

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Baigiamasis bakalauro darbas

**Dalykinės srities modelio transformavimas į UML užduočių
diagramas**

(Deriving use cases from business process)

Atliko: 4 kurso 1 grupės studentas

Aleksandras Sivkovas

(parašas)

Darbo vadovas:

Prof. dr. Saulius Gudas

(parašas)

Recenzentas:

prof. habil. dr. Vardaitis Pavardaitis

(parašas)

Vilnius
2018

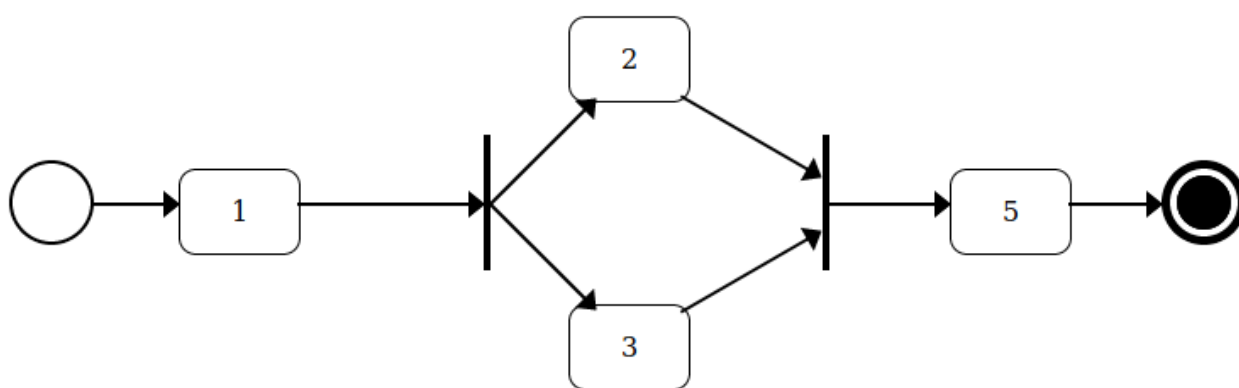
Turinys

Įvadas	2
1. BPMN diagrama	5
2. Detalizuotas vertės grandinės modelis	8
3. Užduočių diagrama	10
4. UML diagramų transformavimo algoritmai	11
4.1. Algoritmas BPMN modeliui transformuoti į Užduočių diagramą	11
5. Užduočių diagramos išvedimas iš BPMN modelio	12
5.1. Ryšiai tarp BPMN ir užduočių diagramų	12
5.2. Ryšių tarp diagramų panaudojimas transformacijai	12
6. Programa BPMN transformacijai į užduočių diagramą	18
Išvados	19
Conclusions	20
Literatūra	21
Santrumpos	22
Priedas Nr.1	
Priedas Nr.2	

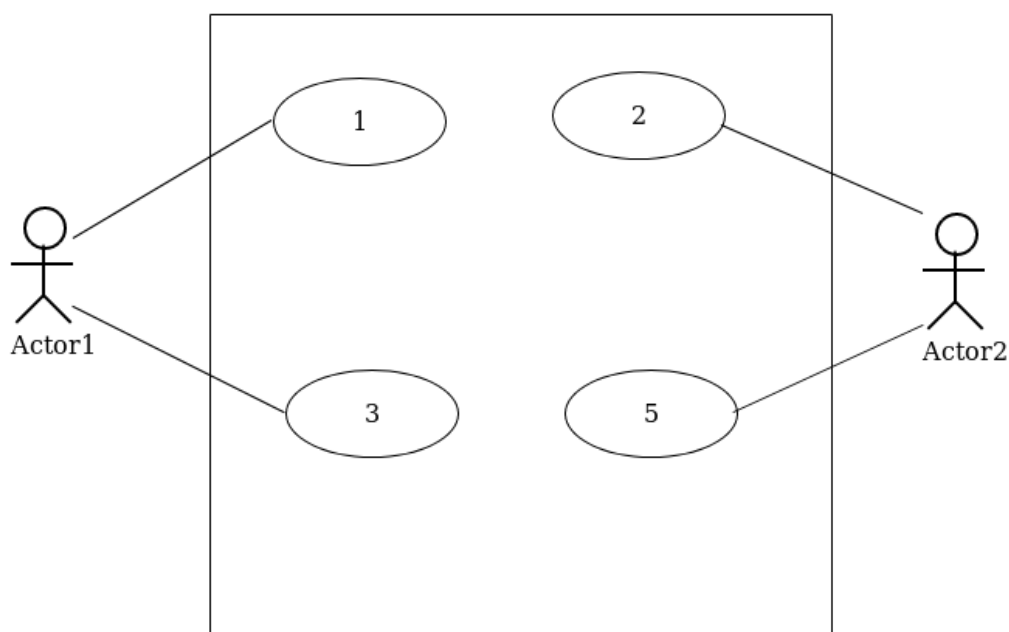
Įvadas

Reikalavimų inžinerija yra sudėtinga programų kūrimo dalis. Proceso sudėtingumas dažnai tampa klaidų priežastimi. Čia atsiradusios klaidos sunkiai aptinkamos ir sukelia brangiai kainuojančias pasekmes, nes sekančiuose etapuose bus kuriama neteisingai apibrėžta programa. Norint išvengti klaidų galima kai kurias proceso veiklas automatizuoti. Nors yra daug modeliavimo įrankių, juose trūksta automatinio vieno modelio generavimo iš kito.

Projektuojant sistemą gali atsitikti toks atvejis (2 pav.). Čia pavaizduota, kad analitikas aptiko kokias funkcijas reikia įgyvendinti. Analizuodamas 1 pav. pavaizduotą veiklą jis aptiko vartojimo atvejus pažymėtus 1, 2, 3, 5. Taigi nuspręsta kurti sistemą pavaizduotą 2 pav.

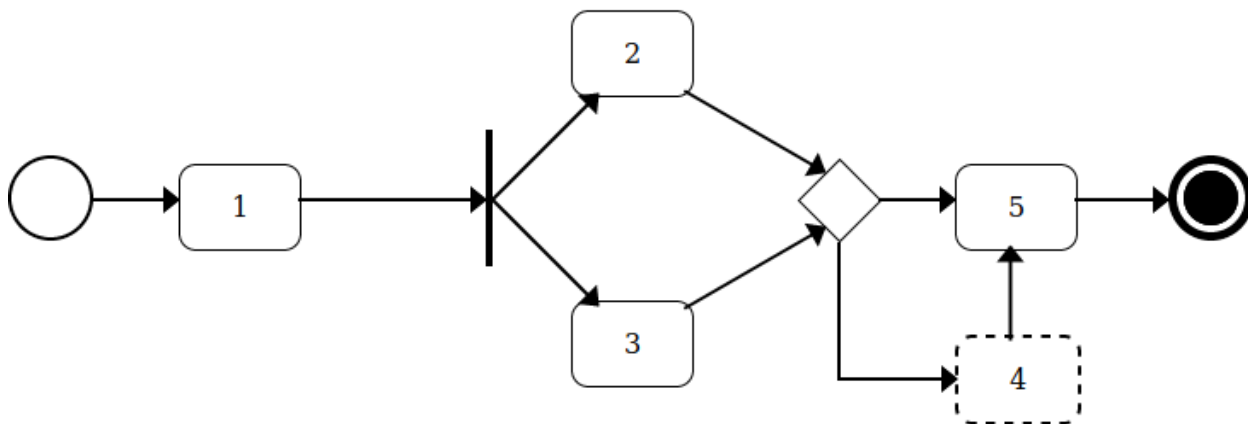


1 pav. Veiklos modelio pavyzdys

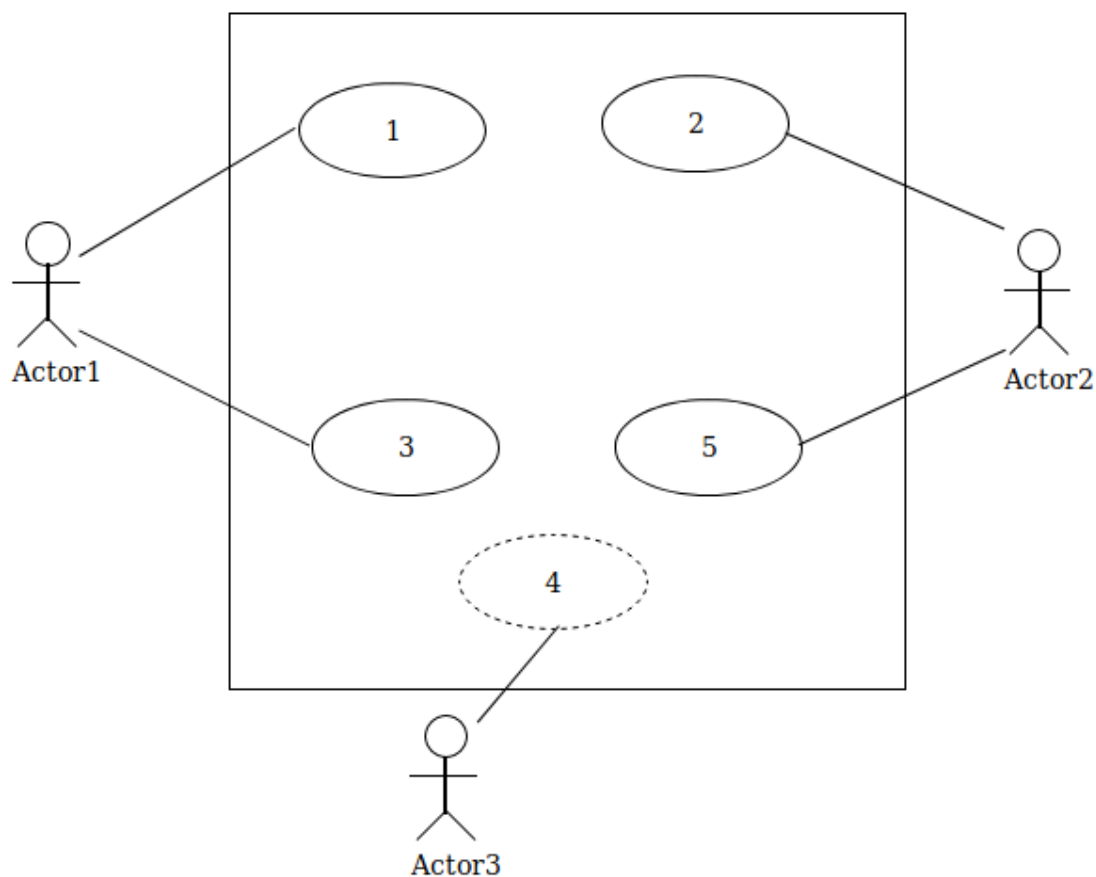


2 pav. Vartojimo atvejų diagramos sukurtos pagal 1 pav. veiklą pavyzdys

Bet vėliau paaiškėjo, kad norint įgyvendinti veiklą pažmėtą 5 kartais reikia duomenų iš 4 veiklos pažymėtos punktyrine linija (3 pav.). Taigi vartojimo atvejų diagramoje trūko 4 vartojimo atvejo (4 pav.). Analitikas to nepastebėjo ir įvyko projektavimo klaida apie kurią niekas nesužinojo.



3 pav. Pataisyto veiklos modelio pavyzdys



4 pav. Vartojimo atvejų diagramos sukurtos pagal 3 pav. pavyzdys

Sumažinti klaidų kiekį galima užrašant turimus duomenis ir automatiškai generuojant mode-

lius. Generavimo įrankis galėtų patikrinti ar įvesties duomenys atitinka keliamus reikalavimus. Taip bus parodyti netikslumai ir analitikas galės pakoreguoti modelį.

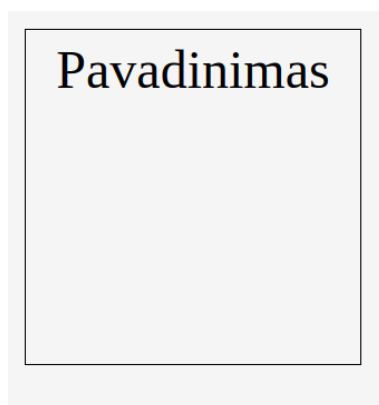
Šio darbo tikslas – sukurti algoritmą **BPMN** modelio transformacijai į **užduočių diagramas** ir įgyvendinti programos prototipą. **Užduočių diagramos** yra svarbi reikalavimų inžinerijos dalis, kadangi ji apibrėžia naudotojo reikalavimus. Įmonės dažniausiai žino kaip ir kokias veiklas jos vykdo. Verslo procesą galima apibrėžti **BPMN** diagramomis. Bet ne viską, kas yra **BPMN** modelyje, galima perkelti į **užduočių diagramą**, todėl darbe bus apibrėžtas modifikuotas **BPMN** modelis, kuriame bus vaizduojama tik algoritmui aktuali informacija. Taip pat gali tekti pridėti papildomų atributų, kurie padės pasiekti tikslesnius rezultatus. Čia bus tiriamas **BPMN** modelio transformacijos į **užduočių diagramas** algoritmas.

Siekiami rezultatai yra:

1. Algoritmas galintis transformuoti **BPMN** modelį į **užduočių diagramą**(angl. Use case diagram).
2. Programa demonstruojanti algoritmo veikimą.

1. BPMN diagrama

BPMN specifikacija leidžia atvaizduoti gana nemažai verslo proceso atributų [Obj11]. Bet šiame darbe ji bus nagrinėjama tik kaip įvesties duomenų formatas, naudojamas apibrėžti informaciją pagal kurią bus kuriama **užduočių diagrama**. Taigi daugelį **BPMN** komponentų galima tiesiog ignoruoti, nes jie neturi jokios įtakos algoritmo vykdymo rezultatui. Norint pabrėžti svarbią informaciją, darbe bus tiriamos tik tos **BPMN** savybės kurios gali įtakoti algoritmo vykdymo rezultatą. Toliau pateikiami pagrindiniai BPMN komponentai.



5 pav. Juostos žymėjimo pavyzdys

Juosta (pool) žymi diagramos dalyvį. Šio komponento paskirtis yra parodyti už kokias veiklas ir koks vykdytojas yra atsakingas. Juosta gali būti skaidoma smulkiau norint konkrečiau nurodyti vykdytojų grupes ir jų pareigas. Juosta žymima apibraukiant tam tikrą sritį (5 pav.). Viduje yra vieta komponentams už kurių atlikimą atsakingas dalyvis.



6 pav. Įvykio žymėjimo pavyzdys

Įvykis (event) žymi, kad įvyko kažkas, kas įtakojo proceso būseną. Šis komponentas dažniausiai turi priežastį (trigger) dėl ko jis įvyko ir pasekmės (result). Specifikacijoje įvykiai skirstomi į tris tipus: pradžios, pabaigos ir tarpinius. Žymėjimas yra tuščias apskritimas (6 pav.), viduje paliekant vietos tipo konkretizacijai.



Veiklos pavadinimas

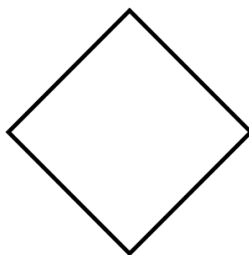
7 pav. Veiklos žymėjimo pavyzdys

Veikla (Activity) yra darbas atliekamas organizacijos procesuose. Ji gali būti atominė ir turėti tik pavadinimą arba skaidoma labiau, tokiu būdu tapdama subprocesu. Visais atvejais žymėjimas yra stačiakampis su užapvalintais kampais (7 pav.).



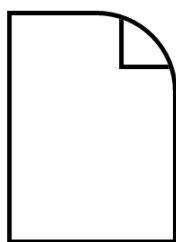
8 pav. Sekos srauto žymėjimo pavyzdys

Sekos srautas (Sequence Flow) žymi veiklų seką. Jeigu nenurodyta lygiagretumo veiklos modelyje vykdomos iš eilės. Šis komponentas parodo kokia tvarka tai vyks. Jį galima apibūdinti kaip grafo su kryptimis briauna, kryptis parodo kuri veikla turi būti įvykdyta vėliau. Sekos srautas žymimas solidžia linija ir užpildyto trikampio formos rodykle parodančia kryptį (8 pav.).



9 pav. Sprendimo žymėjimo pavyzdys

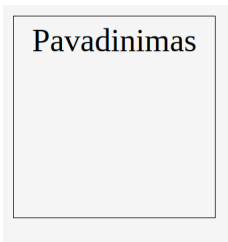

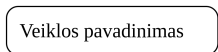

Sprendimas (gateway) gali būti įterptas sekos srautuose tarp veiklų. Jis žymi srautų išsišakojimą arba susijungimą. Šis komponentas parodo, kad priklausomai nuo konkretaus proceso būsenos bus vykdomos veiklos į kurias eina sekos srautai atitinkantys sprendimo sąlygas. Žymėjimas yra kvadratas pasuktas 45 laipsniu kampu (9 pav.). Viduje yra vieta sprendimo konkretizacijai.




10 pav. Duomenų objekto žymėjimo pavyzdys

Duomenų objektas (data Object) pateikia informaciją apie tai kokių duomenų reikalauja veiklos vykdytojas norėdamas ją atlikti ir kokie duomenys pagaminami po jos vykdymo. Komponentas gali žymėti tiek neskaidomus tiek sudėtinius duomenis. Jis vaizduojamas (?? pav.) kaip stačiakampis su nukirptu dešiniuoju viršutiniu kampu ir trikampių prie jo (arba kaip stačiakampis lapas su užlenktu dešiniuoju viršutiniu kampu).

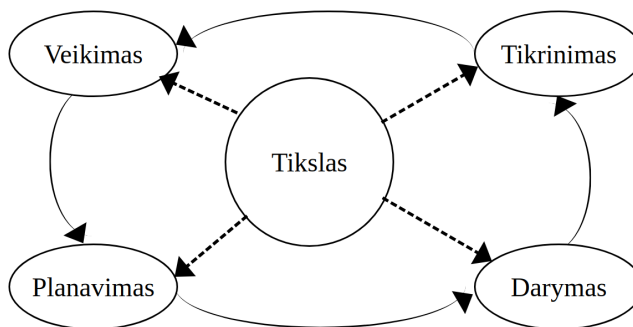
1 lentelė. BPMN diagramos komponentai

Nr.	Komponentas	Aprašymas	Žymėjimo pavyzdys
1	Juosta	Komponentas žymintis diagramos dalyvį ir nurodantis, kad jis atsakingas už veiklų esančių šiame komponente vykdymą.	
2	Įvykis	Komponentas žymintis, kad įvyko kažkas kas įtakojo proceso būseną.	
3	Veikla	Komponentas žymintis užduoties vykdymo procesą	
4	Sekos srautas	Komponentas žymintis veiklų seką.	

5	Transakcija	Subprocesas kuris užtikrina, kad veiklos jame būtų pilnai įvykdytos arba atšauktos.	
---	-------------	---	---

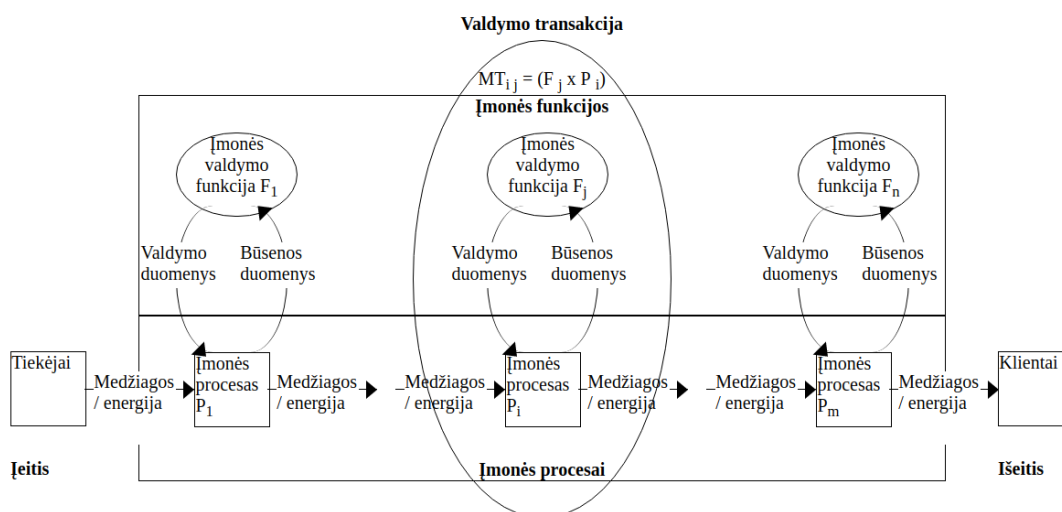
2. Detalizuotas vertės grandinės modelis

BPMN modelyje galima pastebėti įmonės valdymo veiklų supratimo neapibrėžtumus [GL16]. Taip yra todėl, kad išorinio modeliavimo metodai neparodo informacijos arba resursų transformavimo priežasčių. Tačiau įmonę galima analizuoti ir transakcinių darbų sekų modelio (10 pav) požiūriu.



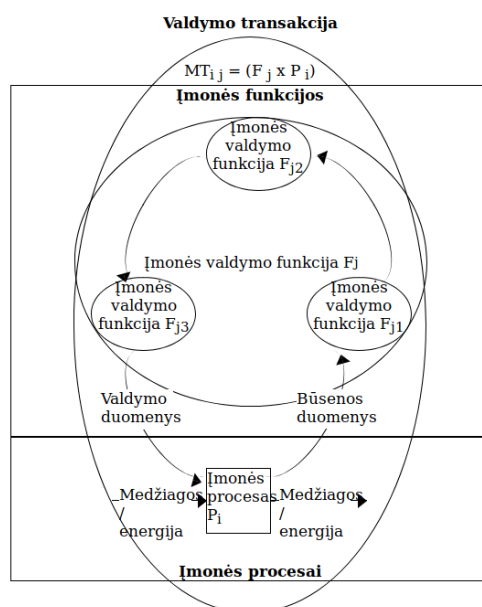
11 pav. Save valdančio proceso pavyzdys

Darbe įmonės procesas bus nagrinėjamas kaip transakcijų visuma. Materialios veiklos atskiriamos nuo valdymo veiklų. P_i žymi veiklos procesą, kuris transformuoja žaliavas, medžiagas, energiją ir formuoja materialią išėigą. F_j yra veiklos valdymo funkcija, informacijos (duomenų, žinių) transformavimo veikla, būtina valdant procesą P_i . Modelis yra suskirstytas į valdymo transakcijas $MT_{ij} = F_j \times P_i$. Tokiu būdu pateikiama daugiau informacijos apie įmonę. Diagrama bus vaizduojama kaip detalizuotas M. Porterio vertės grandinės modelis (11 pav).



12 pav. Detalizuotas M. Porterio vertės grandinės modelis


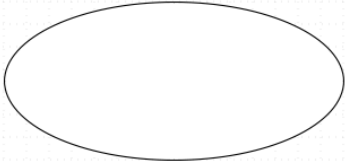

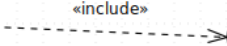
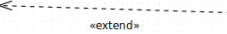
Valdymo funkcija F_j gali būti suskaidyta smulkiau. (12 pav) parodytas pavyzdys kai F_j susideda iš smulkesnių dalių F_{j1} , F_{j2} ir F_{j3} . Kartu visa tai suteikia veiklos procesui P_i valdymo duomenis kuriuos jis panaudoja vykdymui. Vėliau grąžinami būsenos duomenys, jie panaudojami valdymo funkcijoje ir ciklas kartojasi.

13 pav. Valdymo funkcijos F_j išskaidymo pavyzdys

3. Užduočių diagrama

Čia pateikiamas **užduočių diagramos** naudojamos darbe apibrėžimas. Tai bus algoritmo rezultato formatas. Siekiama gauti kiek įmanoma tikslesnę užduočių diagramą. Jos komponentai pateikiam 2 lentelėje.

2 lentelė. Užduočių diagramos komponentai

Nr.	Komponentas	Aprašymas	Žymėjimo pavyzdys
1	Aktorius	Komponentas žymintis sistemos naudotojo tipą.	
2	Vartojimo atvejis	Komponentas specifikuojantis elgsenų ai-bę, kuri generuoja rezultatą.	
3	Asociacija	Komponentas žymintis ryšį tarp aktoriaus ir vartojimo atvejo.	
4	Įtraukia	Komponentas žymintis elgseną kuri yra bendra keliems vartojimo atvejams, todėl ji parodoma atskirai nuo jų, kad būtų galima perpanaudoti.	
5	Išplečia	Komponentas žymintis, kad esant tam tikram atvejui vartojimo atvejis įtraukia papildomus vartojimo atvejus.	

4. UML diagramų transformavimo algoritmai

4.1. Algoritmas BPMN modeliui transformuoti į Užduočių diagramą

Literatūroje yra parašyta apie **vartojimo atvejų diagramos** išvedimą iš **BPMN** modelio [DJ02]. Straipsnyje aprašytas algoritmas atlieka (3) transformaciją. Imamas modifikuotas **BPMN** modelis (1) ir tie **vartojimo atvejų diagramos** komponentai, kurie gali būti iš jo išvesti (2).

$$BPMNElements = \{Start, End, Role, Branch, Task, Transition\}; \quad (1)$$

$$UseCasesElements = \{Actor, Generalization, Association, UseCase, Include, ExtensionPoint, Extend\}; \quad (2)$$

$$BPMN(BPMNModelElements) \Rightarrow UseCases(UseCasesElements); \quad (3)$$

Algoritmo autoriai pirmiausia siūlo surasti ryšius tarp modelių. Juostos atitinka aktorius. Užduotys tuo tarpu grupuojamos, kol nepasiekia maksimalaus skaičiaus vykdomų be pertraukos, priklausančių tai pačiai juostai ir pagaminančių rezultatą. Tokia grupė pavadinama žingsniu ir yra laikoma atitinkančia vartojimo atvejį. Tuomet lieka surasti kaip dar galima būtų panaudoti informaciją, patikslinti ir suprastinti gautoms diagramoms.

Vėliau pristatomas algoritmas. Jis Pirmiausia sudėlioja užduotis į proceso žingsnius. Vėliau juostos tampa aktoriais, o žingsniai jose – vartojimo atvejais. Galiausiai pasikartojančios užduotis išimamos iš žingsnių ir prijungiamos asociacija įtraukia arba išplečia pagal situaciją.

5. Užduočių diagramos išvedimas iš BPMN modelio

Šio darbo tikslas, algoritmas galintis gauti **užduočių diagramas** iš **BPMN** modelio, bus kuriamas pagal 4.1 aprašytą algoritmo sukūrimo pavyzdį. Pirmiausia bus rasti ryšiai tarp diagramų, vėliau sukurtas būdas juos panaudoti, galiausiai panaudota likusi modelio informacija patikslinti ir suprastinti diagramoms.

5.1. Ryšiai tarp BPMN ir užduočių diagramų

Norint duomenis iš vieno modelio perkelti į kitą galima pasinaudoti ryšiais esančiais tarp jų.

3 lentelė. Ryšiai tarp **BPMN** ir **užduočių diagramų**

	Aktorius	Vartojimo atvejis	Asociacija	Ištraukia	Išplečia
Juosta	+		+		
Veikla		+	+	+	+
Transakcija		+		+	+
Sekos srautas					
Įvykis					
Duomenų objektas					
Pranešimų srautas					
Sprendimas					

1. Aktorius – galima gauti iš informacijos esančios juostoje.
2. Vartojimo atvejis – gaunamas iš informacijos veiklose arba transakcijose. Vartojimo atvejis paprastai yra transakcijos tipo.
3. Asociacija – ryšys tarp juostos ir veiklų joje.
4. Ištraukia – veiklos kurias apima transakcija.
5. Išplečia – ryšys randamas kai veikla pasikartoja skirtingose transakcijose.

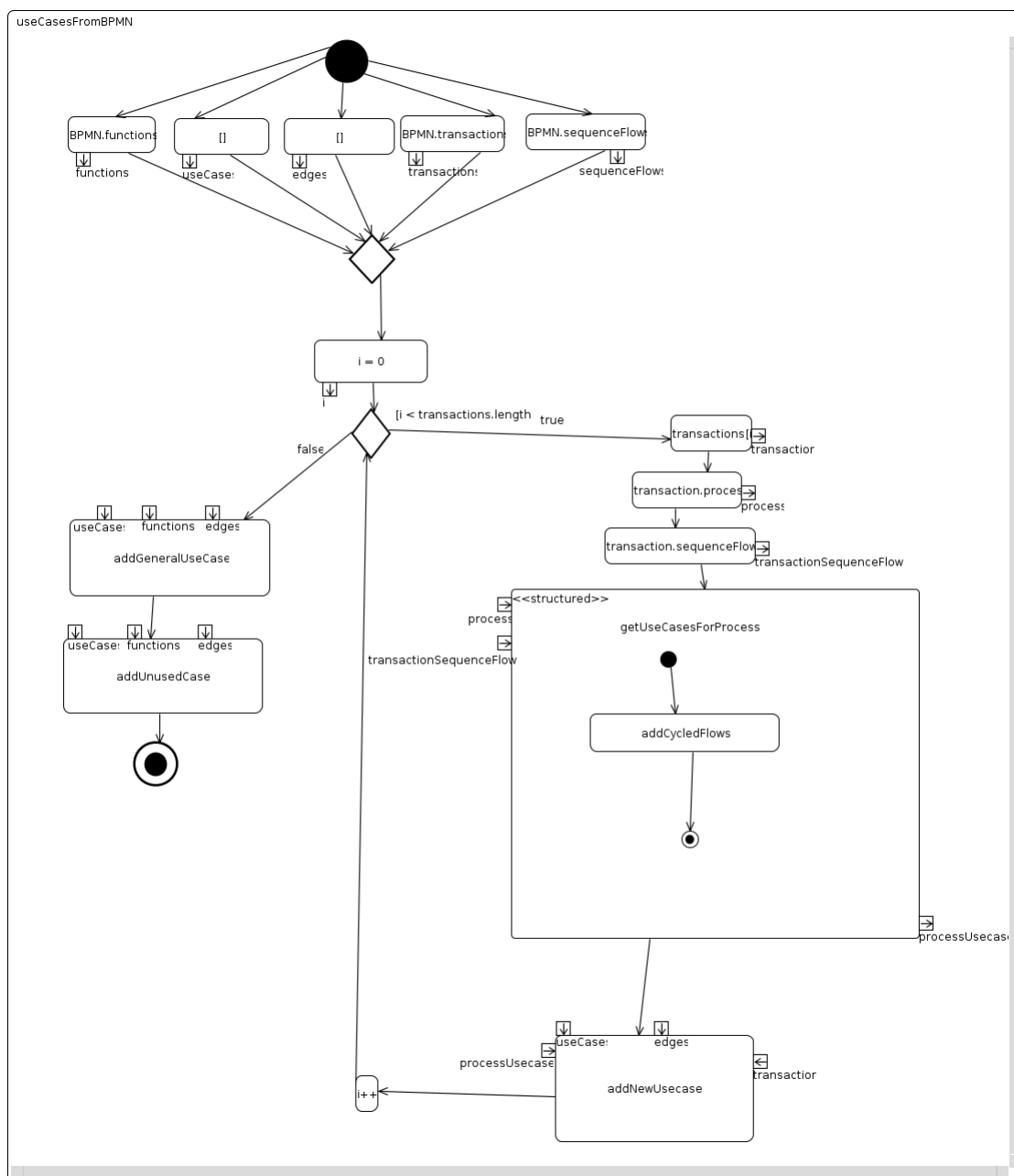
Rasti ryšiai taip pat parodo kurie komponentai bus imami ir kurie gaunami. Taigi galima apibrėžti algoritmo įvesties ir išvesties duomenis.

5.2. Ryšių tarp diagramų panaudojimas transformacijai

Rasti ryšiai parodo į kokius **BPMN** komponentus reikia žiūrėti išvedant **užduočių diagramos** dalis. Toliau peržiūrimi diagramos variantai ir sudėliojami konkretūs žingsniai kuriuos reikia at-

likti. Galiausiai gaunamas pseudokodas (Pseudokodas 1). Jo veikimas pažingsniui aprašomas, taip pat pavaizduotas (pav. 13).

14 pav. Algoritmo diagrama



1. Iškviečiama funkcija useCasesFromBPMN (Pseudokodas 1) paduodant jai **BPMN** modelį.

Pseudokodas 1: **UML Užduočių diagramos** gavimo iš **BPMN** modelio algoritmo pseudokodas

```

1 useCasesFromBPMN = function(BPMN){
2   transactions = BPMN.transactions;
3   sequenceFlows = BPMN.sequenceFlows;

```

```

4   useCases = [];
5   edges = [];
6   for(transaction in transactions){
7       process = transaction.process;
8       transactionSequenceFlows = transaction.sequenceFlows;
9       processUsecases = getUseCasesForProcess(process,
10          transactionSequenceFlows);
11       addNewUsecases(edges,useCases,processUsecases,transaction);
12   }
13   functions = BPMN.functions;
14   addGeneralUseCases(edges,useCases,functions);
15   addUnusedCases(useCases,functions);
16
17   UseCaseDiagram = {useCases,edges};
18   return UseCaseDiagram;
19 }

```

2. Po duomenų inicializavimo pirmiausia imamas transakcijos procesas (eil. 7) ir gaunami sekos srantai jungiantys komponentus transakcijoje (eil. 8).
3. Vėliau sukuriama vartojimo atvejai apibrėžiantys proceso valdymą (eil. 9) funkcija getUseCasesForProcess (Pseudokodas 2).

Pseudokodas 2: Funkcija getUseCasesForProcess

```

1   getUseCasesForProcess = function(process,sequenceFlows){
2       processFlows = sequenceFlows.filter(flow => (
3           flow.source == process
4       ));
5       cycledFlows = [];
6       for(flow in processFlows){
7           addCycledFlows(cycledFlows,[],
8               flow,process,sequenceFlows)
9       }
10      processUsecases = [];
11      for(flow in cycledFlows){
12          usecase = createUseCase(flow);
13          processUsecases.add(usecase);
14      }
15      return processUsecases;

```


16 }

4. Joje randami sekos srautai išeinantys iš proceso (eil. 2) ir kiekvienam iš jų rekursijos būdu surandami ciklai su procesu funkcija `addCycledFlows` (Pseudokodas 3). Kiekvienam iš cikle esančių sekos srautų sukuriamas vartojimo atvejis (eil. 10 - 14). Jų kolekcija ir yra (Pseudokodas 2) grąžinamas rezultatas.

Pseudokodas 3: Funkcija `addCycledFlows`

```

1 addCycledFlows = function(cycledFlows,currentPath,
2   newFlow,process,sequenceFlows){
3   if(newFlow.target == process){
4     return true;
5   }
6   if(currentPath.contains(newFlow)){
7     return false;
8   }
9   if(cycledFlows.contains(newFlow)){
10    return true; // We already found path this way
11  }
12  nextFlows = sequenceFlows.filter(flow => (
13    flow.source == newFlow.target
14  ));
15  if(nextFlows.length == 0){
16    return false;
17  }
18  cycledFlows.add(newFlow);
19  currentPath.add(newFlow);
20  pathFound = false;
21  for(flow in nextFlows){
22    if(addCycledFlows(cycledFlows,currentPath,
23      flow,process,sequenceFlows)){
24      pathFound = true;
25      // No brake because we need to recursively add all paths
26    }
27  }
28  if(!pathFound){
29    cycledFlows.remove(newFlow);
30  }
31  currentPath.remove(newFlow);

```

```

32  return pathFound;
33  }

```

Minėta Funkcija (Pseudokodas 3) pirmiausia patikrina ar jau nėra ciklo su procesu ir jei taip patvirtina, kad parametras `currentPath` turi savyje kelia į procesą (eil. 3 - 5). Jei kelias užsiciklino grąžinamas neigiamas atsakymas (eil. 6 - 8), tai reiškia grįžimą atgal. Jei žingsnis veda į jau išsaugotą sėkmingą kelio atkarpą patvirtinamas jo teisingumas (eil. 9 - 11). Jei nei viena iš šių sąlygų nepasitvirtino paimami sekantys žingsniai (eil. 12). Jų neradus pranešama apie aklavietę (eil. 15 - 17). Kol kas pažymima, kad žingsnis yra sėkmingas (eil. 18) ir žengtas (eil. 19). Toliau ieškoma ciklų su procesu einant sekančiais sekos srautais (eil. 21 - 27). Neradus nei vieno kelio į procesą ištrinamas pažymėjimas apie žingsnio teisingumą (eil. 28 - 30). Kadangi keliai žengus šį žingsnį ištyrinėti grįžtama atgal (eil. 31).

5. (Pseudokodas 2) sukurti vartojimo atvejai pridedami prie jau gautų vartojimo atvejų kartu su ryšiais tarp jų funkcija `addNewUseCases` (Pseudokodas 4).

Pseudokodas 4: Funkcija `addNewUseCases`

```

1  addNewUseCases = function(edges,useCases,newUseCases,transaction){
2      if(newUseCases.length < 1){
3          return;
4      }
5      mainUseCase = createUseCase(transaction);
6      useCases.add(mainUseCase);
7      if(newUseCases.length == 1){
8          useCase = newUseCases[0];
9          mainUseCase.createdFromFlow = useCase.createdFromFlow;
10         mainUseCase.usedBy = useCase.usedBy;
11         return;
12     }
13     for(useCase in newUseCases){
14         useCases.push(useCase);
15         edge = createIncludeEdge(mainUseCase,useCase);
16         edges.push(edge);
17     }
18 }

```

Joje sukuriamas vartojimo atvejis visai transakcijai (eil. 5), pažiūrima kiek vartojimo atvejų rasta, jei vienas tai jo informacija išsaugoma į pagrindinį vartojimo atvejį (eil. 7 - 12). Radus daugiau, jie išsaugomi kaip įeinantys į transakciją (eil. 13 - 17).

6. Galiausiai randamos bendros funkcijos tarp transakcijų ir sukuriami apibendrinantys vartojimo atvejai (Pseudokodas 5).

Pseudokodas 5: Funkcija addGeneralUseCases

```

1 addGeneralUseCases = function(edges,useCases,functions){
2   for(f in functions){
3     useCasesUsingThisFunction = useCases.filter(useCase => (
4       useCase.createdFromFlow.to == f
5     ));
6     if(useCasesUsingThisFunction.length < 2){
7       continue;
8     }
9     generalUseCase = createGeneraluseCase(f);
10    useCases.push(generalUseCase);
11    for(useCase in useCasesUsingThisFunction){
12      edge = createExtendEdge(generalUseCase,useCase);
13      edges.push(edge);
14    }
15  }
16 }

```

7. Jeigu liko nepanaudotų funkcijų, iš jų sukuriami vartojimo atvejai su perspėjimais (Pseudokodas 6).

Pseudokodas 6: Funkcija addUnusedCases

```

1 addUnusedCases = function(useCases,functions){
2   for(function in functions){
3     if(useCases.includes(useCase =>(
4       useCase.createdFromFlow.target == f
5     ))) {
6       continue;
7     }
8     useCase = createUnusedUsecase(function);
9     useCases.push(useCase);
10  }
11 }

```

Atlikusi šio algoritmo veiksmus programa iš Nr. 1 priede pavaizduoto vertės grandinės modelio gauna Nr. 2 priede pavaizduotą **užduočių diagramą**.

6. Programa BPMN transformacijai į užduočių diagramą

Išvados

Conclusions

Literatūra

- [DJ02] R.M. Dijkman and Stef M.M. Joosten. An algorithm to derive use cases from business processes. 08:679–684, 2002-03.
- [GL16] Saulius Gudas ir Audrius Lopata. Towards internal modelling of the information systems application domain. *Informatica, Lith. Acad. Sci.*, 27(1):1–29, 2016. URL: <http://content.iospress.com/articles/informatica/inf1085>.
- [Obj11] Object Management Group (OMG). Business process model and notation (BPMN). OMG Document Number formal/2011-01-03 (<http://www.omg.org/spec/BPMN/2.0>), 2011. Version 2.0.
- [Obj15] Object Management Group (OMG). Omg unified modeling language (OMG UML). OMG Document Number formal/2015-03-01 (<http://www.omg.org/spec/UML/2.5>), 2015. Version 2.5.

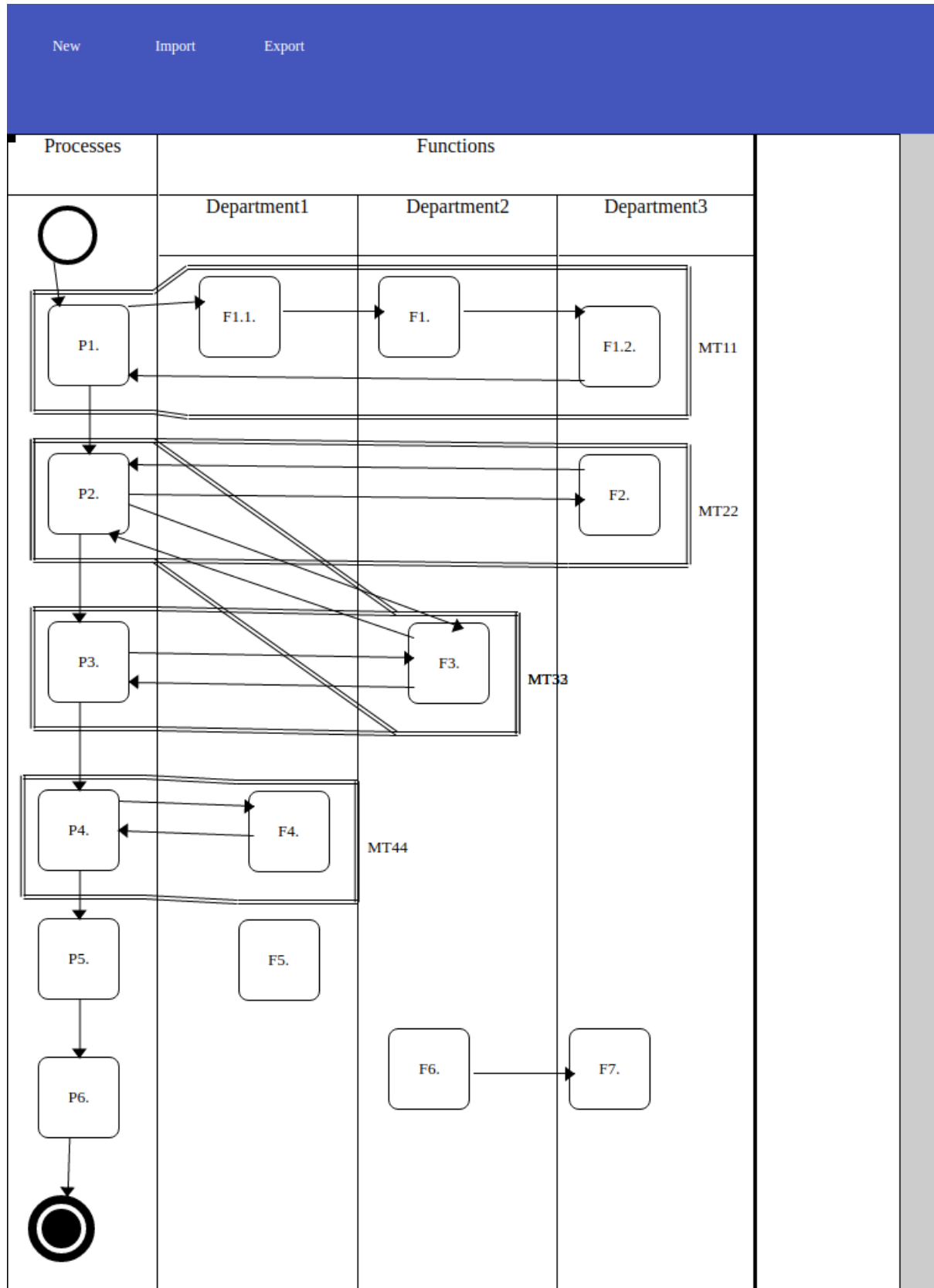
Santrumpos

Šiame darbe naudojami žymėjimai:

1. **BPMN** – modeliavimo kalba, skirta pavaizduoti informaciją plačiai auditorijai. **BPMN** buvo sukurta ir dažniausia naudojama pavaizduoti verslo procesams [Obj11].
2. **UML** – modeliavimo kalba, skirta suteikti standartinę sistemos analizės, architektūros, veikimo ir kūrimo pavaizdavimą [Obj15].
3. **Sekų diagrama** (angl. sequence diagram) – **UML** diagrama, skirta pavaizduoti žinučių tarp apibrėžtų objektų sekai tų objektų gyvavimo metu [Obj15].
4. **Užduočių diagrama** (angl. use case diagram) – **UML** diagrama, skirta pavaizduoti pavaizduoti kaip gali būti naudojama programų sistema [DJ02].

Priedas Nr. 1

Vertės grandinės modelis programos lange



Priedas Nr. 2**Transformuotas užduočių diagrama programos lange**