

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

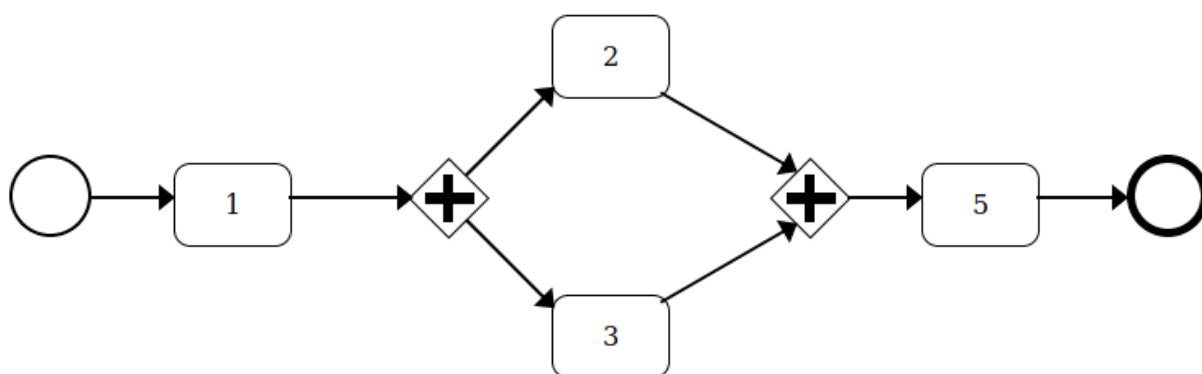
## Turinys

Įvadas .....	2
1. <b>BPMN</b> diagrama .....	5
1.1. <b>BPMN</b> apimtis .....	5
1.2. <b>BPMN</b> komponentai .....	5
1.3. <b>BPMN</b> komponentų tarpusavio ryšiai .....	8
2. Detalizuotas vertės grandinės modelis .....	10
3. Užduočių diagrama .....	12
3.1. Užduočių diagramos apimtis .....	12
3.2. Užduočių diagramos komponentai .....	12
3.3. Užduočių diagramos komponentų tarpusavio ryšiai .....	14
4. <b>UML</b> diagramų transformavimo algoritmai .....	16
4.1. Algoritmas <b>BPMN</b> modeliui transformuoti į <b>Užduočių diagramą</b> .....	16
5. <b>Užduočių diagramos</b> išvedimas iš <b>BPMN</b> modelio .....	17
5.1. Ryšiai tarp <b>BPMN</b> ir <b>užduočių diagramų</b> .....	17
5.2. Ryšių tarp diagramų panaudojimas transformacijai .....	17
6. Programa <b>BPMN</b> transformacijai į <b>užduočių diagramą</b> .....	23
Išvados .....	24
Conclusions .....	25
Literatūra .....	26
Santrumpos .....	27
Priedas Nr.1	
Priedas Nr.2	

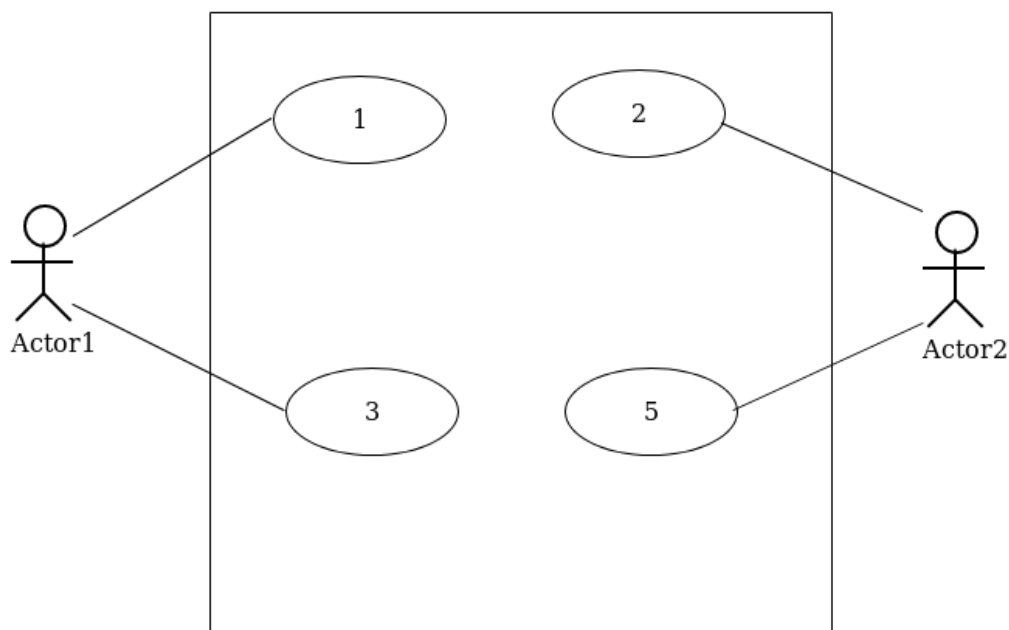
## Įvadas

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.

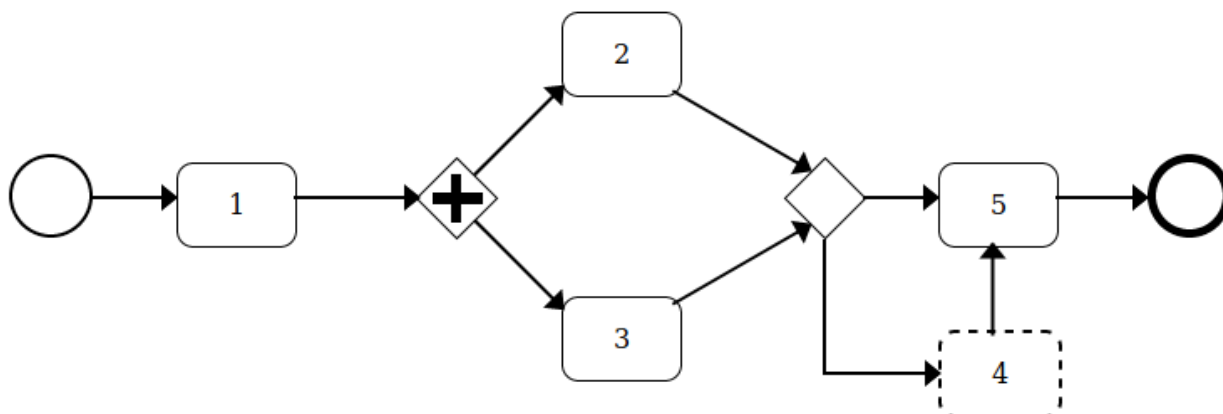


1 pav. Veiklos modelio pavyzdys

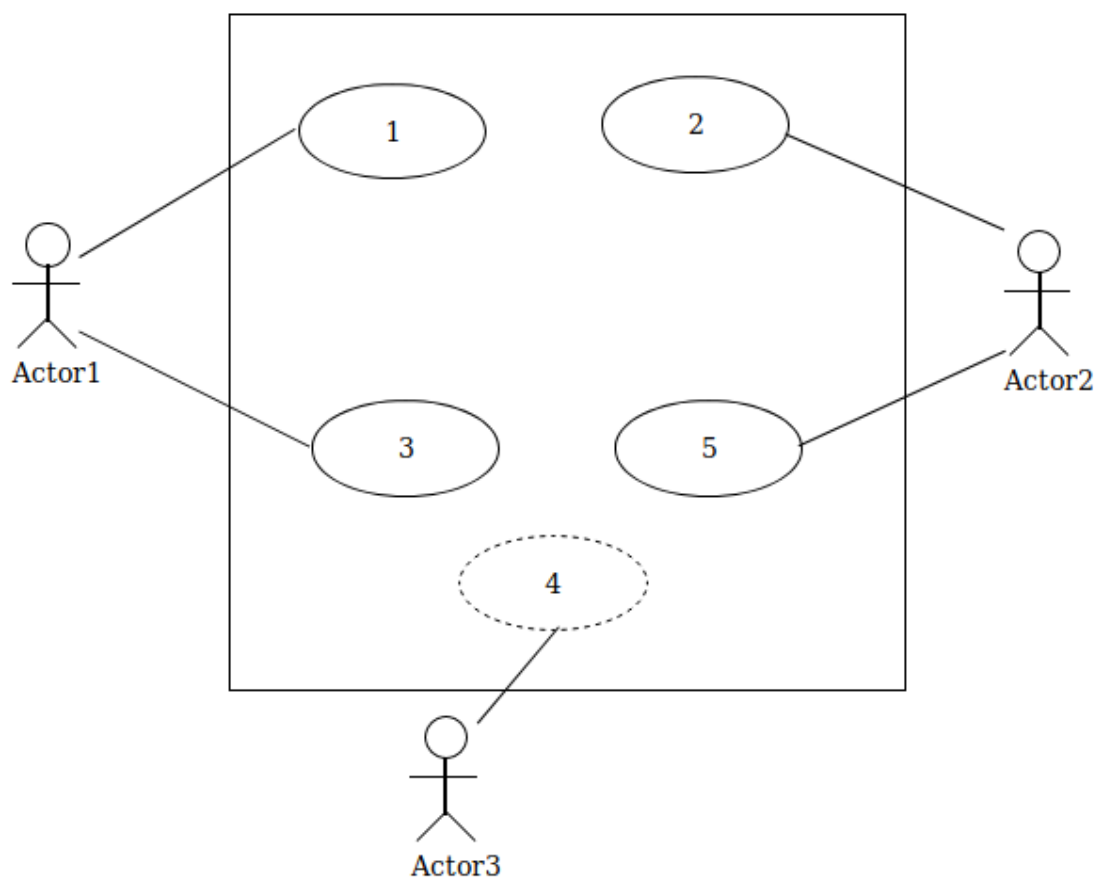


2 pav. Vartojimo atvejų diagramos sukurtos pagal 1 pav. veiklą pavyzdys

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

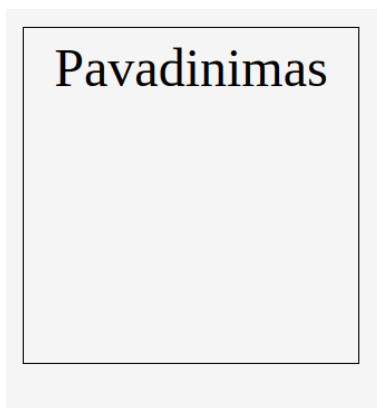
Norint standartizuoti verslo modelių atvaizdavimą 2004 metais organizacija BPMI išleido BPMN 1.0 specifikaciją. Ji leido tiek vaizduoti esamus tiek apsieisti kuriamų procesų reikalavimais. BPMN greitai išpopuliarėjo tarp vadybininkų, verslo analitikų ir programuotojų, nes pasiūlė pažįstamą verslo procesų atvaizdavimą ir turėjo matematinį pagrindą. Vėliau BPMI susijungė su **OMG** ir 2013 metais buvo išleista BPMN 2.0 versija, kurioje BPMN įgavo geresnį skaitomumą, lankstumą ir išplečiamumą.

## 1.1. BPMN apimtis

BPMN specifikacija standartizuoja verslo procesų modelių atvaizdavimo būdą ir duomenų apsieitimo formatą perduoti tiek modelį, tiek jo atvaizdavimą. Sutelkti dėmesiiui į skirtingas proceso dalis pateikiami procesų ir choreografijos submodeliai. Bendradarbiavimo submodelis, į save įtraukiantis kitus, gali būti naudojamas pateikti iš karto viskam. Specifikacijos apimtis yra verslo procesai taigi dalykai kaip strategija, duomenų struktūros, resursai, taisyklės ir įstatymai neįeina į ją.

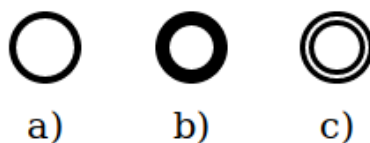
## 1.2. BPMN komponentai

**BPMN** specifikacija leidžia atvaizduoti gana nemažai verslo proceso atributų [Obj11]. Specifikacijoje jie yra suskirstyti į pagrindinius ir išvestinius. 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, tuomet ji gali būti vadinama linija (lane). Juosta žymima apibraukiant tam tikrą sritį (5 pav.). Viduje yra vieta komponentams už kurių atlikimą atsakingas dalyvis.

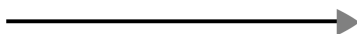


6 pav. Įvykio žymėjimo pavyzdys

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

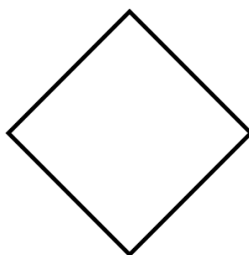
7 pav. Veiklos žymėjimo pavyzdys

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.).



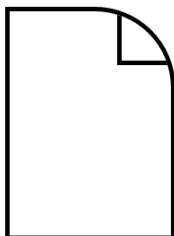
8 pav. Sekos srauto žymėjimo pavyzdys

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.).



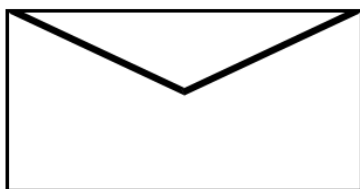
9 pav. Sprendimo žymėjimo pavyzdys

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 konkretizavimui.



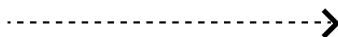
10 pav. Duomenų objekto žymėjimo pavyzdys

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 (10 pav.) kaip stačiakampis su nukirptu dešiniuoju viršutiniu kampu ir trikampiu prie jo (arba kaip stačiakampis lapas su užlenktu dešiniuoju viršutiniu kampu).



11 pav. Pranešimo žymėjimo pavyzdys

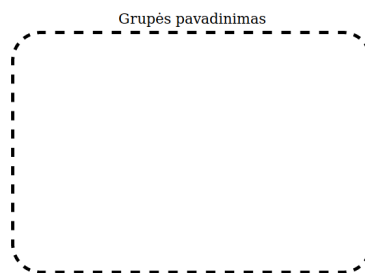
Pranešimas (message) žymi bendravimą tarp dalyviu. Šis komponentas skirtas apibrėžti perduodamai informacijai tarp jų. Vaizduojamas (11 pav.) stačiakampiu su trikampiu viduje (vokas).



12 pav. Pranešimų srauto žymėjimo pavyzdys

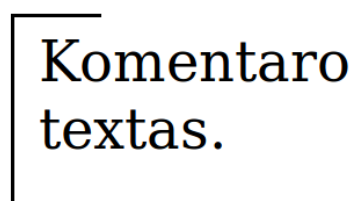
Pranešimų srautas (Message flow) žymi duomenų perdavimą. Tai yra kryptinė grafo briauna, kuri jungia duomenų objektus ir žinutes su juos sukuriančiomis arba naudojančiomis veiklomis. Duomenų srautas į objektą ar žinutę rodo, kad tai yra veiklos išeiga, priešingu atveju tai yra veiklos įeiga, duomenys reikalingi jai įvykdyti. Vaizduojama (11 pav.) punktyrine linija su rodykle rodančia kryptį.





13 pav. Grupės žymėjimo pavyzdys

Grupė (group) skirta nurodyti modelio komponentų kategorijas. Ji neturi įtakos sekų ar duomenų srautams, o tik pasako kas priklauso tai pačiai kategorijai. Vaizdavimas diagramoje yra komponentų apibraukimas ir grupės pavadinimo nurodymas (13 pav.).



14 pav. Komentaro žymėjimo pavyzdys

Komentaras (text annotation) yra mechanizmas skirtas pateikti papildomai informacijai diagramos skaitytojams. Šis komponentas neįtakoja sekos ar duomenų srautų, o tik paaiškina kas ir kodėl vyksta. Žymimas skliaustu iš kairės komentaro teksto pusės (14 pav.).



15 pav. Asociacijos žymėjimo pavyzdys

Asociacija (association) skirta komentarams susieti su modelio komponentais. Ji nurodo kuri diagramos dalis yra komentuojama. Asociacija vaizduojama punktyrine linija (15 pav.).

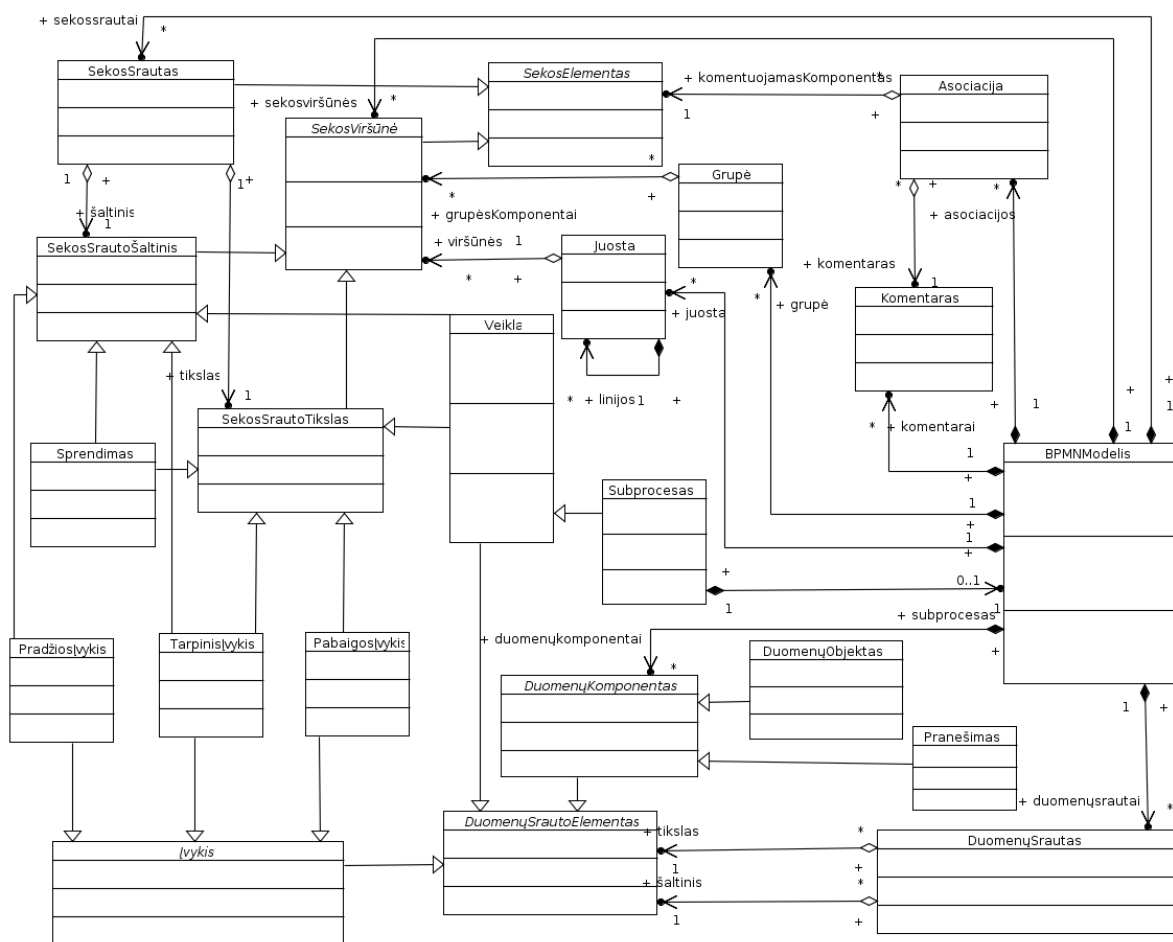
### 1.3. BPMN komponentų tarpusavio ryšiai

Norint modeliuoti verslo procesus vien komponentų nepakanka, taip pat reikia žinoti ir kaip jie sąveikauja tarpusavyje. Tarpusavio ryšius gana neblogai apibūdina metamodelis. Šiame darbe jis bus dažnai naudojamas tam tikslui. Skyriuje 1.2 aprašytų komponentų sąveiką galima pavaizduoti (16 pav.) metamodeliu.

Šiame metamodelyje tipas BPMNModelis yra šakninis komponentas. Jis savyje laiko sekos viršūnes, sekos srautus, juostas, grupes, duomenų komponentus, duomenų srautus, komentarus ir asociacijas. Komponentus metamodelyje galima suskirstyti į tris grupes: sekos komponentus, duomenų komponentus ir komentavimo komponentus. Sekos komponentams priklauso SekosElemento

tipų hierarchija, duomenų komponentams priklauso DuomenųKomponento hierarchija ir DuomenųSrautas, komentavimui priklauso Komentarai ir Asociacija. Taigi galima iš eilės apibūdinti šias grupes iš kurių susideda **BPMN** modelis.

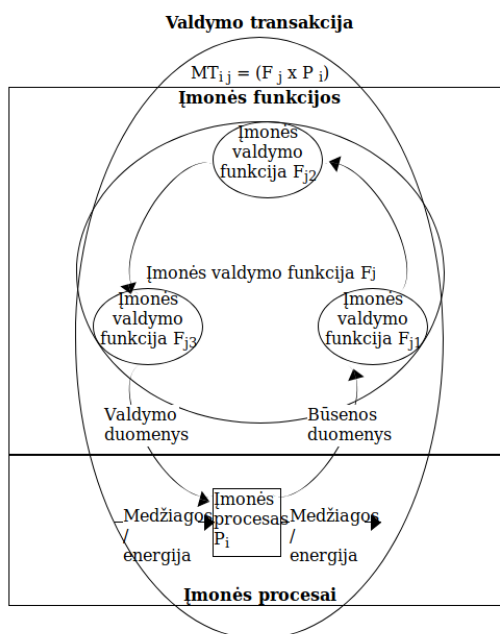
SekosViršūnė yra abstraktus tipas nurodantis, kad tai yra komponentai jungiami sekos srautais, tokie kaip Veikla, Sprendimas ir Įvykio subtipai. Jis išplėstas abstrakčiais tipais: SekosSrautoŠaltinis ir SekosSrautoTikslas, nes kai kurios sekos viršūnės negali turėti ir įeinančių ir išeinančių sekos srautų. Pavyzdžiui komponentas PradžiosĮvykis gali turėti tik išeinančius, PabaigosĮvykis gali turėti tik įeinančius. Įvykis yra abstraktus tipas nes modelyje būna kurie nors iš jo subtipų. Sekos srautas turi vieną šaltinį ir vieną tikslą. Grupės ir Juostos nurodo kurios sekos viršūnės joms priklauso. Juosta dar gali turėti linijų. Metamodelyje taip pat parodoma, kad Veikla gali būti Subprocesas, tokiu būdu savyje laikydama kitą procesą. DuomenųSrautoElementas yra abstraktus tipas parodantis, kad jo subtipai gali būti jungiami duomenų srautais. Įvestas abstraktus tipas DuomenųKomponentas norint apibendrinti Pranešimą ir DuomenųObjektą. Komentarai jungiamas Asociacija ir gali komentuoti sekos komponentus.



16 pav. BPMN pagrindinių komponentų metamodelis



Valdymo funkcija  $F_j$  gali būti suskaidyta smulkiau. (19 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.



19 pav. Valdymo funkcijos  $F_j$  išskaidymo pavyzdys

### 3. Užduočių diagrama

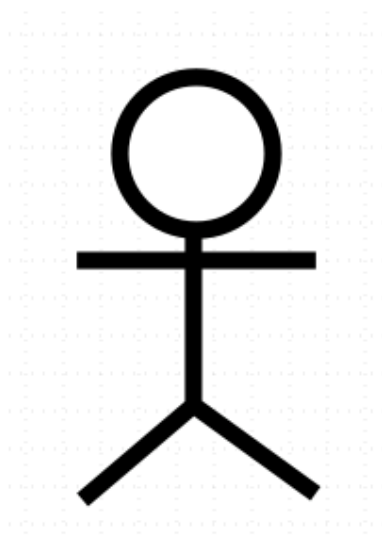
1992 metais Ivar Jacobson apibrėžė metodologiją specifikuoti vartojimo atvejus [JCJ<sup>+</sup>92]. Jis pateikė būdą apibrėžti atvejus tiek tekstu (TODO: pav), tiek diagrama (TODO: pav). Tarp programų sistemų kūrėjų ši metodologija išpopuliarėjo kaip funkcinių reikalavimų apibrėžimo technika. **OMG** savo specifikacijose UML [Obj15] ir SysML [OMG17] standartizuoja vartojimo atvejų modelį ir priskiria jį prie elgsenos apibūdinimo diagramų..

#### 3.1. Užduočių diagramos apimtis

Užduočių diagrama skirta apibūdinti kaip naudotojai siekia savo tikslų pasitelkdami sistemą. Ji parodo kokie yra funkciniai reikalavimai, naudotojų taksonomiją, duomenų srautus tarp naudotojų ir sistemos. Taip pat pateikia funkcijų hierarchijos modeliavimą ir leidžia jas suskaidyti (naudojant įtraukimo ryšį). Užduočių diagrama apibūdina funkcinius reikalavimus, ji nepateikia nei funkcijų atlikimo tvarkos, nei duomenų struktūrų naudojamų sistemoje.

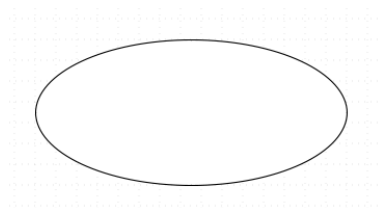
#### 3.2. Užduočių diagramos komponentai

Skirtingi šaltiniai pateikia kiek skirtingus komponentus ir jų apibrėžimus. Šiame darbe daugiausia taikomi **OMG** standartuose pateikti apibrėžimai. Toliau bus pateikiami UML standarte apibrėžtos užduočių diagramos komponentai.



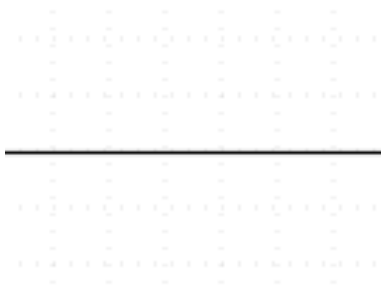
20 pav. Aktoriaus žymėjimo pavyzdys

Aktorius (actor) žymi naudotojo rolę. Ją gali atlikti tiek žmogus, tiek išorinė programų sistema. Aktoriai sąveikauja su bendravimo kanalais prijungtais naudojimo atvejais. Šis komponentas žymimas žmogumi iš pagaliukų (20 pav.).



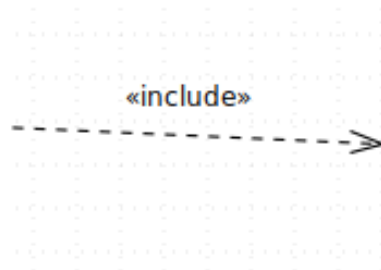
21 pav. Naudojimo atvejo žymėjimo pavyzdys

Naudojimo atvejis (use case) žymi veiklą kurias atlikus gaunamas naudingas rezultatas aibę. Rezultatas gali būti naudingas tiek vykdytojui, tiek kitiems suinteresuotiems žmonėms. Šis komponentas vaizduojamas elipse (21 pav.), naudojimo atvejo pavadinimas gali būti tiek elipsėje, tiek po ja.



22 pav. Bendravimo kanalo žymėjimo pavyzdys

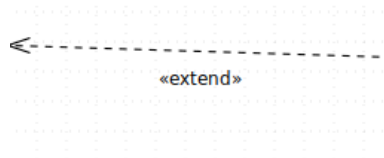
Bendravimo kanalas (communication path) žymi sąveika tarp aktorius ir sistemos. Šis komponentas diagramoje jungia aktorių su naudojimo atveju, taip parodydamas, kad prieš atlikdamas veiklas (sistemos funkcijas) naudotojas pateikia įvestį o po jų atlikimo gaunamas rezultatas. Jeigu bendravimo kanalo ryšio su aktoriumi gausa yra daugiau nei 1 reiškia naudojimo atvejui atlikti reikalingi keli vykdytojai, jeigu bendravimo kanalo ryšio su naudojimo atveju gausa yra daugiau nei 1 reiškia aktorius gali atlikti tas pačias veiklas daugiau nei vieną kartą. Asosijacija žymima solidžia linija (22 pav.).



23 pav. Įtraukimo žymėjimo pavyzdys

Įtraukimas (includes) žymi naudojimo atvejo suskaidymą. Šis komponentas modeliuoja sąryši tarp dviejų vartojimo atvejų, taip parodydamas, kad įtraukiamo naudojimo atvejo veiklos yra atliekamos įtraukiančiame naudojimo atvejiuje. Įtraukimą numatyta naudoti tuomet, kai tos pačios veiklos pasikartoja keliuose naudojimo atvejuose. Tos veiklos įdedamos į atskirą naudojimo atvejį

ir prijungiamos šiuo ryšiu, tokiu būdu iškeliamas pasikartojantis funkcionalumas. Įtraukimas vaizduojamas punktyrine linija su rodykle prie įtraukiamo naudojimo atvejo ir patikslinimu dvigubuose kampiniuose skliaustuose (23 pav.).



24 pav. Išplėtimo žymėjimo pavyzdys

Išplėtimas (extends) žymi, kad esant tam tikroms sąlygoms naudojimo atvejis įtraukia veiklas iš kitų naudojimo atvejų. Šis komponentas numatytas naudoti bendram funkcionalumui iškelti, bet kitaip nei ryšys „Įtraukia“, parodo, kad veiklos įtraukiamos ne visada. Jis vaizduojamas punktyrine linija su rodykle prie išplečiamo naudojimo atvejo ir patikslinimu dvigubuose kampiniuose skliaustuose (24 pav.).



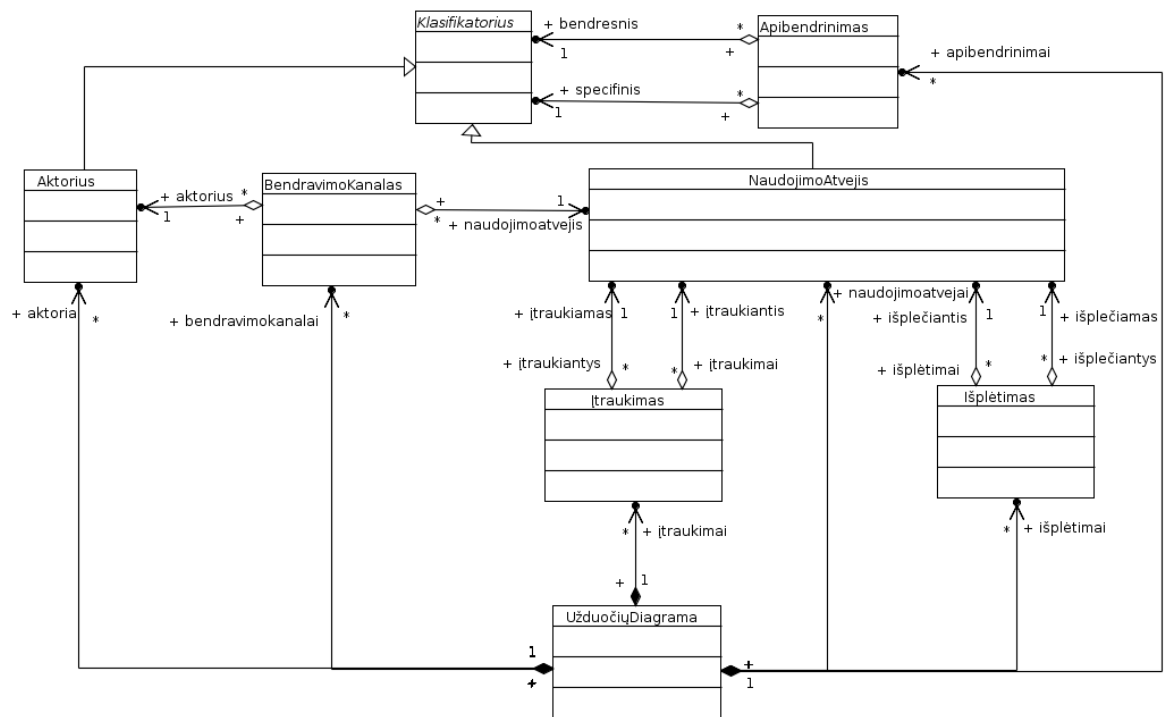
25 pav. Apibendrinimo žymėjimo pavyzdys

Apibendrinimas (generalization) žymi, kad elementas yra bendresnio elemento variantas. Nuo išplėtimo naudojimo atvejo apibendrinimas skiriasi tuo, kad pasakoma jog bent vienas iš apibendrinamų naudojimo atvejų funkcionalumų įtraukiamas į apibendrinančio naudojimo atvejo funkcionalumą. Žymimas solidžia linija su rodykle prie apibendrinamo naudojimo atvejo (25 pav.).

### 3.3. Užduočių diagramos komponentų tarpusavio ryšiai

Komponentų aprašytą 3.2 skyriuje tarpusavio ryšiai pavaizduoti metamodeliu (26 pav.). Jis sudarytas pagal UML ir SysML standartuose pateiktą informaciją.

Užduočių diagrama yra paketas savyje laikantis aktorius, naudojimo atvejus, bendravimo kanalus, įtraukimus, išplėtimus ir apibendrinimus. Bendravimo kanalas jungia vieną aktorį su vienu naudojimo atveju. Įtraukimas yra ryšys tarp įtraukiamo ir įtraukiančio naudojimo atvejo. Išplėtimas jungia išplečiamą naudojimo atvejį su išplečiančiu. UML modelis leidžia abstraktaus tipo klasifikatoriaus apibendrinimą. Aktorius ir naudojimo atvejis yra klasifikatoriaus subtipai, todėl gali būti apibendrinti.



26 pav. Užduočių diagramos metamodelis



## 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 vykdymų 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 bendravimo kanalu 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.

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

	Aktorius	Vartojimo atvejis	Bendravimo kanalas	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. Bendravimo kanalas – ryšys tarp juostos ir veiklų joje.
4. Ištraukia – veiklos kurias apima transakcija.
5. Išplečia – ryšys randamas kai veikla pasikartoja skirtingose transakcijose.

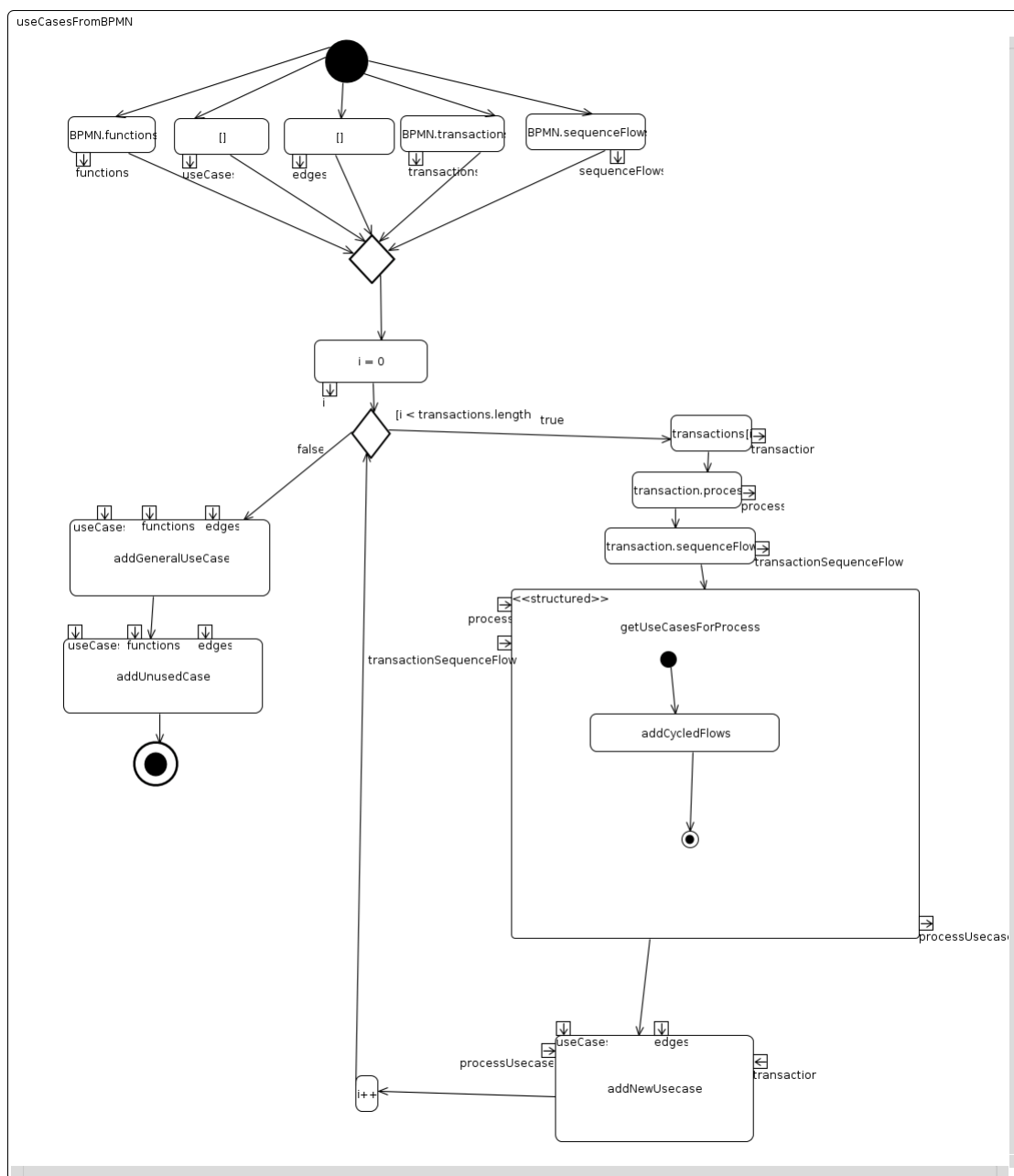
Rasti ryšiai taip pat parodo kurie komponentai bus imami ir kurie gaunami. Taigi galima apibrėžti algoritmo įvesties ir išvesties duomenis.

### 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. 27).

27 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 }  
 17

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>.
- [JCJ<sup>+</sup>92] Ivar Jacobson, Magnus Christerson, Patrik Jonsson ir Gunnar Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, 1992.
- [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.
- [OMG17] OMG. OMG Systems Modeling Language (OMG SysML), Version 1.5, Object Management Group, 2017. URL: <https://www.omg.org/spec/SysML/1.5>.

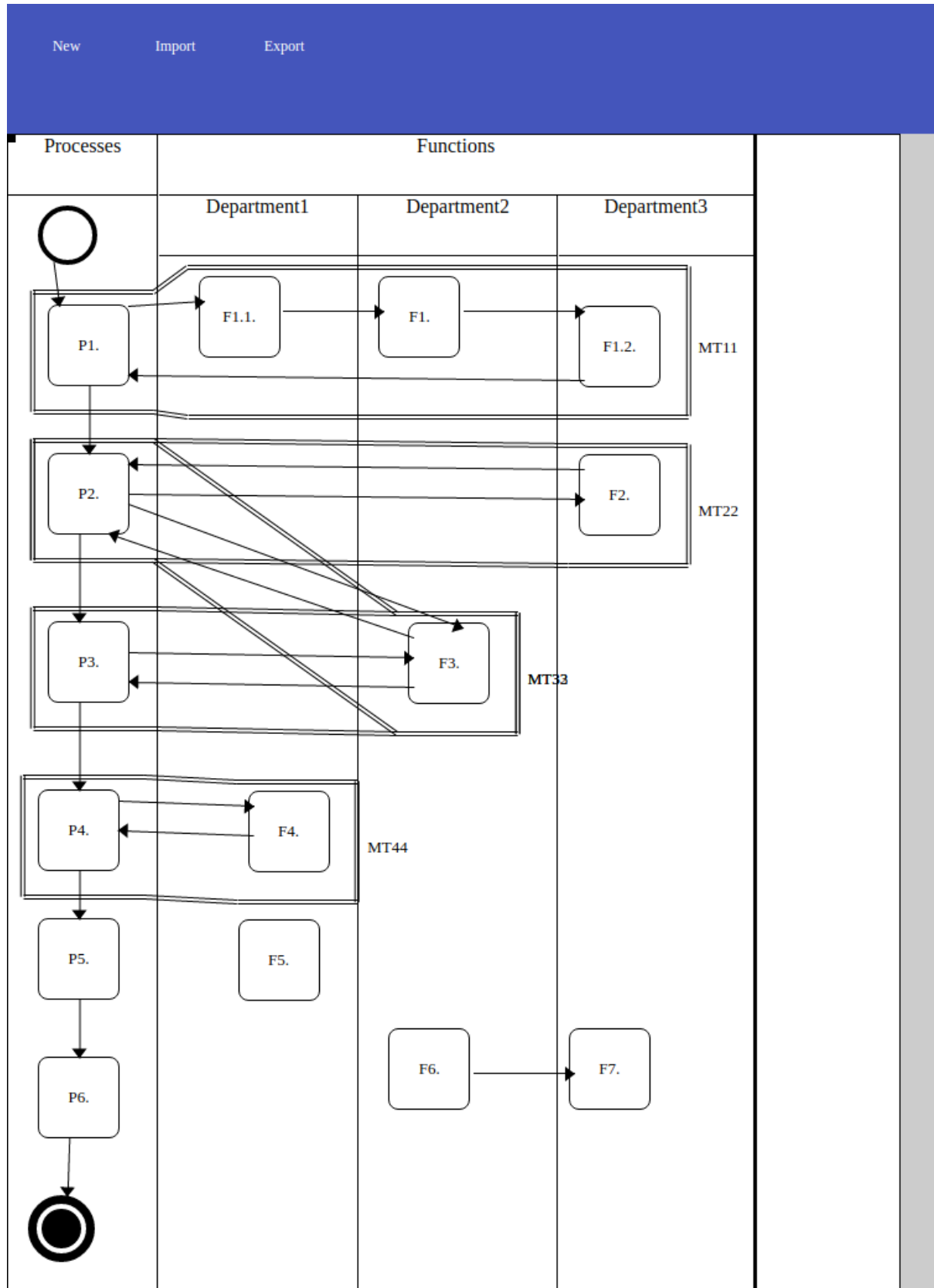
## Santrumpos

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].
5. **OMG** (angl. Object Management Group) – atviras, tarptautinis ne pelno siekiantis technologijų standartų konsorciumas (<https://www.omg.org/>).

# Priedas Nr. 1

## Vertės grandinės modelis programos lange



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