Kauno technologijos universitetas

Informatikos fakultetas

Modeliais grindžiama gedimų medžio analizė

Baigiamasis magistro studijų projektas

Lukas Ružauskas

Projekto autorius

Doc. dr. Lina Čeponienė

Vadovė

Lekt. Gintarė Kriščiūnienė

Konsultantė

Kaunas, 2019

Kauno technologijos universitetas
Informatikos fakultetas

Modeliais grindžiama gedimų medžio analizė

Baigiamasis magistro studijų projektas
Informacinių sistemų inžinerija (6211BX009)

Lukas Ružauskas
Projekto autorius

Doc. dr. Lina Čeponienė
Vadovė

Lekt. Gintarė Kriščiūnienė
Konsultantė

Lekt. dr. Lina Bisikirskienė
Recenzentė

Kaunas, 2019

Kauno technologijos universitetas

Informatikos fakultetas

Lukas Ružauskas

Modeliais grindžiama gedimų medžio analizė

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Luko Ružausko, baigiamasis projektas tema „Modeliais grindžiama gedimų medžio analizė“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	(parašas)

Ružauskas, Lukas. Modeliais grindžiama gedimų medžio analizė. Magistro baigiamasis projektas / vadovė doc. dr. Lina Čeponienė; Kauno technologijos universitetas, Informatikos fakultetas.

Mokslo kryptis ir sritis: Informatikos inžinerija, technologijos mokslai

Reikšminiai žodžiai: *gedimų medžio analizė, modeliais grindžiama sistemų inžinerija, SysML*

Kaunas, 2019. 68 p.

SANTRAUKA

Gedimų medžio analizė – vienas iš kelių pagrindinių saugumo analizės procesų, kuris leidžia atlikti saugumo ir patikimumo analizę kuriant sudėtingas sistemas, dar jų projektavimo metu. Atsekamumas tarp gedimų medžio analizės ir sistemos modelio šiuo metu yra užtikrinamas dokumentais. Tai reiškia, jog žmogiškosios klaidos tikimybė yra didelė, nes viskas turi būti atliekama rankiniu būdu. Atsekamumo ryšiai modeliais grindžiamoje architektūroje yra svarbi sistemos kūrimo dalis, leidžianti greitai ir be klaidų užtikrinti, jog konkrečios sistemos dalys yra tarpusavyje susijusios.

Šiame darbe buvo išanalizuotas gedimų medžio analizės kūrimo procesas kartu su kitais saugumo ir patikimumo analizių procesais. Taip pat išanalizuota modeliais grindžiama sistemų inžinerija, *SysML* kalba. Analizės metu buvo įvertinti gedimų medžio analizės kūrimo įrankiai, nustatant jų trūkumus. Projektavimo metu buvo panaudotas *SysML* lėktuvo modelis, kurio pagalba buvo atrasta koreliacija tarp sistemos modelio ir gedimų medžio analizės. Atradus jų tarpusavio ryšį, buvo surastas būdas atlikti gedimų medžio analizę išplečiant *SysML* kalbą naujais elementais. Pasinaudojant *SysML* parametrikų diagrama ir modelio simuliacija,galima atlikti kiekybinę gedimų medžio analizę, kuri leidžia paskaičiuoti nenorimų įvykių tikimybes. Realizacijos metu buvo sukurtas *SysML* kalbos plėtinys (profilis) ir biblioteka, leidžianti atlikti gedimų medžio analizę modeliais grindžiamoje architektūroje. Eksperimento metu buvo įsitikinta, jog su pasiūlytu sprendimu galima atlikti gedimų medžio analizę viename įrankyje, kuris leistų užtikrinti atsekamumą tarp sistemos modelio ir gedimų medžio analizės.

Ružauskas, Lukas. Model Based Fault Tree Analysis: Master‘s thesis in Information Systems Engineering / supervisor assoc. prof. dr, Lina Čeponienė. The Faculty of Informatics, Kaunas University of Technology.

Research area and field: Informatics Engineering, Technology Science

Key words: Fault Tree Analysis (FTA), Model Based System Engineering, SysML

Kaunas, 2019. 68 p.

SUMMARY

Fault Tree Analysis (FTA) is one of the most important processes to ensure safety and reliability in systems engineering when developing high-hazard systems. Currently, traceability is being ensured by using document-based methods. It means that this traceability has to be applied manually, which leads to a high risk of making human mistakes. Traceability relations take an important role in the model based architecture. They enable quick and precise connection between system‘s parts, by showing relations among them.

This project encompasses fault tree analysis together with analysis of other safety and reliability analysis processes. It also includes the review of model based system engineering and SysML. Many different tools that enable fault tree analysis modeling were also analyzed and evaluated. During the design step in this project, an aircraft SysML model was used which helped to find correlation between system model and fault tree analysis. Once the correlation was found, a way to model fault tree analysis was created by extending SysML language. This method also enables application of SysML parametrics diagram to calculate probabilities of the fault tree events. During the implementation a new profile to extend SysML was created together with a library, which enables fault tree analysis creation in model based system engineering. During the experiment, the provided solution was tested on different models and ensured that it could be used to successfully trace relations between fault tree analysis elements and system model elements.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

FTA (angl. *Fault Tree Analysis*)	Gedimų medžio analizė.
FMEA (angl. *Failure Mode and Effects Analysis*)	Nesėkmės modelio ir pasekmių analizė
SysML	Sistemų modeliavimo kalba (angl. Systems Modeling Language)
MBSE	Modeliais grindžiama sistemų inžinerija (angl. *Model Based System Engineering*)
API (angl. *Application Programming Interface*)	Aplikacijų programavimo sąsaja leidžianti susieti aplikacijos programinį kodą su nuosavu kodu
Loginiai vartai	Loginiai elementai, kurie atlieka bulio algebros operacijas

ĮVADAS

Šis magistro baigiamasis darbas priklauso Informacinių sistemų inžinerijos studijų programai.

Darbo problematika ir aktualumas

Kokybės inžinieriai (angl. *quality engineers*) ar patikimumo inžinieriai (angl. *reliability engineers*) nuolat susiduria su gedimų medžių analizės kūrimo procesu ar šio objekto analizavimu. Kiekvieną kartą analizuojant gedimų medį, jiems reikalingas papildomas dokumentas (ar kitas informacijos šaltinis), kuris leistų nustatyti klaidų medžio elementų sąsaja su kuriamu sistemos modeliu. Taip pat dėl to, kad projektuojant ir kuriant gedimų medžio analizę yra naudojama skirtinga įranga – kuriant *FTA* projektą tenka kurti pakartotinius sistemos elementus. Dėl to yra gaištamas papildomas laikas ir atsiranda daugiau finansinių išlaidų. Visa tai sunkina ne tik saugumo inžinierių, tačiau ir visų kitų (vadovų, analitikų ir kt.) darbą norint išsianalizuoti gedimų medžio diagramas.

Tyrimo tikslas – palengvinti gedimų medžio analizę modeliais grindžiamoje architektūroje, pasiūlant metodiką, susiejančią gedimų medį ir kuriamos sistemos modelį ir užtikrinančią jų tarpusavio suderinamumą.

Tyrimo uždaviniai:

1. išanalizuoti:
   1.1. gedimų medžio analizės (*FTA*) procesą;
   1.2. *SysML* (angl. *Systems modelling language*) darinį;
   1.3. *FMEA* (angl. *Failure Mode Effects Analysis*) procesą;
   1.4. esamus sprendimus gedimų medžio analizei atlikti;
2. aprašyti gedimų medžio analizės atlikimo metodiką modeliais grįstoje architektūroje;
3. suprojektuoti siūlomo sprendimo prototipą, jį realizuoti ir ištestuoti;
4. atlikti sukurtam sprendimui eksperimentą, jį įvertinti ir parašyti išvadas;

Darbo rezultatai suteikia naujas galimybes atlikti kompiuterizuotą gedimų medžio analizę kartu su atsekamumo ryšiu tarp kuriamos sistemos modelio. Pasinaudojant SysML išplėstu profiliu ir biblioteka galima atlikti kiekybinę gedimų medžio analizę, kuri leidžia paskaičiuoti nenorimų įvykių tikimybes.

Darbą sudaro septyni skyriai. Pirmajame skyriuje yra atlikta darbo analizė, kurioje yra aptarta gedimų medžio analizės samprata, kūrimo procesas, svarba. Atlikta analizė modeliais grindžiamai sistemų inžinerijai, jos standartams ir praktiniam taikymui. Antrajame skyriuje yra aprašytas sukurtas koncepcinis sistemos modelis, kuris leidžia turėti *SysML* modelio pagrindą, kuriam bus taikomas sprendimas. Šio modelio pagalba pagalba buvo daroma sprendimo paieškos ir projektavimas, generuojamos idėjos kaip *FTA* procesas turėtų sietis su sistemos modeliu. Taip pat, šiame skyriuje buvo apsibrėžti reikalavimai bei formalus sprendimo aprašas. Trečiame skyriuje aptariama sprendimo realizacija bei jo testavimas. Ketvirtajame skyriuje yra aprašytas eksperimentas bei jo rezultatai. Darbo pabaigoje yra penktasis skyrius, kuriame pateikiami darbo rezultatai ir išvados. Šeštasis skyrius sudaro literatūros sąrašą, o septintasis skirtas darbo priedams.

1. GEDIMŲ MEDŽIO ANALIZĖS TYRIMAS

Analizės skyrių sudaro: analizės tikslo įvardijimas, apibrėžto tyrimo objekto, tyrimo srities bei problemos nustatymas. Šiame skyriuje paaiškinta gedimų medžio analizės samprata, jos procesas. Taip pat aptarta modeliais grindžiamos sistemų inžinerijos svarba. Išanalizuota *SysML* kalba ir atlikta nesėkmės modelio ir pasekmių analizės apžvalga. Išanalizuoti naudotojai, jų poreikiai. Padaryta esamų sprendimo būdų analizė.

1.1. Analizės tikslas

Šios analizės tikslas – apibrėžti kas yra gedimų medžio analizė (angl. *Fault Tree Analysis – FTA*), kokius artefaktus ji apima, kokie gedimų medžio analizės kūrimo procesai bei kaip jie integruojasi į modeliais grindžiamą sistemų inžineriją. Kitas analizės tikslas – suprasti kas yra *SysML*, kam jis naudojamas ir susipažinti bei perprasti pagrindinius *SysML* metodus, artefaktus ir diagramas. Palyginti komplemetinę analizę gedimų medžio analizei – nesėkmės modelio ir pasekmių analizę (*FMEA*). Išanalizuoti naudotojus bei esamus sprendimus *FTA* atlikti.

1.2. Tyrimo objektas, sritis ir problema

Šio darbo **tyrimo sritis** yra gedimų medžio analizės modeliavimo metodai ir įrankiai, gedimų medžio sąsaja su modeliais grindžiama sistemų inžinerija (angl. *Model based system engineering – MBSE)* sistemos modeliu. **Tyrimo objektas** – gedimų medžio analizės proceso integracija į sistemos modelius.

Problema. Šiuo metu sistemų inžinerijoje kuriant sudėtingas sistemas nėra užtikrinamas atsekamumas tarp gedimų medžio analizės ir kuriamo sistemos modelio. Vykstant projektavimo procesui yra kuriamos dvi atskiros dalys: sistemos modelis ir rizikos analizė. Kadangi šie etapai yra dvi atskiros dalys – tai tarp komponentų neįmanoma užtikrinti atsekamumo ryšio. Šio ryšio nebuvimas gali sukelti grėsmes ar kitus pavojus, kai sistema bus sukurta ar atnaujinta. Taip pat, abiems dalims yra naudojami du skirtingi programinės įrangos įrankiai. Iš to matyti, kad, norint atlikti suprojektuotos sistemos gedimų medžio analizę, vėl tenka modeliuoti tuos pačius jau sumodeliuotus sistemos elementus, o tai kainuoja papildomus kaštus ir laiko. Dėl šios priežasties taip pat padidėja žmogiškoji rizika padaryti daugiau klaidų.

1.3. Tyrimo objekto

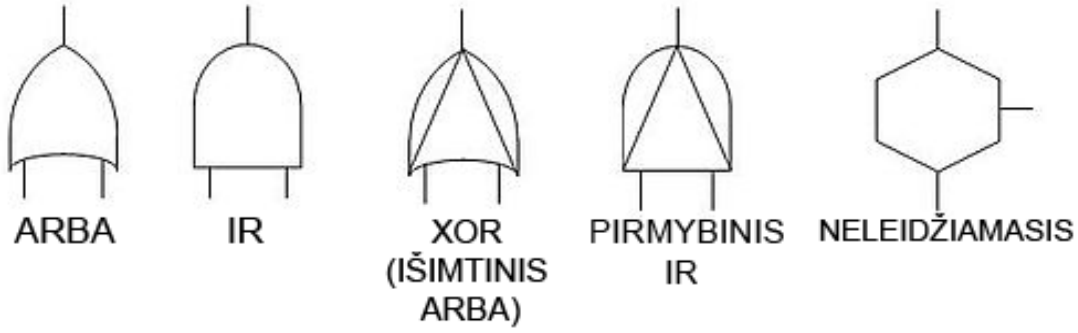
1.3.1. Gedimų medžio analizė

Gedimų medžio analizė (*FTA*) – vienas plačiausiai naudojamų patikimumo analizės ir kiekybinio rizikos įvertinimo būdų, kuris įvertina sistemos gedimo riziką [1]. *FTA* metodas susideda iš loginių schemų kūrimo, kurios yra skirtos atvaizduoti gendančius komponentus. Pastarieji komponentai visuomet turi galimybę sugesti, o jų sugedimas gali sukelti (specifinę) klaidą ar gedimą, kuris sustabdys visos sistemos veikimą. Gedimų medžio analizė paprastai yra sudaroma iš dviejų dalių: kokybinė (kurios tikslas yra identifikuoti ir sudaryti kuo mažesnę sugedimo įvykio tikimybę) ir kiekybinė (kurios tikslas apskaičiuoti nenorimo įvykio tikimybę).

Gedimų medžio analizė yra grindžiama grafiniais modeliais (diagramomis), kurios pagrindiniai elementai - loginiai vartai (angl. *logic gates*) ir gedimų įvykiai (angl. *fault events*) [2]. Diagramas taip pat gali sudaryti ir perdavimo ryšiai (angl. *transfer*).

Loginiai vartai nurodo santykį (bulio algebros, angl. *boolean*) ir ryšį tarp gedimo įvykių. Gedimų medžio analizėje yra naudojami penki loginių vartų simboliai (1.1 pav.) :

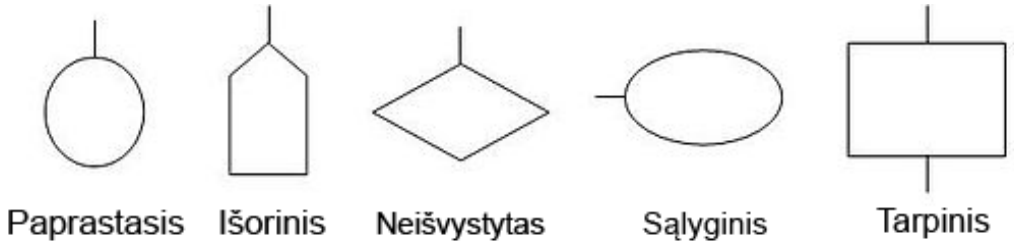
- „ARBA“ loginiai vartai (angl. *OR gate*) – išvestis įvyksta tuo atveju, jei įvyksta bet kuris kitas įvesties įvykis.
- „IR“ loginiai vartai (angl. *AND gate*) – kur išvestis įvyksta tuo atveju, jei įvyksta visi sujungti įvesties įvykiai.
- „IŠIMTINIS ARBA“ (angl. „*Exclusive OR*“ arba *XOR*) – išvestis įvyksta tuo atveju, jei įvyksta vienas ir tik vienas įvesties įvykis (jei yra sujungti du).
- „PIRMYBINIS IR“ (angl. „*Priority AND*“ gate) – išvestis įvyksta, jei įvesties įvykiai įvyksta specialiai numatyta seka, kuri yra nurodyta prie sąlyginio įvykio
- „NELEIDŽIAMASIS „ (angl. „*Inhibit gate*“) – išvestis įvyksta, jei įvestis įvyksta, kai ją įgalina sąlyginis įvykis.



1.1 pav. Loginių vartų simboliai

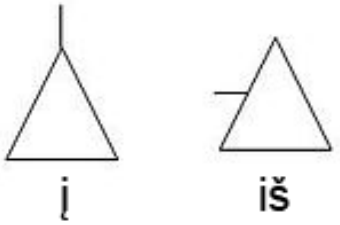
Gedimų įvykiai nurodo įvykį (paprastai gedimą ar tragediją), kurie yra jungiami su vartų logika. Gedimų medžio analizėje paprastai yra naudojami penki įvykių diagramos tipai (1.2 pav.). Įvykiai taip pat yra skirstomi į pirminius ir antrinius. Pirminiai įvykiai paprastai yra neskaldomi į mažesnius ir neturi įvesties ryšių. Antriniai įvykiai yra randami loginių vartų išeityje. Pagrindinis nenorimas analizuojamas įvykis yra žymimas pačiame viršuje ir šis neturi išeities įvykio.

- Paprastasis įvykis (angl. *Basic Event*) – klaida (gedimas) ar nesėkmė sistemos komponente ar elemente (pavyzdžiui: išsikrovė baterija).
- Išorinis įvykis (angl. *External Event*) – paprastai tikimasi, kad įvyks ne dėl savo kaltės (pavyzdžiui: lis lietus).
- Neišvystytas įvykis (angl. *Undeveloped Event*) – įvykis, apie kurį informacijos turime nedaug ir nežinome dėl ko jis įvyksta.
- Sąlyginis įvykis (angl. *Conditioning event*) – tai yra sąlygos, kurios apriboja ar daro įtaką loginiams vartams.
- Tarpinis įvykis (angl. *Intermediate event*) – tai yra įvykis, kuris gali būti naudojamas jungiant su pirminiu įvykiu leidžiant jį aprašyti.



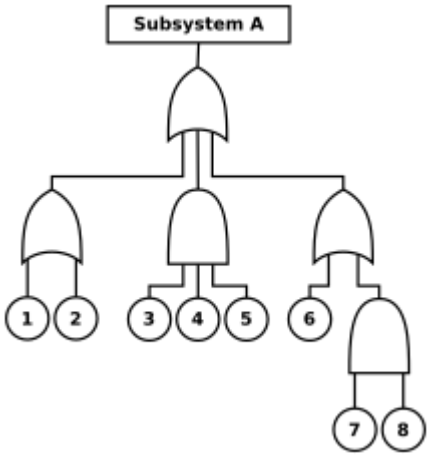
1.2 pav. Įvykių simboliai gedimų medžio analizėje

Perdavimo ryšiai skirti sujungti posistemes (subdiagramas) į didesnes sistemas. Tai yra tiesiog nuorodos, kurios nurodo, kad posistemė toje vietoje yra sutraukiama ar išskleidžiama. Perdavimo ryšiai yra atvaizduojami dviem būdais: „į“ (angl. *in*) ir „iš“ (angl. *out*) (1.3 pav.).



1.3 pav. Perdavimo ryšių simboliai gedimų medžio analizėje

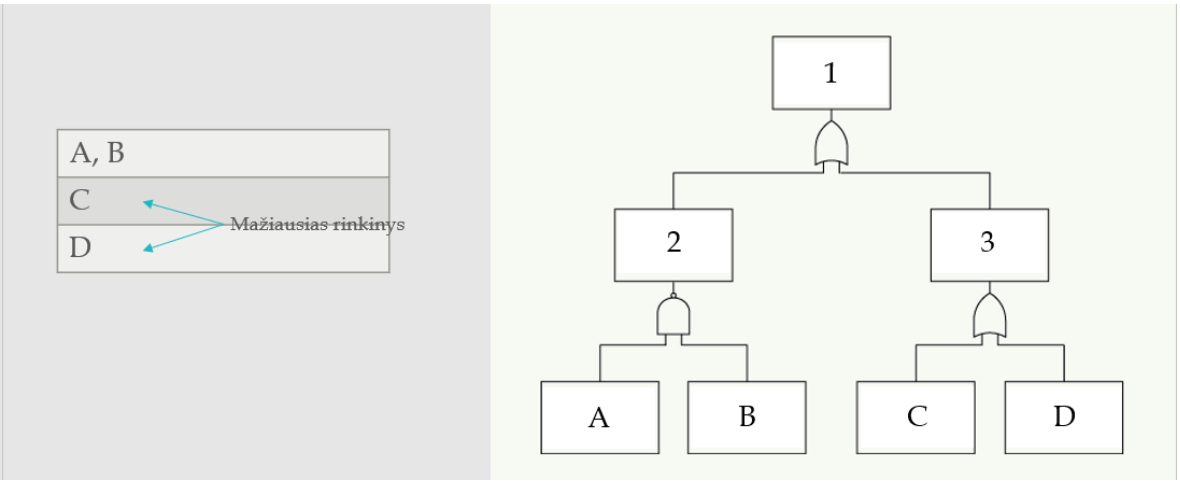
Sujungus šiuos išanalizuotus elementus galima gauti jau pilnas gedimų medžių analizės diagramas (1.4 pav.).



1.4 pav. Gedimų medžio analizės diagrama

Kiekvieną medžio analizę galima skirstyti į dvi dalis: į kokybinę (angl. *qualitative*) ir kiekybinę (angl. *quantitative*).

Kokybinis vertinimas susideda iš įvykimo rinkinių sudarymų (angl. *cut sets*). Rinkiniai sudaromi sudarant sąrašą (1.5 pav.), kurie įvykiai turi įvykti tam, kad būtų pasiektas pagrindinis nenorimas įvykis (angl. *top event*).



1.5 pav. Įvykimo rinkinys su gedimų medžio analizės diagrama

Kokybinio gedimų medžio analizė susideda iš trumpiausio (mažiausio rinkinio) kelio nustatymo, kad įvyktų įvykis ir dažniausiai įvykstančių priežasčių nustatymo. Du iš populiariausių būdų norint nustatyti mažiausius rinkinius gedimų medžiams yra Monte Carlo simuliacija ir deterministinis metodas [3].

Monte Carlo simuliacijos procedūra yra skirta surasti mažiausius rinkinius, pradžioje priskiriant atsitiktinį laiką iki nesėkmės kiekvienam komponentui. Tuomet būna simuliuojamos realios klaidos / problemos su egzistuojančiais probleminiais komponentais, tačiau su atsitiktiniu sistemos elgesiu. Šis metodas išaiškina problemą, kuri įvyksta su įvykių realiais veiksmais, bet simuliuotame laike. Vykdant Monte Carlo simuliaciją viskas vyksta su realistiškais scenarijais, nes realybėje problemų suplanuoti neįmanoma ir dažniausiai nutikimai atsitinka atsitiktinai [4].

Paprasčiausia idėja slypinti po deterministiniu metodu – tai yra tiesioginis praplėtimas ar sumažinimas gedimų medžio analizės pagrindinio nenorimo įvykio (angl. *top event*) remiantis paprasčiausiais įvykiais naudojant bulio (angl. *boolean*) algebrą [3].

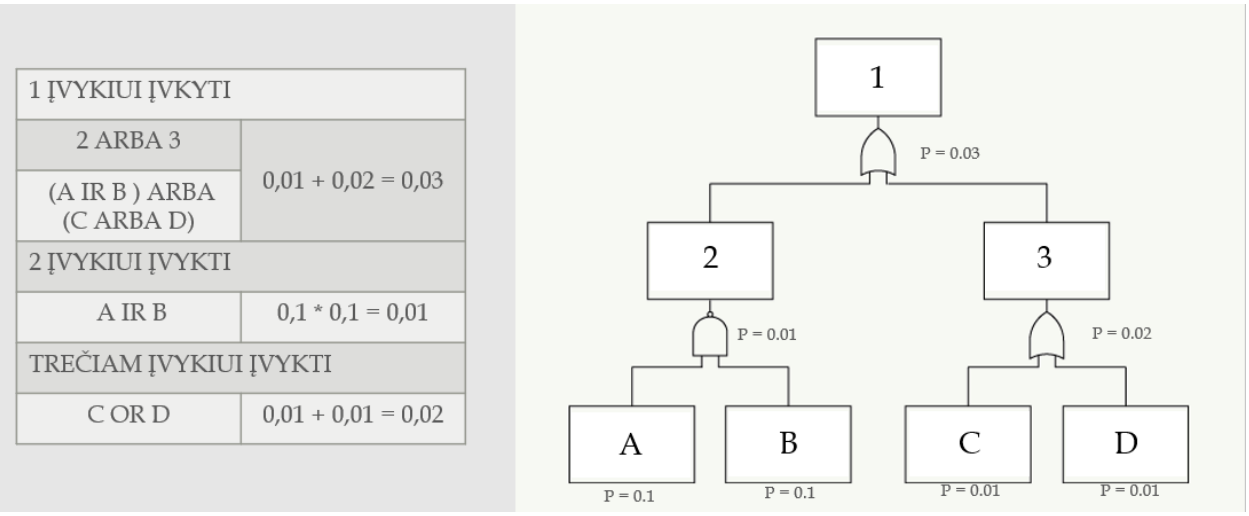
Kiekybinė gedimų medžio analizė susideda iš to, jog pirmiausia reikia susirasti mažiausią rinkinį (angl. *cut set*). Tuomet, jei yra žinoma visų paprastųjų įvykių (esančių analizės apačioje) įvykimo tikimybė – tuomet naudojantis algebrine logika galima apskaičiuoti kiekvieno egzistuojančio analizėje įvykio tikimybę [3]. Šitokiu būdu galima paskaičiuoti ir analizuojamo pagrindinio nenorimo įvykio tikimybę bei statistiką.

Paprastai viršutinių elementų tikimybės yra skaičiuojamos algebriniu principu (1.6 pav.):

IR vartai: P (A and B) = P (A ∩ B) = P(A) P(B). Paprasčiau matematiškai galima užrašyti, jog tikimybės tiesiog susidaugina.

ARBA vartai: P (A or B) = P (A ∪ B) = P(A) + P(B) - P (A ∩ B) . Dėl to, kad P (A ∩ B) dalis yra ženkliai mažesnė už P(A) + P(B) dalį, paprastai yra traktuojama, kad P (A ∩ B) = 0, tuomet formulė yra tiesiog P(A) + P(B).

XOR vartai: P (A xor B) = P(A) + P(B) - 2P (A ∩ B). Naudojamas tas pats principas, jog formulės dalis 2P (A ∩ B) ir daug mažesnė už P(A) + P(B) dalį, neretai traktuojama, jog 2P (A ∩ B) yra lygu 0 ir formulė susiprastina iki to paties P(A) + P(B). [5]



1.6 pav. Logikos pavyzdys norint apskaičiuoti tikimybes aukštesniojo lygio įvykių

1.3.2. Modeliais grindžiama sistemų inžinerija ir *SysML* kalba

Ilgą laiką programinės įrangos ir sistemų inžinerijoje dviejų žodžių kombinacija „krioklys“ ir „didysis sprogimas“ (angl. *big bang*) buvo labai populiari. „Krioklys“ – sistemų kūrimo būdas, kuris leidžia vizualizuoti ir susiprojektuoti kuriamą sistemą, o tik tuomet pradėti ją kurti. „Didysis sprogimas“ reiškia, kad kai projektavimas jau įvykdytas – sistemą galima bus sukurti per vieną iteraciją. Šie du principai gali atrodyti patraukliai dėl kelių priežasčių: paprasta paaiškinti, nuoseklu ir logiška, numatomas trumpas kuriamos sistemos terminas [6].

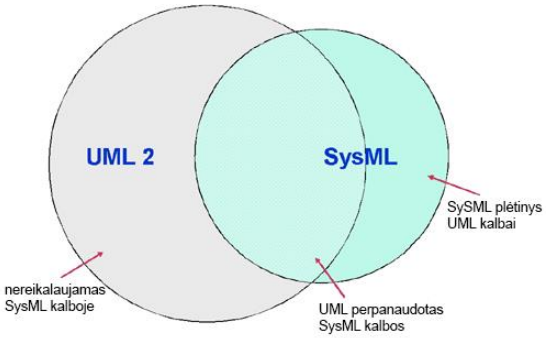
Tačiau praktika rodo, kad sudėtingų ir kompleksiškų sistemų suprojektuoti iš karto yra neįmanoma, nes didelė dalis apribojimų išaiškėja kūrimo (angl. *development*) etape. Pilni reikalavimai pradinėje kūrimo stadijoje taip pat gali būti nežinomi ir užsakovai juos visada gali pakeisti, kai gaus progą patys išbandyti sukurtą sistemos modelį, prototipą ar jau užbaigtą produktą [6].

Objektinis inžinerijos metodas (angl. *object-oriented systems engineering method – OOSEM*) yra modeliais grindžiamos sistemų inžinerijos pagrindas [7]. Modeliais grindžiama architektūra remiasi tuo, kad viską ką jau sukūrėme – galime perpanaudoti. Perpanaudojant sukurtus artefaktus galima stipriai sumažinti sistemų kūrimo kaštus. Taip pat, pasikeitus reikalavimams, nėra būtinybės visko perprojektuoti iš naujo, modeliais grindžiama architektūra leidžia išlaikyti nuoseklumą projekte: pakeičiant artefaktą vienoje vietoje – jis pasikeis visame projekte.

Kitas svarbus privalumas modeliais grindžiamoje sistemų inžinerijoje yra simuliacijos galimybė. Sudėtingos sistemos, kurios yra kuriamos be modeliavimo kainuoja daug papildomų kaštų, nes tenka kurti prototipus bei maketus (angl. *mock-up*). Sukurti realų maketą (pavyzdžiui: erdvėlaivio) ir jį ištestuoti neretai kainuoja milijonus. Modeliais grindžiamoje sistemų inžinerijoje simuliacija leidžia ištestuoti sukurtą lėktuvą virtualiai. Tokiu būdu yra sutaupoma labai daug finansų bei laiko.

Modeliais grindžiama sistemų inžinerija remiasi ne tik modeliavimo kalbos notacija, tačiau ir savo metodika bei karkasais. *MBSE* metodologija – priemonė, kuri pasako nuo ko reikėtų pradėtų modeliuoti, kaip sukurti modelio struktūrą, kokius artefaktus pasirinkti, kokia tvarka ir t.t. Norint užtikrinti *MBSE* sėkmę, modeliavimo kalbą būtina susieti su tam tikra metodologija [8]. Karkasas nurodo kokiais pjūviais reikia atvaizduoti kuriamą modelį: funkciniu, struktūriniu aspektais, kūrimo fazių ir perėjimo (evoliucijos) aspektais [9].

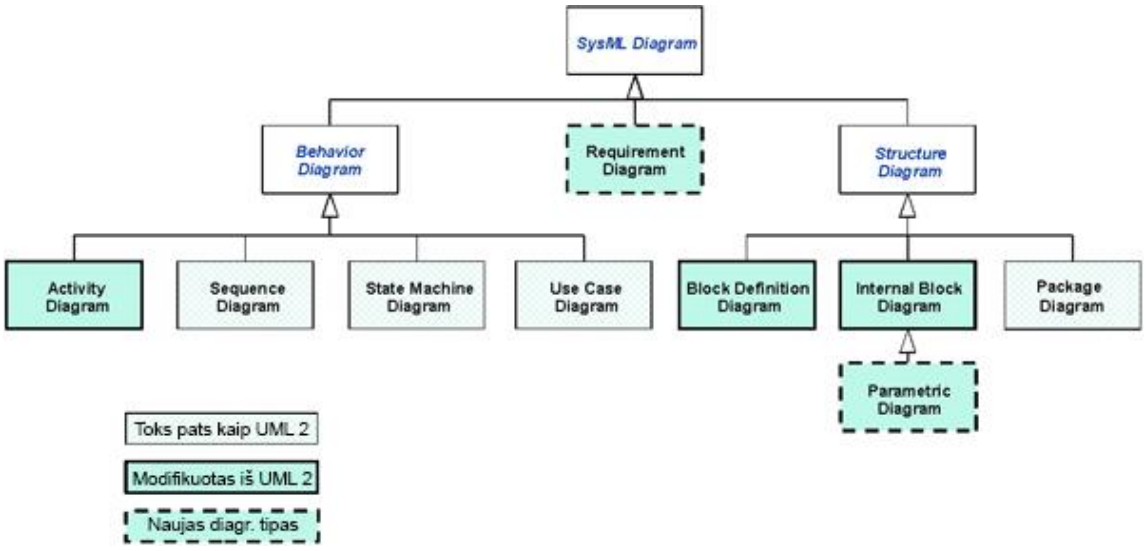
SysML (angl. *Systems modelling language*) – grafinė modeliavimo kalba skirta modeliuoti techninę ir programinę įrangą, duomenų srautus. Šios kalbos pagalba taip pat galima galima apsibrėžti sistemos proceso specifikaciją, dizainą, analizę ir verifikavimą [10]. Šis darinys kilo iš kitos plačiai žinomos kalbos - *UML 2*. *SysML* kalba specialiai buvo sukurta kaip *UML* 2 praplėtimas (1.7 pav.), nes *UML* žodynas yra per daug orientuotas tik į programinės įrangos kūrimą (objektai, klasės ir t.t.).



1.7 pav. *UML 2* ir *SysML* ryšys [11]

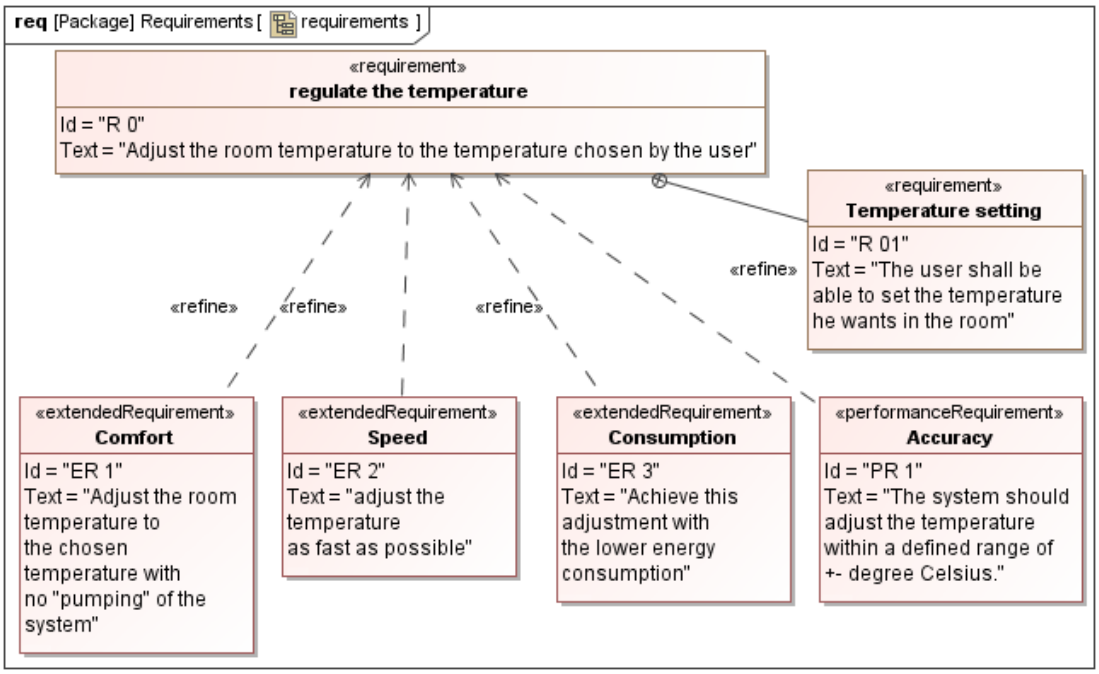
SysML pirmoji versija buvo išleista 2006 metų birželio mėnesį OMG kompanijos. *SysML* pateikia 9 skirtingus diagramų tipus, iš kurių keturi yra identiški *UML 2* modeliavimo kalboje, trys yra modifikuoti iš *UML 2*, o dvi diagramos yra visiškai naujo tipo.

Visiškai identiškos diagramos yra: sekų (angl. *sequence*), būsenų (angl. *state machine*), panaudojimo atvejų (angl. *use case*) ir paketų (angl. *packages*). Modifikuotos diagramos yra: veiklos (angl. *activity*), bloko aprašymo (angl. *block definition*) – pakeičia klasių (angl. *class*) diagramą, vidinio bloko (angl. *internal block*), pakeičia kompozicijos (angl. *composite structure*) diagramą. Naujos diagramos yra: reikalavimų (angl. *requirement*) ir parametrikų (angl. *parametric*) [11] (1.8 pav.).



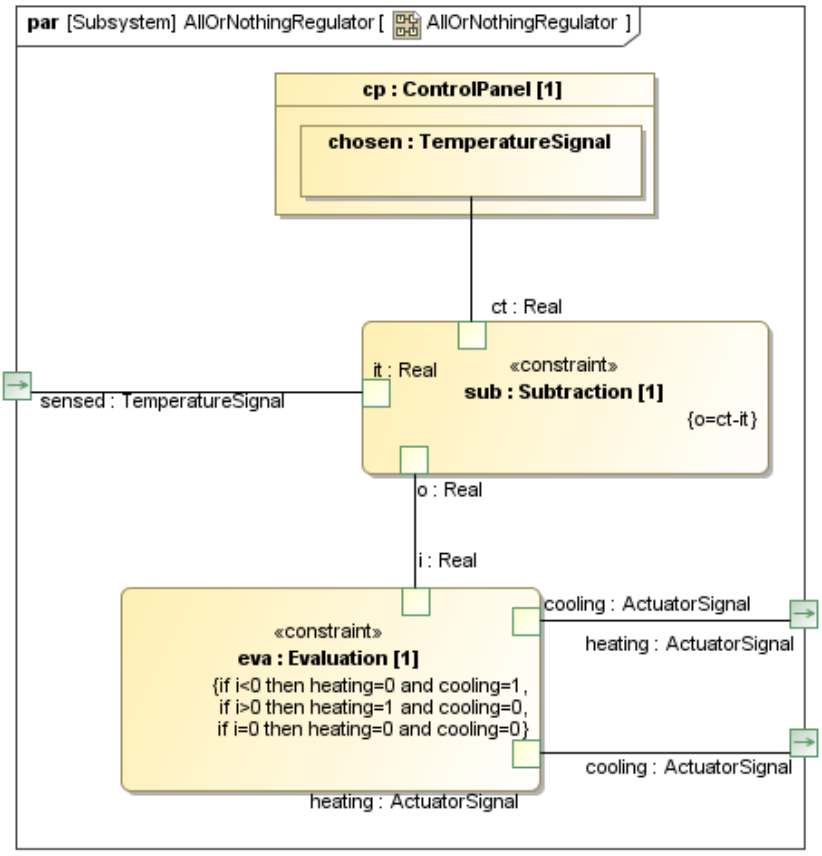
1.8 pav. *SysML* diagramų tipai [11]

SysML reikalavimų diagrama (1.9 pav.) – skirta atvaizduoti reikalavimus tekstiniu pavidalu diagramose. Sukurtas reikalavimas paprastai atvaizduoja sąlygas, kurias sistema privalo tenkinti arba atvaizduoja funkcijas, kurias sistema privalo atlikti. Jų hierarchija sudaro kuriamos sistemos reikalavimų specifikaciją.



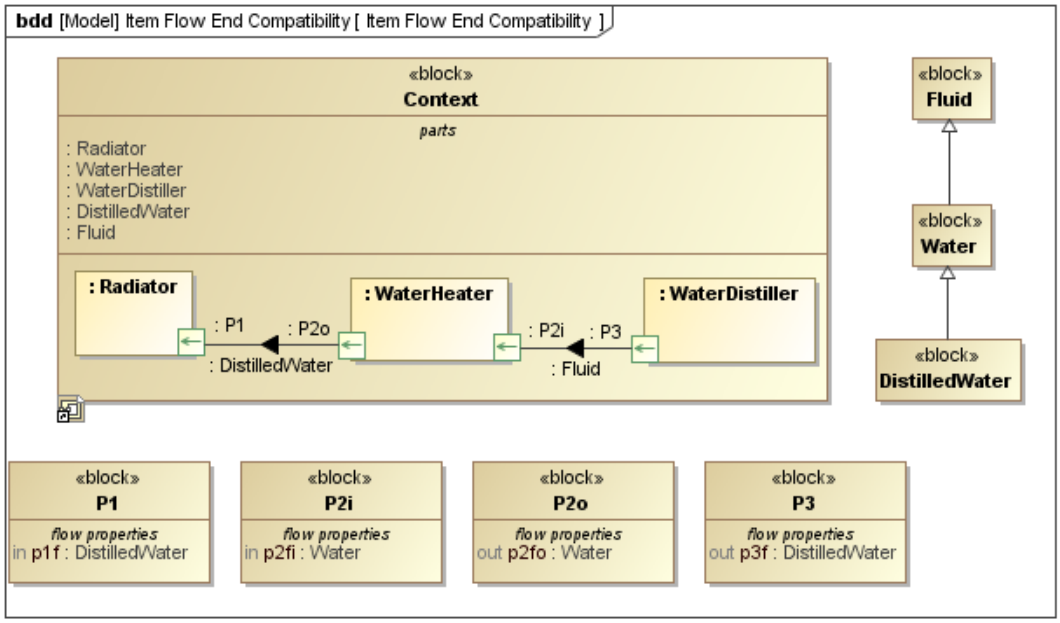
1.9 pav. *SysML* reikalavimų diagrama [12]

SysML parametrikų diagrama – skirta padėti sistemų analizei (įvykdymui, patikimumui ir t.t.) aprašant apribojimus blokams. Ji yra tiesiogiai susijusi su vidinė bloko diagrama (angl. *internal block diagram*) ir tik sukūrus šią diagramą galima plėtotis į parametrikų diagramą skirtą rašyti apribojimams bei kaip blokai bendraus tarpusavyje, bei kokiais parametrais(1.10 pav.).

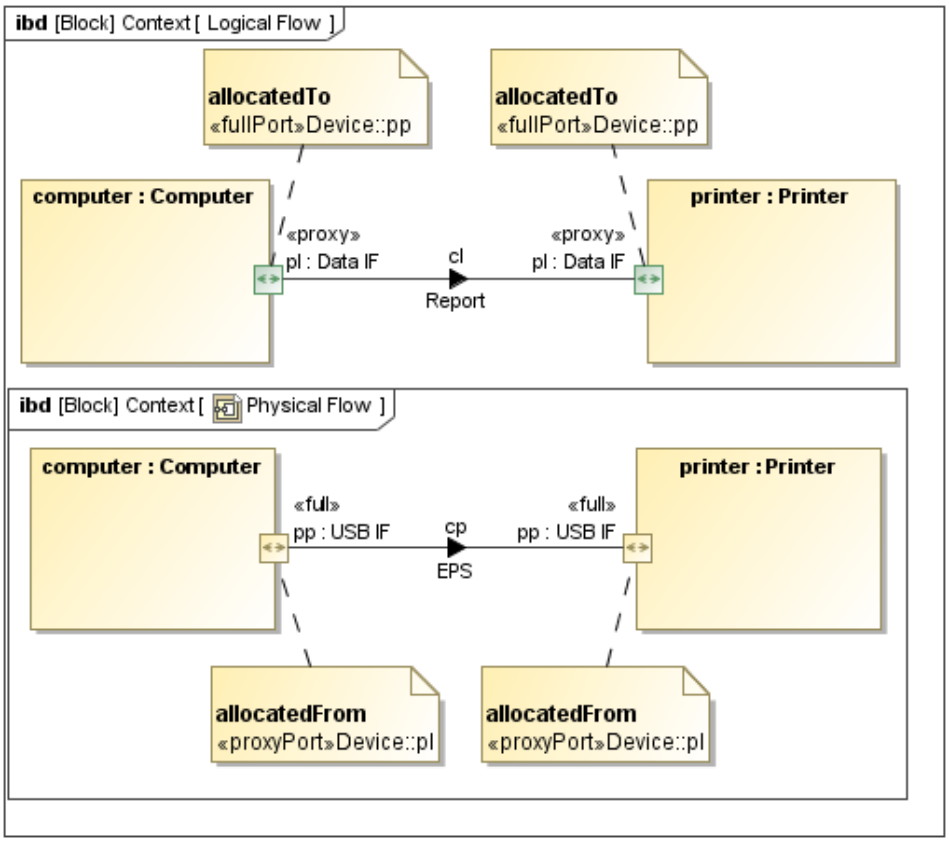


1.10 pav. *SysML* parametrikų diagrama [13]

Šiame darbe svarbu suprasti kaip galima būtų įdiegti atsekamumo ryšį tarp *SysML* modelio ir gedimų medžio analizės. Dėl to, jog gedimų medžio analizė paprastai yra atliekama sistemos komponentams, šis ryšys turėtų sieti blokus (*SysML* komponentus), kurie atvaizduoja realaus pasaulio sistemos dalis. Blokų aprašymo diagrama (1.11 pav.) yra skirta aprašyti sistemos hierarchiją iš komponentų, o vidinė bloko aprašymo diagrama (1.12 pav.) aprašo vidinę sistemos struktūrą iš jos dalių, portų ir jungiklių (angl. *parts, ports and connectors*) [11].



1.11 pav. *SysML* blokų aprašymo diagrama (angl. *block definition diagram*) [14]



1.12 pav. *SysML* vidinė blokų aprašymo diagrama (angl. *internal block diagram*) [15]

Analizuojant komponento gedimo priežastis galime naudoti gedimų medžio analizę ir kiekvienam elementui sukurti atskirą *FTA*. Tačiau sukurtų technologijų užtikrinti ryšiui tarp sukurtos analizės ir sumodeliuoto sistemos bloko nėra. Dėl šios priežasties tenka pasikliauti papildomomis programomis, kurios tekstiniu pavidalu aprašo atsekamumą tarp šių dviejų komponentų (*SysML* bloko ir *FTA*), tačiau tai kainuoja daug papildomų kaštų ir didėja galimybė atsirasti žmogiškosioms klaidoms.

1.3.3. Nesėkmės modelio ir pasekmių analizė

Nesėkmės modelio ir pasekmių analizė (angl. *FMEA – Failure mode and effects analysis*) – kitas dažnai naudojamas metodas, kuris yra skirtas atlikti sistemos veikimo kokybės analizę. Nors šis analizavimo procesas oficialiai buvo skirtas tik saugumo analizės sutvirtinimui, tačiau jis greit tapo universaliai priimtinu ir pripažįstamu įrankių atlikti patikimumo užtikrinimui ir rizikos valdymui [16].

Skirtingai nei gedimų medžio analizėje (*FTA*) – visas požiūris į sistemos geidimus vyksta iš kitos pusės. *FTA* remiasi „Iš viršaus į apačią“ (angl. „*Top-Down*“) technika, o *FMEA* remiasi „Iš apačios į viršų“ (angl. „*Bottom-Up*“) technika. Tai reiškia, jog pirmiausia orientuojamės į priežastis, kurios gali nutikti sistemai (įvairios klaidos, komponentų sugedimai ir t.t.), o po to bandome atsekti kokią pasekmę komponentų sugedimas turės analizuojamoje sistemoje.

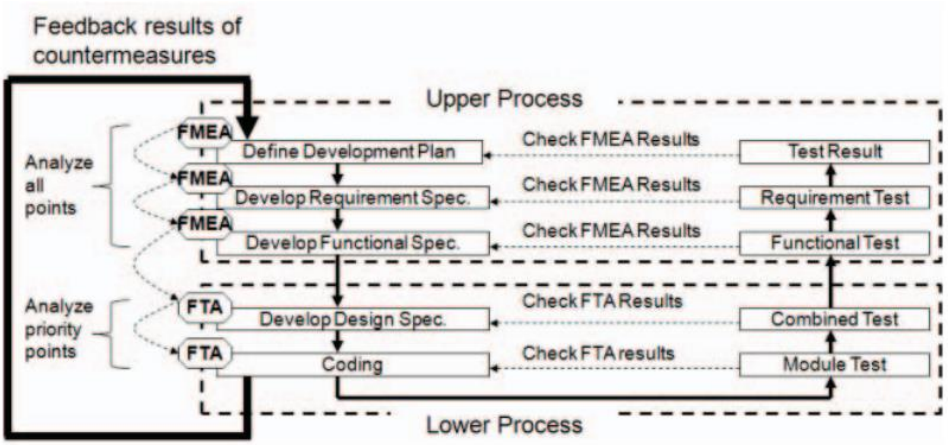
FMEA paprastai yra atvaizduojama ne diagrama, o lentele (1.13 pav.). Kiekviena lentelės eilutė reprezentuoja tiriamą objektą, jo klaidų (neveikimo) priežastis ir kitus atributus (tiriamo įvykio svarbą, prevencijos priemones, atsakingus žmones, datas, rezultatus ir t.t.).

#	Name	Item	Failure Mode	Local Effect Of Failure	Final Effect Of Failure	SEV	Cause Of Failure
1	F1	airbag : Airbag	Bag does not open on impact		Injure Passenger	4.0	Sensor is not functioning properly Broken wire Controller is not functioning prope
2	F2	light : Light	Light does not turn on	Car inoperable at night Car inoperable under ba		3.0	Battery dead
3	F3	light : Light	Light does not turn on	Car inoperable at night Car inoperable under ba		3.0	Broken wire
4	F4	light : Light	Light does not turn off	Car won't start		3.0	Short circuit in switch
5	F5	light : Light	Light does not turn on	Car inoperable at night Car inoperable under ba		3.0	Headlight out
6	F6	light : Light	Light does not turn off	Car won't start		2.0	Operator error (left on)
7	F7	light : Light	Light does not turn on	Car inoperable at night Car inoperable under ba		2.0	Switch broken
8	F8	light : Light	Light does not turn on	Car inoperable at night Car inoperable under ba		2.0	Switch corroded

1.13 pav. *FMEA* analizės pavyzdys [17]

Šio darbo tiriamasis objektas yra *FTA*, todėl svarbu pabrėžti kaip šios dvi analizės dera tarpusavyje. Japonijos mokslininkai teigia, jog norint užtikrinti didesnį saugumą ir sukurti mažesnę riziką *FMEA* analizė ir *FTA* analizės turi būti atliekamos kartu, tačiau neatsižvelgiant į viena kitą (1.14 pav.).

Viršutiniame sistemos kūrimo sluoksnyje (plano sudarymo, reikalavimų specifikavimo, funkcinių reikalavimų specifikavimo dalyse) turi būti analizuojami punktai remiantis nesėkmės modelio per pasekmių analize (*FMEA*). Tuomet, kai yra pasiekiamas apatinis sistemos kūrimo sluoksnis (realizavimas), tuomet kiekvienas kuriamas komponentas turi būti tiriamas ir analizuojamas remiantis gedimų medžio analizės principais. Nors *FTA* yra atliekama vėliau (apatiniame sistemos kūrimo etape), tačiau jos rezultatus pamatome greičiau, nes ištestuoti ką tik realizuotus komponentus bei jų derėjimą galima dar pilnai nesukūrus visos sistemos. Tik tuomet, kai sistemos kūrimas praeina visą *FTA* analizę ir prasideda surinktų ir išpildytų reikalavimų testavimas, tuomet galime pamatyti *FMEA* analizės rezultatus [18]. Tokiu būdu atliekant atskiras dvi analizes iš dviejų skirtingų požiūrių (nuo viršaus į apačią ir nuo apačios-žemyn) galima atrasti daug daugiau klaidų, nei atlikus vieną analizę ir tik viename sistemos kūrimo etape.



1.14 pav. *FTA* ir *FMEA* kartu [18]

Norint efektyviai išnaudoti nesėkmės modelio ir pasekmių analizės ir gedimų medžio analizės rezultatus reikalingas galintis kurti sistemos modelį, *FTA* ir *FMEA* analizes įrankis (arba keli integruoti įrankiai), kuris gebėtų apjungti šiuos punktus (turėti atsekamumo ryšius).

1.4. Tyrimo objekto naudotojų analizė

Sukurtas patikimas ir saugus produktas ar sistema yra kone kiekvienos įmonės vienas iš svarbiausių tikslų. Norint gauti produktą, kuris bus patikimas – reikalingi specialistai galintys analizuoti sistemos keliamas grėsmes. Analizuoti gali darbuotojai, kuriems yra svarbus gedimų medžio analizės procesas. Dažniausiai šie specialistai yra vadinami kokybės užtikrinimo inžinieriais ar patikimumo užtikrinimo inžinieriais. Norint gerai suprasti saugumo spragas reikia gerai išmanyti dalykinę sritį, taip pat turėti aukštą kvalifikaciją kuriamo projekto srityje (pavyzdžiui, kuriant informacinę sistemą, reikia išmanyti informacinės sistemos kūrimo procesus). Būtent tokiems naudotojams yra svarbus tiriamas objektas – gedimų medžio analizė. Šis tyrimas atliekamas tam, kad būtų palengvinamas darbas atitinkamos profesijos atstovams.

Išanalizavus kokybės užtikrinimo inžinierių darbą buvo atrastos problemos su gedimų medžio analizės kūrimo procesu ir surašytos į 1.1. lentelę.

1.1 lentelė. Naudotojų problemų aprašymas

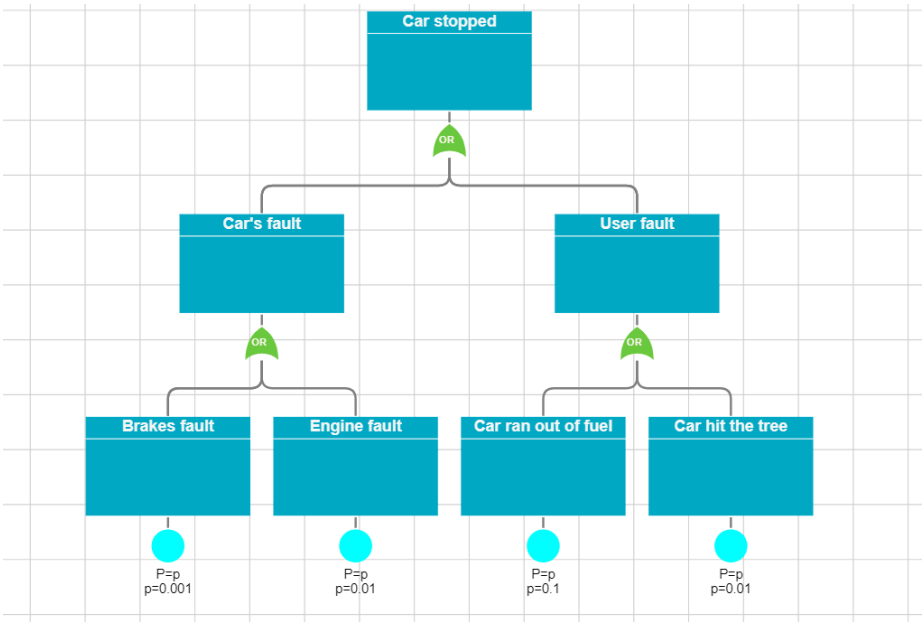
Problema	Kaip viskas vyksta dabar
Trūksta vieningos sistemos programinės įrangos sistemos projekto modeliui kurti ir gedimų medžio analizei atlikti.	Naudojama atskira programinė įranga gedimų medžio analizei atlikti ir atskira programinė įranga projekto modeliui kurti.
Nėra galimybės valdyti atsekamumo tarp projekto modelio ir gedimų medžio analizės	Atsekamumas paprastai aprašomas papildomuose dokumentuose tekstiniu pavidalu

1.5. Esamų problemos sprendimo metodų analizė

Esamų problemos sprendimo metodų analizei buvo pasirinkti įrankiai, kurie gali atlikti gedimų medžio analizę: „Fault Tree Analyser“, „HiP-HOPS“, „Isograph Reliability Workbench“, . Buvo išbandytas jų funkcionalumas, kuris buvo įvertintas pagal numatytus punktus.

1.5.1. „Fault Tree Analyser“

„Fault Tree Analyser“ (sukurtas „ALD Reliability Engineering LTD“) yra specialus įrankis, kuris yra skirtas tik gedimų medžio analizei atlikti. Šis įrankis turi dvi versijas: nemokamą, skirtą naudotis interneto naršyklėje (1.15 pav.) ir „Enterprise“ versiją, kuri yra skirta dirbti įmonėms, komandose. Mokama šios programos versijos kaina prasideda nuo 2290 JAV dolerių [19]. Už šią papildomą kainą prie vienos gedimų medžio analizės duomenų bazės gali dirbti keli žmonės, leidžia į vieną modelį dėti kelias gedimų medžio analizės diagramas, kombinuoti šias diagramas su perdavimo ryšių elementais (angl. *transfers*). Atlikinėti kitas saugumo ir patikimumo analizes (pvz. *FMEA*, *FMECA* ir t.t.).



1.15 pav. „Fault Tree Analyser“ pagalba sukurta gedimų medžio analizės diagrama

Nemokama įrankio versija yra skirta atlikinėti tik gedimų medžio analizei. Šios programinės įrangos pagalba galima skaičiuoti tikimybes gedimų medyje. Gedimų medžio analizę galima eksportuoti į XML failą. Įrankyje taip pat egzistuoja mygtukai „Import from“, kurie ateityje leis importuoti gedimų medžius iš kitų programų kaip „Isograph“ ar „Cafta“, tačiau šiuo metu tokios galimybės nėra.

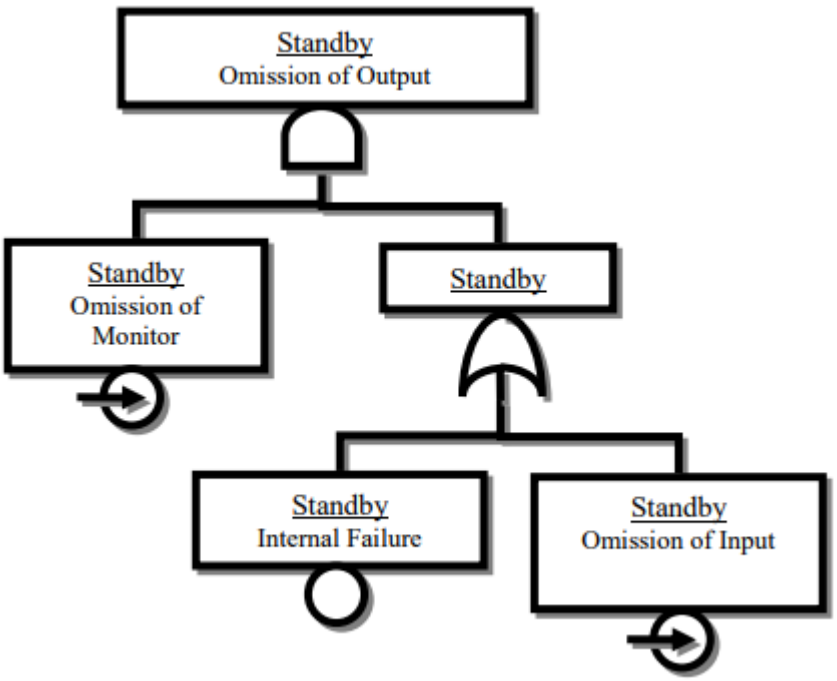
Naudotis programa yra labai paprasta, nes kiekvienas mygtukas ir spustelėjimas yra intuityvus ir suprantamas. Sukūrus gedimų medžio analizę galime padaryti ataskaitas, kurios rodys kaip tikslingai veikia mūsų gedimų medis: rinkinius (angl. *cut sets*), įvykių tikimybes.

Deja, įrankyje (net ir mokamoje versijoje) negalime kurti jokio sistemos modelio ar kažkaip kitaip matyti kuriamos sistemos komponentų, dėl to, jokio sąryšio tarp gedimų medžio ir sistemos modelio be papildomos dokumentacijos, būti negali.

1.5.2. „HiP-HOPS“

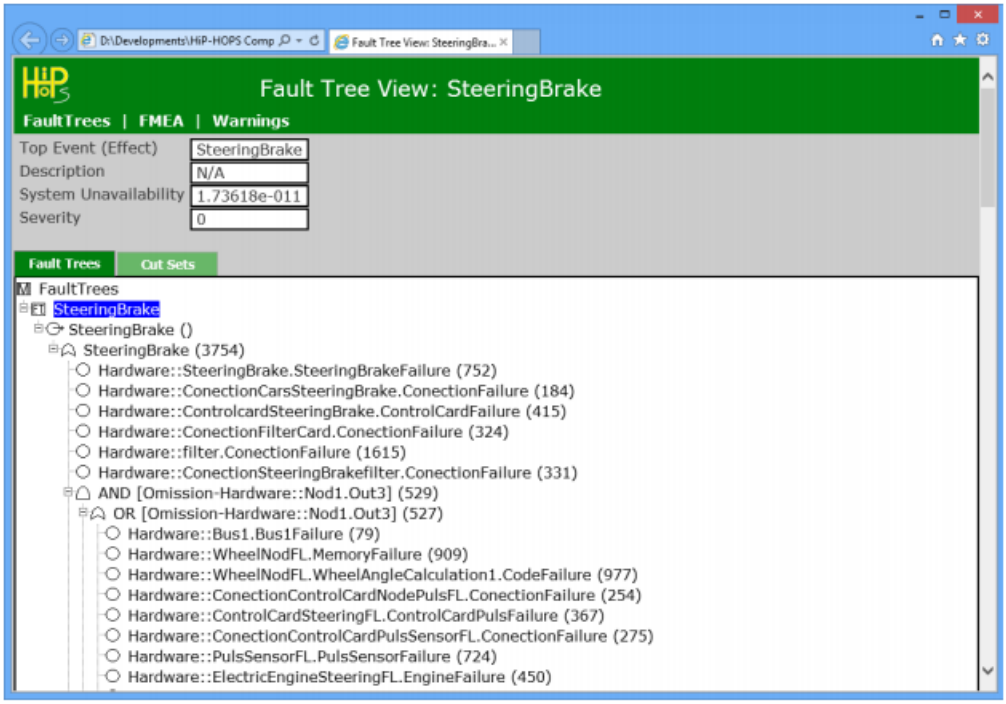
„HiP-HOPS“ – tai įskiepis skirtas programoms „Matlab Simulink“ ir „Simulation X“. Įdiegus šią programą-įskiepį kitos programos įgyja galimybes modeliuoti rizikos valdymo ir patikimumo diagramas (*FTA, FMEA* ir kitas). Šio įskiepio nemokama versija riboja apimtį iki 20 nemokamų komponentų, o jo įdiegimas yra gana sudėtingas.

Sukurta gedimų medžio analizė programoje „MatLab Simulink“ matoma 1.16 pav. Šis įrankis taip įgalina naudotojus skaičiuoti tikimybes tinkamai susikonfigūravus serverį kompiuteryje, tačiau tai tikrai sunkus ir kruopštus darbas ir veikiantis tik ant Windows operacinės sistemos.



1.16 pav. „HiP-HOPS“ Gedimų medžio analizė [20]

Visa analizė yra atliekama interneto naršyklėje (1.17 pav.) tam specialiai sugeneruotame įrankio puslapyje. Įrankis nėra nei patogus naudoti, nei paprastas, o ir skirtas jis tik patyrusiems IT specialistams išmanantiems ir „Windows“ operacinės sistemos konfigūracijas ir failų sistemos ypatybes. Naudotojo vadovą sudaro 43 puslapiai, o funkcionalumo lyginant su konkurentų programomis – mažai.



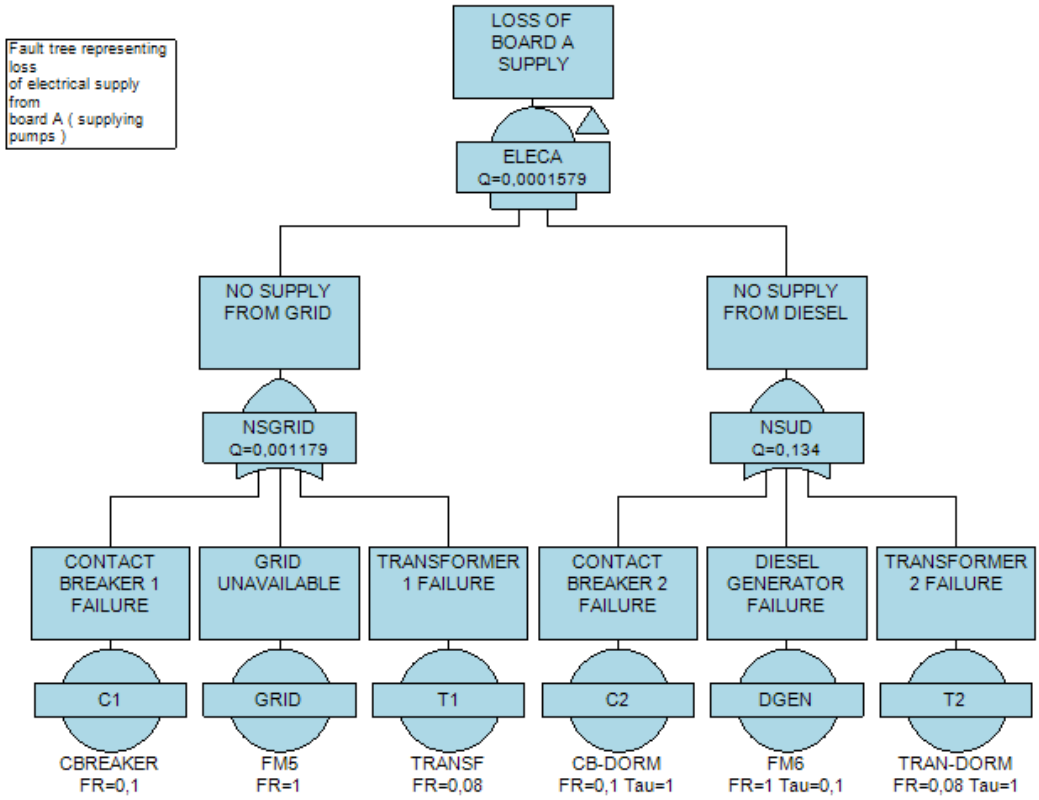
1.17 pav. „HiP-HOPS“ gedimų medžio analizės tikimybės interneto naršyklėje

1.5.3. „Isograph Reliability Workbench“

„Reliability Workbench“ – vienas iš lyderiaujančių patikimumo, saugos ir palaikomumo programinės įrangos įrankių. Ši programa yra kuriama jau nuo 1980 metų. Ji leidžia atlikt daugybę įvairių analizių: *FMECA, FMEA,* Veibulio (angl. *Weibull*) istorinę analizę ir kitas, o tarp jų ir gedimų medžio analizę [21].

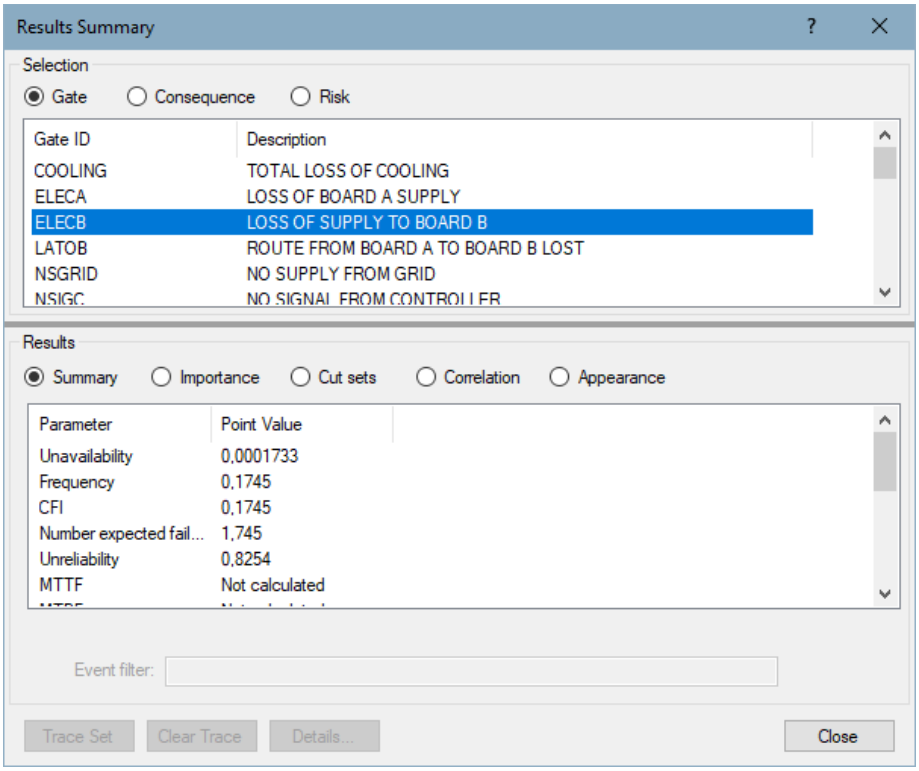
Naudotis programa nėra labai paprasta darbuotojui, kuris yra nesusiduręs su panašiais įrankiais (pvz. „MagicDraw“), nes naudotojo sąsaja nėra intuityvi ir norint pasiekti tikslą reikia praleisti nemažai laiko ieškant įvairių funkcijų. Atliekant gedimų medžio analizę (*1.18 pav.*) „Reliability Workbench“ įrankyje ją galima apžiūrėti iš rezultatų santraukos perspektyvos, kuri įgalina naudotoją peržiūrėti kiekvieno loginio varto tikimybę, dažnumą ir kitus aspektus, taip pat ir galimas pasekmes.

Programa nėra nemokama, o kaina yra neviešinama; dėl licenzijos reikia kreiptis į pardavimų skyrių. Bandomoji versija yra suteikiama tik su slaptažodžiu gautu el. paštu ir atsiunčiama per 24 valandas nuo užklausos pateikimo.



1.18 pav. „Isograph Reliability Workbench“ gedimų medžio analizės vaizdinys

Šiame įrankyje taip pat nėra galimybės kurti sistemos modelio kas padėtų įgalinti atsekamumą tarp gedimų medžio analizės ir kuriamo sistemos modelio, dėl to šis įrankis taip pat nėra tinkamas pagrindiniam šio darbo tikslui pasiekti.



1.19 pav. Rezultatų santrauka įrankyje „Reliability Workbench“

1.5.4. Gedimų medžio analizės įrankių palyginimas

Atliktą gedimų medžio analizės įrankių palyginimo analizę patogiausia yra matyti lentelės pavidale (1.2 lentelė).

1.2 lentelė. Sprendimų palyginimo lentelė

Lyginimo kriterijai	**Sprendimai**			
	„Fault Tree Analyser“	„HiP-HOPS“	„Reliability Workbench“	Kuriamas sprendimas
Galimybė modeliuoti *FTA*	+	+	+	+
Galimybė modeliuoti *FMEA*	+	+	+	+
Galimybė modeliuoti sistemos modelį (*SysML*)	-	-	-	+
Galimybė atlikti *FTA* tikimybių skaičiavimus	+	+	+	+
Galimybė matyti *FTA* statistiką	+	+	+	-
Įrankis savarankiškas (nepriklausomas nuo kitų programų)	+ / - (priklauso nuo interneto naršyklės)	-	+	-
Atsekamumo (angl. *traceability*) funkcionalumas	-	-	-	+
Nemokamas	+ / - (*FTA* analizės)	-	-	-
Lengva naudotis	Labai lengva	Labai sudėtinga	Vidutiniškai	Vidutiniškai

Kiekvienas tirtas įrankis veikia savo platformose (interneto naršyklėje, kitos programos pagalba arba kaip atskira programa). Visi tirti įrankiai gali modeliuoti *FTA*, *FMEA*, atliktų tikimybių skaičiavimus bei matyti statistiką. Ištirti sprendimai turi savo įkainius, tačiau vienintelis „Fault Tree Analyser“ tiesiogiai neapriboja *FTA* analizės funkcionalumo.

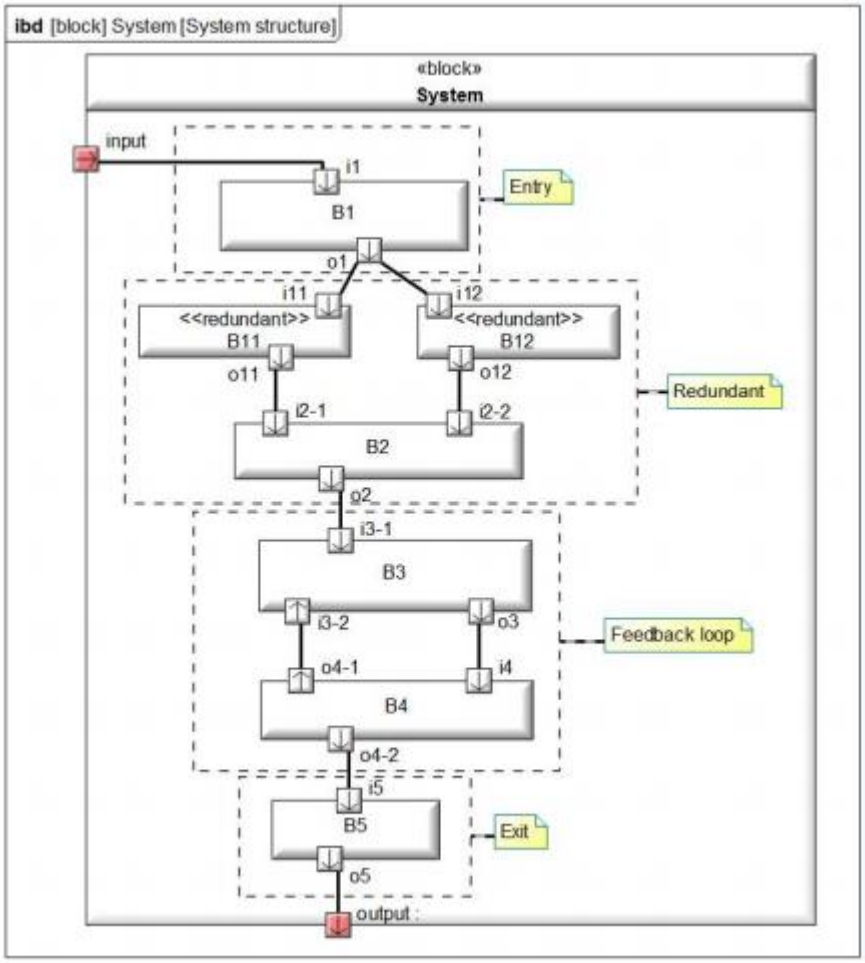
Nė vienas iš tiriamų gedimų medžio analizės įrankių neturėjo galimybės modeliuoti sistemos modelio, todėl nė vienas įrankis tiesiogiai negali atlikti darbe suteikto tikslo – įgalinti tiesioginio atsekamumo ryšių tarp gedimų medžio analizės elementų ir sistemos modelio elementų. Nė vienas iš šių trijų įrankių nepateikia galimybės susieti du modelio elementus juos apjungiant ryšiais (apart įvykio – loginių vartų

ryšio) arba susieti elemento su dokumentu ar kitu fiziniu failu (byla). Galbūt toks sprendimas įgalintų apėjimus (angl. *workaround*) atsekamumo srityje.

1.6. Modeliais grindžiama automatinė gedimų medžio analizė

Kritinės sistemos privalo turėti aukštus patikimumo, apsaugos, saugumo ir prieinamumo lygius. Dėl šios priežasties nusistovėję standartai rodo, jog paralelėje kartu yra sistemos inžinerinis modelis su įvairiomis saugos vertinimo veiklomis [22]. Mokslininkai buriasi į grupes ir siūlo savo sukurtus, tačiau skirtingus būdus kaip galima būtų automatizuoti patikimumo analizių procesą iš *SysML* kalba sumodeliuoto sistemos modelio. Vienas iš tokių būdų yra naudoti tarpinę modeliavimo kalbą „AltaRica“ su sukurtu saugumo modeliavimo karkasu skirtu gedimų medžio generavimui ir analizei (angl. *safety modelling framework for FT generation and analysis – SMF-FTA*) [22] ir sukurtais algoritmais generuoti statinę gedimų medžio analizę, tačiau naujesniais mokslininkų duomenimis galima atlikti tai paprasčiau.

Pagal [23] straipsnyje Paryžiaus Supméca mechanikos instituto mokslininkų pateiktą metodą yra būdas kurti gedimų medžio analizes pagal sukurtas *SysML* vidinė blokų diagramas (angl. *Internal Block Diagrams – IBD*).



1.20 pav. Vidinė blokų diagrama naudojama kaip pavyzdys aprašyti gedimų medžio analizės generavimo automatizavimo algoritmus [23]

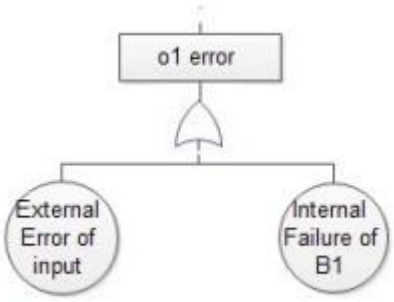
Atlikus šių dviejų straipsnių analizę (1.3 lentelė), paaiškėjo, jog paprastenis metodas generuoti gedimų medžio analizę yra naudojantis tiesogiai *IBD* diagramomis, nes toks metodas nereikalauja papildomų transformacijų į kitas kalbas.

1.3 lentelė. Automatinio *FTA* diagramų generavimo palyginimas

Kriterijai	**Naudojant *SMF-FTA*** [22]	**Naudojant *IBD*** [23]
Tarpinė transformacija	Dvi: į *AltaRica* ir *ARC*	Nereikalinga
Reikalinga papildoma anotacija sistemos modelyje	Taip	Ne
Reikalingos žinios	*AltaRica, SysML, FTA*	*SysML, FTA*

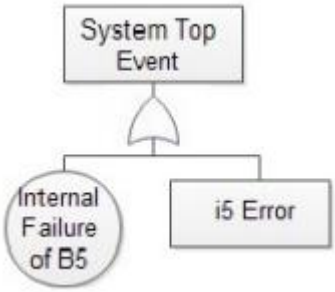
Vidinės bloko diagramos suteikia informacijos apie vidinę sistemos struktūrą, o juose yra komponentai sąveikaujantys per standartus ir srauto prievadus (angl. *ports*), o pastarieji būna sujungti su elementais vadinamais jungtimis (angl. *connectors*). Sugedus vienam iš komponentų, sekant šių jungčių logika galime matyti, kurie kiti komponentai pradės nebeveikti ar streikuoti. Pastebėję šį atsekamumą *SysML* diagramose mokslininkai [23] straipsnyje parašė keturis šablonus – metodus, kuriais remiantis reikėtų kurti automatines gedimų medžio analizes:

1) Įėjimo šablonas (angl. *Entry pattern*) – jei blokas turi bent vieną įeinantį prievadą (angl. *input port*), tai ta įėjimo dalis, kuri yra *SysML* vidinėje bloko diagramoje yra vadinama įėjimo dalimi. Šis įėjimo prievadas turėtų būti transformuotas į paprastąjį įvykį, kuris reprezentuoja **išorinio komponento** gedimą ar klaidą. Dėl to yra sukuriamas ARBA loginiai vartai, kurie sujungia dvi paprastąsias klaidas: išorinio komponento sugedimą arba priimančio vidinio komponento dalių klaidą (1.21 pav.).



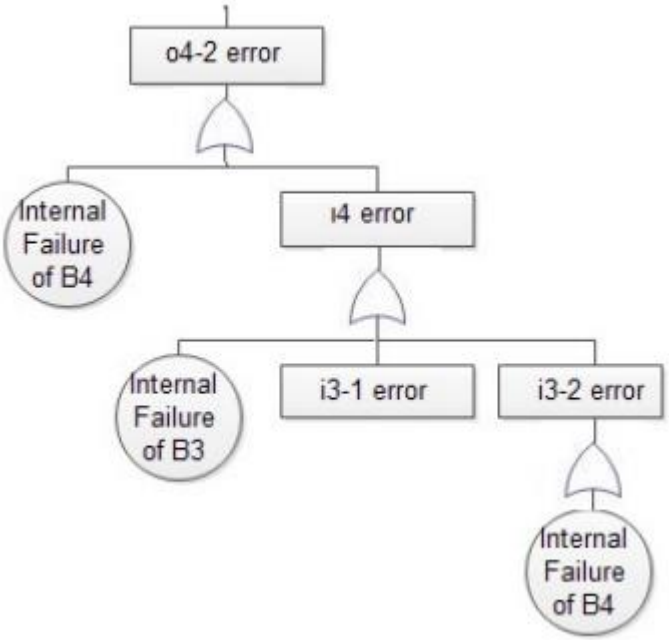
1.21 pav. Įėjimo šablono automatiškai sugeneruota gedimų medžio analizės dalis [23]

2) Išėjimo šablonas (angl. *Exit pattern*) – tai kitas būdas, jei blokas turi bent vieną išeinantį (išvedantį, angl. *output port*) prievadą, kurio srautas išeina į jau realią sistemą ar posistemę. Nutikus taip, kad nesuveikiant šiai bloko daliai – šis turėtų būti traktuojamas kaip labiausiai nepageidautinas viso bloko įvykis ir jis atsirasti gedimų medžio analizės diagramos viršuje. Šiuo atveju taip pat turėsime ARBA loginius vartus, kurios operandai bus vidinė išėjimo dalies klaida ir visi kiti tarpiniai (angl. *intermediate*) įvykiai, kurie ateis iš kitų ateinančių įvadų klaidų (1.22 pav.).



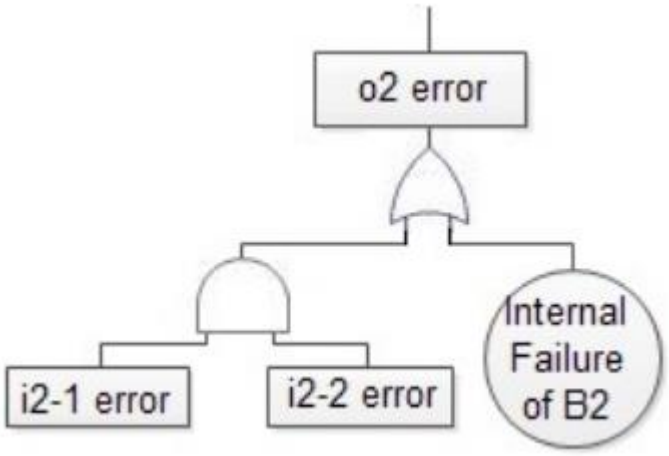
1.22 pav. Išėjimo šablono automatiškai sugeneruota gedimų medžio analizės dalis [23]

3) Atsakomasis šablonas (angl. *Feedback pattern*) – trečiasis variantas, kai sekant vidine bloko diagrama einama kryptingai ir vėl grįžtame tą patį mazgą. Kitaip tariant, tai galime vadinti ciklu arba atsakomuoju ryšiu. Jei pažiūrėtume į 1.20 pav., einant kryptingu srautu galime matyti, kad o4-2 išėjimas yra vienas iš B4 komponentų išėjimų ir jis yra tiesiogiai priklausomas nuo i4 įėjimo įvado. Tačiau i4 yra tiesiogiai priklausomas nuo o3 išeigos, kurios rezultatas yra priklausomas nuo o4-1 išeigos. Todėl šiuo atveju gaunasi grįžtamasis ryšys ir B4 komponento rezultatas tiesiogiai priklauso nuo savęs pačio. Tokiu atveju rinkinys (angl. *cut*) gali būti atvaizduotas kaip 1.23 pav., jei nebeimsime atskirų įėjimo šablonų kiekvienam įėjimui.



1.23 pav. Atsakomojo šablono automatiškai sugeneruota gedimų medžio analizės dalis [23]

4) Atsarginis (angl. *redundant*) šablonas – tai ketvirtasis būdas, kai dalis vidinėje blokų diagramoje yra sumodeliuoti du pertekliniai blokai, kurie atlieka tą pačią funkciją. Sakykime, kad turime atsarginį šabloną su komponentais B2, B11 ir B12 iš 1.20 pav. Šiuo atveju, turi būti naudojami IR vartai dėl to, kad dėl skirtingose įvesčių jungtyse gali būti skirtingos reikšmės, o trumpiau sakant, komponentas B2 neveiks tik tuo atveju, jei abu pertekliniai komponentai pradės nebeveikti. Iš to seka sugeneruota 1.24 pav. diagramos dalis.



1.24 pav. Atsarginio šablono automatiškai sugeneruota gedimų medžio analizės dalis [23]

Taikant aptartus keturis šablonus galime automatiškai generuoti gedimų medžio analizės dalis, kurios būtų išgeneruotos iš vidinių bloko diagramų. Tačiau saugumo ekspertai, taip pat sako, kad norint turėti teisingas gedimų medžio analizės šakas, reikia jas detalizuoti su galimomis skirtingomis įeičių, išeičių ar komponentų klaidomis arba optimizuoti komponentus, jog tos šakos būtų nepasiekiamos pasitelkiant kitų saugumo analizių (pvz. *FMEA*) rezultatais. Automatiškai generuotos gedimų medžio analizės šakos gali per mažai suteikti informacijos apie detalių gedimo priežastis.

1.7. Analizės išvados

1. Išsiaiškinus *FTA* procesus, galima suprasti kaip vyksta pats gedimų medžio analizės procesas, paaiškinti kas yra ši analizė, kam ji reikalinga ir kokias naudas ji atneša. Atliekant tyrimą buvo svarbu išanalizuoti ir gedimų medžio analizės notaciją, logikos ir tikimybių skaičiavimo pagrindus norint geriau suprasti jos prasmę ir svarbą atliekant gyvybiškai svarbius projektus. Gedimų medžio analizė – vienas iš daugelio metodų padedančių mažinti riziką labai svarbiuose (kuriuose negali būti klaidų) projektuose.

2. Kuriant kompleksines sistemas išlaikyti sistemos vientisumą bei mažinti kaštus padeda modeliais grįsta sistemų inžinerija. Jos pagalba, yra galimybė atlikti įvairias analizes dar nesukūrus realios sistemos ar prototipo. Modeliais grįsta architektūra dažniausiai atsiremia į metodologijos ir karkaso rėmus, kurie padeda modeliuoti ir apgalvoti visus sistemos aspektus. Tačiau kartu su metodologija ir karkasu komplektuojasi ir modeliavimo kalba. Atlikus *SysML* kalbos analizę buvo suprasta, jog tai yra projektų modeliavimo kalbos variantas kilęs iš plačiai žinomos *UML* kalbos, tačiau yra pranašesnis tuo, kad juo galima užrašyti ir nefunkcinius reikalavimus. Kai yra kuriamas sistemos modelis, tuomet dažniausiai kuriama sistema turi ypatingą svarbą. Joje turi būti kuo mažiau klaidų. Todėl kartu su modeliu neretai būna atliekamas ir gedimų medžio analizės procesas kaip vienas iš būdų užtikrinti kuo didesnę klaidų aptikimo tikimybę. Tai reiškia, kad *SysML* ir gedimų medžio analizės objektai tarpusavyje turi koreliuoti.

3. Atliktas nesėkmės modelio ir pasekmių analizės proceso išsiaiškinimas parodo, jog tai yra dar vienas plačiai naudojamas patikimumo įvertinimo būdas skirtas sumažinti rizikas projekte. Tačiau ši analizė yra visiškai skirtinga ir remiasi atvirkštiniais principais nei gedimų medžio analizė. Buvo išsiaiškinta, kad dėl šios priežasties šios analizės neretai būna naudojamos kartu, nes atliekant jas iš skirtingų kampų projektuose galima atrasti daug daugiau klaidų nei atliekant tik vieną pasirinktą analizę. Dėl šios priežasties, objektai (sistemos blokai) egzistuojantys vienoje analizėje, gali egzistuoti ir kitoje, o

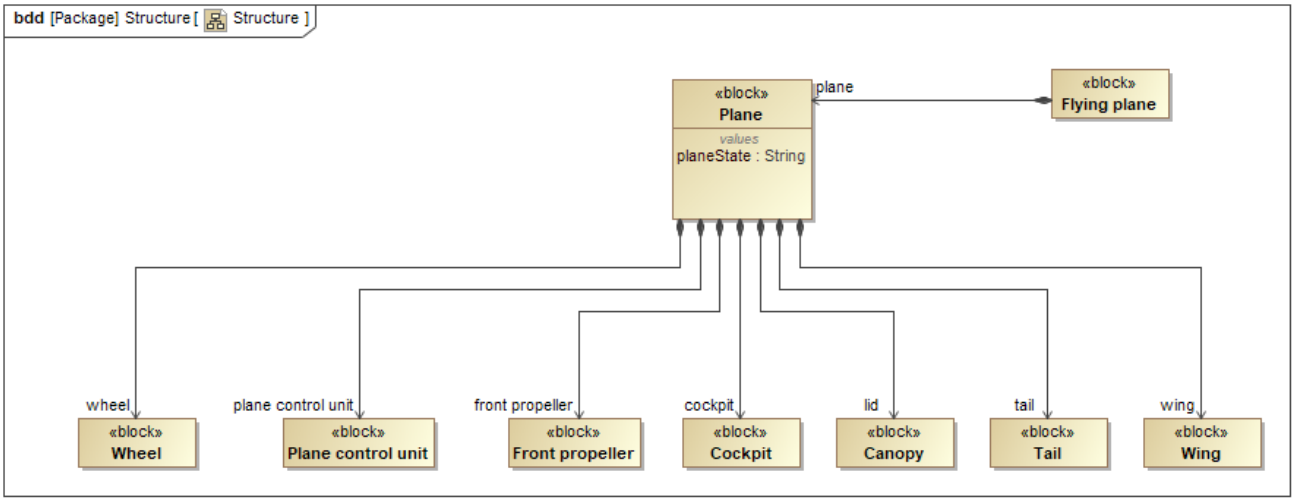
tai reikštų, kad ir čia reikėtų atsekamumo ryšio dėl mažesnio klaidų skaičiaus analizėse bei pačiame projekte.

4. Atlikus klaidų medžio analizės (FTA) esamų problemų sprendimų analizę išaiškėjo, kad šiuo metu nėra vieningo įrankio kurti sistemos projektui ir gedimų medžio analizei atlikti. Dėl šios priežasties sistemos komponentus reikia pakartotinai kurti kelis kartus, o dėl to didėja kaštai ir rizika padaryti daugiau žmogiškų klaidų. Taip pat, dėl šios priežasties yra naudojamas papildomas dokumentas aprašyti sistemos komponentų atsekamumui. To pasekoje, didelė darbo dalis įdedama ne tik kuriant gedimų medžio analizę, tačiau ir ją analizuojant.

2. GEDIMŲ MEDŽIO ANALIZĖS MODELIAIS GRĮSTOJE ARCHITEKTŪROJE REIKALAVIMŲ SPECIFIKACIJA IR PROJEKTAS

2.1. Konceptualus sistemos modelis

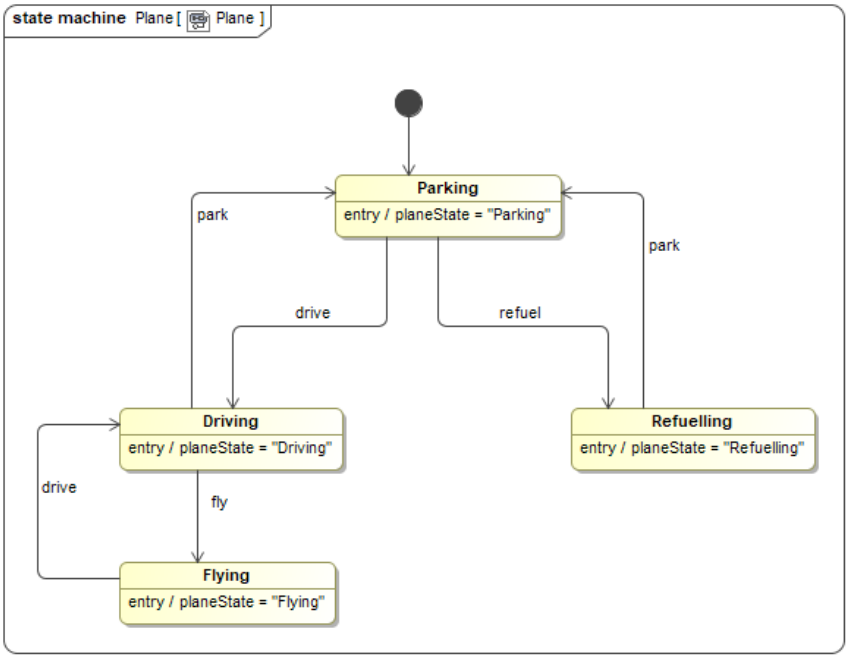
Norint geriau suprasti, bei išsiaiškinti kaip galima būtų realizuoti gedimų medžio analizės procesą, reikalingas sistemos modelis, kurio pavyzdžiu būtų galima projektuoti modeliuojamą gedimų medžio analizę. Tam buvo pasirinkta sukurti supaprastintą vienviečio akrobatinio lėktuvo modelį naudojant *SysML* kalbą (2.1 pav.). Sumodeliuotas lėktuvas turi ratus (angl. *wheels*), centrinį lėktuvo valdymo modulį (angl. *plane control unit*), priekinį propelerį (angl. *front propeller*), lakūno kabiną (angl. *cockpit*), dengiamąjį dangtį (angl. *canopy*), lėktuvo uodegą (angl. *tail*) ir sparnus (angl. *wings*).



2.1 pav. *SysML* kalba sumodeliuotas lėktuvo modelio blokų diagrama.

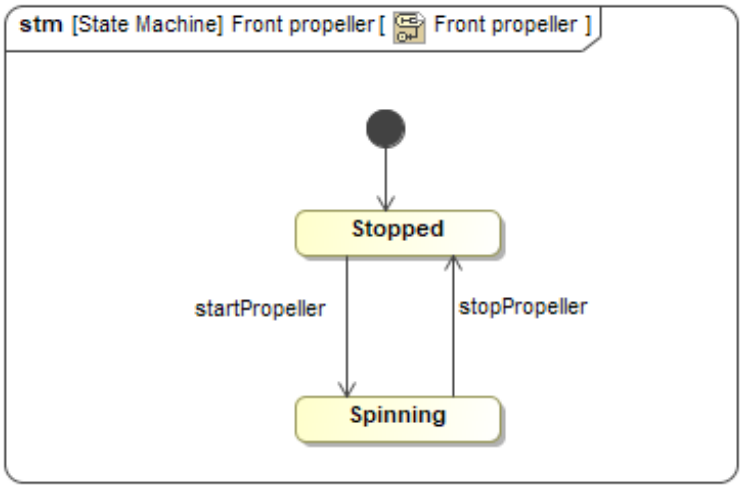
Gedimų medžio analizėje nenorimi įvykiai (angl. *unwanted events*) įvyksta pereinant iš vienos detalės būsenos į kitą (angl. *on state change*). Todėl modelyje taip pat yra labai svarbu turėti būsenų diagramas (angl. *state – machine diagrams*). Pasirinktame koncepciniame sistemos modelyje būsenas turi tik šie komponentai:

Lėktuvas (angl. *plane*) – turintis keturias skirtingas būsenas: stovėjimą (angl. *parking*), važiavimą (angl. *driving*), skridimą (angl. *flying*) ir kuro pylimą (angl. *refuelling*) (2.2 pav.).



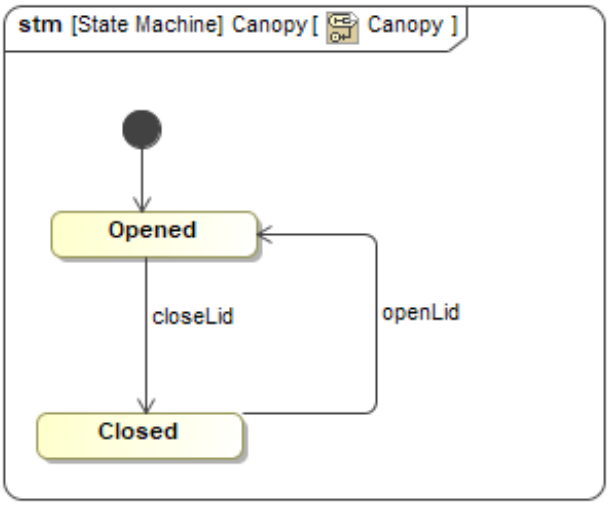
2.2 pav. Lėktuvo būsenų diagrama

Priekinis propeleris (angl. *front propeller*) – turintis tik dvi būsenas: sustojęs (angl. *stopped*) ir besisukantis (angl. *spinning*) (2.3 pav.).



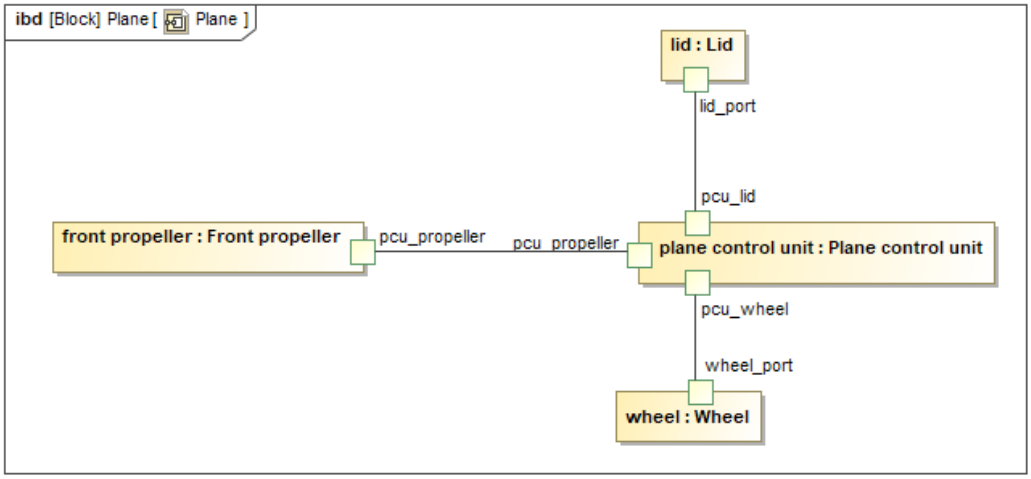
2.3 pav. Priekinio propelerio būsenų diagrama

Dangtis (angl. *canopy*) – turintis taip pat dvi būsenas: atidarytas (angl. *opened*) ir uždarytas (angl. *closed*) (2.4 pav.).



2.4 pav. Dangčio būsenų diagrama

Kiekvienas dinaminis komponentas turintis skirtingas būsenas yra valdomas naudojantis centrinio lėktuvo valdymo modulio pagalba, todėl dinaminiai komponentai (ratai, propeleris ir dangtis) yra tiesiogiai sujungti su lėktuvo valdumo moduliu. Tai galima matyti vidinėje bloko diagramoje (angl. *Internal Block Diagram*) (2.5 pav.).



2.5 pav. Vidinė lėktuvo bloko diagrama

Šis sukurtas koncepcinis sistemos modelis bus skirtas ieškoti koreliacijos tarp būsimos gedimų medžio analizės. Sukurtas išsamus *SysML* lėktuvo sistemos modelis yra pamatas būsimo sprendimo realizavimui ir testavimui.

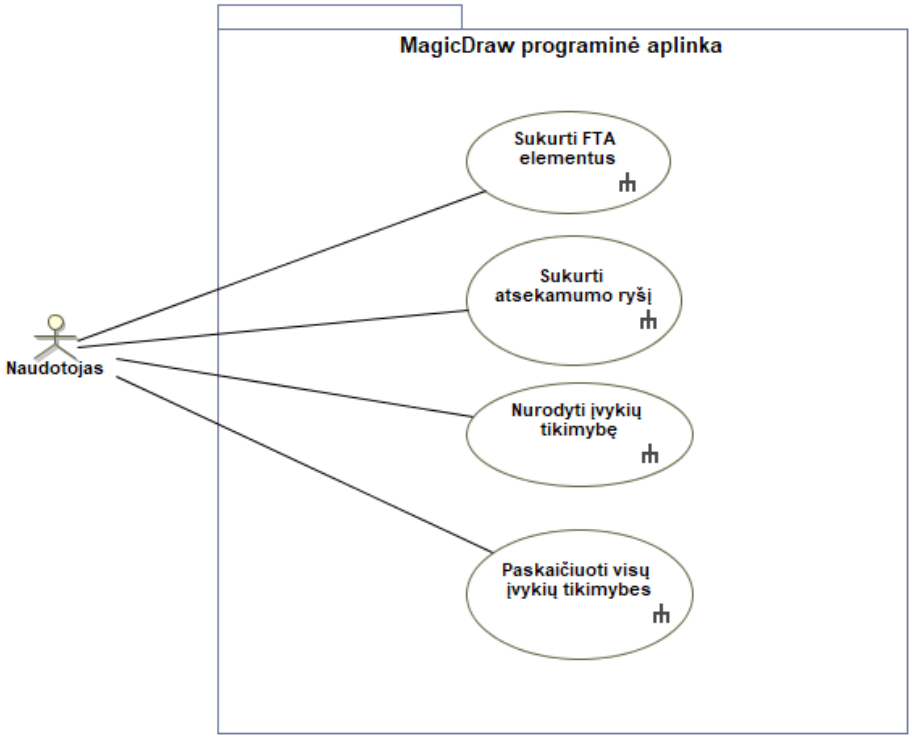
2.2. Reikalavimų specifikacija

Norint pagerinti gedimų medžio analizės procesą, pirmame skyriuje buvo atlikta analizė, kuri turėtų padėti suprasti problemas ir galimus jų sprendimo būdus. Šios išanalizuotos problemos iškėlė reikalavimus, kurie buvo performuluoti į *SysML* reikalavimų diagramą (7.1 priedas), kurioje aiškiau matomas išsikeltas tikslas.

Pagal išsikeltus reikalavimus buvo reikalinga programinė įranga, kuri iš karto atitiktų daugiausiai reikalavimų. Taip pat, norint sugebėti realizuoti papildomą funkcionalumą, reikalinga programinė įranga turi būti atviro kodo arba turėti prieinamą atvirą aplikacijų programinę sąsają (angl. *Application programming interface - API*). Šiuos kriterijus atitinka „No Magic Inc.“ kuriamas įrankis

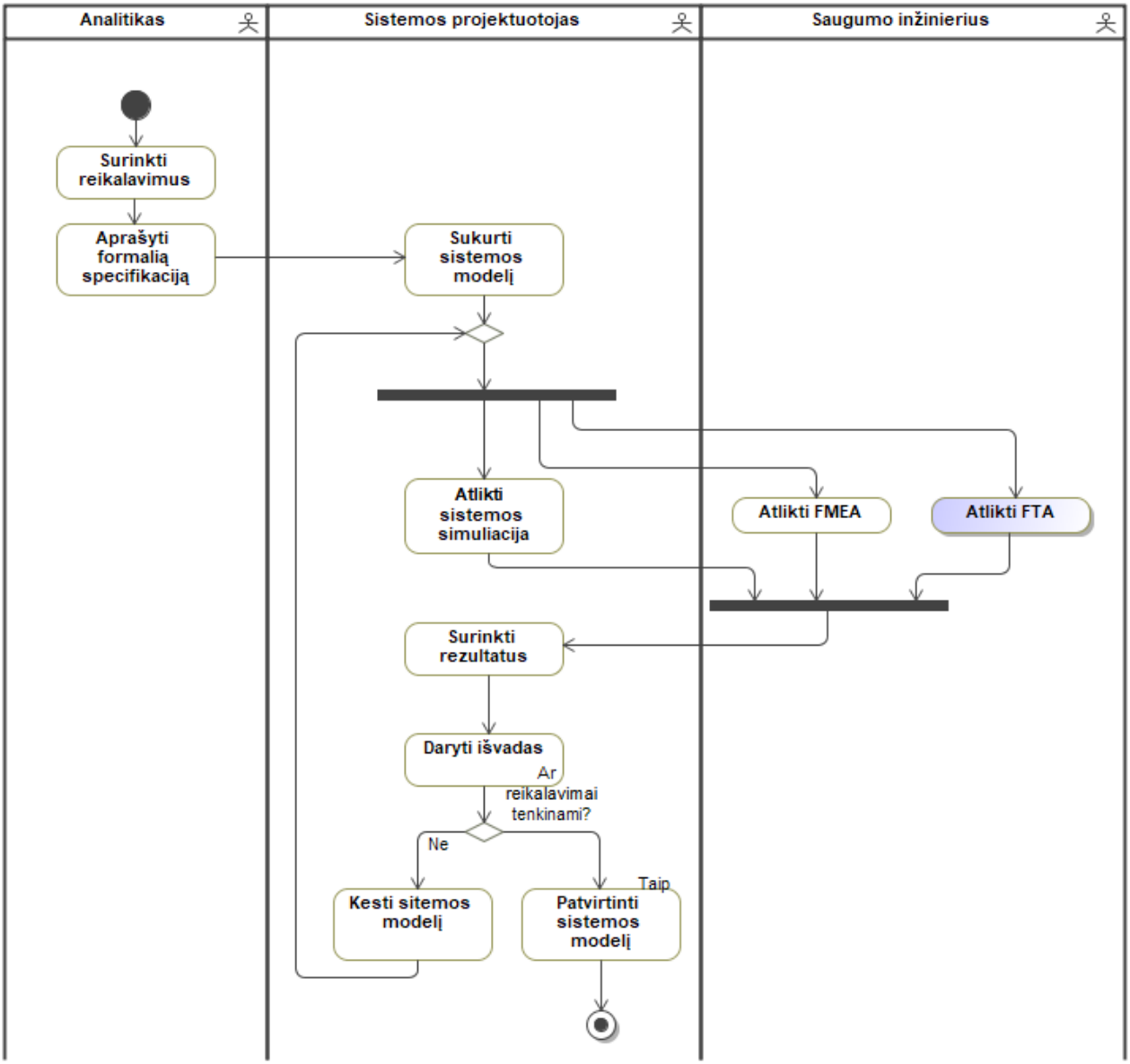
„MagicDraw“, kuris turi realizuotą visą *SysML* modeliavimo dalį ir plačiai dokumentuotą aplikacijų programinę sąsają per kurią būtų realizuota gedimų medžio analizės implementacija.

Dėl šių priežasčių buvo nuspręsta, jog reikia naudotis modeliavimo įrankiu „MagicDraw“. Iš suprojektuotos reikalavimų diagramos buvo nustatyta, kurie reikalavimai jau yra įgyvendinti pasirinktame įrankyje. Taip pat, buvo išgryninti kiti funkciniai reikalavimai, kuriuos vis dar reikia realizuoti. Šie reikalavimai yra atvaizduoti *UML* panaudojimo atvejų diagramoje (2.6 pav.). Kiekvienam iš funkcinių reikalavimų buvo parašyta specifikacija ir suprojektuotas vykdymo scenarijus.



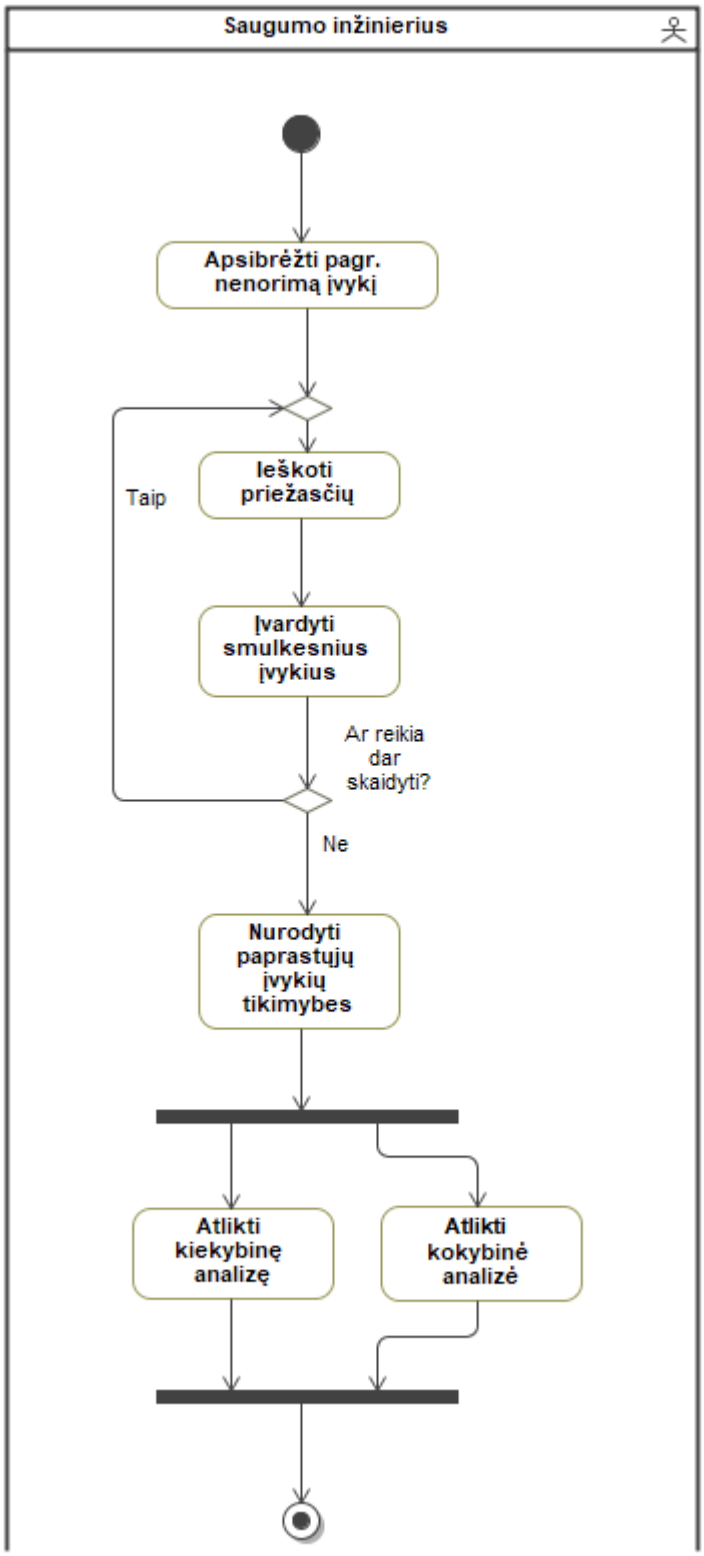
2.6 pav. Panaudojimo atvejų diagrama

Taip pat, norint sėkmingai užtikrinti *FTA* analizės procesą ir suprojektuoti realizaciją korektiškai, reikia suprasti kurioje kūrimo stadijoje yra atliekama gedimų medžio analizė, todėl tam buvo sumodeliuota *UML* veiklos diagrama, kuri atvaizduoja kada ir kur vyksta *FTA* kūrimo procesas (2.7 pav.).



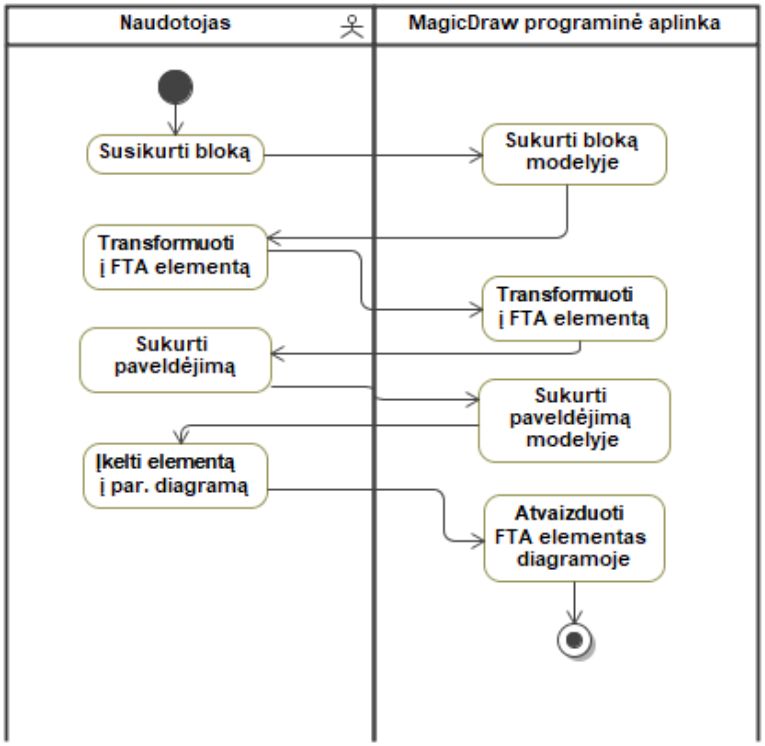
2.7 pav. Sistemos projektavimo etapai atliekant *FTA*

Sumodeliavus diagramą, kuri mums rodo kurioje kūrimo stadijoje yra atliekama *FTA* analizė – kitas svarbus aspektas suprasti kokiu principu ji yra kuriama. Tam buvo sumodeliuota dar viena veiklos diagrama (2.8 pav.), kuri aprašo *FTA* kūrimo procesą.



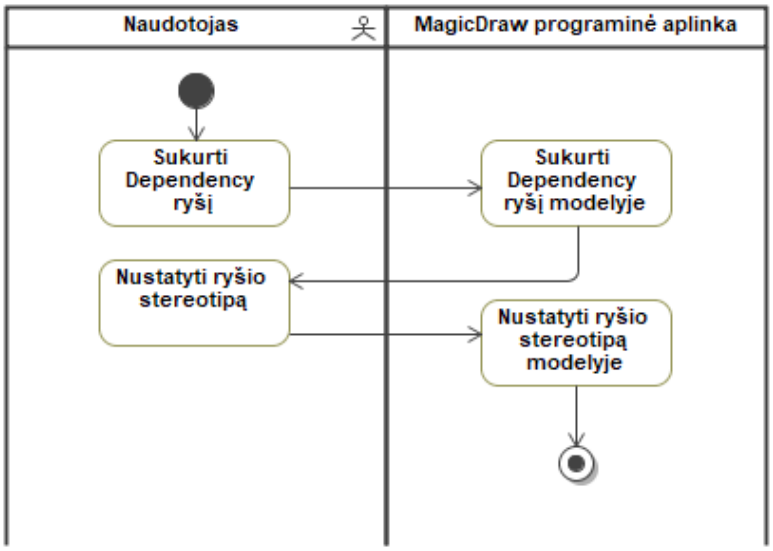
2.8 pav. Gedimų medžio analizės atlikimo procesas

Panaudojimo atvejo „Sukurti *FTA* elementus“ vykdymo scenarijus pavaizduotas 2.9 pav.



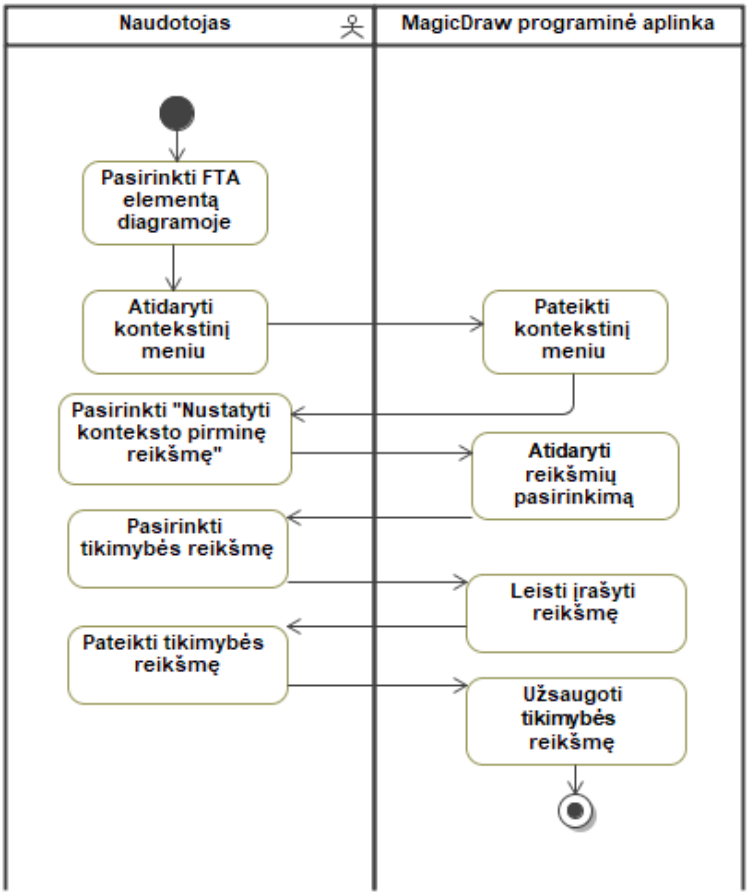
2.9 pav. PA „Sukurti *FTA* elementą“ vykdymo scenarijus

Panaudojimo atvejo „Sukurti atsekamumo ryšį vykdymo scenarijus pavaizduotas 2.10 pav.



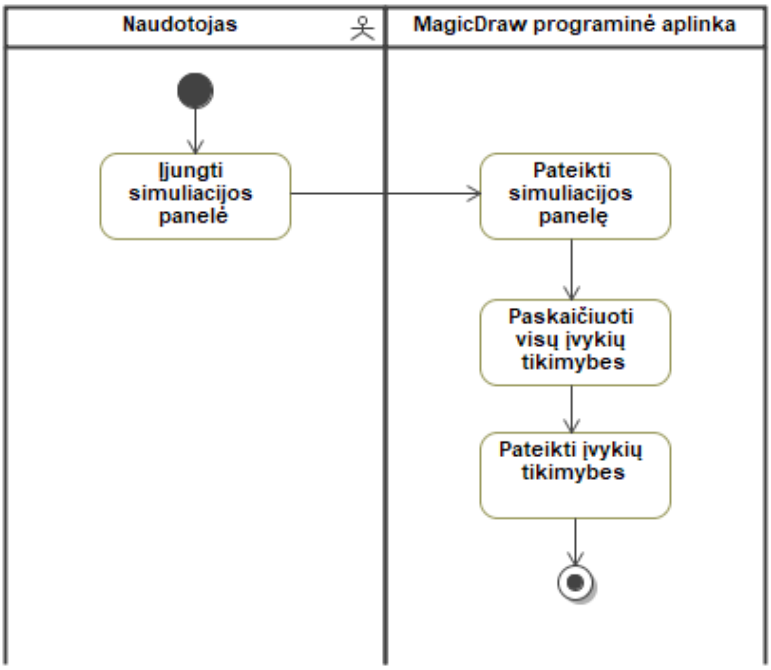
2.10 pav. PA „Sukurti atsekamumo ryšį“ vykdymo scenarijus

Panaudojimo atvejo „Nurodyti įvykio tikimybę“ vykdymo scenarijus pavaizduotas 2.11 pav.



2.11 pav. PA „Nurodyti įvykio tikimybę“ vykdymo scenarijus

Panaudojimo atvejo „Automatiškai apskaičiuoti įvykio tikimybę“ vykdymo scenarijus pavaizduotas 2.12 pav.



2.12 pav. PA „Automatiškai apskaičiuoti įvykio tikimybė“ vykdymo scenarijus

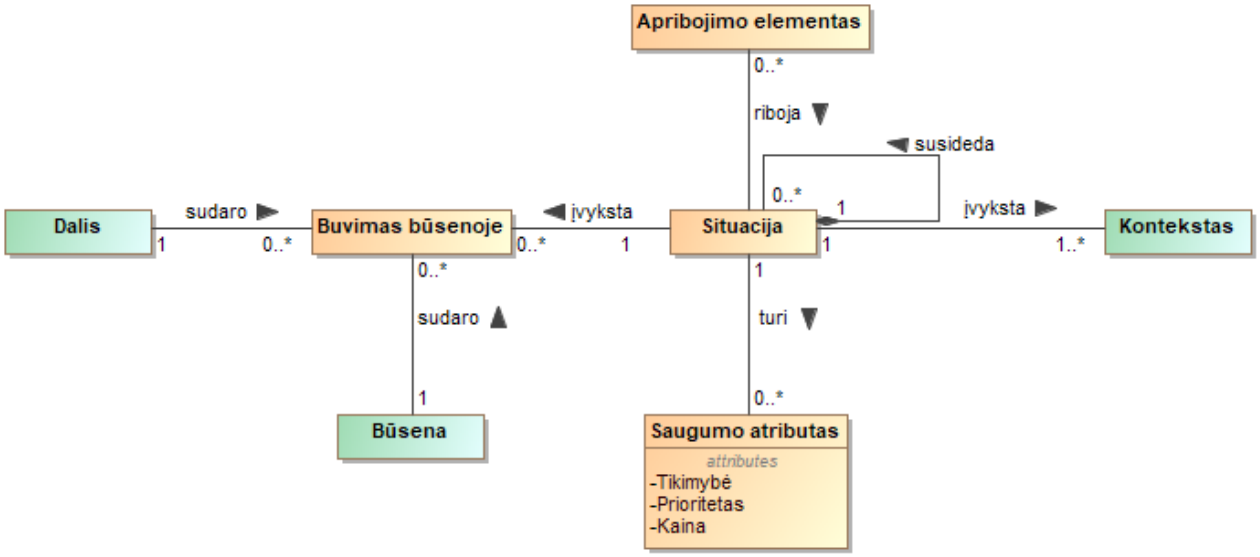
2.3. Formalus sprendimo aprašas

Išsianalizavus *SysML* modeliavimo kalbą, papildomai buvo atlikta MagicDraw įrankio modelio simuliacijos galimybių analizė. Analizės metu buvo nustatyta, kad norint atlikti skaičiavimus simuliacijoje geriausiai tam tinka *SysML* parametrikų diagrama. Taip pat buvo nustatyta, jog teisingai modeliuojant *SysML* modelį, gedimų medžio analizę galima būtų atlikti išplečiant *SysML* metamodelį pridėjus kelis naujus elementus:

- situaciją[1] (angl. *situation*), kuri atitiktų mūsų aptariamą *FTA* įvykį
- apribojimo elementą (angl. *constraint*), kuris atstotų *FTA* loginius vartus

Įvertinti ar sprendimas tinkamas, visų pirma reikia susimodeliuoti naują metamodelio profilį (angl. *profile*), kuris galėtų realizuoti gedimų medžio analizę. Tačiau norint tai padaryti reikia geriau suprasti kiek, kokių elementų ir kokiais ryšiais juos siejančių turėtų būti profilyje. Tam buvo sumodeliuota klasių diagrama kuri parodo numatomą sprendimą (2.13 pav.).

[1] Žodis „situacija“ buvo pasirinktas dėl to, kad šis žodis galėtų būti bendrinis tiek *FTA*, tiek ir *FMEA* analizei. Apsibrėžus bendrinį žodį tinkamą kelioms analizėms atlikti galima pritaikyti tą patį metamodelį, keliems saugumo analizių standartams.



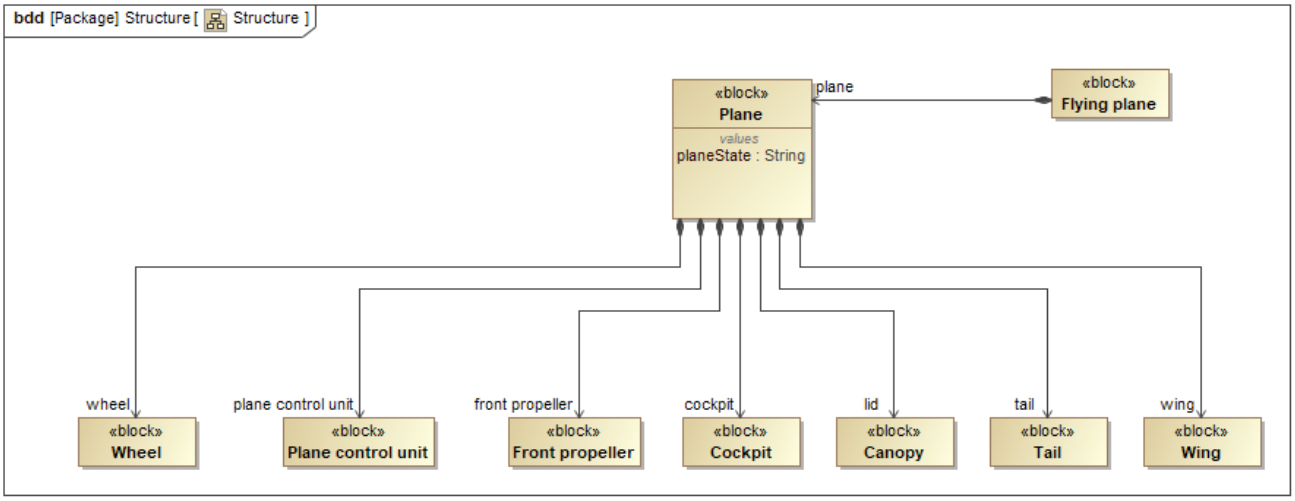
2.13 pav. Klasių diagrama vaizduojanti *FTA* sprendimą

Klasių diagramoje žaliai pavaizduoti elementai atitinka *SysML* jau egzistuojančius elementus:

- Kontekstas – blokas, kuris atitinka sistemos būseną tam tikru momentu.
- Dalis (angl. *part*) – elementas, kuris turėtų būti konkreti sistemos dalis (pvz. lėktuvo propeleris).
- Būsena – elementas, kuris nusako kurioje būsenoje yra dalis.

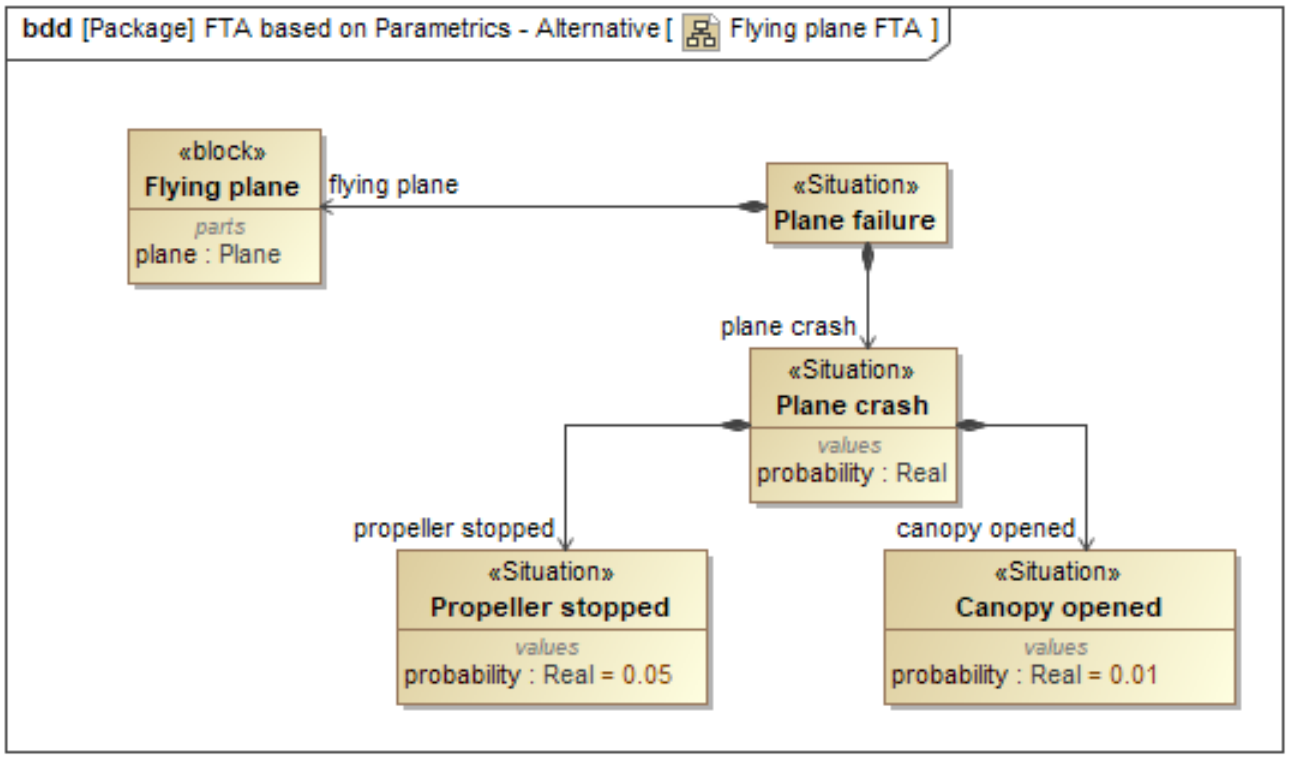
Taigi, pagal sukurtą diagramą galime matyti, jog elementas „Buvimas būsenoje“ (angl. *In State*) apjungia sistemos dalį ir jos būsenas, o šis apjungiantis elementas yra tiesiogiai sujungtas su situacija. Iš to seka, jog situacijoje galime nurodyti kokiose būsenose turi būti tam tikros sistemos dalys, norint sudaryti situaciją. Taip pat, reikia atkreipti dėmesį, kad situacija yra tiesiogiai sujungta su kontekstu, kuris nurodo, jog ta situacija gali susidaryti tik būnant tam tikroje, aukštesnio lygio sistemos būsenoje. Situacija taip pat gali susidėti iš mažesnių situacijų (gedimų medžio analizės principu – aukštesnio lygio įvykiai susideda iš mažesnio lygio įvykių, o šie iš dar mažesnių). Situacija gali turėti įvairius saugumo atributus – svarbiausias iš jų tikimybė, norint apskaičiuoti aukštesnių įvykių tikimybes. Na ir svarbus momentas, jog situacija gali būti sujungta su apribojimo elementais (angl. *constraint*) norint atlikti loginių vartų skaičiavimo logiką.

Norint įvertinti/patvirtinti *FTA* metamodelį, reikia jį pritaikyti analizės dalyje aprašytam kencepciniam sistemos modeliui. Prieš atliekant *FTA* analizę, sukurtame sistemos modelyje buvo papildomai sukurtas naujas lėktuvo kontekstas – blokas „Skrendantis lėktyvas“ (angl. *Flying plane*) (2.14 pav.).



2.14 pav. Papildyta sistemos koncepcinio modelio diagrama

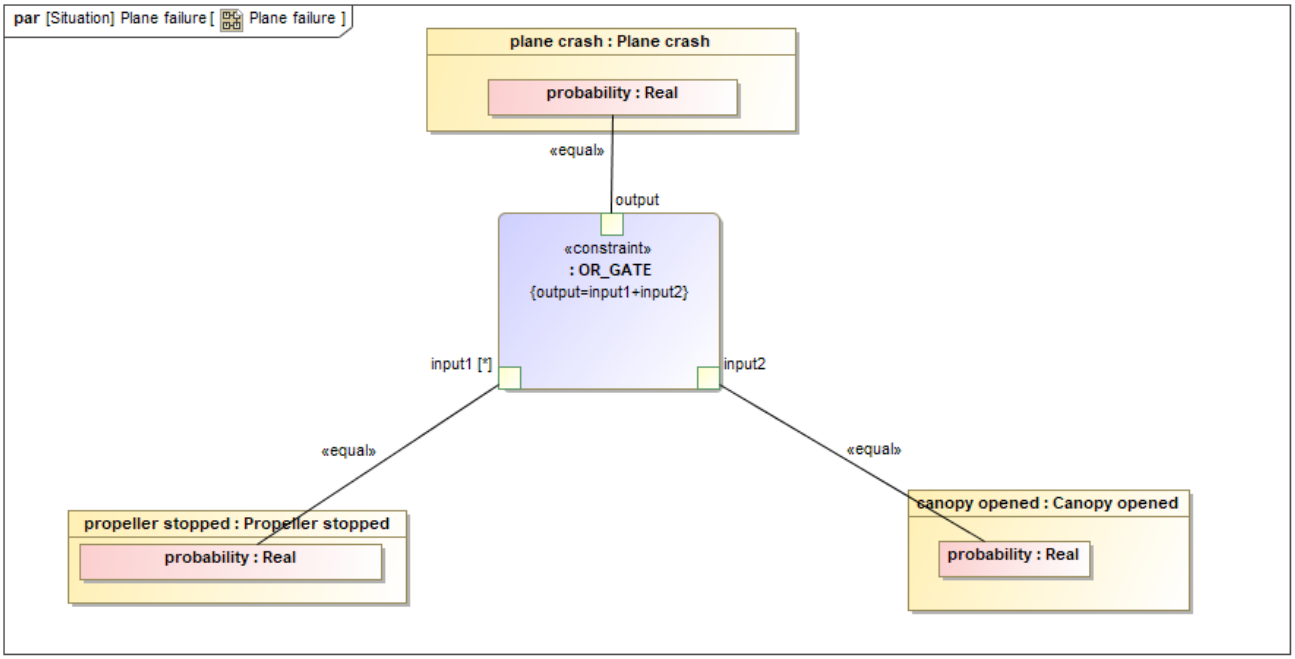
Pagrindinė nesėkmės situacija turėtų būti sudaryta iš šio konteksto. O pagrindinė nesėkmės situacija turėtų apjungti smulkesnes situacijas (įvykius). Tam buvo sumodeliuotas nesėkmės įvykių medis, vaizduojantis hierarchinę situacijų struktūrą (2.15 pav.). Jį sudaro pati aukščiausia situacija (atitikmuo aukščiausio lygio nesėkmės įvykiui) – lėktuvo nesėkmė (angl. *plane failure*), o šią – gali sudaryti smulkesnės lėktuvo situacijos.



2.15 pav. Hierarchinė situacijų struktūra

Pateiktoje hierarchijoje matome, kad lėktuvo nesėkmė įvyks tokiu atveju, jei lėktuvas suduš. Sudužti jis gali tik pasirinktame kontekste – skrendančiame lėktuve, susidėliojus tam tikrai logikai iš smulkesnių situacijų: sustojus propeleriui (*propeller stopped*) ir atsidarius lėktuvo dangčiui (*canopy opened*). Žemiausioms situacijoms yra nurodyta, jog jos įvyksta, jei tam tikra sistemos dalis yra tam tikroje būsenoje. Šiuo atveju, situacija „sustojęs propeleris“ yra sujungta su *SysML* būsena „Stopped“ (iš koncepcinio sistemos modelyje esančio propelerio būsenų sąrašo), o situacija „atidarytas lėktuvo

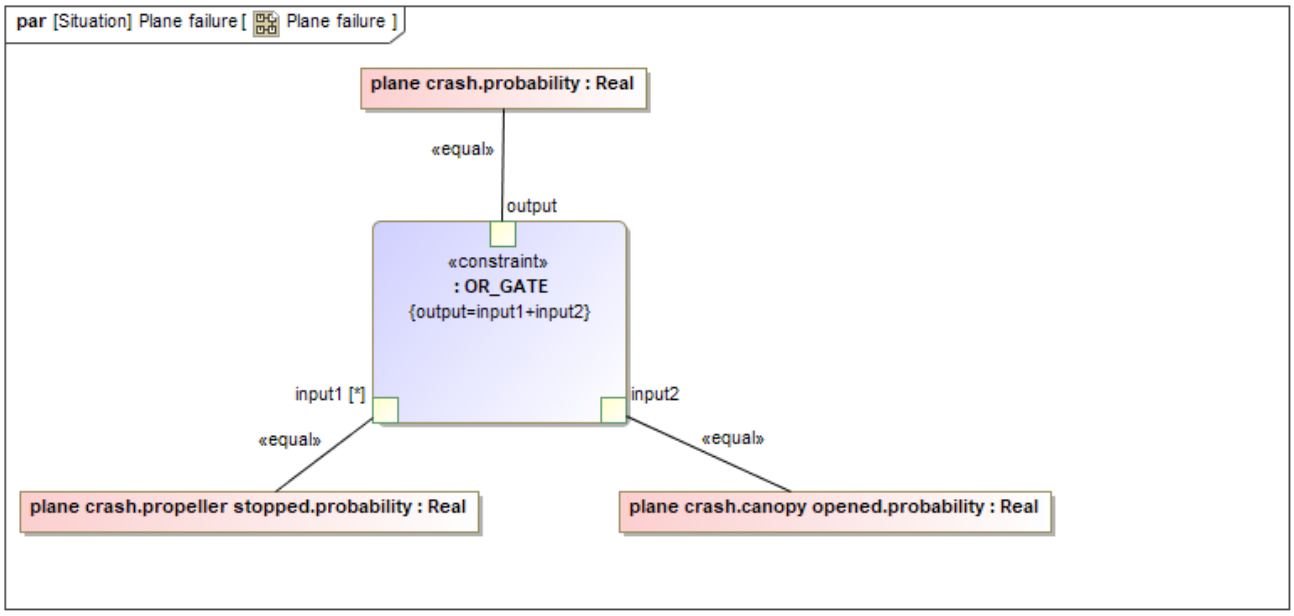
dangtis“ yra sujungta su *SysML* būsena „Opened“ (iš to paties koncepcinio modelio, tačiau jau iš lėktuvo dangčio būsenų). Tačiau, ši diagrama nėra pakankama atvaizduoti konkrečios gedimų medžio analizės logikos, todėl pasinaudojant parametrikų diagramos pagalba, buvo sumodeliuotos dvi galimos parametrikų diagramomis grįstos *FTA* diagramos, kurios leistų mums paskaičiuoti tikimybes ir konkrečiai nurodyti tam tikrą įvykių logiką (2.16 pav.).



2.16 pav. Parametrikų diagrama grįsta gedimų medžio analizės logika (I variantas)

Aukščiausias įvykis - lėktuvo nesėkmė (angl. *plane failure*) yra sudarytas iš mažesnių dalių – lėktuvo dužimo (angl. *plane crash*), propelerio sustojimo ir dangčio atidarymo (angl. *canopy opened*). Todėl sukūrus parametrikų diagrama, kuri yra priklausanti aukščiausiam įvykiui – galime atvaizduoti visas šias tris situacijas kaip įvykio dalis (angl. *parts*). Žemiausio lygio situacijos turi savo iš anksto apibrėžtas tikimybes, o aukštesnės situacijos turėtų automatiškai apsiskaičiuoti pagal loginių vartų sujungimo taisykles aptartas analizėje. Šioje diagramoje sukūrus loginio varto apribojimą (angl. *constraint*) galime nurodyti tikimybės išvesties formulę bei apjungti visas tris situacijas į tam tikrą loginę seką, pavyzdžiui: lėktuvas suduš tuo atveju, jei propeleris sustos arba dangtis atsidarys (bet tik skrendančio lėktuvo kontekste).

Kitas būdas modeliuot *FTA* grįstą parametrikų diagramomis – rodyti situacijas ne kaip dalis, o kaip konkrečias įvykio tikimybių reikšmes (angl. *value property*). Vadovaujantis šiuo principu – nerodome, jog ta situacija turi tam tikrą tikimybę, o rodome, jog ta tikimybė yra konkrečiai iš pasirinktos situacijos (2.17 pav.)



2.17 pav. Parametrikų diagrama grįsta gedimų medžio analizės logika (II variantas)

Nurodę žemiausių įvykių numatytąsias tikimybes (kaip matoma 2.15 pav.) galime pabandyti paleisti įrankio simuliaciją ir pasitikrinti ar teisingai yra išskaičiuojamos aukštesnio įvykio tikimybės (2.18 pav.). Nustačius, jog dangtis atsidarys – tikimybė lygini 0,01, o propeleriui sustoti tikimybė yra – 0,05, įrankis automatiškai paskaičiuoja, jog lėktuvo sudužimo tikimybė yra lygi 0,06.

Name	Value
Plane failure	Plane failure@7b3c755d
flying plane : Flying plane	Flying plane@65f4c9c0
plane : Plane [Parking]	Plane@4e533153
plane crash : Plane crash	Plane crash@105ba9f9
probability : Real	0,0600
canopy opened : Canopy opened	Canopy opened@2367cf7d
probability : Real	0,0100
: Gate	
: Gate	
propeller stopped : Propeller stopped	Propeller stopped@7c5537d5
probability : Real	0,0500
: Gate	
: Gate	
: Gate	
: Gate	
: Gate	
: Gate	
: OR_GATE {plane crash.probability=plane crash.p...	OR_GATE@471d0df5
input1 [*]	0,0500
input2	0,0100
output	0,0600

2.18 pav. Simulacijos galimybė MagicDraw įrankyje automatiškai apskaičiuoja įvykių tikimybes

Išanalizavus skaičiavimų rezultatus, galime daryti išvadą, jog tai yra teisingas būdas atlikti gedimų medžio analizę norint turėti atsekamumą su *SysML* sistemos modeliu. Norint pilnai realizuoti šį metodą, reikia sukurti *FTA* profilį, kuris išplėstų standartinį *SysML* profilį su aptartomis situacijomis bei sukurti biblioteką, kuri leistų naudotojui naudotis jau aprašytais loginiais vartų apribojimais (angl. *constraints*) su formulėmis, o ne priversti naudotoją kurtis loginius vartus kiekvienam atvejui.

Šiam sprendimui įgyvendint patogiausias naudotojui sprendimas – turėti MagicDraw įskiepį, kuris leistų sukurti gedimų medžio analizes sistemos kūrimo metu naudojant *SysML*.

2.4. Reikalavimų apibendrinimas

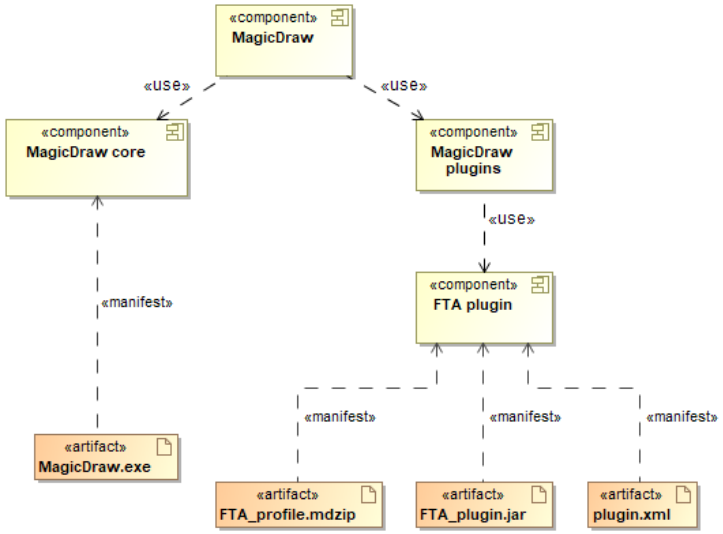
Buvo nustatyta, jog gedimų medžio analizės prototipas parengtas pagal sumodeliuotą koncepcinį lėktuvo modelį yra puikiai atliekantis kiekybinę gedimų medžio analizę bei galintis užtikrinti atsekamumą. Remiantis padarytu projektu, buvo nutarta, jog reikia sukurti MagicDraw įskiepį, kuris įgalins suprojektuotos gedimų medžio analizės kūrimo procesą bei palengvins darbą saugumo inžinieriams.

3. SPRENDIMO REALIZACIJA IR TESTAVIMAS

Šiame skyriuje bus aprašomas realizacijos sprendimas bei jo ištestavimas.

3.1. Sprendimo realizacija

Sprendimo realizacijai įgyvendinti buvo sumodeliuota komponentų architektūra (3.1 pav.). Ši diagrama leidžia suprasti kokių artefaktų reikia norint įgalinti modeliais grindžiama gedimų medžio analizę.

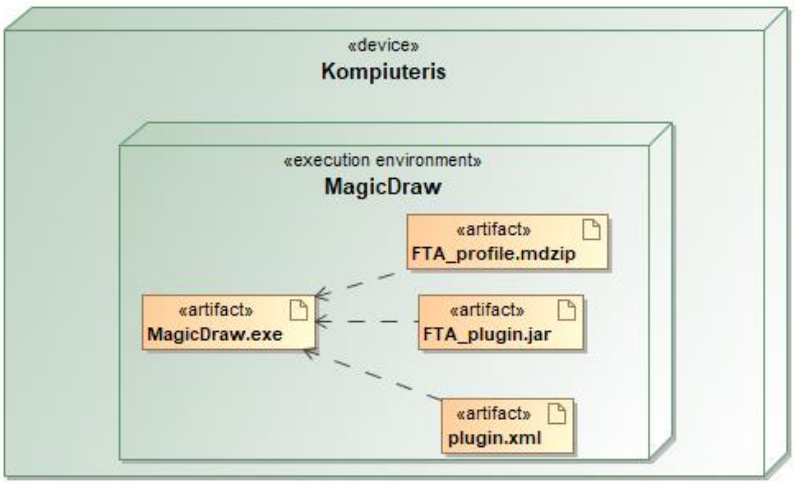


3.1 pav. *FTA* įskiepio komponentų diagrama

Komponentų diagramoje matyti, kad pagrindinis kuriamas komponentas *FTA* įskiepis (angl. *FTA plugin*) bus sudarytas iš trijų artefaktų:

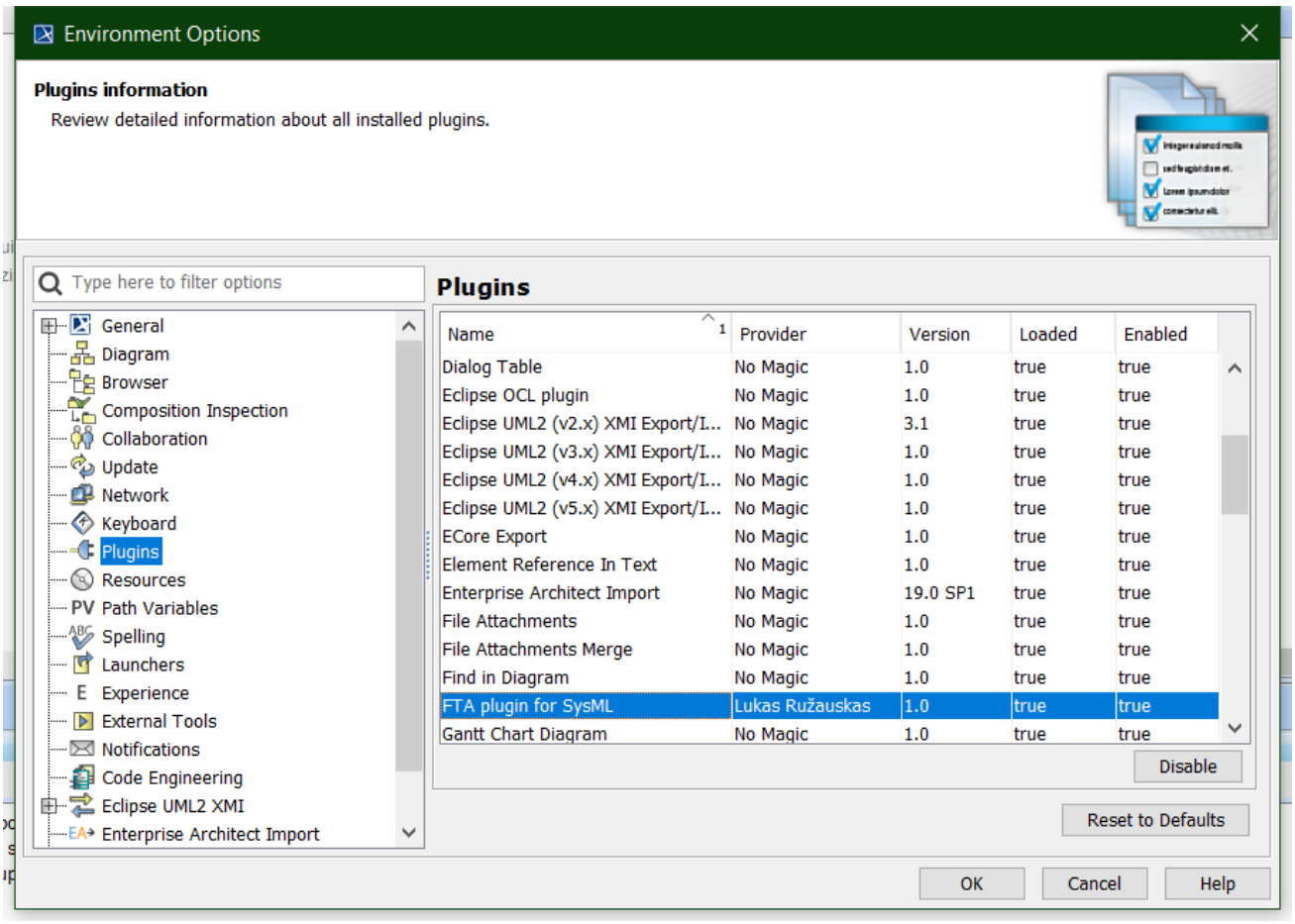
- profilio, kuriame bus *FTA* elementų metaklasės bei biblioteka, kuri skirta apribojimo blokams
- plugin.xml bylos, kuri bus atsakinga už įskiepio informaciją
- FTA_plugin.jar bylos, kuri atsakinga už įskiepio įdiegimo dalį

Iš nustatytų komponentų taip pat galime apsibrėžti įskiepio architektūrą, ji atvaizduota diegimo diagramoje (3.2 pav.)



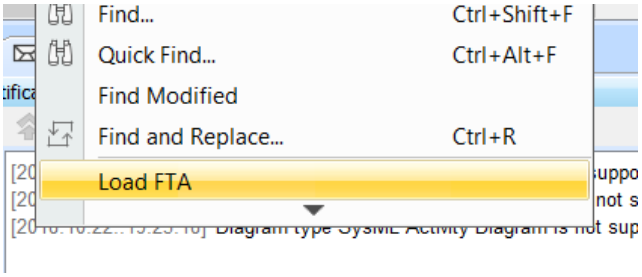
3.2 pav. FTA įskiepio diegimo diagrama

Realizuoti gedimų medžio analizę modeliuose galima tiesiog panaudojant specialų profilį, kurio viduje yra šios analizės metamodelis. Norint palengvinti šį darbą naudotojams, buvo sukurtas įskiepis programai (3.3 pav.) „MagicDraw“, kuris leidžia praktiškai vieno mygtuko paspaudimu įterpti metamodelį į modelį ir jį naudoti. Įskiepio realizacijai buvo pasinaudota kompanijos „No Magic Inc.“ sukurto įrankio „MagicDraw“ įrankio *Open API* teikiamomis funkcijomis, kurios leidžia nesunkiai sukurti įskiepį ir jį sujungti su šiuo įrankiu.



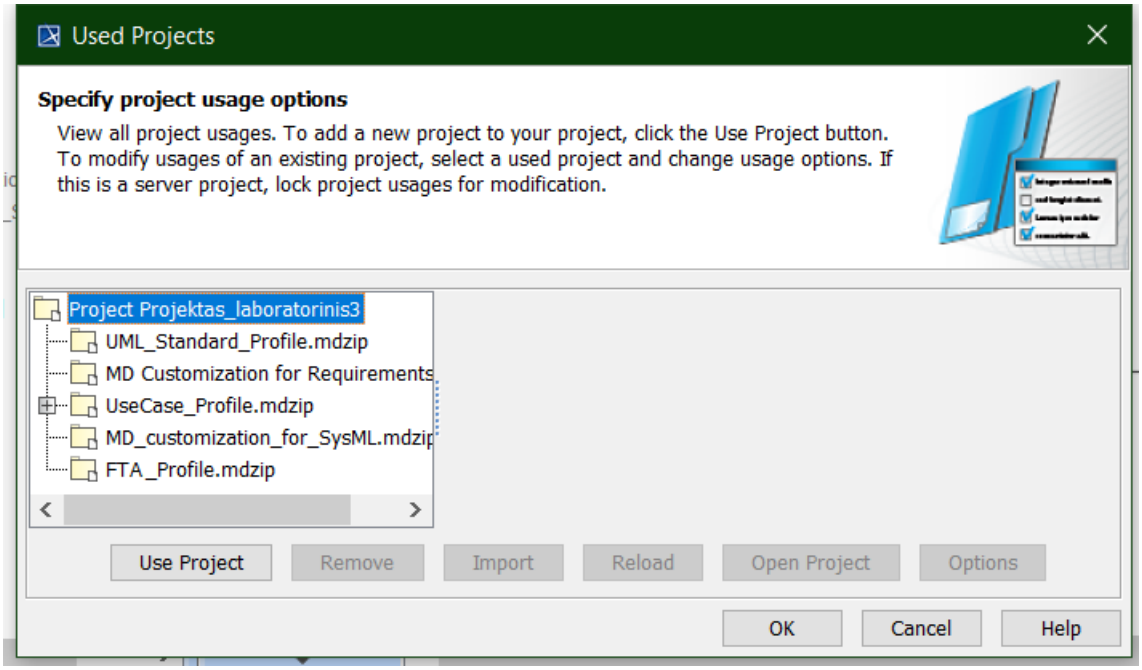
3.3 pav. Įskiepių sąraše matomas sukurtas gedimų medžio analizės įskiepis

Pagrindinė įskiepio funkcija – panaudoti *FTA* metamodelį jau egzistuojančiame sistemos modelyje, tam įskiepio pagalba meniu juostoje buvo sukurtas mygtukas meniu „Load FTA“ (3.4 pav.), kuris automatiškai prideda prieš tai aptartą metamodelį į modelio struktūrą.



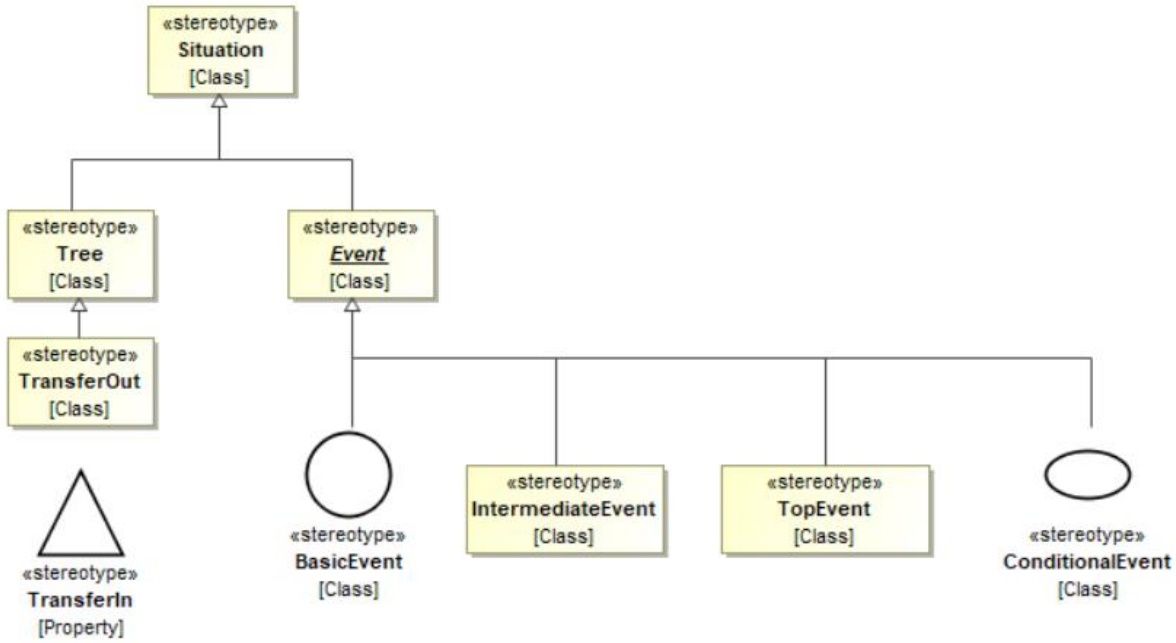
3.4 pav. Meniu juostoje sukurtas mygtukas „Load *FTA*“

Panaudojus šį mygtuką, tarp panaudotų projektų atsiranda ir projektas FTA_profile.mdzip, kurį jau galima naudoti sistemos modelyje (3.5 pav.).



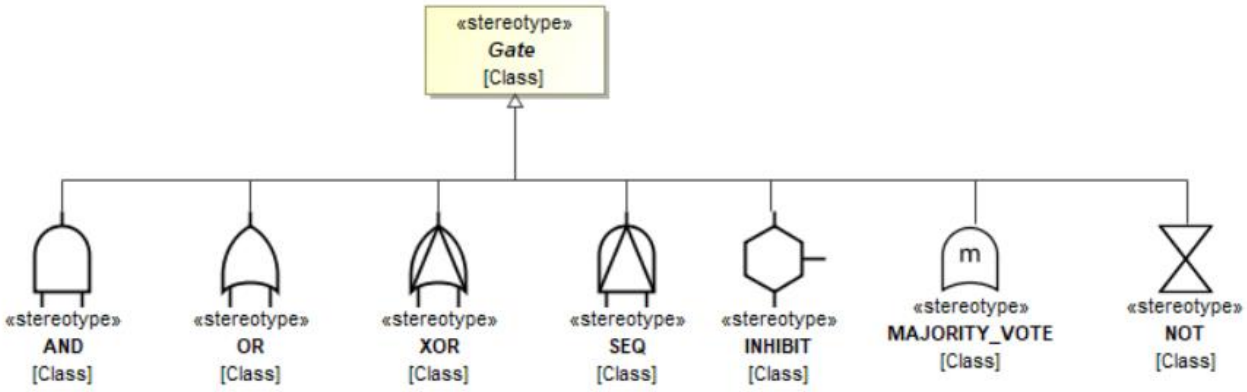
3.5 pav. Matomas panaudotas FTA_Profile sistemos modelyje.

Pats gedimų medžio analizės profilis yra sudarytas iš pagrindinių *FTA* elementų - įvykių (3.6 pav.). Iš paveikslėlio galime matyti, kad pagrindinis elementas yra situacija, kuri yra po to detalizuojama į įvykių tipus: medį (angl. *tree*) ir įvykį (angl. *event*). Medis – tai elementas, kuris nurodo, jog konkretus elementas yra visas savarankiškas gedimų medis, kuris turėtų turėti tiek pagrindinį nenorimą įvykį (angl. *top event*), bei jį detalizuojančius įvykius. Šio medžio reikšmė gali būti atitinkanti „Iš“ perdavimo loginio varto reikšmę (angl. *transfer out*). Tai reiškia, kad šis elementas, gali būti panaudotas kaip smulkinantis medį (po-medžio, detalesnio medžio) elementas.



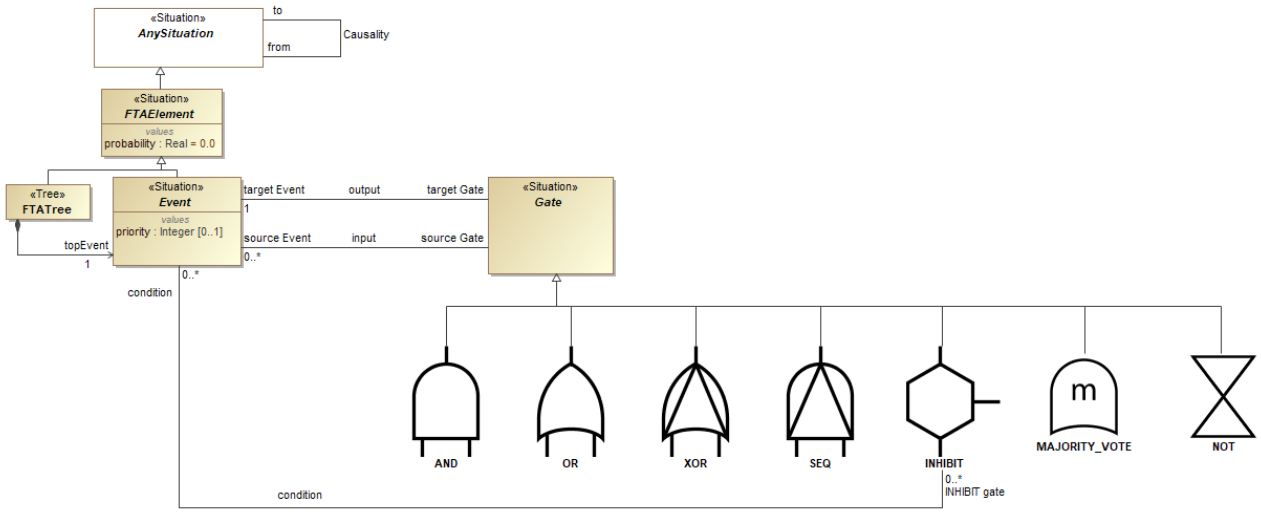
3.6 pav. Profilyje esantis gedimų medžio analizės įvykių metamodelis

Taip pat į realizuotą profilį patenka ir loginių vartų stereotipai, kurie leidžia apibrėžti kiekvieną loginius vartus (3.7 pav.).



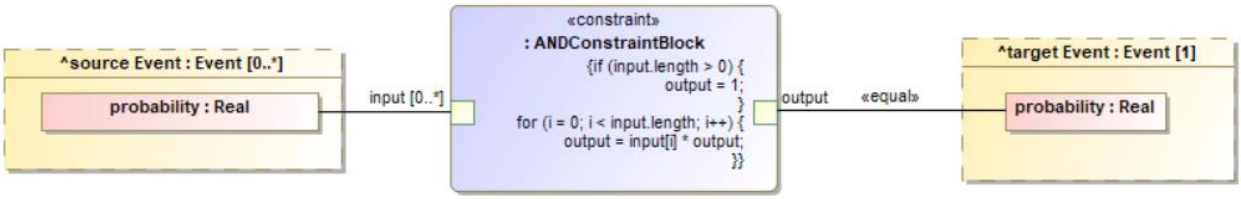
3.7 pav. Profilyje esantis gedimų medžio analizės loginių vartų metamodelis

Profilyje buvo paruošta biblioteka, kuri leidžia naudoti loginius vartus (3.8 pav.). Kiekvienas loginio varto tipas yra paveldėtas iš loginių vartų elemento. Šie loginiai vartai turi du ryšius – įeigą (įeinančius įvykius) ir išeigą – (išeinančius įvykius). Kiekvienas įvykis yra tiesiog gedimų medžio analizės elementas ir jis gali priklausyti tam tikram gedimų medžiui. Kaip matome iš paveikslėlio, NELEIDŽIAMASIS (angl. *INHIBIT*) loginiai vartai taip pat gali turėti trečią ryšį – sąlyginį (angl. *condition*), kuris gali įvykti tiesiog nurodyti kokiomis sąlygomis esant gali įvykti tas įvykis.



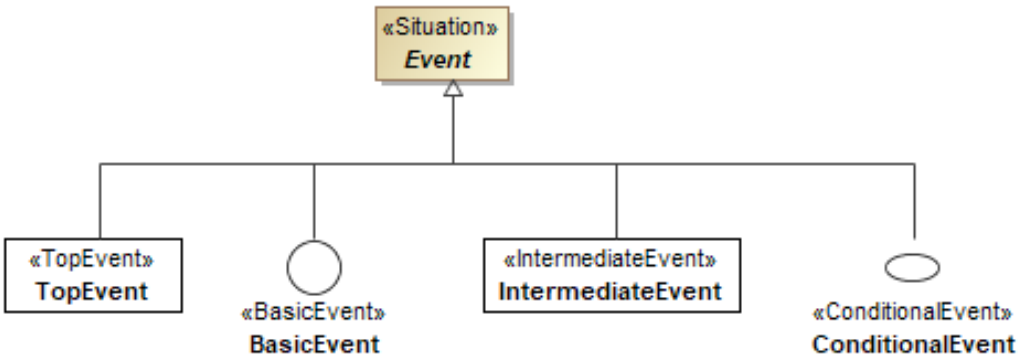
3.8 pav. Bibliotekoje esančios loginių vartų priklausomybės

Šie loginiai vartai yra pranašesni už savo susikurtus loginius vartus tuo, kad juose (tipuose) jau yra suprogramuotos formulės pasinaudojant parametrikų diagrama. Jos pagalba yra lengvai leidžiama apskaičiuoti įvykių tikimybes (3.9 pav.). Kitų loginių vartų programiniai kodai buvo pateikti prieduose (7.2. skyriuje).



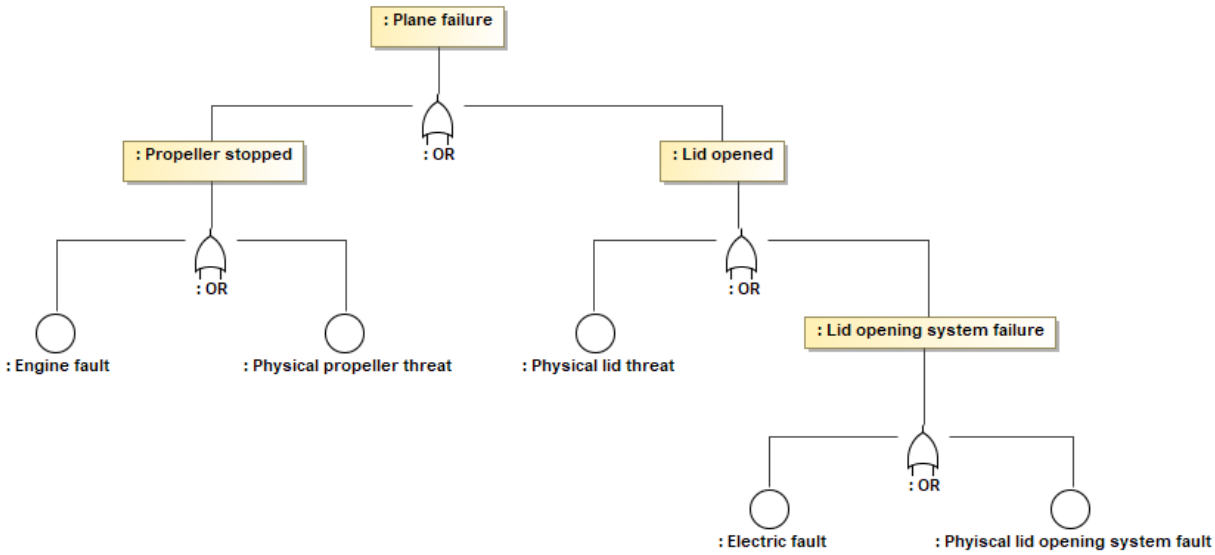
3.9 pav. Parametrikų diagrama aprašyta IR loginių vartų logika

Į biblioteką taip pat įeina ir įvykių paveldimumas (3.10 pav.). Jame galime pamatyti, jog turime abstraktų įvykį (angl. event), iš kurio yra paveldima visa informacija apie bendras įvykių savybes (pvz. tikimybių atributas). Detalesnių įvykiai yra pagrindinio nenorimo įvykis (angl. *top event*), paprastasis įvykis (angl. *basic event*), tarpinis įvykis (angl. *intermediate event*) ir sąlyginis įvykis (angl. *conditional event*).



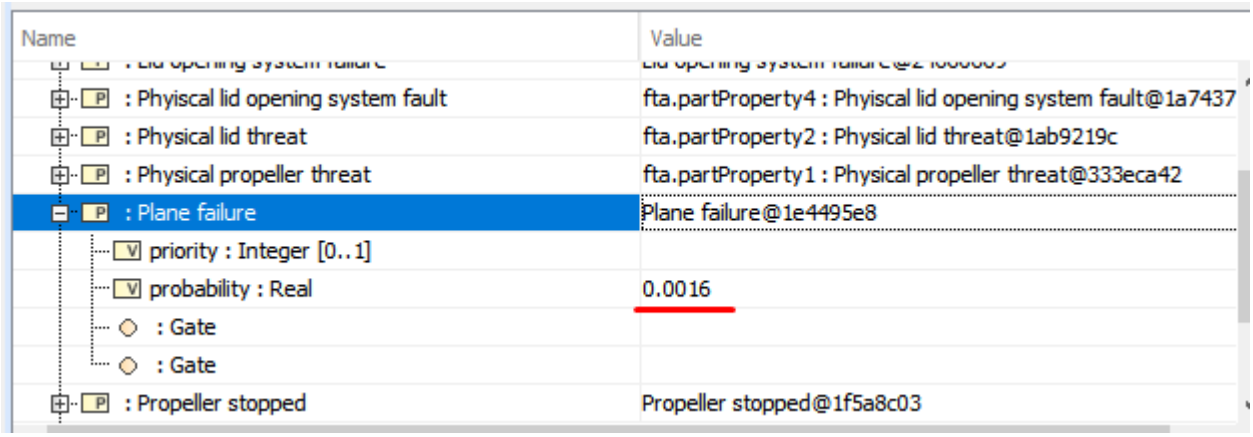
3.10 pav. Įvykių paveldimumas sukurtoje bibliotekoje

Pritaikytas sprendimas buvo ištestuotas ant koncepcinio lėktuvo modelio. Pagal sukurtą *SysML* lėktuvo modelį buvo sukurta gedimų medžio analizė (3.11 pav.). o paprastiesiems įvykiams nurodytos tikimybės.



3.11 pav. Gedimų medžio analizė sukurta koncepciniam lėktuvo modeliui

Panaudojant *MagicDraw* įrankio teikiamą simuliacijos funkcionalumą, sukurtai medžio analizei buvo paskaičiuota aukščiausio nenorimo įvykio tikimybė (3.12 pav.)



3.12 pav. Paskaičiuota nenorimo įvykio tikimybė pagal sukurtą gedimų medžio analizę

Pagal atliktą testavimą, sukurtas sprendimo prototipas ir siūlomas metodas yra tenkinamas norint atlikti kiekybinę gedimų medžio analizę modeliais grindžiamoje architektūroje. Atsekamumo užtikrinimą tarp architektūriškai skirtingų SysML modelių siūlomame metode atskleis atliktas eksperimentas 5 skyriuje.

4. EKSPERIMENTINIS SPRENDIMO GEDIMŲ MEDŽIO ANALIZĖS MODELIAIS GRINDŽIAMOJE ARCHITEKTŪROJE TYRIMAS

4.1. Eksperimento planas

Eksperimento metu siekiama išbandyti sukurti gedimų medžio analizes su paskaičiuojamomis tikimybėmis skirtingų struktūrų *SysML* modeliams. Eksperimentinis tyrimas turėtų parodyti ar pateikiamas sprendimas yra tinkamas daugeliui *SysML* metodologijų, tai yra – ar galima užtikrinti atsekamumą tarp *SysML* modelio bei gedimų medžio analizės.

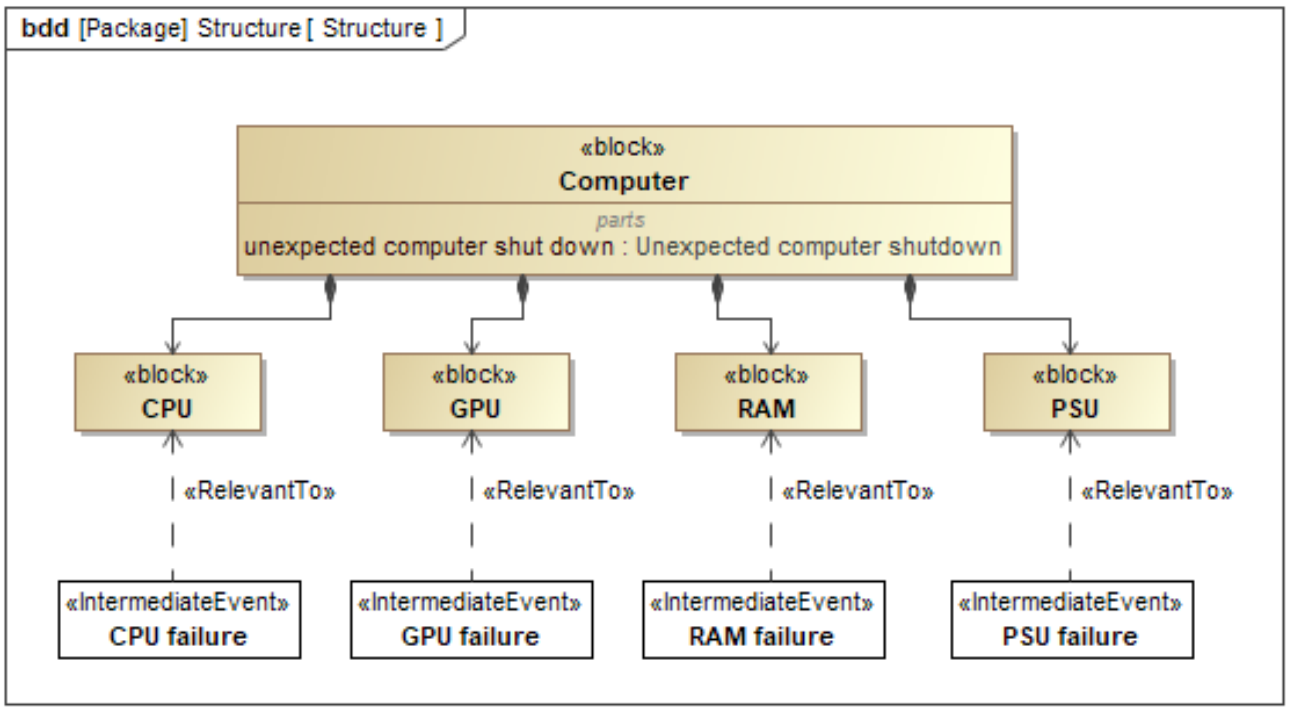
Eksperimento planas:

1. ištestuoti sukurtą sprendimą, kuris būtų pritaikytas sukurtam *SysML* sistemos modelui;
2. ištestuoti sukurtą sprendimą, kuris būtų pritaikytas sukurtiems *SysML* sistemos reikalavimams (neturint sistemos modelio);

4.2. Eksperimento rezultatai

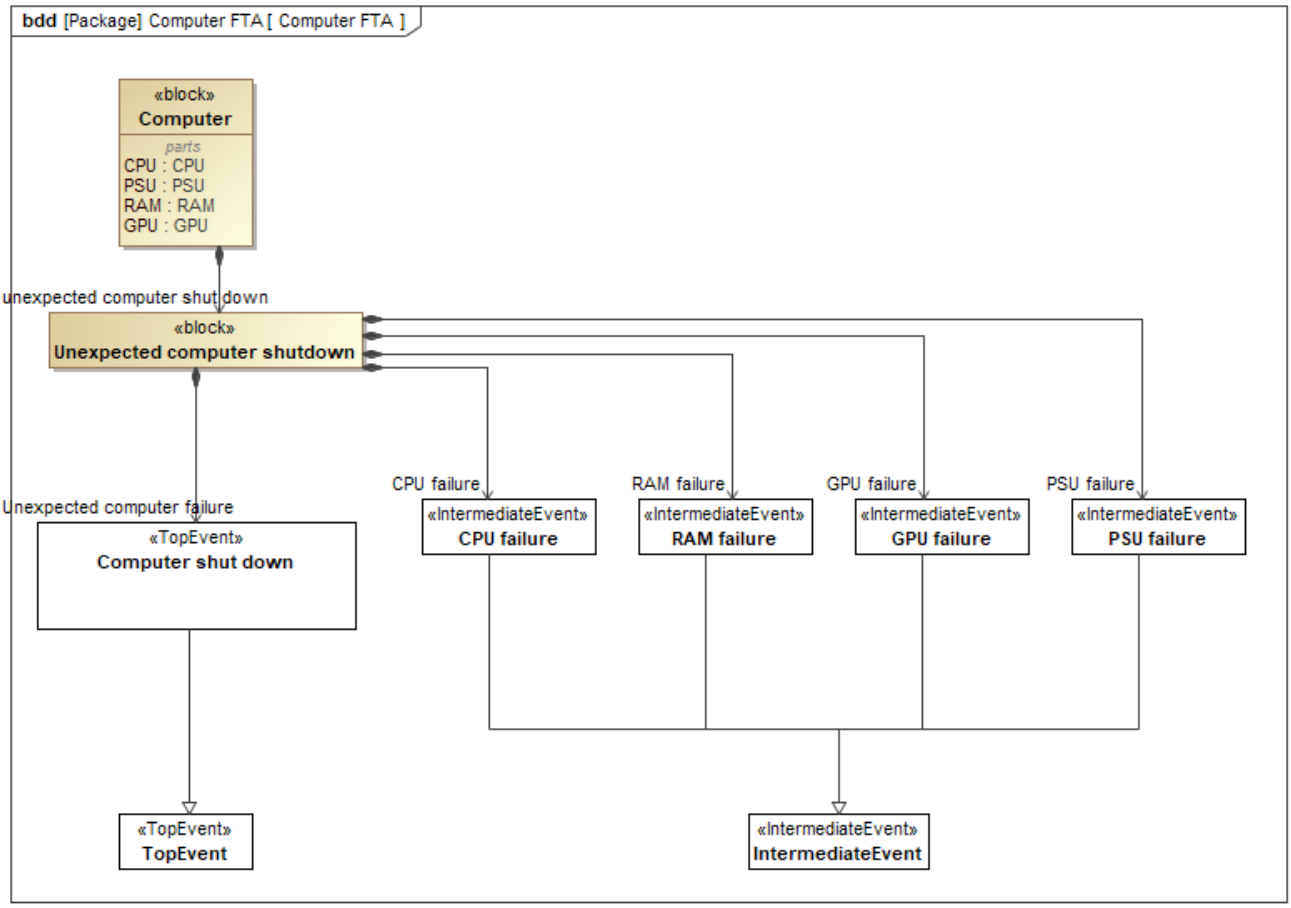
4.2.1. Kompiuterinės sistemos eksperimentas

Kompiuterinės sistemos eksperimento metu buvo panaudotas sistemos modelis, kuris neturi sumodeliuotų reikalavimų. Kompiuterinė sistema susideda iš keturių detalių: procesoriaus (angl. *computing processing unit – CPU*), spačiosios atminties (angl. *random access memory – RAM*), grafinio procesoriaus (angl. *graphics processing unit – GPU*) bei maitinimo blokelio (angl. *power supply unit – PSU*) (4.1 pav.).



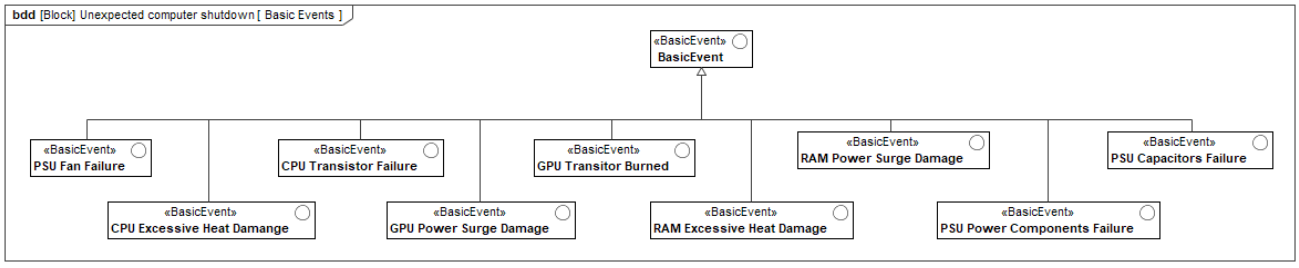
4.1 pav. Kompiuterinės sistemos eksperimento modelio struktūra

Pagrindiniam „*Computer*“ blokui buvo sukurtas kontekstas neigiamui įvykiui laikyti „Netikėtas kompiuterio išsijungimas“ (angl. *Unexpected computer shutdown*). Šiame kontekste buvo sukurti nenorimi įvykiai kompiuteriui (4.2 pav.).



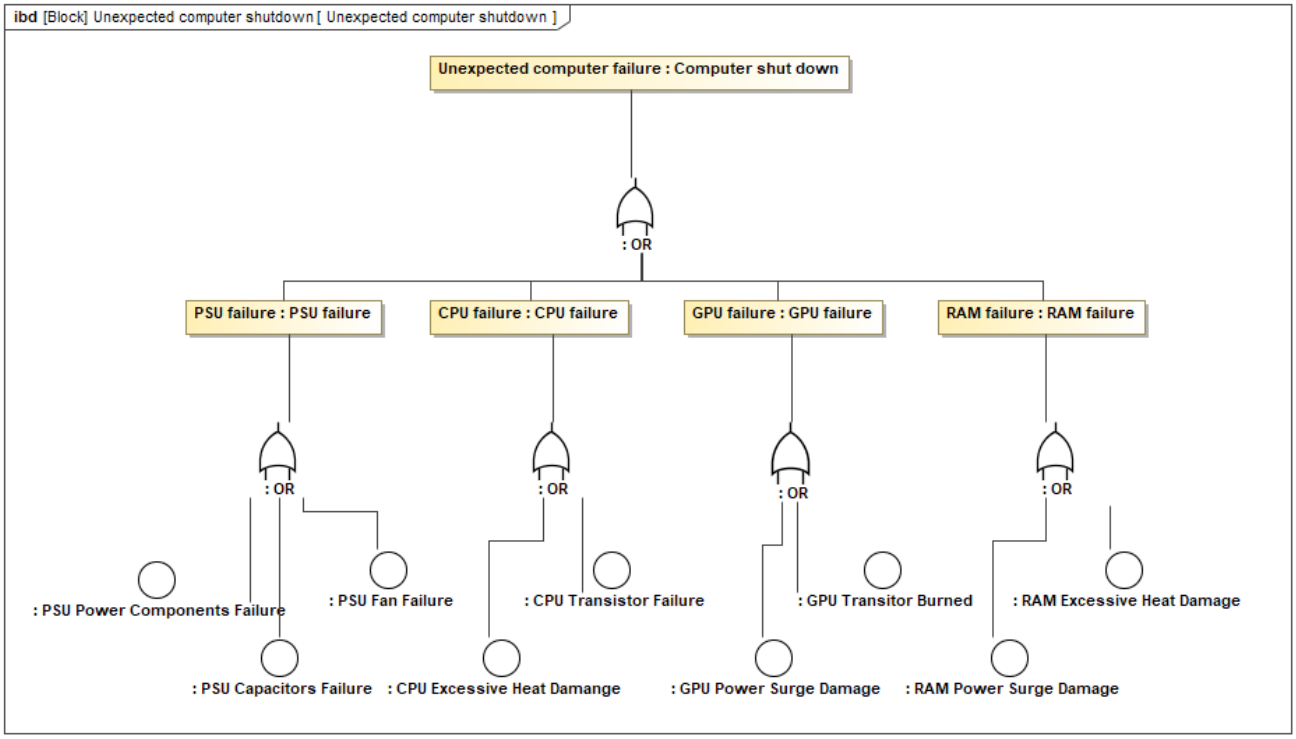
4.2 pav. Kompiuterio sistemos nenorimų įvykių diagrama

Sukurtiems nenorimiems įvykiams buvo sukurti paprastieji įvykiai, kurie dar labiau nustato nenorimų įvykių priežastis (4.3 pav.).



4.3 pav. Paprastieji kompiuterio sistemos įvykiai

Kai kuriems nenorimiems įvykiams buvo nubrėžti atsekamumo ryšiai „*RelevantTo*“, kurie leidžia užtikrinti atsekamumą tarp kompiuterio sistemos dalių (blokų) bei nenorimų įvykių (*4.1 pav.*). Visiems sumodeliuotiems įvykiams buvo nubrėžta viena gedimų medžio analizės diagrama, kuri parodo reikiamas sąlygas įvykti nenorimam kompiuterio išsijungimui (*4.4 pav.*).



4.4 pav. Gedimų medžio analizės diagrama kompiuterio sistemai

Visiems nenorimiems įvykiams buvo nurodytos tikimybės ir pasinaudojant sistemos simuliacijos funkcionalumo pagalba automatiškai buvo paskaičiuota pagrindinio nenorimo įvykio tikimybė (4.5 pav.).

P Unexpected computer failure : Compute...	Computer shut down@1d1a337d
V priority : Integer [0..1]	
V probability : Real	0.0017

4.5 pav. Paskaičiuota pagrindinio nenorimo įvykio tikimybė

Atlikus šį eksperimentą, buvo paskaičiuota statistika (4.1 lentelė), kurioje galime pastebėti, jog nors ir susietų *FTA* įvykių bei *SysML* elementų porų skaičius yra vos 5 (iš jų yra tik keturi atsekamumo ryšiai buvo sukurti tarp *SysML* kompiuterio modelio ir *FTA* nenorimų įvykių), tačiau pažiūrėjus iš loginės perspektyvos, galime pasakyti, jog kiekvienas pagrindinis įvykis įeina į tarpinį įvykį, kuris turi šį atsekamumo ryšį su sistemos modeliu, o pagrindinis įvykis tiesiogiai yra kylantis iš kompiuterio sistemos konteksto. Iš to darome išvadą, jog **kiekvienas** *FTA* sumodeliuotas nenorimas įvykis turi sukurtą atsekamumą su sistemos modeliu

4.1 lentelė. Kompiuterinės sistemos eksperimentas

Sukurtų bendrinių[2] *FTA* įvykių skaičius	14
Sukurtų konkrečių[3] *FTA* įvykių skaičius	14
Susietų fizinių *FTA* įvykių bei *SysML* elementų porų skaičius	5
Panaudotas loginių vartų skaičius	5
SysML elementų skaičius modelyje (prieš atliekant *FTA*)	5

Sudarius lentelę pagal atliktą eksperimentą (4.6 pav.), galime matyti, jog kiekviena kompiuterio sisteminė dalis turi tiesiogiai sukurtą atsekamumo ryšį „RelevantTo“ tarp nenorimų įvykių, o visus nenorimus įvykius savyje turi kompiuterio sistema bei jos kontekstas „Netikėtas kompiuterio išsijungimas“ (angl. *Unexpected computer shutdown*).

#	Name	Relevant From	△ Owning
1	CPU	CPU failure	
2	GPU	GPU failure	
3	PSU	PSU failure	
4	RAM	RAM failure	
5	Unexpected computer shutdown		OR Computer shut down PSU Fan Failure RAM Power Surge Damage RAM failure PSU Capacitors Failure CPU Transistor Failure GPU failure CPU failure PSU failure ...
6	Computer		PSU RAM CPU Unexpected computer shutdown GPU

4.6 pav. Kompiuterio sistemos ir *FTA* ryšius atvaizduojanti lentelė

[2] Bendriniai įvykiai – nenorimi įvykiai, kuris gali būti panaudotas keliuose kontekstuose bei diagramose
[3] Konkretūs įvykiai – konkretūs nenorimi įvykiai, kurie yra priskirti konkrečiam kontekstui bei diagramai

4.2.2. Krovininio automobilio ratų eksperimentas

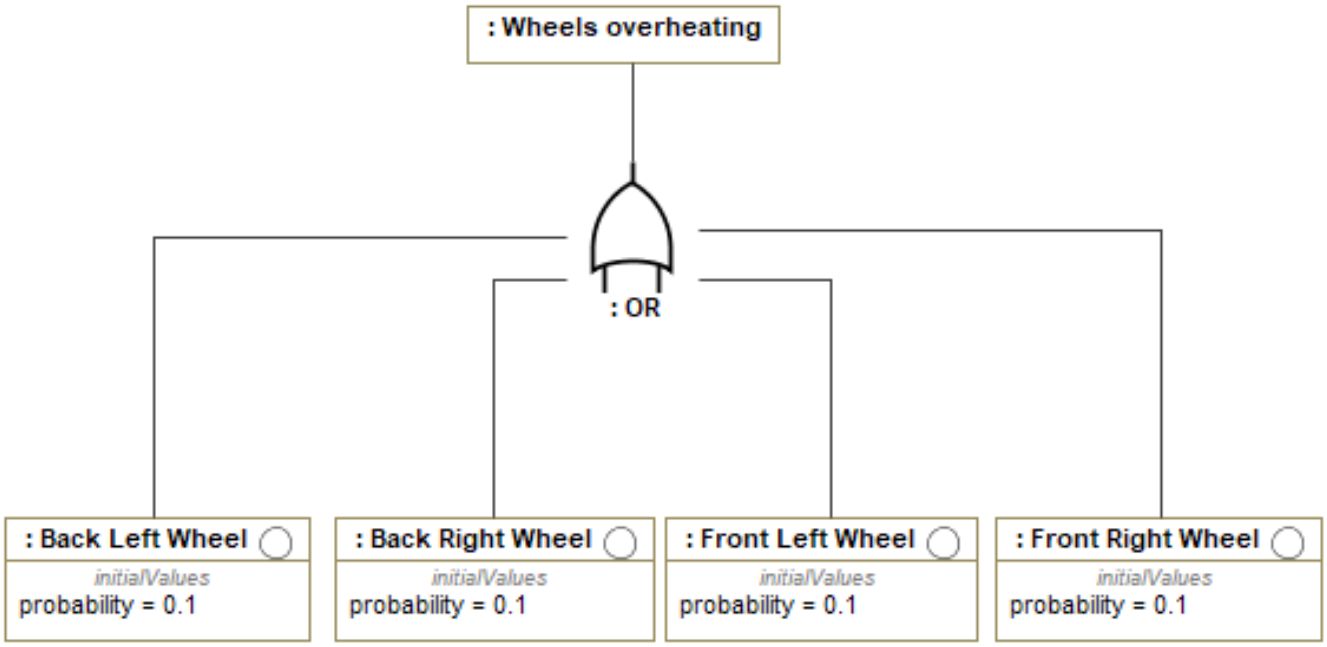
Šis eksperimentas skirtas išbandyti sukurtą sprendimą pritaikyti turint tik sistemos reikalavimus: dar neturint viso krovininio automobilio modelio. Neretai daugelis inžinerinių dizaino sprendimų būna priimama prieš pradedant modeliuoti visą sistemą, tai yra turint tik reikalavimus. Vienas iš būdų pasirinkti tam tikrą dizainą prieš modeliuojant – atlikti gedimų medžio analizę reikalavimams.

Eksperimento metu buvo pasiimti krovininio automobilio reikalavimai (4.7 pav.). Vienas iš reikalavimų teigia, jog vienos pusės ratų perkaitimo tikimybė turėtų būti mažesnė nei 0,1.

#	Name	Text
1	Stopping Distance	The Vehicle stopping distance shall not exceed the values in Table 1.
2	Brake Heating	Braking at 100 kilometers per hour shall generate less than 53 kW of heat at each wheel.
3	Brake Pad Life	Brake pads shall have a projected life of at least 57,500 kilometers under normal driving conditions, as per industry standard assumptions.
4	Vehicle Weight	The vehicle weight shall be equal to or less than 4000 kilograms.
5	Tires	The tires shall have 22-inch rolling diameter
6	Rotor Diameter	The brake rotors shall have a 0.28 meter diameter.
7	Pad Width	The Pad width shall be between 0.04 and 0.065 meters
8	Pad Center Length	The Pad Center Length shall be between 0.075 and 0.14 meters.
9	Pad Center Thickness	The Pad Center Thickness shall be between 5 and 11.5 millimeters.
10	Wheels overheating probability	One side wheels overheating probability shall be less than 0.1.

4.7 pav. Krovinio automobilio reikalavimai

Žinant tokį reikalavimą, bei ratų modelį (perkaitimo tikimybė 0,1) galime sukurti gedimų medžio analizę, kuri leidžia ištirti, ar užtektų uždėti po vieną ratą kiekvienoje pusėje (4.8 pav.).



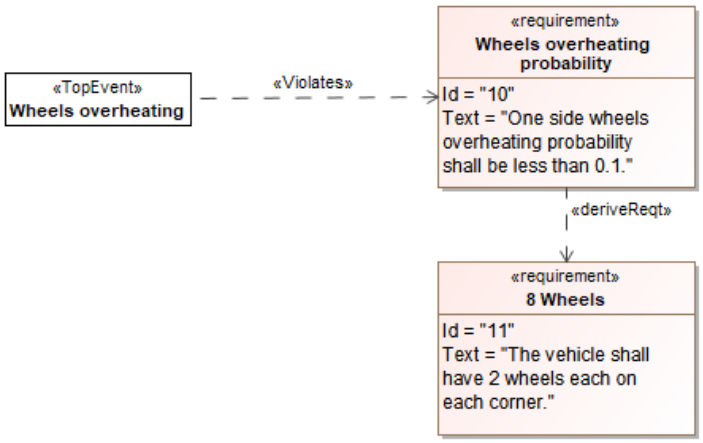
4.8 pav. Gedimų medžio analizė keturiems ratams (po vieną visose pusėse)

Pasinaudojant sukurtu sprendimu ir MagicDraw simuliacijos teikiamu funkcionalumu, galime automatiškai ir greitu būdu paskaičiuoti tikimybę (4.9 pav.), kuri yra lygi 0,3439.

Name	Value
Wheel heating FTA	Wheel heating FTA@fb32d5
: Wheels overheating	Wheels overheating@73d09432
priority : Integer [0..1]	
probability : Real	0.3439

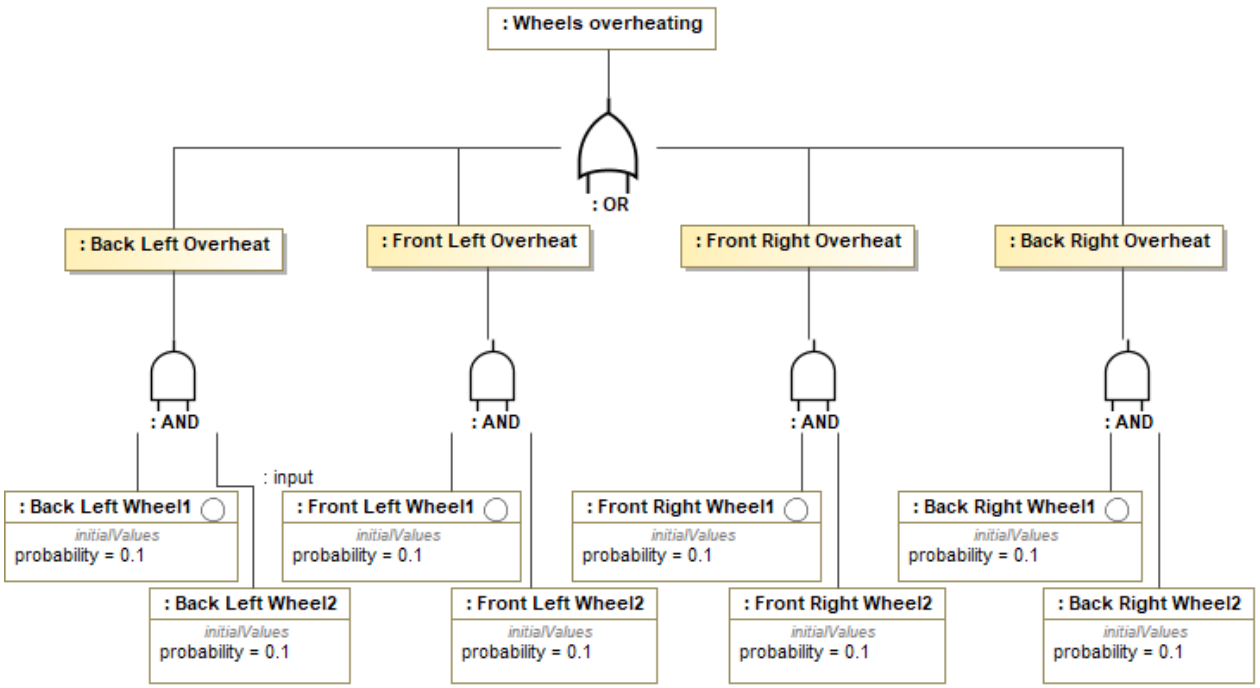
4.9 pav. Gedimų medžio kiekybinė analizė keturiems ratams

Matant tokį rezultatą, iš karto galime teigti, jog tokia tikimybė pažeidžia pirminius mūsų reikalavimus (4.10 pav.). Gedimų medžio analizė, kuri yra atlikta tik reikalavimams jau parodo galimas sistemos klaidas.



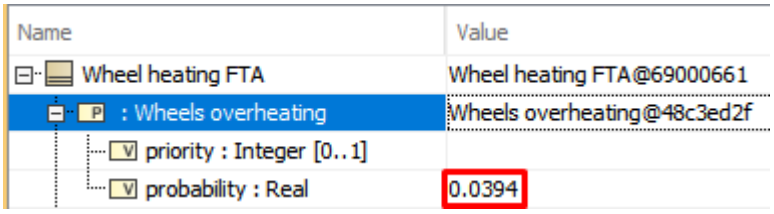
4.10 pav. Pažeidžiamas ratų reikalavimas

Todėl dar prieš kuriant konkretų sistemos modelį, sistemos inžinieriai turi galvoti apie kitus dizaino sprendimus ir priimti naujus papildomus reikalavimus, pavyzdžiui, jog uždėti po du ratus, į kiekvieną krovininio automobilio pusę (4.10 pav.) bei šiam atvejui atlikti gedimų medžio analizę (4.11 pav.).



4.11 pav. Gedimų medžio analizė aštuoniems ratams.

Atlikus šią analizę reikia pažiūrėti ir jos kiekybinius rezultatus (4.12 pav.). Šie rodo, jog perkaitimo tikimybė yra 0,0394, o tokie rezultatai – neviršija pradinio reikalavimo, todėl jau reikalavimų kūrimo metu panaudota gedimų medžio analizė gali nuspręsti dizaino sprendimus.



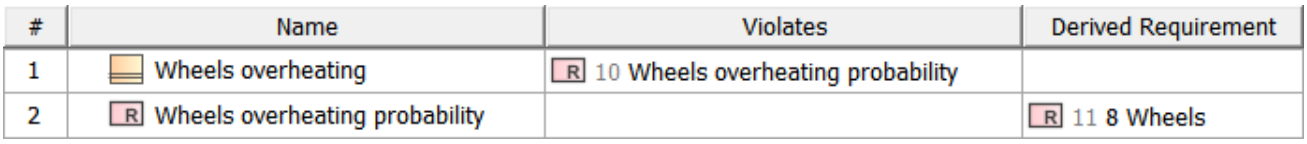
4.12 pav. Kiekybinė gedimų medžio analizė aštuoniems ratams

Tai įrodo, jog sukurtas sprendimas tinka ne tik susieti gedimų medžio analizę su esančiais sistemos komponentais, tačiau dar ir reikalavimų etape, prieš pradedant modeliuoti konkrečią sistemą. Eksperimento metu taip pat buvo paskaičiuota statistika (4.2 lentelė), kurioje galime pastebėti, jog siūlomą sprendimą galima naudoti konkrečiam vienam reikalavimui, kuris yra aprašytas *SysML* aplinkoje. Tam tereikėjo užveisti vienintelį atsekamumo (pažeidžiamumo (angl. *violates*)) ryšį su reikalavimu pagrindiniam nenorimam įvykiui.

4.2 lentelė. Krovininio automobilio ratų eksperimento statistika

Sukurtų bendrinių *FTA*[4] įvykių skaičius	13
Sukurtų konkrečių *FTA*[5] įvykių skaičius	13
Susietų fizinių *FTA* įvykių bei *SysML* elementų porų skaičius	1
Panaudotas loginių vartų skaičius	5
SysML elementų skaičius modelyje (prieš atliekant *FTA*)	10

Atlikus *FTA* analizę *SysML* reikalavimų modelis buvo tobulinamas ir pasitelkiant *FTA* analizės rezultatus buvo išvestas naujas reikalavimas sistemai. Gedimų medžio analizės susiejimas su reikalavimais matomas 4.13 pav., kuriame matosi, jog pagrindinis nenorimas įvykis „*Wheels overheating*“ pažeidžia reikalavimą „*Wheels overheating probability*“, o iš šio reikalavimo yra išvedamas (angl. *derive*) naujas reikalavimas sistemai.



4.13 pav. Krovininio automobilio ratų eksperimento naujų reikalavimų išvedimas naudojant *FTA*

4.3. Eksperimento rezultatai

Eksperimento metu buvo panaudoti du skirtingi modeliai, kuriems buvo kuriamos gedimų medžio analizės. Abu atvejai naudojantis sukurtu metodu bei prototipu buvo išanalizuoti jiems pritaikant gedimų medžio analizės principus ir abiems buvo sukurti atsekamumo ryšiai tarp *SysML* modelio ir

[4] Bendriniai įvykiai – nenorimi įvykiai, kuris gali būti panaudotas keliuose kontekstuose bei diagramose
[5] Konkretūs įvykiai – konkretūs nenorimi įvykiai, kurie yra priskirti konkrečiam kontekstui bei diagramai

FTA. Tai įrodo, jog sukurtas sprendimas nėra sukurtas konkrečiai modeliavimo metodologijai, tačiau pasiūlytas metodas yra universalus visoms modeliais grindžiamoms architektūroms.

5. REZULTATŲ APIBENDRINIMAS IR IŠVADOS

1. Darbo metu buvo išanalizuoti saugumo analizės procesai (*FTA* ir *FMEA*) ir išsiaiškinta, kad šie procesai dažniausiai atliekami sistemos architektūros kūrimo metu ir neretai jų sukurti elementai turi ryšių su sistemos modeliu (*SysML*).

2. Išanalizavus esamus sprendimus gedimų medžio analizei atlikti paaiškėjo, kad nėra sukurto sprendimo, kuris leistų sukurti sistemos *SysML* modelį ir kartu leistų atlikti gedimų medžio analizę viename įrankyje, ar kitokiu būdu užtikrintų atsekamumą tarp šių dviejų projekto dalių.

3. Kuriant metodą bei projektuojant prototipą buvo nustatyta, jog egzistuojančios *SysML* diagramos leistų atlikti gedimų medžio analizę, jei *SysML* kalba būtų išplėsta keliais naujais elementais bei tikimybių apskaičiavimams būtų sukurta biblioteka, leidžianti panaudoti universalius loginių vartų modelius grįstus parametrikų diagramomis.

4. Prototipo realizacijos metu buvo nutarta, jog sprendimui naudoti patogiausia būtų įsidiegti papildomą MagicDraw įskiepį, kuris užkrautų tiek *SysML* kalbos plėtinį, tiek ir biblioteką su *FTA* reikiamais elementų tipais. Realizacijos metu taip pat buvo sukurtas koncepcinis lėktuvo modelis, kuriam atlikta gedimų medžio analizė ir automatinis tikimybių skaičiavimas, įrodantis, kad sprendimas yra tinkamas sistemos *SysML* modeliui.

5. Eksperimento metu buvo sukurti du skirtingi pavyzdžiai, kurie parodo, jog pateiktas sprendimas gali būti naudojamas ne tik sukurto sistemos modelio saugumo testavimui, tačiau dar ir prieš pradedant sistemos projektą – reikalavimų projektavimo metu. Pateiktas sprendimas leidžia kelių mygtukų paspaudimu sukurti atsekamumą tarp sistemos ar reikalavimų modelio bei gedimų medžio analizės bei automatiškai paskaičiuoti nenorimų įvykių tikimybes.

6. LITERATŪRA

[1] Gabriel Petrica, Ionut-Daniel Barbu, Sabina-Daniela Axinte, Cristian Pascariu, „Reliability analysis of a Web server by FTA method," *2017 10th International Symposium on Advanced Topics in Electrical Engineering,* 2017.

[2] Ericson, Clifton A., II, Hazard Analysis Techniques for System Safety, 2005.

[3] LEE, Wen-Shing, et al., „Fault Tree Analysis, Methods, and Applications," *IEEE transactions on reliability,* pp. 194-203, 1985.

[4] K. Durga Rao, V. Gopika, V.V.S. Sanyasi Rao, H.S. Kushwaha, A.K. Verma, A. Srividya, „Dynamic fault tree analysis using Monte Carlo simulation in probabilistic," *Reliability Engineering & System Safety,* t. 94, nr. 4, pp. 872-883, 2009.

[5] B. S. Dhillon and Canan, „TEXAS A&M UNIVERSITY, Chapter 4. Fault Trees and Common Cause Failures," [Tinkle]. Available: http://ece.tamu.edu/~c-singh/enggreliability/Chap4.pdf. [Kreiptasi 13 11 2017].

[6] Ogren Ingmar, „On Principles forModel-Based Systems Engineering," 1998.

[7] Muhammed Waseem, M. U. Sadiq, „Application of Model-Based Systems Engineering in Small Satellite Conceptual Design - A SysML approach," *IEEE Aerospace and Electronic Systems Magazine,* 2018.

[8] Donatas Mažeika, Aurelijus Morkevičius, Aistė Aleksandravičienė, „MBSE driven approach for defining problem domain," *System of Systems Engineering Conference (SoSE),* 2016.

[9] Yaniv Mordecai, D. Dori, „Agile modeling of an evolving ballistic missile defense system with Object-Process Methodology," *Annual IEEE Systems Conference (SysCon) Proceedings,* 2015.

[10] Sanford Friedenthal, Alan Moore and Rick Steiner, A Practical Guide to SysML. The Systems Modeling Language, 2008.

[11] „What is SysML?," OMG, [Tinkle]. Available: http://www.omgsysml.org/what-is-sysml.htm. [Kreiptasi 10 12 2019].

[12] „Requirement Diagram," No Magic, 09 12 2015. [Tinkle]. Available: https://docs.nomagic.com/display/SYSMLP185/Requirement+Diagram. [Kreiptasi 10 12 2017].

[13] „SysML Parametric diagram," No Magic, 11 04 2017. [Tinkle]. Available: https://docs.nomagic.com/display/SYSMLP185/SysML+Parametric+Diagram. [Kreiptasi 10 12 2017].

[14] „SysML block definition diagram," No Magic, 04 04 2017. [Tinkle]. Available: https://docs.nomagic.com/display/SYSMLP185/SysML+Block+Definition+Diagram. [Kreiptasi 10 12 2017].

[15] „SysML internal block diagram," No Magic, 04 04 2017. [Tinkle]. Available: https://docs.nomagic.com/display/SYSMLP185/SysML+Internal+Block+Diagram. [Kreiptasi 10 12 2017].

[16] Darryl W. Kellner, „Software FMEA: A successful application for a complex service oriented architecture system," įtraukta *Reliability and Maintainability Symposium (RAMS),* Orlando, FL, USA, 2017.

[17] „Cameo safety and reliability analyzer plugin," No magic, [Tinkle]. Available: https://www.nomagic.com/product-addons/no-cost-add-ons/cameo-safety-and-reliability-analyzer-plugin#screenshots. [Kreiptasi 11 12 2017].

[18] Masakazu Takahashi, Riki Kosaka, Reiji Nanba, Yunarso Anang, Yoshimichi Watanabe, „A study of methodology for securing control software based FMEA-FTA coordination," *IEEE/SICE International Symposium on System Integration,* 2016.

[19] „RAM Commander - 2017 Pricing and Implementation Cost,“ ITQlick, [Tinkle]. Available: https://www.itqlick.com/ram-commander/pricing. [Kreiptasi 11 12 2017].

[20] HiP-HOPS, *HiP-HOPS User Manual. Automated Fault Tree, FMEA and Optimisation Tool,* 2013.

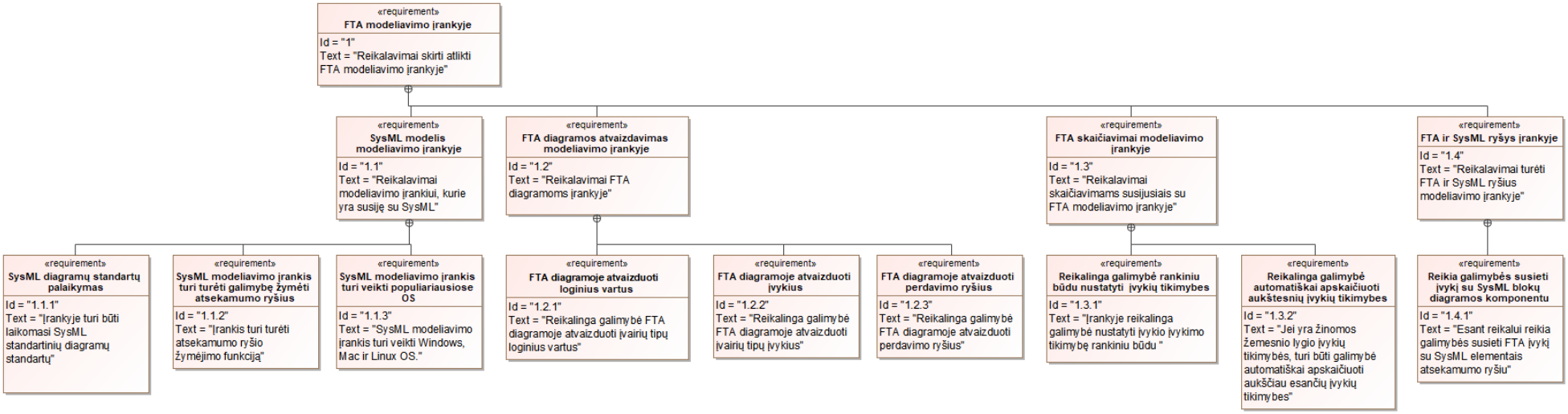
[21] „Fault Tree Analysis,“ Isograph, [Tinkle]. Available: https://www.isograph.com/software/reliability-workbench/fault-tree-analysis/. [Kreiptasi 14 12 2017].

[22] Nataliya Yakymets, Hadi Jaber, Agnes Lanusse, „Model-Based System Engineering for Fault Tree Generation and Analysis,“ *MODELSWARD,* 2013.

[23] Faida Mhenni, Nga Nguyen and Jean-Yves Choley, „Automatic Fault Tree Generation From SysML System Models,“ *Advanced Intelligent Mechatronics,* 2014.

7. PRIEDAI

7.1. priedas. Reikalavimų diagrama



7.1 pav. Sumodeliuota *SysML* reikalavimų diagrama skirta pagerinti gedimų medžio analizės procesą modeliais grindžiamoje architektūroje

7.2. priedas. Loginių vartų formulės

Loginių vartų IR skaičiavimas programiniu kodu atvaizduotas 1.2 pav.

```
if (input.length > 0) {
    output = 1;
}
for (i = 0; i < input.length; i++) {
    output = input[i] * output;
}
```

1.2 pav. Loginių vartų IR skaičiavimo formulė

Loginių vartų ARBA skaičiavimas programiniu kodu atvaizduotas 1.3 pav.

```
if (input.length > 0) {
    output = 1;
}
for (i = 0; i < input.length; i++) {
    output = (1 - input[i]) * output;
}

output = 1 - output;
```

1.3 pav. Loginių vartų ARBA skaičiavimo formulė

Loginių vartų XOR skaičiavimas programiniu kodu atvaizduotas 1.4 pav.

```
if (input.length > 0) {
    output = input[0];
}
for (i = 1; i < input.length; i++) {
    output = (output + input[i]) - (2 * output * input[i]);
}
```

1.4 pav. Loginių vartų XOR skaičiavimo formulė

Loginių vartų NE (angl. *NOT*) skaičiavimas programinu kodu atvaizduotas 1.5 pav.

```
output = 1 - input;
```

1.5 pav. Loginių vartų NE skaičiavimo formulė

Loginių vartų NELEIDŽIAMŲJŲ (angl. *INHIBIT*) skaičiavimas programiniu kodu atvaizduotas 1.6 pav.

```
if (input.length > 0) {
    output = 1;
    for (i = 0; i < input.length; i++) {
        output = input[i] * output;
    }
}
if (condition.length > 0) {
    for (i = 0; i < condition.length; i++) {
        output = condition[i] * output;
    }
}
```

1.6 pav. Loginių vartų INHIBIT skaičiavimo formulė

Loginių vartų MAJORITY VOTE skaičiavimas programiniu kodu atvaizduotas 1.7 pav.

```
function getCombinations(array, min) {
    var res = [];
    function iter(i, temp) {
        if (i === array.length) return;
        var t = temp.concat(array[i]);
        iter(i + 1, t);
        iter(i + 1, temp);
        if (t.length >= min) {
            res.push(t);
        }
    }

    iter(0, []);
    return res;
}

function and(arrayOfProbabilities) {
    if (arrayOfProbabilities.length > 0) {
        output = 1;
    }
    for (i = 0; i < arrayOfProbabilities.length; i++) {
        output = arrayOfProbabilities[i] * output;
    }
    return output;
}


function or(arrayOfProbabilities) {
    if (arrayOfProbabilities.length > 0) {
        output = arrayOfProbabilities[0];
    }
    for (i = 1; i < arrayOfProbabilities.length; i++) {
        output = (output + arrayOfProbabilities[i]) - (output * arrayOfProbabilities[i]);
    }
    return output;
}

var allCombinations = getCombinations(input, m);
var andsArray = [];

for (j = 0; j < allCombinations.length; j++) {
    andsArray.push(and(allCombinations[j]));
}

output = or(andsArray);
```

1.7 pav. Loginių vartų MAJORITY VOTE skaičiavimo formulė