

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

Sąvokų apibrėžimai .....	2
Įvadas .....	3
1. Algoritmas kurti <b>sekų diagramas</b> pagal <b>BPMN</b> pateiktą informaciją .....	4
1.1. <b>UML</b> diagramų transformavimo algoritmai .....	4
1.1.1. <b>Užduočių diagramos</b> išvedimas iš <b>BPMN</b> modelio .....	4
1.2. <b>Užduočių diagramos</b> išvedimas iš <b>BPMN</b> modelio .....	4
1.2.1. Ryšiai tarp <b>BPMN</b> ir <b>sekų diagramų</b> .....	5
1.2.2. <b>BPMN</b> diagrama .....	5
1.2.3. <b>Užduočių diagrama</b> .....	7
1.2.4. Ryšių tarp diagramų panaudojimas transformacijai .....	8
1.2.5. Likusios informacijos panaudojimas suprastinti ar patikslinti diagramai .....	12
2. Programa <b>BPMN</b> transformacijai į <b>sekų diagramą</b> .....	13
Išvados .....	14
Conclusions .....	15
Literatūra .....	16

## Sąvokų apibrėžimai

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].

## 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.

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 kaip sistema bus naudojama. 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**.
2. Programa demonstruojanti algoritmo veikimą.

# 1. Algoritmas kurti sekų diagramas pagal BPMN pateiktą informaciją

Šio darbo tikslas yra algoritmas atlikti diagramų transformacijai. Pirmiausia pateikiama tai, kas literatūroje rašoma apie **UML** diagramų transformavimo algoritmus. Vėliau sukuriamas siekiamas algoritmas.

## 1.1. UML diagramų transformavimo algoritmai

### 1.1.1. Užduočių diagramos išvedimas iš BPMN modelio

Literatūroje yra parašyta apie **vartojimo atvejų diagramos** išvedimą iš **BPMN** modelio [DJ02]. Darbe nagrinėjamas [DJ02] straipsnyje aprašytas algoritmas (3). 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, tos pačios juostos ir pagaminančių rezultatą. Tokia grupė pavadinama žingsniu ir yra laikoma atitinkančia vartojimo atvejį. Taip išsiaiškinami transformacijos žingsniai. Tuomet lieka tik formaliai užrašyti kaip jie gali būti vykdomi ir surasti kaip dar galima būtų panaudoti informaciją, patikslinti ir suprastinti gautoms diagramoms.

Algoritmas Pirmiausia sudėlioja pavienes 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.

## 1.2. Užduočių diagramos išvedimas iš BPMN modelio

Šio darbo tikslas, algoritmas galintis gauti **užduočių diagramoms** iš **BPMN** modelio, bus kuriamas pagal 1.1.1 aprašytą algoritmo sukūrimo pavyzdį. Pirmiausia bus rasti ryšiai tarp diagramų, vėliau sukurtas būdas juos panaudoti, galiausiai panaudota likusi modelio informacija patikslinti ir suprastinti diagramoms.

### 1.2.1. Ryšiai tarp BPMN ir sekų diagramų

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

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

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

1. Aktorius – galima gauti iš informacijos esančios juostoje.
2. Vartojimo atvejis – gaunamas iš informacijos veiklose.
3. Asociacija – ryšys tarp juostos ir veiklų joje.
4. Įtraukia – pasikartojančios veiklos.
5. Išplečia – ryšys randamas kai veikla yra atskiras kitos veiklos atvejis.

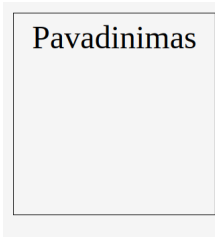

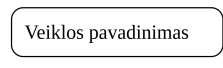
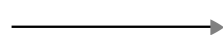
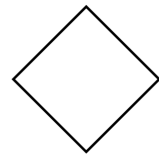
Rasti ryšiai taip pat parodo kurie komponentai bus imami ir kurie gaunami. Taigi galima apibrėžti **BPMN** ir **Užduočių diagramas**. Tai taps algoritmo įvestimi ir išvestimi.

### 1.2.2. BPMN diagrama

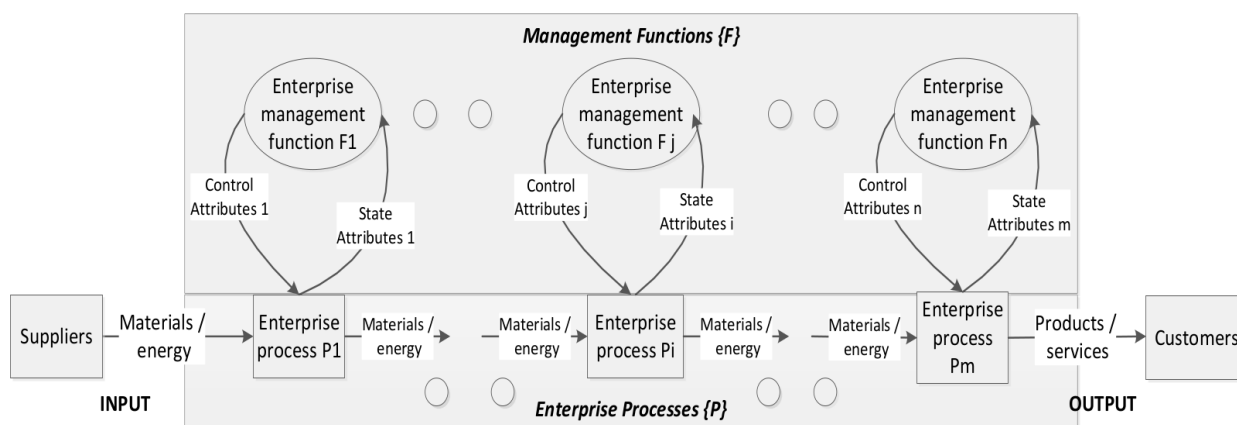
**BPMN** specifikacija leidžia atvaizduoti gana nemažai verslo proceso atributų [Obj11]. Bet šiame darbe ji bus nagrinėjama tik kaip įvesties duomenų formatas, naudojamas apibrėžti informaciją pagal kurią bus kuriama **sekų diagrama**. Taigi daugelį **BPMN** komponentų galima tiesiog ignoruoti, nes jie neturi jokios įtakos algoritmo vykdymo rezultatui. Norint pabrėžti svarbią informaciją, darbe bus tiriamos tik tos **BPMN** savybės kurios gali įtakoti algoritmo vykdymo rezultatą. Atitinkami komponentai pavaizduoti 2 lentelėje.

2 lentelė. BPMN diagramos komponentai

Nr.	Komponentas	Aprašymas	Žymėjimo pavizdys
-----	-------------	-----------	-------------------

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	Sprendimas	Komponentas žymintis sekos srautų išsišakojimą.	

Darbe bus nagrinėjamas procesas kuriam kuriama programų sistema, todėl materialios veiklos bus atskirtos nuo duomenų tvarkymo veiklų [GL16]. **BPMN** diagrama bus vaizduojama kaip detalizuotas M. Porterio vertės grandinės modelis (1 pav), todėl į apibrėžimą įtrauktas sprendimas, kuris bus naudojamas išsišakojimui pavaizduoti. Tokiu būdu dėmesys telkiamas ties informacijos procesais.

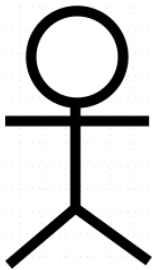
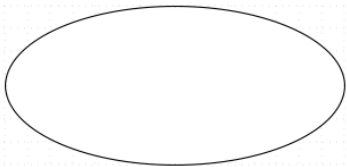



1 pav. Detalizuotas M. Porterio vertės grandinės modelis

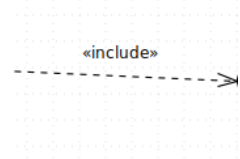

### 1.2.3. Užduočių diagrama

Čia pateikiamas **užduočių diagramos** naudojamos darbe apibrėžimas. Tai bus algoritmo rezultato formatas. Siekiama gauti kiek įmanoma tikslesnę užduočių diagramą. Jos komponentai pateikiam 3 lentelėje.

3 lentelė. Užduočių diagramos komponentai

Nr.	Komponentas	Aprašymas	Žymėjimo pavidzys
1	Aktorius	Komponentas žymintis sistemos naudotojo tipą.	
2	Vartojimo atvejis	Komponentas specifikuojantis elgsenų aiškę, kuri generuoja rezultatą.	
3	Asociacija	Komponentas žymintis ryšį tarp aktoriaus ir vartojimo atvejo.	



4	Ištraukia	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.	

#### 1.2.4. 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 atlikti. Galiausiai gaunamas pseudokodas (Listing 1).

Listing 1: **UML Sekų diagramos** gavimo iš **BPMN** modelio algoritmo pseudokodas

```

1 useCasesFromBPMN = function (BPMN) {
2     { transactions , sequenceFlows } = BPMN;
3     useCases = [];
4     edges = [];
5     for(transaction in transactions){
6         flowUseCases = getUseCasesForProcess(transaction .
              process , sequenceFlows );
7         addNewUseCases(edges , useCases , flowUseCases ,
              transaction );
8     }
9     functions = getFunctions(sequenceFlows);
10    addGeneralUseCases(edges , useCases , functions );
11    addUnusedCases(useCases , functions );
12
13    UseCaseDiagram = { useCases , edges };
14    return UseCaseDiagram ;
15 }
```

1. Pirmiausia imamas transakcijos procesas.

2. Vėliau jam sukuriami vartojimo atvejai funkcija `getUseCasesForProcess` (Listing 2).

Listing 2: Funkcija `getUseCasesForProcess`

```

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

```

Joje funkcija `addCycledFlows` (Listing 3) paima tik tuos sekos srautus, kurie sudaro ciklą su procesu.

Listing 3: Funkcija `addCycledFlows`

```

1 addCycledFlows = function(cycledFlows , currentPath , newFlow ,
2     process , sequenceFlows){
3     if(newFlow.to == process){
4         return true;
5     }
6     if(newFlow.to.type != type.function){
7         return false;
8     }
9     if(currentPath.contains(newFlow)){
10        return false;
11    }
12    if(cycledFlows.contains(newFlow)){
13        return true; \\ We already found path this
14        way
15    }
16    nextFlows = sequenceFlows.filter(flow => flow.from =
17        newFlow.to);
18    if(nextFlows.length == 0){
19        return [];
20    }

```

```

17     }
18     cycledFlows.add(newFlow);
19     currentPath.add(newFlow);
20     pathFound = false;
21     for(flow in nextFlows){
22         if(getCycledFlows(cycledFlows, currentPath,
23             flow, process, sequenceFlows)){
24             pathFound = true;
25         }
26     }
27     if(!pathFound){
28         cycledFlows.remove(newFlow);
29     }
30     currentPath.remove(newFlow);
31     return pathFound;
32 }

```

3. Sukurti vartojimo atvejai pridedami prie jau gautų vartojimo atvejų kartu su ryšiais tarp jų funkcija `addNewUsecases` (Listing 4).

Listing 4: Funkcija `addNewUsecases`

```

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

```

4. Vėliau gaunamos visos diagramos funkcijos (Listing 5).

Listing 5: Funkcija getFunctions

```

1 getFunctions = function(sequenceFlows){
2     functions = new Set();
3     for(flow in sequenceFlows){
4         if(flow.from.type == type.function){
5             functions.add(flow.from);
6         }
7         if(flow.to.type == type.function){
8             functions.add(flow.to);
9         }
10    }
11    return functions;
12 }

```

5. Randamos bendros funkcijos tarp tranzakcijų ir sukuriami apibendrinantys vartojimo atvejai (Listing 6).

Listing 6: Funkcija addGeneralUseCases

```

1 addGeneralUseCases = function(edges, useCases, functions){
2     for(function in functions){
3         useCasesUsingThisFunction = useCases.filter(
4             useCase => useCase.createdFromFlow.to ==
5                 function);
6         if(useCasesUsingThisFunction.length < 2){
7             continue;
8         }
9         generalUseCase = createGeneraluseCase(
10            function);
11        useCases.push(generalUseCase);
12        for(useCase in useCasesUsingThisFunction){
13            edge = createExtendEdge(
14                generalUseCase, useCase);
15            edges.push(edge);
16        }
17    }
18 }

```

6. Galiausiai randamos nepanaudotos funkcijos ir sukuriami vartojimo atvejai su perspėjimais 7).

Listing 7: Funkcija addUnusedCases

```

1 addUnusedCases = function(useCases , functions){
2     for(function in functions){
3         useCasesUsingThisFunction = useCases.filter(
4             useCase => useCase.createdFromFlow.to ==
5             function);
6         if(useCasesUsingThisFunction.length > 0){
7             continue ;
8         }
9         useCase = createUnusedUsecase(function);
10        useCases.push(useCase);
11    }

```

### 1.2.5. Likusios informacijos panaudojimas suprastinti ar patikslinti diagramai

## **2. Programa BPMN transformacijai į sekų 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.