windows šeimos operacinė sistema bei .NET 2.0 (arba vėlesnis) vykdymo karkasas.



a) **b)**

14 pav. Programos grafinė sąsaja

Paleidus programą, reikia paspausti ant nuorodos „Click here to browse Continuation System file“ (14 a pav.). Atsidariusiame lange reikia pasirinkti failą, kuriame yra aprašyta darbo sekos sistema. Failo struktūra:

1. Užrašas [SystemID=<ID>] reiškia, kad prasideda sistemos, kurios identifikatorius yra ID, aprašas. Sistema privalo turėti bent vieną lygtį.

2. Užrašas [Equation] reiškia, kad prasideda einamosios sistemos lygties aprašas. Lygties aprašas laikomas užbaigtu, jei: sutinkamas sekantis [Equation] raktažodis, [SystemID=<ID>] raktažodis, ar pasibaigia failas.

Pavyzdinis darbo sekos aprašas:

```
[SystemID=C1]
[Equation]
x1 = Invoke Begin; x2;
[Equation]
x2 = System C2; Invoke End;
[SystemID=C2]
[Equation]
x1 = Invoke Begin; Repeat Invoke B; While (s1); Invoke End;
```

Sistemos lygtys yra aprašomos taip, kaip buvo paaiškinta šiame darbe anksčiau.

Pasirinkus korektišką sistemos aprašo failą, programos formoje, srityje *Systems* bus parodytos visos sistemos/posistemės su identifikatoriais ir lygčių skaičiumi (14-as b pav.). Po mygtuko *Solve* paspaudimo, sistema bus išspręsta ir lange pasirodys teksto redaktoriuje atidarytas sprendimo failas, kuriame bus pateiktas nuoseklus sistemos sprendimas su minimaliais paaiškinimais bei galutinis sistemos sprendinys, jei sistema yra išsprendžiama.

2.5.3. Rezultatai

Tęstinumo semantika buvo aprašytos kelios sudėtingos darbo sekos, kuriose buvo naudojami sudėtingi cikliniai perėjimai, bei kitos *BPN* struktūros (tarp pavyzdžių buvo ir šiame darbe detaliai aprašytas pavyzdinis transformavimas). Programos pagalba buvo nagrinėta, kaip sprendžiamos aprašytos sudėtingos darbo sekų lygtys. Tokiu lygčių sprendimas „rankiniu“ būdu užtruktų daug laiko, o nagrinėti programos pateiktą lygties sprendimą yra žymiai lengviau. Programos rezultatai parodė, kad šis algoritmas yra korektiškas.

2.6. Gautos struktūros validavimas

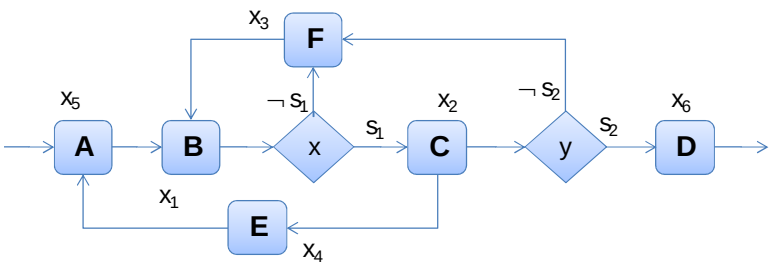
Transformavus BPD į atitinkamą WF sekų modelio XAML pavidalo struktūrą, bendruoju atveju gausime ekvivalentųjį procesą pradiniam. Bet šis ekvivalentumas yra teisingas, deja, ne visais atvejais. Jei procesą nagrinėsime konkrečios dokumentų valdymo sistemos arba kitos darbo sekų vykdymo sistemos kontekste, turėsime atsižvelgti į tai, kad tam tikrų veiklų realizavimas vykdymo sistemoje gali įtakoti mūsų transformuotos darbo sekos teisingumą. Vadinasi transformacijos teisingumas priklauso nuo pačios sistemos realizuojančios darbo sekų vykdymą. Šiame skyrelyje aptarsime kaip nustatyti ar transformuota darbo seka yra ekvivalenti pradiniai konkrečioje sistemoje, kuri vykdo tą darbo seką. Tokioms situacijoms valdyti anksčiau

buvo įvesta transformavimo pradinio reikalavimo sąvoka, kurios bendrinis pavidalas yra: [x_i: $x_j \not\rightarrow s_k$]. Tokie reikalavimai atsiranda tik faktorizavimo ir sąlyginio išskirstymo taisyklėse ir jų tenkinimas reiškia, kad transformuota darbo seka yra ekvivalenti pradiniai duotoje vykdymo aplinkoje.

Pasinaudojus sekančiu algoritmu galima patikrinti ar konkretus pradinis transformavimo reikalavimas [x_i: $x_j \not\rightarrow s_k$] yra tenkinamas konkrečioje vykdymo aplinkoje:

- surasti visus įmanomus grafo kelius $t_k \in (t_i, ..., t_n)$ nuo x_j iki x_i (neimtinai);
- jei $\forall t_k$ nei viena veikla $a_i \in t_k$ neįtakoja s_k sąlygos, tokiu atveju reikalavimas yra tenkinamas.

Taigi, transformuotas į WF sekų modelį procesas W yra ekvivalentus pradiniam procesui B $\in$ BPD tada ir tiktai tada, kai proceso vykdymo aplinkoje yra tenkinami visi transformavimo pradiniai reikalavimai.



15 pav. Pavyzdinis procesas

Kad lengviau būtų suvokti pradinių reikalavimų tenkinimą panagrinėsime paprastą pavyzdį. Tarkime turimą darbo seką X pavaizduotą 8-me paveikslėlyje. Transformavome ją į Y. Tarkime transformavimo metu buvo nustatyti sekantys pradiniai reikalavimai vykdomajai darbo sekai:

(0) [x_1: $x_2 \not\rightarrow s_1$]
(1) [x_2: $x_3 \not\rightarrow s_2$]

Toliau ieškome atitinkamų grafo kelių, bei juose pasitaikančių veiklų. Iš x_2 į x_1 egzistuoja du skirtingi keliai: $r_1 = (x_2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_1)$, $r_2 = (x_2 \rightarrow x_3 \rightarrow x_1)$. Į kelią r_1 įeina C, E, A veiklos, į r_2 - C, F.

Iš x_3 į x_2 egzistuoja tik vienas kelias $r_3 = (x_3 \rightarrow x_1)$. Į kelią r_3 įeina F, B veiklos.

Taigi, kad Y būtų ekvivalenti X vykdomojoje aplinkoje, turi būti tenkinami sekantys reikalavimai:

- vykdomos veiklos C, E, A, F negali įtakoti sąlygos s_1;
- vykdomos veiklos F, B negali įtakoti sąlygos s_2.

Rezultatai

Apibendrinus, šiame darbe buvo atlikti šie darbai ir pasiekti planuoti rezultatai:

- išrinkti BPMN žymėjimai skirti DVS sistemoms:
- sukurta specifikavimo kalba skirta BPMN žymėjimų formaliam specifikavimui bei aprašytas algoritmas, kaip tokius žymėjimus specifikuoti;
- sukurtas algoritmas leidžiantis transformuoti formaliai aprašytą BPMN modelį į pavidalą skirtą paprastam konvertavimui į WF sekų modelį;
- aprašyta, kaip galima supaprastinti transformuotą modelį pasinaudojant loginėmis išvadomis;
- sukurtas algoritmas leidžiantis patikrinti ar transformuotas modelis yra ekvivalentus pradiniam pasirinktoje darbo sekų vykdymo aplinkoje;
- sukurta speciali programa eksperimentinėms transformacijoms atlikti.

Išvados

Darbo pradžioje suformuotas pagrindinis tikslas buvo pasiektas, tai yra, buvo sukurtas dokumentų valdymo sistemų darbo sekų transformavimo algoritmas, aprašantis, kaip transformuoti BPMN į WF sekų modelį. Kadangi buvo nagrinėjami tik DVS sistemoms pritaikomi procesai, atsižvelgiant į tai, buvo išrinktos ir formalizuotos tik tam tikri BPMN žymėjimai skirti būtent nagrinėjamai sričiai bei įvesti papildomi apribojimai užtikrinantys transformuojamų darbo sekų apibrėžtumą vykdymo metu.

Darbe pateiktas algoritmas yra grindžiamas keturiomis pagrindinėmis transformavimo taisyklėmis, kurių naudojimo tvarka nėra griežtai reglamentuojama. Taigi transformuojant pasirinktą darbo seką, pasinaudojant pateiktu algoritmu, galima gauti skirtingos struktūros ekvivalenčius procesus. Šiame darbe nebuvo nagrinėjama, kaip pasirinkti algoritmo taisyklių tvarką, kad transformuojamas modelis gautųsi optimalus.

Nagrinėjant sukurto algoritmo rezultatus buvo nustatyta, kad transformuoti modeliai yra ekvivalentūs ne visose darbo sekų vykdymo aplinkose. Todėl šiame darbe buvo sukurtas algoritmas, leidžiantis pagal aprašytas sąlygų ir veiklų priklausomybes patikrinti ar transformuotas modelis yra tapatus pasirinktoje darbo sekų vykdymo aplinkoje.

Literatūros sąrašas

[ACW05] P. Andrew, J. Conard, S. Woodgate, J. Flanders, G. Hatoun, I. Hilerio, P. Indurkar, D. Pilarinos, J. Willis. Presenting Windows Workflow Foundation, Beta Edition. Sams, 2005. 312p.

[AH02] W. van der Aalst, K. van Hee. Workflow management. MIT Press, 2002. 375p.

[Amm92] Z. Ammarguellat, A control-flow normalization algorithm and its complexity. Software Engineering. 1992.

[BJ66] C. Böhm, G. Jacopini. Flow diagrams, Turing machines and languages with only two formation rules**.** Communications of the ACM, 1966. 366-371p.

[BPM04a] BPMI.org. Business process modeling notation. Version 1.0. www.bpmi.org, 2004. 296p.

[BPM04b] BPMI.org. Business Process Modeling Notation Specification. www.bpmi.org, 2004. 308p.

[Gao06] Y. Gao. BPMN - BPEL Transformation and Round Trip Engineering. eClarus software, 2006. 6p.

[Hol95] D. Hollingsworth. The Workflow Reference Model. Workflow Managment Coalition. http://www.wfmc.org/standards/docs/tc003v11.pdf. 210KB, 1995.

[KH04] J. Koehler, R. Hauser. Untangling unstructured cyclic flows – a solution based on continuations. IBM Zurich Research Laboratory, 2004. 18p.

[Kit07] T. Kitta. Professional Windows Workflow Foundion. Wrox Press, 2007. 432p.

[MSD07a] MSDN. Windows Workflow Foundation Overview. http://msdn2.microsoft.com/en-us/library/ms734631.aspx. 48K, 2007.

[MSD07b] MSDN. XAML Syntax Terminology. http://msdn2.microsoft.com/en-us/library/ms788723.aspx. 150K, 2007.

[MSD07c] MSDN. XAML Overview. http://msdn2.microsoft.com/en-us/library/ms752059.aspx. 128K, 2007.

[OAD06] C. Ouyang, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede. Translating BPMN to BPEL. BPMcenter.org, 2006. 22p.

[ODB05] C. Ouyang, M. Dumas, S. Breutel, A. H. M. ter Hofstede. Translating standard process models to BPEL. BPMcenter.org, 2005. 17p.

[OR03] M. Owen, J. Raj. BPMN and Business Process Management. Introduction to the New Business Process Modeling Standard. www.popkin.com, 2003. 27p.

[Rey98] J. Reynolds. The discovers of continuations. LISP and Symbolic Computation. 1998.

[Whi04a] S. A. White. Introducing to BPMN. IBM, 2004. 11p.

[Whi04b] S. A. White. Process modeling notations and workflow patterns. IBM, 2004. 25p.