Politechnika Wrocławska
Wydział Elektroniki
Instytut Informatyki, Automatyki i Robotyki

Rozprawa doktorska

Metody analizy systemów oparte na czasowych rozszerzeniach drzew niezdatności

Autor:

Marcin Kowalski

Promotor:

Prof. dr hab. inż. Jan Magott

Wrocław, 2013

Spis treści

Spis treści **i**

Wykaz skrótów **v**

Wykaz oznaczeń **vii**

Spis modeli i schematów **viii**

1 Wstęp **3**

2 Metody analizy niezawodności systemów technicznych **5**

2.1 Niezawodność systemów z zależnościami czasowymi 6

2.2 Języki modelowania niezawodności 7

2.3 Dotychczas opracowane warianty drzew niezdatności 8

2.4 Sieci Petriego w rozstrzyganiu znaczenia języków formalnych 10

2.5 Prace wokół stworzenia uniwersalnego języka modelowania 12

2.5.1 Rozszerzenia UML'a dla opisu niezawodności 13

2.5.2 Dotychczasowe osiągnięcia w definicji semantyki diagramów aktywności . 15

2.6 Oprogramowanie wspierające analizę niezawodności 17

2.7 Narzędzia metamodelowania i transformacji międzymodelowych 19

3 Cele i organizacja pracy **21**

3.1 Koncepcje poszukiwań języków opisu niezawodności 21

3.2 Teza i cele dysertacji 22

3.3 Struktura rozprawy 23

4 Etap 1: Opracowanie Probabilistycznych Drzew Niezdatności z Zależnościami Czasowymi **25**

4.1 Wprowadzenie do drzew niezdatności 26

4.2 Składnia i semantyka PDNZC 27

4.2.1 Bramy uogólniające 27

4.2.2 Bramy przyczynowe 29

4.3 Modelowanie i analiza polityk napraw systemu komputerowego 30
4.3.1 Globalna naprawa systemu 31
4.3.2 Odnowa systemu komputerowego poprzez równoczesną konserwację wszystkich uszkodzonych komponentów 32
4.3.3 Odnowa systemu poprzez sekwencyjną naprawę komponentów 32
4.3.4 Badania symulacyjne i weryfikacja wyników 35
4.4 Moc ekspresji PDNZC 35

5 Koordynacja opóźnień wielokrotnej redundancji gorącej zabezpieczeń odległościowych w elektroenergetyce z wykorzystaniem PDNZC 37
5.1 Struktura i działanie redundantnych zabezpieczeń odległościowych 38
5.2 Składnia abstrakcyjna języka do opisu redundancji w zabezpieczeniach 40
5.3 Składnia konkretna języka do opisu redundancji w zabezpieczeniach 41
5.4 Obiektowa specyfikacja PDNZC 45
5.5 Transformacja języka dziedzinowego w PDNZC 46
5.5.1 Zabezpieczenia zdalne 46
5.5.2 Zabezpieczenia lokalne 50
5.6 Inne zastosowania translatora 52
5.7 Omówienie wyników 54

6 Etap 2: Włączenie elementów sieci Petriego - Grafy Niezdatności 57
6.1 Podstawy Grafów Niezdatności 58
6.1.1 Zdarzenia proste i parametryczne 58
6.1.2 Semantyka bram i przejść 59
6.2 Polityki napraw systemu komputerowego 65
6.2.1 Zdarzenia parametryczne w modelu globalnej polityki napraw 65
6.2.2 Model polityki opartej o zasoby 66
6.3 Automatyczna transformacja grafów w symulator niezawodności 69
6.3.1 Obiektowa specyfikacja GN 69
6.3.2 Koncepcja symulatora 69
6.4 Modelowanie i symulacja modelu kolejkowego 72

7 Modelowanie i symulacja szeregowego systemu z zimną rezerwą funkcjonalną, dostawą uszkodzonego elementu po naprawie oraz rezerwą czasową 75
7.1 Pojęcie rezerwy czasowej w systemie wsparcia logistycznego 75
7.2 Analiza danych rzeczywistych o uszkodzeniach 76
7.3 Sieć Petriego wybranego systemu z rezerwą czasową i dostawą 81
7.4 Badania symulacyjne GN 84
7.5 Omówienie wyników 86
7.5.1 Zgodność wyników symulacyjnych z rzeczywistymi 86
7.5.2 Porównanie sieci Petriego z modelem GN 89

8 Etap 3: Rozszerzone Niezawodnościowo Diagramy Aktywności 91
8.1 Składnia READ jako profil UML . . . 91
8.1.1 Elementy READ bazujące na konstrukcjach UML . . . 93
8.1.2 Przejście wzorowane na sieciach Petriego . . . 97
8.1.3 Bramy drzew niezdatności . . . 97
8.1.4 Model czasowy . . . 98
8.1.5 Składnia języka wyrażeń . . . 98
8.2 Semantyka READ . . . 100
8.2.1 Problemy napotkane przy opisie READ sieciami Petriego . . . 100
8.2.2 Szkielet sieci Petriego odpowiadającej modelowi READ . . . 102
8.2.3 Brama przyczynowa OR . . . 104
8.2.4 Brama przyczynowa AND . . . 107
8.2.5 Brama uogólniająca OR . . . 108
8.2.6 Brama uogólniająca AND . . . 108
8.2.7 Brama opóźniająca . . . 109
8.2.8 Cykliczny generator obiektów . . . 110
8.2.9 Przejście . . . 111

9 Metody wyznaczania niezawodności systemów z czasem aktywacji rezerwy zimnej, dostawą oraz rezerwą czasową 113
9.1 Metodyka obliczeń . . . 114
9.2 Metoda Ia: oszacowanie oparte o rozkłady hipowykładnicze . . . 118
9.3 Metoda Ib: oszacowanie z użyciem statystyk pozycyjnych . . . 119
9.4 Metoda Ic: oszacowanie wykorzystujące zmienne wykładnicze . . . 120
9.5 Metoda II: oszacowanie przy pomocy rozkładów warunkowych . . . 120
9.6 Eksperyment obliczeniowy . . . 121
9.6.1 Model M/W/M . . . 122
9.6.2 Model W/W/M . . . 124
9.6.3 Model W/W/W i podsumowanie eksperymentu . . . 127

10 Analiza i modelowanie niskokosztowych procesów eksploatacji bez rezerwy, z naprawami oraz przeglądami 131
10.1 Niektóre założenia niskokosztowych linii lotniczych . . . 132
10.2 Model strukturalny systemu niskokosztowych linii lotniczych . . . 132
10.3 Główny model procesu eksploatacji . . . 134
10.4 Detekcja i obsługa uszkodzeń . . . 138
10.5 Model inspekcji dziennych . . . 139
10.6 Algorytm zastępowania lotów . . . 141

11 Podsumowanie i osiągnięcia 143
11.1 Rozszerzenia drzew niezdatności . . . 143

11.2 Nowe metody analizy niezawodności 144
11.3 Weryfikacja metody w wybranych klasach systemów 145

Bibliografia **149**

A Generacja symulatora niezawodności z modelu wyrażonego w języku GN **161**
A.1 Bramy uogólniające 161
A.2 Bramy przyczynowe 163
A.3 Przejścia 164

B Wprowadzenie do Języka Ograniczeń Obiektów (OCL) **167**

C Opis wykorzystanych narzędzi projektowania sieci Petriego **169**
C.1 CPN Tools 169
C.2 Snoopy 171

Wykaz skrótów

analiza SWOT Analiza Mocnych i Słabych Stron, Szans oraz Zagrożeń - *ang. Strengths, Weaknesses, Opportunities and Threats Analysis* 15

BPMN Notacja Modelowania Procesów Biznesowych - *ang. Business Process Modeling Notation* 14

CSP Komunikujące się Procesy Sekwencyjne - *ang. Communicating Sequential Processes* 16

DA diagramy aktywności 14–19, 91–94, 97, 100

DFT Dynamiczne Drzewa Niezdatności - *ang. Dynamic Fault Trees* 9, 10, 13, 17, 18, 22, 36

DRBD Diagramy Dynamicznych Bloków Niezawodności - *ang. Dynamic Reliability Block Diagrams* 10, 11

DS diagramy sekwencji 11, 14

DSL język dziedzinowy - *ang. domain specific language* 19, 37

EBNF Rozszerzona Postać Backus'a-Naur'a - *ang. Extended Backus-Naur Form* 40, 98

FMEA Procedura Analizy Rodzajów i Skutków Uszkodzeń - *ang. Failure Mode and Effect Analysis* 13, 15

FTA Analiza Drzew Niezdatności - *ang. Fault Tree Analysis* 15, 17, 18

GFT Uogólnione Drzewa Niezdatności - *ang. Generalized Fault Trees* 22

GN Grafy Niezdatności ii, iv, viii, ix, 23, 57–60, 65, 67, 69, 70, 73–76, 84–86, 88–91, 94, 97, 98, 100, 101, 112, 143, 145–147, 161, 163

GSPN Uogólnione Stochastyczne Sieci Petriego - *ang. Generalized Stochastic Petri Nets* 10, 11

HLPN Sieci Petriego Wysokiego Poziomu - *ang. High Level Petri Nets* 11, 18, 57, 75, 144, 146, 169

Marte Analiza i Modelowanie Systemów Czasu Rzeczywistego oraz Systemów Wbudowanych - *ang. Modeling and Analysis of Real Time and Embedded Systems* 15, 98, 99, 134

MDA Architektura Sterowana Modelem, - *ang. Model Driven Architecture* 12, 19, 143, 161

MSC Diagram Sekwencji Komunikatów - *ang. Message Sequence Chart* 12, 18

NDNZC Niedeterministyczne Drzewa Niezdatności z Zależnościami Czasowymi 9, 10, 14, 36, 54, 60, 145

NLL niskokosztowe linie lotnicze ix, 131–133, 135, 144

OCL Język Ograniczeń Obiektów - *ang. Object Constraint Language* iv, 13, 19, 37, 93, 103, 104, 167, 168

OMG Object Management Group 12, 13, 91

PDNZC Probabilistyczne Drzewa Niezdatności z Zależnościami Czasowymi viii, 23, 25, 27, 29, 30, 33, 35–38, 40, 41, 46, 52, 54, 57, 58, 60, 65, 66, 69, 76, 98, 101, 112, 131, 143, 145, 146, 167

PERT Technika Oceny i Weryfikacji Projektów - *ang. Program Evaluation and Review Technique* 36

PFT Parametryczne Drzewa Niezdatności - *ang. Parametric Fault Trees* 9, 10, 17, 18, 22

PIM Model Niezależny od Platformy - *ang. Platform Independent Model* 161

PSM Model Zależny od Platformy - *ang. Platform Specific Model* 161

QVT Query/View/Transform 19, 37, 46, 145

RBD diagramy bloków niezawodności - *ang. reliability block diagrams* 7

READ Rozszerzone Niezawodnościowo Diagramy Aktywności - *ang. Reliability Enhanced Activity Diagrams* ix, 15, 18, 23, 91, 93–98, 100–104, 111, 122, 131–133, 135, 138, 141, 143, 144, 146, 147, 167, 169

RFT Naprawialne Drzewa Niezdatności - *ang. Repairable Fault Trees* 9, 10, 15, 17, 22

SWN Stochastyczne Dobrze Zbudowane Sieci Petriego - *ang. Stochastic Well-Formed Colored Nets* 10, 146

SysML Język Modelowania Systemów - *ang. Systems Modeling Language* 12–17

TPN Czasowe sieci Petriego - *ang. Timed Petri Nets* 10, 11

UML Zunifikowany Język Modelowania - *ang. Unified Modeling Language* 11–19, 23, 91, 93, 94, 97, 98, 100, 102–104, 143, 144, 167, 169

XPDL Język Definicji Procesów Oparty o XML - *ang. XML Process Definition Language* 14

Wykaz oznaczeń

n liczba elementów rezerwy funkcjonalnej 76, 78, 115, 118, 119, 121, 123, 124, 127

A zmienna losowa czasu pomiędzy kolejnymi awariami vii, 76, 78–80, 113–115, 123, 124, 127, 147

CAND brama przyczynowa logicznej koniunkcji viii, 28, 29, 32, 60, 93, 171

COR brama przyczynowa logicznej alternatywy viii, ix, 29, 60, 61, 86, 161, 163, 164

D zmienna losowa czasu dostawy obejmująca czas naprawy uszkodzonego elementu, jak i opóźnienie do wznowienia funkcjonowania vii, 76–80, 113–115, 123, 124, 127, 147

E zmienna losowa czasu wymiany, po którym dostępny element zapasowy przejmuje zadania uszkodzonego vii, 76–80, 113, 114, 123, 124, 127, 147

Ecore język modelowania będący metamodelem dla języków PDNZC oraz GN 69, 144

FT czas trwania uszkodzenia w systemie z rezerwą czasową 114

GAND brama uogólniająca logicznej koniunkcji viii, 28, 59, 60, 86, 161

GOR brama uogólniająca logicznej alternatywy 60

H zdarzenie oznaczające hazard w systemie z rezerwą czasową 114, 123, 124

HT czas trwania hazardu 114, 115, 123, 124

M/M/M klasa systemów z rezerwą czasową, A, E i D rozkłady wykładnicze 128

M/W/M klasa systemów z rezerwą czasową, A i D rozkłady wykładnicze, E rozkład Weibulla iii, 122–124, 128, 145

$\mathbf{R_C}$ rezerwa czasowa rozumiana jako odcinek czasu, w którym uszkodzony element musi zostać dostarczony lub wymieniony na element zapasowy ix, 76, 78, 114–117, 123–127

W/W/M klasa systemów z rezerwą czasową, A i E rozkłady Weibulla, D rozkład wykładniczy iii, ix, 124–126, 128, 145

W/W/W klasa systemów z rezerwą czasową, A, E i D rozkłady Weibulla iii, ix, 127, 128, 145

Spis modeli i schematów

4.1 Model nieodnawialnego systemu komputerowego z redundancją 26
4.2 PDNZC systemu komputerowego z redundancją zaprojektowane w narzędziu autorskim 27
4.3 Model czasowy bram GAND i CAND 28
4.4 PDNZC opisujące globalną politykę odnowy 31
4.5 PDNZC opisujące równoległą politykę odnowy 33
4.6 Szkielet polityki szeregowej 34

5.1 Linia elektroenergetyczna 38
5.2 Model obiektowy języka dziedzinowego koordynacji zabezpieczeń 40
5.3 Diagram głównych klas języka PDNZC 45
5.4 Diagram klas bram uogólniających i przyczynowych 45
5.5 Drzewo niezdatności dla podsekcji a1 sekcji S1; P3 zabezpieczenie wielosystemowe . 47
5.6 Rozłączenie linii, gdy zabezpieczenie jest jednosystemowe z członem startowym . . . 51
5.7 Drzewo niezdatności dla podsekcji a1 sekcji S1; P3 zabezpieczenie jednosystemowe . 53
5.8 Linia elektroenergetyczna z zabezpieczeniami umieszczonymi na końcach sekcji . . . 54
5.9 Drzewo niezdatności dla podsekcji a2 sekcji S3, gdy zabezpieczenia są na końcach linii 55

6.1 Zdarzenia występujące w GN 58
6.2 Interpretacja bramy COR 61
6.3 Czas trwania zdarzenia wyrażony przez bramę CNOT 63
6.4 Przykładowe przejście GN 63
6.5 Modelowanie strumienia uszkodzeń 64
6.6 Uproszczenie modelu polityki globalnej 67
6.7 Polityka odnowy bazująca na zasobach współdzielonych 68
6.8 Obiektowy metamodel GN 70
6.9 Zmienne losowe i ich dystrybuanty wspierane przez symulator GN 71
6.10 Algorytm symulacji GN 71
6.11 Model kolejki G/G/k 73

7.1 Wykresy gęstości oraz Q-Q dla przypadków 1, 2, 5, 6 79
7.2 Wykresy gęstości oraz Q-Q dla przypadków 3, 4, 7, 8 80

7.3 Sieć Petriego opisująca niezawodność systemu tramwajowego . . . 82
7.4 Model w języku GN opisujący niezawodność systemu tramwajowego . . . 85
7.5 Porównanie prawdopodobieństwa hazardu i średniego czasu hazardu . . . 87
7.6 Histogramy czasów niezdatności . . . 89

8.1 Profil UML stanowiący składnię READ . . . 92
8.2 Diagram klas naprawialnego systemu komputerowego . . . 93
8.3 Diagram klas systemu tramwajowego . . . 94
8.4 Model w języku READ polityk napraw . . . 95
8.5 Model w języku READ niezawodności systemu tramwajowego . . . 96
8.6 Metamodel UML dotyczący READ . . . 102
8.7 Wzorzec translacji bramy COR [91] . . . 105
8.8 Wzorzec translacji bram CAND . . . 108
8.9 Wzorzec translacji bram GOR . . . 109
8.10 Wzorzec translacji bram GAND . . . 109
8.11 Wzorzec translacji bram opóźniającej . . . 110
8.12 Wzorzec translacji generatora obiektów . . . 110
8.13 Wzorzec translacji przejścia READ . . . 111

9.1 Błędy względne w szacowaniu $Pr(H)$ w modelu $W/W/M$, R_C=41 min . . . 125
9.2 Błędy względne w szacowaniu $Pr(H)$ w modelu W/W/M, R_C=101 min . . . 125
9.3 Błędy względne w szacowaniu $E(HT)$ w modelu W/W/M, R_C=41 min . . . 126
9.4 Błędy względne w szacowaniu $E(HT)$ w modelu W/W/M, R_C=101 min . . . 126
9.5 Porównanie jakości oszacowań hazardu modelu W/W/W . . . 129

10.1 Diagram klas systemu NLL . . . 133
10.2 Główny model READ niezawodności NLL . . . 135
10.3 Diagram uszkodzeń i napraw . . . 139
10.4 Diagram inspekcji . . . 140
10.5 Algorytm zastępowania lotów . . . 142

A.1 Działanie bramy CNOT . . . 163
A.2 Przebiegi czasowe zdarzeń bramy *COR* . . . 164
A.3 Przejście T3 z rys. 6.7 atomowo przydziela konserwatora do uszkodzonego dysku . . . 166

C.1 Sieć Petriego reprezentująca proces realizacji zamówienia . . . 170
C.2 Sieć Petriego specyfikująca działanie bramy CAND . . . 172

Streszczenie

W trzech częściach niniejszej rozprawy doktorskiej omówiono nową metodę analizy niezawodności systemów technicznych z zależnościami czasowymi będącą hybrydą drzew niezdatności oraz diagramów aktywności UML. W proponowanym narzędziu włączono bramy przyczynowe o naturze stochastycznej, akcje i bufory centralne, a także przejścia sieci Petriego, co ostatecznie zaowocowało językiem modelowania niezawodności o ekspresji wystarczającej do intuicyjnego i precyzyjnego opisu procesów konserwacji z zależnościami czasowymi. Tym samym wypełniono istniejącą lukę wśród narzędzi modelowania. Semantyka metody została rozstrzygnięta na gruncie Kolorowanych Sieci Petriego.

Równolegle z podstawami formalnymi proponowanej metody poszukiwano i dokumentowano obszary jej zastosowań. W pierwszej części omówiono transformację opisu koordynacji opóźnień wielokrotnej rezerwy gorącej w elektroenergetyce w rozszerzone drzewa niezdatności. W drugiej części zbudowano model niezawodności szeregowego systemu z rezerwą czasową, zimną rezerwą funkcjonalną oraz dostawą elementu po naprawie, który po transformacji w kod wykonywalny analizowano metodami symulacyjnymi. Istotnym elementem prezentowanych w rozprawie metod obliczeniowych są omówione w trzeciej części oszacowania analityczne pozwalające na uniknięcie badań symulacyjnych dla pewnych przedziałów parametrów zmiennych losowych systemów z rezerwą czasową i zimną rezerwą funkcjonalną. Rozprawę kończy demonstracja siły ekspresji nowej metody na problemie modelowania niskokosztowego procesu eksploatacji pozbawionego rezerwy i obejmującego naprawy oraz przeglądy.

Rozdział 1

Wstęp

Bez wątpienia rozległe systemy techniczne stały się motorem rozwoju cywilizacji. Olbrzymia zależność ludzkości od nieprzerwanych dostaw energii, zdolności współdzielenia informacji i szybkości przemieszczania oznacza, że zwiększenie ich niezawodności przynosi nie tylko wymierne korzyści finansowe, ale również w wielu przypadkach przyczynia się do wyższego bezpieczeństwa życia ludzkiego.

Choć projektowanie z myślą o wysokiej niezawodności nie jest nowością, zasadniczą trudnością pozostaje przewidywanie scenariuszy prowadzących do awarii. Warto zauważyć, że żadna z największych katastrof z udziałem czynników technologicznych nie wydarzyła się pod wpływem pojedynczego błędu [155]. Powodem awarii systemów projektowanych z myślą o wysokiej niezawodności jest nieprzewidziany ciąg wydarzeń, na który składają się zarówno błędy w budowie systemu, jak i jego obsłudze. W istocie, to na styku człowieka i technologii zachodzi największa liczba wypadków [142]. Potwierdza to między innymi analiza wypadku w reaktorze jądrowym w Three Mile Island, gdzie po uszkodzeniu zaworu cieczy chłodzącej personel niepoprawnie rozpoznał prawidłową, choć nieintuicyjną sygnalizację jej poziomu zakomunikowaną przez system informatyczny. Ten i wiele innych wypadków nakazuje postrzegać postęp technologiczny również jako zapewnienie wyższego poziomu niezawodności poprzez wsparcie pracy operatorów.

To stawia nowe wyzwania autorom metod analizy niezawodności. Każda modyfikacja istniejącego systemu technicznego pociąga za sobą albo jego komplikację, albo istotną przebudowę, co w obydwu przypadkach skłania do wzmożonych badań niezawodności. Nie ulega wątpliwości natomiast, że im szybszy postęp techniczny i bardziej skomplikowane systemy, tym efektywniejsze muszą być metody badawcze. Niniejsza dysertacja wpisuje się w szeroki nurt poszukiwań takich metod, czyli w dyscyplinę niezawodności.

Historyczna akceptacja niezawodności jako integralnej części nauki trwała około 20 lat. Problemem okazało się nie tyle uzasadnienie jej kosztów jako procesu wówczas jedynie pośrednio związanego z wytwarzaniem, co przyznanie na gruncie probabilistycznym, że produkt może ulec awarii. Chociaż jej podstawy teoretyczne oraz praktyczne istniały już na początku XX wieku, to dopiero olbrzymia awaryjność lamp elektronowych podczas II Wojny Światowej była katali-

zatorem badań trwałości komponentów elektronicznych [154].

W prekursorskiej pracy Johna von Neumanna [127] z 1956 roku pokazano metodę syntezy zwielokrotnionej bramy logicznej typu NAND o niezawodności wyższej niż podstawowe elementy, z których się składa. W odpowiedzi dwóch innych Wielkich Informatyków, Edward Moore oraz Claud Shannon na przykładzie przekaźników zaproponowali ulepszenie podejścia von Neumanna wymagające jednak niższego poziomu redundancji dla uzyskania podobnej niezawodności [126]. Tym samym stało się jasne, że niezawodność można specyfikować, mierzyć, przewidywać i adaptować, co pozwoliło ją uznać za dyscyplinę nauki. Kolejne badania w owym czasie koncentrowały się na analizie zbieranych od użytkowników informacji o defektach i unikaniu udokumentowanych błędów w nowo projektowanych komponentach [39].

Za sprawą rozwoju krytycznych dla życia ludzkiego systemów w dziedzinach lotnictwa i aeronautyki (programy Merkury, Gemini, Apollo) w latach 60. wzrosło zainteresowanie badaniami na poziomie całych systemów. Chęć estymacji czasu do uszkodzenia bez uprzedniego uruchomienia obiektu badań wymagała innego spojrzenia na awaryjność. Odpowiedzią na rosnące wymagania względem metod predykcji niezawodności okazały się między innymi udokumentowane po raz pierwszy w 1962 roku drzewa niezdatności (ang. *fault trees*). Katalizatorem rozwoju tej i innych metod, wynikającym po części z wypadku w Three Mile Island, stały się zastosowania w energetyce nuklearnej w latach 70 i 80 [76].

W ostatnim dwudziestoleciu ludzkość ustanowiła rozległe systemy techniczne, takie jak transport, energetyka, telekomunikacja, centralną osią cywilizacji. Gwałtowny rozrost geograficzny i funkcjonalny, a także towarzysząca ich komplikacja zaowocowały w nowe wyzwania dotyczące analizy niezawodności systemów przejawiających następujące cechy:

1. możliwość realizacji tego samego zdania różnymi środkami technicznymi, co pozwala na adaptację choreografii usług po zajściu niekorzystnego zdarzenia w systemie powodującą, że pomimo awarii z perspektywy użytkownika system pozostaje niezawodny [187],

2. wielostanowość komponentów składowych skutkująca tym, że cześć systemu może być technicznie sprawna, ale pracować z mniejszą wydajnością, przez co inna część może szybciej ulec awarii [188],

3. sieciowość i rozproszenie powodujące, że wyzwaniem staje się oszacowanie wpływu awarii jednego węzła o określonej wadze na niezawodność całego systemu [75].

Istotnym elementem wielu takich systemów są zależności czasowe pomiędzy zdarzeniami składające się na procesy odnowy i rekonfiguracji. W kolejnych rozdziałach dysertacji zaproponowano nową metodę badawczą wspierającą analizę niezawodności określonych klas systemów z zależnościami czasowymi na drodze modelowania, symulacji i estymacji.

Rozdział 2

Metody analizy niezawodności systemów technicznych

Niezawodność jest naturalnie obciążona nieprzewidywalnością, niepewnością oraz ograniczoną wiedzą, toteż należy oczekiwać od metod jej opisu adekwatnych możliwości ekspresji, a te zagwarantować mogą między innymi modele probabilistyczne. W niniejszej dysertacji wykorzystuje się definicję niezawodności zaczerpniętą z pracy [186]:

$$R(t) = Pr(\tau \geqslant t)$$

w której:

- $R(t)$ niezawodność,
- t założony (lub wymagany) czas pracy bez uszkodzenia.
- τ czas pracy bez uszkodzenia,

Zmienna losowa τ jest wyznaczana na podstawie parametrów modeli, czyli zazwyczaj innych zmiennych losowych o określonych rozkładach. To one powodują, że poszukiwania dokładnych rozwiązań niezawodności są znacznie bardziej skomplikowane niż dla innych problemów o naturze deterministycznej. Gdy zmienne losowe mają rozkłady wykładnicze, często udaje się wyprowadzić formuły zamknięte niezawodności. Zastosowanie innych rozkładów może zaowocować nazbyt skomplikowanymi, praktycznie nieobliczalnymi formułami, które w takiej sytuacji ustępują miejsca z natury przybliżonym estymacjom oraz symulacjom.

Należy jednak odróżnić niezawodność od bezpieczeństwa rozumianego według [128] jako własność oznaczająca, że system swoim działaniem nie stanowi zagrożenia dla życia ludzkiego, mienia czy środowiska. Na podstawie modelu probabilistycznego można drogą symulacyjną lub aproksymacyjną wykazać, że system jest wysoce niezawodny, na przykład poprzez udowodnienie, że średni czas do uszkodzenia jest bardzo duży. Natomiast badania czy awaria systemu prowadzi do katastrofy (czyli system nie jest bezpieczny) wymagają użycia modeli innych typów.

Choć najliczniejsze, modele probabilistyczne nie są jedynymi w grupie niedeterministycznych. Jeśli celem badań jest sprawdzenie bezpieczeństwa systemu, to warto zwrócić uwagę na modele ujmujące moment zaistnienia awarii przedziałem o końcach oznaczających czas minimalny i maksymalny [118]. Poprzez analizę takich przedziałów można wykluczyć zagrożenie, co jest niezbędne w systemach, w których czynnikiem krytycznym jest życie ludzkie (np. przejazd kolejowy). Gdzie jednak to niezawodność jest przedmiotem kontraktu, modele probabilistyczne umożliwią osiągnięcie porozumienia pomiędzy ceną budowy i utrzymania systemu a dostępnością świadczonej przez niego usługi.

W niniejszej rozprawie poszukiwane są metody analizy niezawodności systemów przejawiających opisane w kolejnym podrozdziale zależności czasowe.

2.1 Niezawodność systemów z zależnościami czasowymi

Choć czas jest naturalnym czynnikiem w każdym modelu niezawodności, to nie każdy język pozwala na opis zależności czasowych [53,102] pomiędzy zdarzeniami w modelowanym systemie. W kolejnych paragrafach scharakteryzowano pojęcie zależności czasowych.

W modelu niezawodnościowym uwzględniającym zależności czasowe istnieją dwa typy zdarzeń. Pierwsze to naturalne i antycypowane zdarzenia związane z funkcjonowaniem systemu. Druga grupa obejmuje zdarzenia związane z awariami i zawodnością komponentów.

Przykładowo: w systemie elektroenergetycznym dystrybucję mocy w linii można wstrzymać wyłącznikiem. Rozłączenie linii przez ten wyłącznik jest zdarzeniem należącym do pierwszej grupy. Dla kontrastu, zwarcie linii elektroenergetycznej, które nie jest antycypowane w modelu funkcjonalnym dystrybucji mocy, należy do drugiej grupy.

Pojęcie zależności czasowych odnosi się do obydwu grup i jest relacją przyczynowo-skutkową pomiędzy zdarzeniami. Cechami zależności czasowych są struktura i dynamika. Struktura definiuje które zdarzenia są przesłankami względem innych stanowiących ich konsekwencje. Dynamika określa chwile czasu, w których zależności są realizowane. Komplikacja współczesnych systemów technicznych powoduje, że ani struktura, ani dynamika nie mogą zostać uznane za deterministyczne.

W sytuacjach ekstremalnych, na przykład podczas intensywnego ataku wydajnościowego na sieć komputerową, pełna i jednoznaczna identyfikacja relacji przyczynowo-skutkowych pomiędzy zdarzeniami bywa niemożliwa [88]. Inną przyczyną niedeterministycznej natury struktury może być wielokrotna redundancja o typie gorącym, taka jak w systemie elektroenergetycznym opisanym w rozdziale 5.

Dynamika zależności czasowych wymaga opisu ilościowego, czyli modelu czasowego, który w ogólności również nie jest przewidywalny. Jeśli wyłącznik ma za zadanie rozłączenie linii po wykryciu zwarcia, to chwilę wejścia impedancji zwarcia w charakterystykę impedancji strefy zabezpieczenia uruchamiającego ten wyłącznik należy uznać za niedeterministyczną. Zatem opóźnienie pomiędzy chwilą zwarcia a rozłączeniem linii jest również niedeterministyczne.

Niezawodność jest z natury probabilistyczna, toteż w jej badaniach z uwzględnieniem zależności czasowych należy przyjąć, że podobnie struktura i dynamika zależności są stochastyczne.

Zdarzenia przyczynowe, ich konsekwencje, a także stowarzyszone zależności czasowe są podstawowymi elementami procesów zachodzących w systemie, takich jak eksploatacja czy naprawa. Uzupełnienie wzorców modelowania procesów ogólnych [152] o poprawny opis zależności czasowych jest niezbędne do precyzyjnej definicji procesów z zależnościami czasowymi wpływającymi na niezawodność systemu.

Chęć uwzględnienia zależności czasowych w modelach niezawodności istotnie ogranicza repertuar możliwych do wykorzystania języków modelowania.

2.2 Języki modelowania niezawodności

Matematyczne metody wyznaczania niezawodności wymagają opisu systemu w języku formalnym. Bazując na takim modelu można następnie zaprojektować system informatyczny automatyzujący proces analizy.

Pełny opis niezawodności systemu jest możliwy po zdefiniowaniu przestrzeni stanów, takiej że każdy stan zawiera informację o wszystkich komponentach składowych. Takie podejście widać w łańcuchach Markowa oraz sieciach Petriego. Owa przestrzeń stanów jest z jednej strony gwarantem bardzo wysokiej ekspresji metody, ale z drugiej powoduje, że problem wyznaczenia niezawodności sprowadza się do czasochłonnego jej przejrzenia. W praktyce, dla systemów składających się z setek elementów składowych, w których czasy do uszkodzenia są niewykładnicze, metody oparte o przestrzenie stanów są obecnie nierozwiązywalne obliczeniowo.

Przekrojowe spojrzenie na wyniki badań nad metodami wyznaczania niezawodności, potwierdza, że moc ekspresji zwykle ogranicza możliwości analityczne. Stąd istotą narzędzi w drugiej rodzinie metod niezawodnościowych nazywanych kombinatorycznymi stało się założenie o statystycznej niezależności uszkodzeń w modelowanym systemie. To pozwoliło na zaprojektowanie wielu nowych metod takich jak: tytułowe drzewa niezdatności, diagramy bloków niezawodności - *ang. reliability block diagrams* (RBD) czy grafy ataków. W wielu przypadkach dla metod kombinatorycznych udało się opracować efektywniejsze algorytmy wyznaczenia niezawodności niż dla modeli opartych o przestrzenie stanowe, choć te drugie mogą być użyte do opisu znacznie szerszej grupy problemów.

Przynależność języka do określonej rodziny metod nie implikuje jednak metody jego analizy. Badania nad metodami opartymi o przestrzenie stanów często kończą się znalezieniem określonej klasy problemów, dla których istnieje rozwiązanie analityczne ([16,104,105] i wiele innych). Natomiast analiza opisanych w kolejnym podrozdziale rozmaitych rozszerzeń drzew niezdatności wymagała podejścia symulacyjnego stanowiącego z kolei kompromis pomiędzy jakością rozwiązania a wysiłkiem obliczeniowym. Inną klasą metod analizy niezawodności znajdującą zastosowanie w obydwu rodzinach języków są analityczne schematy aproksymacyjne dostarczające przybliżonych rozwiązań, ale w czasie krótszym niż symulacja funkcjonowania całego systemu [119,122,183].

Dwa opisane czynniki decydujące o wyborze metody analizy niezawodności: moc ekspresji oraz możliwości analizy uzupełnia trzeci - intuicyjność języka dla inżynierów badanego systemu. Znaczenie tej cechy omówiono w kolejnym podrozdziale na tle drzew niezdatności, które są jednym z najbardziej intuicyjnych narzędzi rozpatrywania niezawodności.

2.3 Dotychczas opracowane warianty drzew niezdatności

Przegląd literatury w zakresie zastosowań narzędzi analizy niezawodności wykazuje, że drzewa niezdatności [176] to najpopularniejsza wśród ekspertów wielu dziedzin metoda analizy bezpieczeństwa lub niezawodności. Faktycznie, nietrudno o znalezienie raportów nietrywialnych zastosowań drzew niezdatności w dziedzinach takich jak:

- inżynieria procesowa [79],
- aeronautyka [140],
- transport [49],
- elektroenergetyka [29],
- energetyka nuklearna [157],
- budownictwo [40],
- inżynieria sanitarna [179],
- medycyna [189],
- hydrogeologia [74].

Cechą wspólną wymienionych prac jest akceptacja drzew niezdatności z powodu ich prostoty i czytelności dla ekspertów wymienionych dziedzin. Łączenie zdarzeń przy pomocy bram AND i OR oraz dedukcyjna rozbudowa drzewa od awarii systemu do zdarzeń elementarnych to cechy, których próżno szukać wśród modeli Markowa, sieci Petriego czy sieci Bayesowskich. Podobną do drzew niezdatności intuicyjność oferują jedynie modele kolejkowe oraz opisy typu *data-flow*, jednak tych nie da się w podstawowej postaci wykorzystać do badań niezawodności.

Wraz z rozszerzaniem się wachlarza zastosowań drzew niezdatności rosną wymagania autorów modeli dotyczące możliwości wyrażania coraz bardziej skomplikowanych struktur stanowiących często autorskie i badawcze koncepcje ulepszenia obszaru techniki, w których są ekspertami. Barierą okazuje się wielokrotnie raportowana w literaturze niewielka moc opisowa podstawowych drzew niezdatności [9, 19, 31]. Przykładowo duże braki w ich mocy ekspresji są widoczne przy próbach opisu struktur powtarzalnych czy modelowaniu zależności czasowych pomiędzy zdarzeniami. Opisane w kolejnych paragrafach rozszerzenia drzew niezdatności adresują wybrane problemy w ich możliwościach opisu.

Choć drzewa niezdatności mogą być dowolnie duże, to praktycznym ograniczeniem ich wielkości są problemy z percepcją dużego modelu oraz czas jego analizy, na przykład wyznaczania zbiorów przyczyn. Obydwa problemy adresują Parametryczne Drzewa Niezdatności - *ang. Parametric Fault Trees* (PFT) [18]. Niezależnie od liczby kopii danego modułu w systemie, w PFT wystarczy zamodelować ten moduł tylko jeden raz i dodać informację o redundancji. Dzięki efektywnym algorytmom poszukiwania zbiorów przecinających, PFT nadają się zatem do badań rozległych systemów o powtarzalnej strukturze. Autorzy zauważają, że generowane sieci Petriego mogą być miejscem, w którym dodaje się informacje o zależnościach stochastycznych nieobecnych jednak w modelach PFT.

Jednym z prominentnych rozszerzeń języka są Dynamiczne Drzewa Niezdatności - *ang. Dynamic Fault Trees* (DFT) [41] pozwalające na selektywne wskazanie zależności pomiędzy niezawodnością komponentów systemu. Przykładowo dzięki bramie zależności funkcyjnej można zdefiniować zjawisko propagacji awarii jednego komponentu na grupę innych. Z kolei przy pomocy trzech bram rezerw funkcjonalnych wskazuje się elementy mogące wzajemnie się zastępować powodując, że pomimo niektórych awarii system realizuje powierzone funkcje. DFT nie zawierają jednak modelu czasowego zależności pomiędzy niezawodnością komponentów.

Naprawialne Drzewa Niezdatności - *ang. Repairable Fault Trees* (RFT) pozwalają na definicję takiego modelu czasowego ale jedynie dla procesów odnowy. Dzięki translacji w sieci Petriego takie drzewa mogą być następnie analizowane symulatorem GreatSPN [146]. Proponowana brama skrzynki naprawczej nie ma zastosowania poza modelowaniem napraw systemu.

Zależności czasowe pomiędzy zdarzeniami w systemie są bezpośrednio adresowane przez Niedeterministyczne Drzewa Niezdatności z Zależnościami Czasowymi (NDNZC) [117], w których elementy mają przypisane czasy wyrażone przedziałami. Analiza czasowa takiego drzewa pozwala wykluczyć bądź potwierdzić zagrożenie. NDNZC są zatem dedykowane analizie bezpieczeństwa, a ich bezpośrednie zastosowanie do niezawodności jest niemożliwe.

Przed opracowaniem omawianego w rozprawie języka nie istniał wariant drzew niezdatności pozwalający na pełną specyfikację probabilistycznych zależności czasowych pomiędzy zdarzeniami w systemie. Próby zastosowania drzew niezdatności w tym obszarze prowadziły zazwyczaj do jednego z trzech rozwiązań:

1. ograniczenie roli drzew niezdatności do poglądowej, podczas gdy faktyczna analiza odbywa się innym narzędziem [179],

2. całkowite porzucenie drzew niezdatności i przeprowadzenie analizy bardziej ekspresywnymi, lecz mniej intuicyjnymi modelami opartymi o generację przestrzeni stanów, tj. sieciami Petriego [98], czy modelami Markowa. Brak wsparcia tych języków dla specyfikacji procesów eksploatacji daje w efekcie rozległe i nazbyt szczegółowe opisy.

3. Równoległe wykorzystanie drzew niezdatności i modeli Markowa modelujące system na różnych poziomach abstrakcji [25]. Brak rozszerzeń składniowych drzew niezdatności powoduje, że projektowanie i rozbudowa modeli wymaga biegłości w obydwu językach.

W kolejnym podrozdziale wyjaśniono dlaczego częste zastosowanie sieci Petriego do analizy drzew niezdatności nie jest przypadkowe.

2.4 Sieci Petriego w rozstrzyganiu znaczenia języków formalnych

Już w latach 80. zauważono, że drzewa niezdatności i sieci Petriego są w wielu aspektach wzajemnie komplementarne. Sieci Petriego mają wysoką moc ekspresji i niską intuicyjność, natomiast klasyczne drzewa niezdatności - niewielką moc ekspresji, ale są bardzo intuicyjne [26]. To pozwoliło na wyrażenie bram AND i OR przy pomocy klasycznych miejsc i przejść, a w konsekwencji na stworzenie nowych metod analizy drzew niezdatności, na przykład algorytmów detekcji i izolacji uszkodzeń określonego komponentu systemu [64]. Wraz z rozwojem drzew niezdatności znajdowały zastosowania kolejne warianty sieci Petriego, takie jak:

1. Czasowe sieci Petriego - *ang. Timed Petri Nets* (TPN) [117],
2. Uogólnione Stochastyczne Sieci Petriego - *ang. Generalized Stochastic Petri Nets* (GSPN) [17],
3. Stochastyczne Dobrze Zbudowane Sieci Petriego - *ang. Stochastic Well-Formed Colored Nets* (SWN) [18]

To dzięki powyższym językom rozstrzygnięto znaczenie wielu wariantów drzew niezdatności, między innymi opisanych w poprzednim podrozdziale PFT, RFT czy NDNZC.

W pracy [159] zebrano część z zalet sieci Petriego jako narzędzia definicji semantyki. Są nimi:

- solidne podstawy matematyczne wynikające z wieloletnich badań nad językiem,
- wysoka moc ekspresji, która jest warunkiem koniecznym języka definicji semantyki,
- możliwość walidacji i symulacji wsparta dojrzałym oprogramowaniem,
- składnia graficzna pozwalająca na wizualne projektowanie sieci.

Jednak w tej samej pracy postuluje się wprowadzenie warstwy pośredniej pomiędzy językiem definiowanym i sieciami Petriego określanej jako *Fundamental Modeling Concepts for Petri Net Diagrams* [86], gdyż same sieci Petriego często stają się trudne w odbiorze i analizie. Istotnie, jeśli wygenerowane modele są często modyfikowane lub rozbudowywane przez użytkowników, to wprowadzenie warstwy pośredniej upraszcza ich prace.

Diagramy Dynamicznych Bloków Niezawodności - *ang. Dynamic Reliability Block Diagrams* (DRBD) to jeden z języków opisu niezawodności, który zdefiniowano przy pomocy sieci Petriego [148]. Formalizacja pozwoliła na automatyczną weryfikację diagramów oraz identyfikację błędów w modelach DRBD. Warto jednak zauważyć, że DRBD mają siłę opisu niewiele większą od DFT i podobnie do tego języka koncentrują się na jakościowym zamiast ilościowym opisie dynamiki. Planowane przez Autorów rozszerzenie DRBD o aspekty czasowe będzie prawdopodobnie wymagało zwrócenia się do GSPN.

Warto zauważyć, że dla Sieci Petriego Wysokiego Poziomu - *ang. High Level Petri Nets* (HLPN) dostępne są środowiska definicji oraz wykonania reguł transformacji międzymodelowych [114]. Oznacza to, że po zdefiniowaniu przez użytkowników reguł transformacji do dostarczonego metamodelu sieci Petriego zapewniona zostaje integracja z wieloma dotychczas opracowanymi narzędziami do analizy i weryfikacji tego języka. W podobny sposób ułatwiona jest na przykład analiza wsteczna wyników działania tych narzędzi w języku faktycznie wykorzystywanym przez użytkownika. To powoduje, że stworzenie podobnych do opisanej w poprzednim paragrafie metody analizy niezawodności z wykorzystaniem DRBD wymagałoby obecnie mniejszego wysiłku programistycznego.

Sieci Petriego zostały z powodzeniem wykorzystane do definicji języka opisu dynamiki - diagramów sekwencji (DS), których formalizacja, w przeciwieństwie do np. diagramów aktywności, wydaje się obecnie dojrzała. Wynika to przede wszystkim z faktu, że druga wersja standardu UML przyniosła znaczącą, choć konsekwentną rozbudowę DS [123]. W efekcie bez wysiłku można znaleźć propozycje semantyki DS w sieciach Petriego [43], w tym transformacji rozszerzeń dalej opisanego profilu QoS [45]. Wśród metod analizy wynikowych sieci Petriego można znaleźć symulacje w CPN Tools [78], a także dalsze transformacje w język opisu implementacji sprzętowej [158]. W ostatniej pracy oraz w [46] podkreśla się również związki DS z diagramami przypadków użycia, choć te ostatnie ze względu na semiformalną naturę są rzadko rozstrzygane na gruncie sieci Petriego.

Definicję znaczenia diagramów sekwencji warto przeanalizować jeszcze raz mając na względzie typy oferowanych przez język współbieżności. Bloki $CombinedFragment$ umożliwiają wskazanie typu współbieżności w projektowanej interakcji. Zatem istotne znaczenie dla rozstrzygnięcia semantyki diagramów sekwencji mają typy współbieżności oferowane przez język definicji. Może to być faktycznie równoległe wykonanie wielu procesów, czyli tzw. „prawdziwa współbieżność" (ang. *true concurrency*) lub „semantyka przeplotowa" (ang. *interleaving semantics*) oznaczająca, że w danej chwili wykonuje się jeden proces, a współbieżność osiąga się na drodze cyklicznego przeplotu wielu z nich. Należy uznać, że pełna definicja diagramów sekwencji wymaga włączenia obydwu wariantów.

Sieci Petriego pozwalają na wyrażenie zarówno pełnej jak i przeplotowej współbieżność [143]. Tymczasem pomimo szeregu rozszerzeń, jak dotąd nie udało się w pełni wyrazić w algebrach procesowych współbieżności oferowanych przez diagramy sekwencji [23].

Diagramy maszyn stanowych jako najstarsze z dostępnych w UML doczekały się licznych prac dotyczących ich formalizacji i to sieci Petriego stały się częstym użytym narzędziem. Oznacza to, że dorobek w zakresie definicji maszyn stanowych stanowi inspirację do wykorzystania określonych rozszerzeń sieci w celu zaadresowania wymagań definiowanego języka.

W literaturze można odnaleźć zastosowania klasycznych i kolorowanych sieci Petriego do definicji podstawowych elementów składniowych diagramów stanów [180] oraz TPN do rozstrzygnięcia modelu konsumpcji energii [28], a także transformacje dedykowane analizie wydajności przy pomocy GSPN [121]. Z kolei z opisu środowiska projektowania oraz implementacji systemów czasu rzeczywistego [139] wynika, że sieci Petriego w wariancie z sygnałami i zdarzeniami

- ang. *Input-Output Place Transition Nets* (IOPT) bardzo dobrze integrują się z narzędziami generacji kodu i mogą stanowić szkielet produktów o architekturze sterowanej modelem (MDA).

Modele złożonych systemów są rozległe i nierzadko zawierają powtarzalne fragmenty struktury i zachowania. W pracy [24] udokumentowano na przykładzie diagramów bloków wewnętrznych SysML w jaki sposób Hierarchiczne Kolorowane Sieci Petriego umożliwiają wyrażanie: uproszczonych modeli najwyższego poziomu, jak i szczegółowych modeli zagnieżdżonych.

2.5 Prace wokół stworzenia uniwersalnego języka modelowania

Ogół prac wokół Zunifikowanego Języka Modelowania (UML) jest częścią największej jak dotąd inicjatywy zapoczątkowanej przez firmę Rational Software w latach 90 a kontynuowanej przez Object Management Group (OMG) w kierunku stworzenia standardu modelowania oprogramowania. Bezpośrednim bodźcem do opracowania UML'a była popularyzacja paradygmatu obiektowego w analizie i projektowaniu systemów informatycznych i tym samym uzasadnione było wypracowanie respektowanego w przemyśle standardu specyfikacji bazującego na dotychczasowych osiągnięciach w tym obszarze. I choć początkowo głównymi kontrybutorami UML'a zostały narzędzia takie jak: *Object Oriented Software Engineering (OOSE)* [69], metoda Booch'a [21] czy *Object Modeling Technique* [113] będące w latach 90. prekursorami modelowania obiektowego, to standard nie pozostawał nimi ograniczony.

W istocie język UML nigdy nie oderwał się od formalizmów respektowanych w środowisku naukowym. Przykładowo: na aktualną postać diagramów klas olbrzymi wpływ miały diagramy związków encji Petera Chena [30], a diagram maszyny stanowej UML jest w istocie wariantem języka przedstawionego w znanej pracy Davida Harrela [57]. W końcu, z perspektywy diagramów aktywności i w związku z tym z punktu widzenia niniejszej rozprawy, niebywałe znaczenie ma praca doktorska Carla Petriego [141], gdzie Autor zdefiniował pierwszą wersję sieci, które przez lata ewoluowały w istotne narzędzie współczesnej informatyki.

Pomimo że UML jest aktualnie standardem modelowania oprogramowania, istnieją dwie główne bariery powstrzymujące dalszą jego akceptację w inżynierii oprogramowania. Pierwszą wersję języka często uznawano za skomplikowaną i niepełną [52] i to właśnie te problemy adresowane były w wersjach UML 2.x. Z drugiej strony, w mniejszych zespołach wytwarzających oprogramowanie publikowane na rynku o niskiej konkurencji problemy rozwiązywane przez UML są mniej istotne i w związku z tym język jest wolniej wdrażany [54].

Warto zauważyć, że spośród trzech fundamentalnych dla definicji UML'a języków: maszyny stanowej, sieci Petriego oraz związków encji jedynie te ostatnie są dedykowane systemom zorientowanym na dane. Nic więc dziwnego, że skoro pierwsze dwa można z powodzeniem zastosować do opisu algorytmów czy procesów, to cały język UML starano się niemal od początku zaadaptować do systemów czasu rzeczywistego.

Standard w wersji drugiej wzbogacony o komercyjne rozszerzenia pozwolił już na migrację procesu rozwoju systemów bezprzewodowych ze znacznie starszego rozwiązania SDL [44] w firmie Motorola [72]. Łatwa adopcja UML'a wynikała prawdopodobnie z podobieństwa Diagramów

Sekwencji Komunikatów - *ang. Message Sequence Chart* (MSC) [175] do nowych diagramów sekwencji.

Istotnym postępem w poszukiwaniu ogólnego języka modelowania systemów zorientowanych na sterowanie było opracowanie w latach 2004 - 2006 języka Języka Modelowania Systemów - *ang. Systems Modeling Language* (SysML) [58], który następnie udoskonalano w ramach prac OMG [129]. Akceptacja nowych standardów w obszarze systemów krytycznych jest jednak znacznie trudniejsza niż w oprogramowaniu ogólnego przeznaczenia [63] i wciąż jest za wcześnie na jednoznaczną ocenę przyjęcia SysML'a. Na jego korzyść przemawia pokrewieństwo z UML'em i możliwość adaptacji istniejących narzędzi informatycznych, których brak początkowo ograniczał popularność UML'a wśród inżynierów [1].

2.5.1 Rozszerzenia UML'a dla opisu niezawodności

W próbach wykorzystania UML'a do badań niezawodności można wyróżnić kilka podejść. W najstarszym wykorzystuje się stereotypy bądź dokonuje zmian semantyki UML'a [36], dzięki którym modele można łatwo transformować do innego formalizmu dedykowanego analizie niezawodności - na przykład klasycznych drzew niezdatności [62, 107] bądź DFT [136]. Takie rozwiązania są zatem ograniczone możliwościami wyrazu docelowego języka i nie wykorzystują potencjału UML'a w wyrażaniu struktury i zachowania systemów. Ponadto każdorazowo wymagają daleko idących rozszerzeń lub modyfikacji UML'a, co kwestionuje jego zastosowanie.

W nowszym podejściu modelowanie i programowanie obiektowe wykorzystuje się do wsparcia analizy i wytwarzania systemów wymagających wysokiego poziomu bezpieczeństwa, na przykład systemów sterowania koleją [109, 110]. W dwuczęściowym artykule zauważono między innymi, że dzięki spojrzeniu na obiekt jako na „czarną skrzynkę" ułatwione zostaje programowanie wielowersyjne, a także głosowanie i weryfikacja w zbiorze równolegle działających komponentów. Ponadto dzięki enkapsulacji i polimorfizmowi obiekty łatwiej zabezpieczyć przed przypadkową zmianą stanu, co razem z innymi zaletami paradygmatu obiektowego pozwoliło na zaproponowanie metody wytwarzania oprogramowania zmniejszającej ryzyko awarii systemu kolejowego.

Dalsze zwiększenie precyzji w modelach UML można uzyskać dzięki adnotacjom Językiem Ograniczeń Obiektów - *ang. Object Constraint Language* (OCL). Istotnym wyzwaniem dla standaryzacji UML w tym zakresie pozostaje na przykład bliższa integracja definicji klas z ich kontraktami [137].

Część z problemów napotykanych przy adaptacji UML do niezawodności z powodzeniem rozwiązuje SysML, choć kwestia mianowania SysML językiem modelowania niezawodności pozostaje nierozstrzygnięta. W pracy [37] wykorzystano wprawdzie wprowadzone przez SysML diagramy wymagań oraz definiowania bloków do przeprowadzenia Procedury Analizy Rodzajów i Skutków Uszkodzeń - *ang. Failure Mode and Effect Analysis* (FMEA), ale do opisu zachowania prezentowanego systemu wystarczyły diagramy sekwencji. Te natomiast pozostały niezmienione w stosunku do wersji z UML, a zatem nie zawierają rozszerzeń w kierunku niezawodności. Rozpatrywanie podobną metodą problemów przedstawionych w niniejszej rozprawie wymaga-

łaby rozszerzeń w opisie zachowania. Podobną myśl wyrażono w pracy [6], gdzie stwierdzono, że możliwości opisu zachowania oferowane przez SysML są niewystarczające do modelowania niektórych aspektów dynamiki.

Niezgodność w ocenie użyteczności SysML'a w niezawodności może wynikać z następującego spostrzeżenia. W kategoriach opisu zachowania SysML dostarcza nowy diagram parametryczny oraz rozszerzenia diagramów aktywności. Pierwszy z nich dedykowany jest fizycznemu oraz elektromagnetycznemu opisowi zjawisk towarzyszących systemowi, podczas gdy drugi koncentruje się na akcjach strumieniowych i buforowaniu. Jakkolwiek w części przypadków jest to wystarczające do specyfikacji niezawodności systemu (np. [111]), inne mogą wymagać nieobecnych w SysML rozszerzeń czasowych czy wsparcia opisu procesu konserwacji [7].

Głównymi diagramami opisu zachowania w UML są:

- diagramy sekwencji - ukierunkowane na modelowanie interakcji pomiędzy obiektami przy pomocy komunikatów,
- diagramy maszyn stanowych - opisujące przejścia pomiędzy stanami obiektu,
- diagramy aktywności (DA) - wykorzystywane do opisu koordynacji procesów (aktywności) składających się z kroków (akcji), z których każdy ma zdefiniowane wejście i wyjście.

Diagramy maszyn stanowych okazują się niezastąpione przy modelowaniu niezawodności pojedynczych komponentów lub niewielkich ich grup np. puli rezerwowej przedstawionej w [112]. Aby jednak umożliwić ogólne spojrzenie na interakcje w systemie, diagramy stanów są często uzupełniane innymi metodami opisu na przykład drzewami niezdatności. W artykule [83] zaprezentowano metodę analizy bezpieczeństwa oprogramowania łączącą diagramy stanów z klasycznymi drzewami niezdatności, a NDNZC wykorzystano w [118]. W ostatniej pracy zaprezentowano metodę opartą o diagramy stanów do wyznaczania czasów trwania zdarzeń i odpalania bram w drzewach niezdatności z czasem wyrażonym niedeterministycznie w postaci przedziałów. W ogólności jednak diagramy maszyn stanowych posiadają braki w obszarze komunikacji pomiędzy obiektami i modelowanie procesów eksploatacji opartych na wymianie danych pomiędzy elementami systemu jest utrudnione.

Z kolei diagramy sekwencji mają wyraźną przewagę nad innymi metodami modelowania w przypadku systemów sterowania zbudowanych wokół przerwań, sensorów i efektorów, takich jak system zapobiegający przelewowi cieczy przedstawiony w omawianej już pracy [37].

Problemy poruszone w niniejszej rozprawie wymagają modelu pozwalającego na opis procesów z zależnościami czasowymi, stąd rozszerzanym językiem będą diagramy aktywności. W pracach [152, 153] zidentyfikowano w sumie ponad 40 wzorców sterowania wykorzystywanych przy projektowaniu procesów, a następnie zweryfikowano ich dostępność w 14 produktach komercyjnych. Według ostatecznej oceny to: DA, BPMN oraz XPDL stanowiły trzy najbardziej rozbudowane w 2006 roku języki modelowania procesów.

Opis cech niefunkcjonalnych komponentów systemu, w tym niezawodności, adresuje profil *Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS), ale

jego głównym celem jest umożliwienie zastosowania istniejących metod analizy niezawodności i ryzyka, takich jak HazOp, Analiza Drzew Niezdatności - *ang. Fault Tree Analysis* (FTA), czy FMEA, w modelach UML ([173], str. 47). Innymi przykładami mogą być wsparcie dla Analizy Mocnych i Słabych Stron, Szans oraz Zagrożeń (ang. SWOT), modelowania incydentów, czy opisu typów rezerwy w systemach z redundancją. Sam profil nie zawiera jednak rozszerzeń wymienionych metod i choć niektóre z braków w zakresie modelowania dynamiki uzupełnia praca [149], to zastosowanie tych narzędzi do znajdujących się w centralnym obszarze zainteresowań rozprawy zjawisk czasowych pozostaje wykluczone.

Ponieważ w niniejszej rozprawie niezawodność rozpatrywana jest na tle zależności czasowych, to warto przy przeglądzie literatury odwołać się również do uznanych technik modelowania wydajności. Profil: Analiza i Modelowanie Systemów Czasu Rzeczywistego oraz Systemów Wbudowanych - *ang. Modeling and Analysis of Real Time and Embedded Systems* (Marte) [172] wywodzi się bezpośrednio ze starszego narzędzia *Schedulability, Performance and Time* (SPT) [174] i pomimo wielu rozszerzeń w stosunku do poprzedniej wersji pozostaje ograniczony do modelowania wydajności i nie daje się bezpośrednio zastosować do analizy niezawodności. Spośród nieabstrakcyjnych modułów profilu: opisu alokacji, platformy wykonania, wymagań niefunkcjonalnych oraz czasu dwa ostatnie oferują szeroki repertuar środków wyrazu czasu fizycznego, logicznego czy zmiennych losowych przy pomocy dedykowanej składni konkretnej [2]. Część z tych możliwości wykorzystano w rozdziale 8 przy definicji nowego języka modelowania niezawodności z zależnościami czasowymi.

Przegląd literatury wskazuje, że połączenie Marte i drzew niezdatności jest pożądane przy rozwiązywaniu rzeczywistych problemów. Na przykładzie analizy bezpieczeństwa systemu kolejowego w referacie [14] omówiono translację modelu Marte w RFT, co jednak nie daje takiej siły wyrazu i elastyczności, co jawne połączenie syntaktyczne i składniowe UML i drzew niezdatności proponowane w dysertacji.

Istotna część prac nad profilami UML i powiązanymi językami to zapewnienie spójności i kompatybilności, gdyż w praktyce inżynierskiej rzadko kiedy ogranicza się do stosowania jednego języka. Po zbadaniu i rozwiązaniu problemów integracji narzędzi SysML i Marte [48] udało się stworzyć metodologię Mades [145] zawierającą język, bazujące na nim oprogramowanie oraz procesy wspierające rozwój szerokiej gamy systemów czasu rzeczywistego. Podobnie jak SysML i Marte posiadają aktualnie ujednolicony model alokacji oprogramowania i sprzętu, tak Marte i proponowane w rozprawie Rozszerzone Niezawodnościowo Diagramy Aktywności - *ang. Reliability Enhanced Activity Diagrams* (READ) będą posiadały zgodny model opisu czasu.

2.5.2 Dotychczasowe osiągnięcia w definicji semantyki diagramów aktywności

W aktualnej postaci specyfikacji UML [131] ani składnia, ani semantyka DA nie są wolne od błędów, choć rozstrzyganie semantyki postępuje zdecydowanie wolniej. Powodem jest prawdopodobnie fakt, że najczęściej wykorzystywaną metodą definicji znaczenia elementów jest opis w języku naturalnym, który z jednej strony jest dostępny najszerszemu gronu odbiorców, ale

z drugiej strony często prowadzi do dwuznaczności w analizie. Nie ulega wątpliwości, że jednoznaczna definicja semantyki języka jest niezbędna do budowy prawidłowych modeli oraz do rozwijania narzędzi informatycznych przetwarzających te modele. Stąd wiele prac od kilku lat adresuje ten problem.

Semantyka DA w UML 1.0 była związana z diagramami stanów, podczas gdy druga wersja nawiązuje do sieci Petriego ([131], s. 333). Zatem jeśli do pewnej akcji $A3$ skierowane są dwa łuki od akcji poprzedzających $A1$ i $A2$, to w UML 1.0 wystarczyło aby $A1$ *lub* $A2$ zostało wykonane, żeby rozpoczęło się $A3$. W UML 2.x przed startem $A3$ musi wykonać się *zarówno* $A1$ jak i $A2$. Owa zmiana semantyki powoduje, że wiele prac sprzed 2004 roku, takich jak [55, 150], trudno bez poważnych zmian zastosować do wersji drugiej standardu. Ponadto w związku z szeregiem rozszerzeń drugiej wersji DA, choćby z wykorzystywanym w rozprawie nowym elementem $CentralBufferNode$ ([131], s. 362), definicje semantyki w cytowanych pracach nie mogą być aktualnie uznawane za kompletne. Z drugiej strony, wiele problemów zidentyfikowanych przy formalizacji DA w rozległym raporcie [47] można uznać za aktualne, choć ich rozwiązanie w nowej wersji standardu może wymagać innych środków.

Rzadkością w literaturze jest kompleksowe podjęcie problemu formalizacji DA. Przykładowo: wyniki przedstawione w [70] pozwalają na wygenerowanie łańcuchów Markowa stanowiących interpretację DA w wersji rozszerzonej przez SysML o prawdopodobieństwa wykonania oraz stochastyczny model czasowy i to te elementy stanowią trzon pracy. Autorzy nie komentują zakresu składniowego wspieranego przez prezentowaną transformację i nie podejmują dyskusji nad możliwością wyrazu przez łańcuchy Markowa bardziej skomplikowanych elementów niż przedstawione w przykładzie. Tym niemniej, przy wykorzystaniu narzędzia Prism [103] osiągnięto cele pracy polegające na walidacji oraz analizie wydajności DA składających się z bloków sterowania.

W pracy [13] zdefiniowano diagramy aktywności w czasowej wersji języka Komunikujących się Procesów Sekwencyjnych - *ang. Communicating Sequential Processes* (CSP) [61, 135]. Choć omówiona semantyka dotyczy drugiej wersji standardu, to nie pokrywa udoskonalonego w tej wersji przepływu obiektów, który jest istotną częścią proponowanego w rozprawie rozwiązania. Ponadto w pracy założono, że akcje posiadają deterministycznie określoną chwilę początku oraz końca, co w niezawodności jest niewystarczające. Analogicznie do CSP, częściowe definicje semantyki DA można znaleźć w [184], gdzie wykorzystana została Algebra Procesowa lub w [169], gdzie użyto Rachunku Sytuacyjnego.

Sieci Petriego jako narzędzie definicji semantyki wydają się również bardziej przydatne w niniejszej rozprawie niż narzędzia algebraiczne, nie tylko dlatego, że są językiem graficznym, ale również z powodu semantycznej bliskości diagramom aktywności oraz proponowanym rozszerzeniom. Warto zauważyć, że również bramy drzew niezdatności są najczęściej definiowane przy wykorzystaniu sieci Petriego [117, 146].

Z powyższych powodów wydaje się, że definicja semantyki DA opracowana w [56] będzie bardziej odpowiadała potrzebom formalizacji języka niezawodności niż poprzednio omówione. W pracy wykorzystano sieci Petriego, a transformacja wspiera również bufory centralne i przechowywane w nich obiekty. Ze względu na zastosowanie klasycznych sieci Petriego transformacja

wymagałaby jednak rozszerzeń o model czasowy. Co równie istotne, w przedstawionej wersji klasy nie posiadają ani atrybutów, ani asocjacji, tak więc bez dalszych rozszerzeń niemożliwe jest wykorzystanie warunków i akcji.

W artykułach i referatach Haralda Störrle oraz Jana Hausmanna [161–166] widać kompleksowe podejście do formalizacji DA z odrębnym uwzględnieniem: przepływu sterowania, przepływu danych, wyjątków czy węzłów strukturalnych. Przy formalizacji przepływu obiektów zastosowano Kolorowane Sieci Petriego umożliwiając tym samym translację atrybutów klas o typach prostych. Nie odniesiono się natomiast do modelowania asocjacji, a tym samym atrybutów o typach złożonych, co dokonano jednak w niniejszej rozprawie. Niektóre z prac obecnie rozszerzających DA (np. [60] lub [178]) stosują zaproponowane w cyklu prac translacje i zbliżoną metodykę wykorzystano w rozdziale 8 przy włączaniu aspektów niezawodnościowych.

Warto jednak podkreślić, że w ostatniej z cyklu prac [166] stwierdzono, że formalizacja niektórych nowych konstrukcji DA jest kłopotliwa w sieciach Petriego wymagając coraz bardziej wyszukanych rozszerzeń języka semantyki. Przykładem ilustrującym opisane przez autorów zjawisko może być próba formalizacji rozszerzonych w SysML DA dokonana w pracy [3]. Model konsumpcji energii dodany do DA wymusił zastosowanie wyspecjalizowanego wariantu czasowych sieci Petriego - Timed Petri Net with Energy Constraints [170], który jest kolejnym po sieciach klasycznych, czasowych, stochastycznych oraz kolorowanych. Jednak w tej samej pracy [166] Harald Störrle i Jan Hausmann odrzucają zastosowanie innych narzędzi niż sieci Petriego sugerując, że pełna definicja DA wymaga raczej przemyślenia niektórych obszarów ich specyfikacji. Stąd w rozprawie nie umieszcza się pełnej definicji DA, a jedynie dodaje nowe rozszerzenia uznaną między innymi w [60] metodyką.

Faktyczną implementację w systemie AToM^3 [106] transformacji z DA do sieci Petriego można odnaleźć w pracy [168]. Oprogramowanie jest jednak ukierunkowane na zarządzanie projektami i integrację z pakietem Microsoft Project. Tym samym nie odnosi się do inżynierii systemowej, a ponadto wykorzystuje klasyczne sieci Petriego utrudniające modelowanie klas UML.

2.6 Oprogramowanie wspierające analizę niezawodności

Faktyczne zastosowanie wielu z opisanych wcześniej metody ograniczone jest dostępnością stabilnych i wzajemnie kompatybilnych narzędzi informatycznych. Wzorowym przykładem rozwoju nie tylko metody analizy języka, ale również jej implementacji jest narzędzie Galileo [167] pozwalające na uzyskanie metryk niezawodnościowych z DFT niemal od chwili opracowania samego języka. Z drugiej strony, sama obecność narzędzi informatycznych nie jest gwarantem ich zastosowania w praktyce. Wyniki ankiety omówionej w pracy [147] wskazują, że istniało wiele rozwiązań modelowania odrzuconych przez przemysł z powodu niewystarczającej funkcjonalności, pomimo opracowania ich dobrych podstaw formalnych.

Jednym z najpopularniejszych profesjonalnych narzędzi do FTA jest Windchill (wcześniej znany jako Relex). Wspierane są wszystkie bramy DFT, choć wciąż brakuje obsługi bardziej zaawansowanych konstrukcji znanych z nowszych wersji języka (PFT, RFT). Zaletą narzędzia

jest obszerna dokumentacja omawiająca technikę FTA z inżynierskiego punktu widzenia.

W przeciwieństwie do Windchill, OpenFTA [134] jest narzędziem o otwartym kodzie źródłowym. Główne dostarczone przez autorów metody analizy klasycznych drzew niezdatności to: algebraiczna wyznaczania zbiorów przecinających oraz symulacyjna. Istotną cechą oprogramowania jest możliwość wielokrotnego wykorzystywania tych samych zdarzeń podstawowych, a więc i modelowania uszkodzeń o wspólnej przyczynie (ang. *common cause failure*).

Osobną grupą narzędzi istotnych z punktu widzenia rozprawy jest oprogramowanie do projektowania i analizy sieci Petriego. Pakiet CPN Tools [71] jest niewątpliwie najbardziej znanym środowiskiem dedykowanym HLPN, choć ze względu na nieadekwatność modelu czasowego do poruszanych w rozprawie problemów, skorzystano również z narzędzia Snoopy [151]. Porównanie tych dwóch pakietów oprogramowania zawarto w dodatku C.

Ze względu na dojrzałość oraz liczne praktyczne zastosowania warto pamiętać o pakiecie Uppaal [12] bazującym na automatach czasowych. Oprócz środowiska modelowania dostępne są weryfikator czasowych formuł logicznych oraz symulator, którego wykonanie obrazowane jest automatycznie generowanym diagramem MSC.

Choć dostępnych jest już wiele środowisk modelowania UML, to metody analizy diagramów dynamiki są wciąż intensywnie rozwijane. Dwa najpopularniejsze obecnie symulatory diagramów aktywności to Arena [5] oraz moduł dostarczany razem z pakietem IBM Rational Software Architect [65]. Pierwszy z nich, starszy, jest szeroko wykorzystywany na potrzeby wojska czy ochrony zdrowia, podczas gdy drugi jest bardzo dobrze zintegrowany z pozostałymi elementami języka UML. Niestety obydwa narzędzia mają zamknięty kod źródłowy, więc ich wykorzystanie jako bazy programistycznej symulatora READ jest wykluczone.

Najbliższym znalezionym w literaturze narzędziem spokrewnionym z prezentowanym w rozprawie jest Turtle [4] dedykowane modelowaniu i symulacji systemów czasu rzeczywistego. Rozwiązanie również bazuje na diagramach klas do opisu struktury systemu oraz diagramach aktywności do specyfikacji zachowania i choć oryginalnie wykorzystywaną wersją UML jest 1.5, to dla opisu dynamiki zaproponowano zastosowanie diagramów stanów UML 2.0. Tym samym stosowana wewnętrznie translacja z diagramów aktywności 1.5 do języka RT-Lotos [34] mogłaby zostać częściowo wykorzystana, gdyż DA UML 1.5 i diagramy stanów UML 2.0 mają pokrewne semantyki. Metoda Turtle pozwala na wyrażenie wielu typów zależności czasowych w modelach, zarówno deterministycznych jak i stochastycznych, lecz sama w sobie nie jest dedykowana analizie niezawodności. Dzięki włączeniu bram drzew niezdatności READ ma szansę rozszerzyć zakres zastosowań UML w tej dziedzinie.

Oprogramowanie Draw-Net [50] można zaliczyć zarówno do pakietów analizy systemowej, jak i do grupy narzędzi tworzenia nowych języków dziedzinowych opisanych w kolejnej sekcji 2.7. Użytkownicy Draw-Net mają możliwość budowania własnych narzędzi modelowania pod warunkiem, że te przypominają strukturą graf. Do analizy niezawodności oraz wydajności systemów zdefiniowanych w nowych językach można wykorzystać zaimplementowane przez autorów środowiska algorytmy dla: sieci Petriego, DFT, PFT oraz sieci Bayesa. Długofalowym celem projektu jest integracja innych, znanych z literatury naukowej metod analizy systemów w ramach jednej

platformy skierowanej do inżynierów.

2.7 Narzędzia metamodelowania i transformacji międzymodelowych

Podążając za główną myślą pracy [138] język modelowania może albo zostać zaprojektowany od nowa, albo zbudowany jako unifikacja istniejących narzędzi. Do pierwszej grupy należą języki dziedzinowe - *ang. domain specific languages* (DSL), podczas gdy przykładem należącym do drugiej grupy jest UML. W rozprawie zastosowano kolejno obydwa podejścia: najpierw od nowa zaprojektowano metamodel języka posiadającego cechy drzew niezdatności i sieci Petriego, aby w następnym kroku dokonać unifikacji z DA. W ten sposób zapewniono elastyczność w pierwszych krokach, aby w kolejnych skoncentrować się na problemie formalnej integracji dwóch aspektów: modelowania niezawodności oraz opisu procesów.

Przyjęta strategia rozwoju języka niezawodności nie wyklucza jednak równoległego stosowania języków DSL. W istocie, ich zastosowanie ułatwia generację, modyfikację oraz utrzymanie modeli [27]. Wysokie zainteresowanie takimi językami w ostatnim dziesięcioleciu zaowocowało w szereg darmowych i komercyjnych platform wspierających ich projektowanie i przetwarzanie. Choć zastosowań komercyjnych środowisk, takich jak Microsoft DSL Tools [32] oraz MetaEdit+ [77], nie brakuje, to zdaniem autora najszerszą funkcjonalność oferują darmowe rozszerzenia platformy Eclipse [51] w niniejszej rozprawie dodatkowo wsparte narzędziem openArchitectureWare [42] do projektowania gramatyk Antlr [171].

Również od około 10 lat kolejne wersje języka transformacji międzymodelowych Query/View/Transform (QVT) konsekwentnie wzmacniają pozycję swojego standardu wśród metod translacji [35] i obecnie wydaje się, że QVT jest najpopularniejszym językiem tego typu. Systematycznie aktualizowana specyfikacja [124] jest wsparta oprogramowaniem rozwijanym przez firmę Borland [22] dostępnym również w wersji otwartej dla społeczności MDA. Wysoką jakość QVT potwierdziły doświadczenia na praktycznym gruncie wyniesione z pracy nad algorytmem weryfikacji topologii sieci elektroenergetycznej [87], gdzie oprócz QVT stosowano również ATL [73] oraz Xtend [42].

W praktyce, aby doprecyzować metamodel oraz transformacje nowego języka ich definicje muszą być wsparte językiem ograniczeń [181]. Choć ustandaryzowany OCL [132] ma swoje słabości [177], to jest wciąż najlepiej wspierany przez środowiska projektowania DSL i dlatego został wykorzystany w rozprawie. Wprowadzenie do OCL zawarto w dodatku B. Imperatywną alternatywą dla deklaratywnego OCL jest język Alloy [68] adresujący niektóre słabości OCL.

Rozdział 3

Cele i organizacja pracy

3.1 Koncepcje poszukiwań języków opisu niezawodności

Każda technika modelowania powinna posiadać solidne matematyczne i informatyczne podstawy pozwalające na precyzyjne opisywanie systemów i jednoznaczną analizę powstałych modeli. Z drugiej jednak strony, ostatecznym celem rozwoju języka formalnego jest jego zastosowanie przy rozwiązywaniu istotnej grupy problemów. Jakkolwiek ostateczne osiągnięcie obydwu celów nie jest wykluczone, w praktyce w procesie rozwoju nowej metody dominuje zwykle jeden nich, a drugi włączany jest stopniowo.

Kiedy w poszukiwaniach przyjmuje się **koncepcję sterowaną językiem**, nacisk kładziony jest na formalną stronę techniki modelowania. Bez wymagań dotyczących konkretnych aplikacji składnia oraz semantyka języka nie mają ograniczeń dziedzinowych. W ten sposób mogą być adaptowane lub wręcz projektowane, aby sprzyjać istnieniu efektywnych algorytmów analizy modeli. Końcowy kształt metody warunkuje obszar poszukiwań zastosowań i go ogranicza.

W **koncepcji sterowanej problemami** sytuacja jest odwrotna. Składnia i semantyka muszą wspierać analizę jednego lub wielu rzeczywistych i nietrywialnych problemów. Stąd język musi być zaprojektowany mając na uwadze określony obszar zastosowania i nie powinien go ograniczać. Ponadto ponieważ jednoczesne rozważanie wielu skomplikowanych modeli nawet z tej samej dziedziny jest kłopotliwe, rozwój języka zazwyczaj odbywa się w iteracyjny sposób. Na początku każdej iteracji wybierany jest nowy zbiór problemów i do nich stosuje się rozwijaną metodę, co prowadzi do jej ulepszenia. Sposób wyboru problemów warunkuje wynikową składnię oraz semantykę i jest jednocześnie krytyczny dla obszaru zastosowania języka. Nieprawidłowy dobór kolejnych modeli obejmujący zbyt szeroką lub zbyt wąską przestrzeń rzeczywistości może prowadzić do podziału języka lub jego dryftu w kierunku narzędzia dziedzinowego.

To właśnie praktyczne znaczenie metody jest główną przyczyną, dla której badania w niniejszej rozprawie prowadzone są w myśl **koncepcji sterowanej problemami**. Pozwala to nie tylko na faktyczne zastosowanie metody do badań niezawodności rzeczywistych systemów już na wczesnych etapach jej rozwoju, ale także pielęgnację jej intuicyjności poprzez współpracę z

ekspertami dziedzinowymi. Obydwa cele adresowane równolegle dają dużą szansę na znalezienie intuicyjnej i ekspresywnej metody analizy niezawodności systemów czasu rzeczywistego i tym samym wypełnienie istniejącej luki wśród dostępnych narzędzi.

W myśl powyższego, zawartość każdej części niniejszej dysertacji obejmować będzie zarówno język modelowania (na pewnym etapie rozwoju), jak i zbiór modeli rozwiązujących określone problemy techniczne. Ostatni element obejmuje także algorytmy transformacji międzymodelowych oraz implementujące je narzędzia dostarczające miar niezawodności.

3.2 Teza i cele dysertacji

Pomimo opisanego w rozdziale 2 szerokiego wachlarza zastosowań drzew niezdatności, głębsza analiza ich mocy opisowej wykazuje wyraźne słabości. Główne napotykane historycznie problemy to: uchwycenie zależności i sekwencyjności awarii, czego rozwiązaniem okazały się Dynamiczne Drzewa Niezdatności - *ang. Dynamic Fault Trees* (DFT) [41], rozpatrywanie rozległych, ale powtarzalnych systemów, dla których opracowano Parametryczne Drzewa Niezdatności - *ang. Parametric Fault Trees* (PFT) [18], a także uwzględnienie odnowy w Naprawialnych Drzewach Niezdatności (RFT) [146]. Niestety, ani żadne z wymienionych rozszerzeń rozpatrywane osobno, ani wszystkie rozpatrywane razem w postaci Uogólnionych Drzew Niezdatności [31] nie dają siły opisu wystarczającej do modelowania zależności czasowych opisanych w podrozdziale 2.1. Choć w powyższych sytuacjach można wykorzystać procesy Markowa bądź sieci Petriego, to takie modele okazują się skomplikowane, nieczytelne i trudne w analizie przez ekspertów w dziedzinie niezawodności.

Poszukiwania intuicyjnych metody rozpatrywania niezawodności mają w istocie głębokie uzasadnienie. Badania w dziedzinie ergonomii [15] wskazują, że dzięki instynktownym narzędziom zadania wykonywane są nie tylko szybciej, ale również, co szczególnie istotne w kontekście niezawodności, dokładniej. Autorzy wykazują ponadto, że aby narzędzia były postrzegane jako intuicyjne muszą nawiązywać do uznanych standardów tym bardziej, im starsi są jej użytkownicy, co ponownie ma znaczenie w niezawodności jako dziedzinie wymagającej wieloletniego doświadczenia. W świetle cytowanych wyników należy uznać, że przyczyny niskiej popularności sieci Petriego w dziedzinie niezawodności tkwią w unikatowości ich rozszerzeń i metodach analizy wymagających doświadczenia akademickiego.

W pracy [84] stwierdzono, że aby metoda analizy niezawodności była postrzegana jako intuicyjna przez inżynierów określonego systemu, to powinna przede wszystkim umożliwiać budowę modeli przypominających swoją strukturą rozpatrywany system. Potwierdza to niską intuicyjność sieci Petriego, w których odwołanie do struktury może następować jedynie przez nazwy miejsc i przejść. Drzewa niezdatności częściowo spełniają to wymaganie, gdyż zdarzenia podstawowe oznaczają awarie komponentów systemu, a bramy logiczne ukazują ich następstwa.

Niedoskonałości drzew niezdatności i sieci Petriego oraz chęć prowadzenia pracy interdyscyplinarnej stanowią motywację do postawienia następującej tezy niniejszej dysertacji:

Teza *Istnieje język opisu niezawodności bardziej intuicyjny niż sieci Petriego z jednej strony i bardziej ekspresywny niż drzewa niezdatności z drugiej, oraz istnieją metody analizy tego języka pozwalające na jego zastosowanie przy analizie niezawodności rzeczywistych problemów z zależnościami czasowymi w dziedzinach takich jak transport czy elektroenergetyka.*

Na wykazanie tezy składać się będzie osiągnięcie następujących celów:

1. opracowanie intuicyjnej metody badawczej dedykowanej analizie niezawodności systemów technicznych z uwzględnieniem zależności czasowych zarówno w składni jak i semantyce wchodzącego w jej skład języka modelowania,
2. przeprowadzenie nową metodą analizy niezawodności następujących systemów:
   - komputerowego z redundancją poddanego określonym politykom napraw,
   - koordynacji czasowej zabezpieczeń linii elektroenergetycznej,
   - tramwajowego we Wrocławiu,
   - eksploatacji samolotów niskokosztowych linii lotniczych.

Wiążąc poprzednie myśli z dominującą obecnie tendencją do integracji wielu metod przy analizie bezpieczeństwa [38,50,125] poszukiwania ekspresywnej metody należy ukierunkować na umiejętne łączenie ze sobą intuicyjnych i uznanych w świecie inżynierskim narzędzi. Do takich standardów należą: drzewa niezdatności, modele kolejkowe, UML czy systemy o logice rozmytej.

3.3 Struktura rozprawy

Iteracyjny rozwój metody analizy niezawodności przedstawiony w rozprawie uzasadnia jej podział na trzy główne części, z których każda omawia wyniki kolejnego etapu prac.

W części pierwszej, składającej się z rozdziałów 4 oraz 5, przedyskutowana jest stochastyczna natura Probabilistycznych Drzew Niezdatności z Zależnościami Czasowymi (PDNZC). Język udało się z powodzeniem zastosować do pewnego problemu z dziedziny elektroenergetyki, jak i systemu komputerowego poddanego niektórym politykom napraw.

W części drugiej, tj. w rozdziałach 6 oraz 7, mając na względzie moc modelowania zaprojektowane zostały Grafy Niezdatności (GN) posiadające wybrane cechy sieci Petriego. I to właśnie dzięki włączeniu tej techniki GN zawdzięczają możliwość zastosowania do bardziej wymagających problemów niezawodności systemu tramwajowego oraz polityk napraw opartych o zasoby.

Finalnym wynikiem pracy doktorskiej jest język READ i stąd najszerszy jego opis umieszczony w rozdziałach 8, 9 oraz 10. Dyskusji poddano nie tylko składnię READ zdefiniowaną w postaci profilu UML, ale także semantykę, dla której sieci Petriego okazały się efektywnym środkiem wyrazu. Wykazano ponadto, że dzięki uwzględnieniu diagramów aktywności, READ stał się narzędziem specyfikacji procesów eksploatacji z zależnościami czasowymi, co zostało zaprezentowane, między innymi, na systemie niskokosztowych linii lotniczych w rozdziale 10.

Konsekwencją przyjętego w 3.1 założenia o koncepcji badań jest podział każdej z części na dwa rozdziały. W pierwszym z nich, mianowicie w rozdziałach 4, 6 oraz 8 omówiono projekt języka, podczas gdy w rozdziałach 5, 7 oraz 9 i 10, zademonstrowano jego zastosowania.

Aby technika modelowania nie służyła jedynie dokumentacji systemu muszą istnieć efektywne metody analizy jej modeli. Dla wymienionych problemów poszukiwane będą symulacyjne oraz analityczne metody wyznaczania niezawodności. Proponowane podejście symulacyjne, szeroko korzystające z inżynierii sterowanej modelami, przedstawiono w rozdziałach 5 i 7. Narzędzia tam opisane posłużyły następnie do weryfikacji rozwiązań oraz oceny estymacji analitycznych opracowanych w rozdziale 9.

Dyskusja nad najważniejszymi osiągnięciami oraz podsumowanie dysertacji znajdują się w rozdziale 11.

Rozdział 4

Etap 1: Opracowanie Probabilistycznych Drzew Niezdatności z Zależnościami Czasowymi

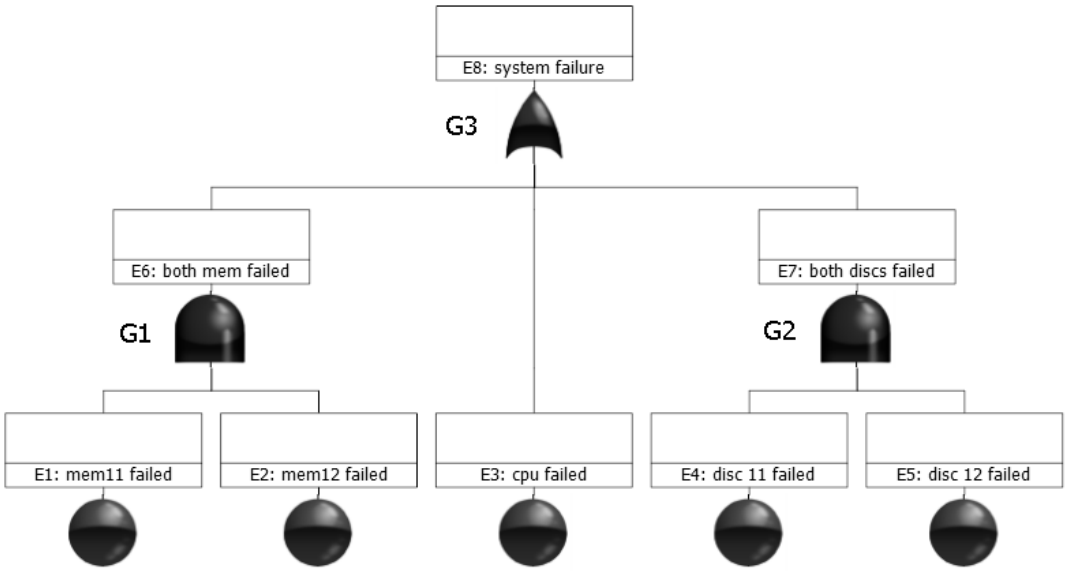
Probabilistyczne Drzewa Niezdatności z Zależnościami Czasowymi (PDNZC) to graficzny język modelowania umożliwiający opis oraz analizę niezawodności systemów czasu rzeczywistego, czyli takich, których poprawność działania zależy od spełnienia wymagań czasowych.

Metoda PDNZC wywodzi się z tradycyjnych drzew niezdatności, które jak dotąd największe uznanie znalazły w badaniach niezawodności reaktorów nuklearnych zapoczątkowanych w latach siedemdziesiątych. W podstawowej postaci drzewa niezdatności nie nadają się jednak do rozwiązywania problemów niezawodności z zależnościami czasowymi. Brak możliwości wyrażania aspektów czasowych zmusza do uproszczeń modelu, zwykle w stronę pesymistyczną, co z kolei prowadzi do rozwiązań określanych w dziedzinie jako zbyt konserwatywne [185]. Nadmierny pesymizm w spojrzeniu na niezawodność przejawia się w niepotrzebnej redundancji i komplikacji systemu, a także, w dłuższym okresie czasu, niechęci do badań niezawodności.

Zdefiniowany w niniejszym rozdziale język PDNZC to rozszerzenie tradycyjnych drzew niezdatności dające możliwość stochastycznego opisu zjawisk czasowych i tym samym odejście od niedokładnej analizy wynikającej z zastosowania języka podstawowego. W rozdziale następują kolejno: wprowadzenie do drzew niezdatności (podrozdział 4.1), opis składni i semantyki PDNZC (4.2) oraz modelowanie i badania symulacyjne niezawodności porównane z dostępnymi w literaturze (4.3). Zagadnienia zostaną omówione na tle naprawialnego systemu komputerowego z redundancją oryginalnie opisanego w [146]. Bardziej złożone zastosowanie PDNZC, nawiązujące do przedstawionego w [9] problemu koordynacji czasowej zabezpieczeń linii elektroenergetycznej, zostanie omówione w rozdziale 5.

4.1 Wprowadzenie do drzew niezdatności na przykładzie systemu komputerowego z redundancją

System komputerowy składa się z 5 komponentów: procesora, dwóch dysków twardych i dwóch bloków pamięci, które mogą ulegać awariom. Aby system działał poprawnie sprawne muszą być: procesor, przynajmniej jeden dysk i przynajmniej jedna pamięć. Do budowy modelu niezawodnościowego oraz wyznaczenia funkcji niezawodności wystarczą tradycyjne drzewa niezdatności.



Rysunek 4.1: Model nieodnawialnego systemu komputerowego z redundancją zaprojektowany w narzędziu Windchill FTA [144]

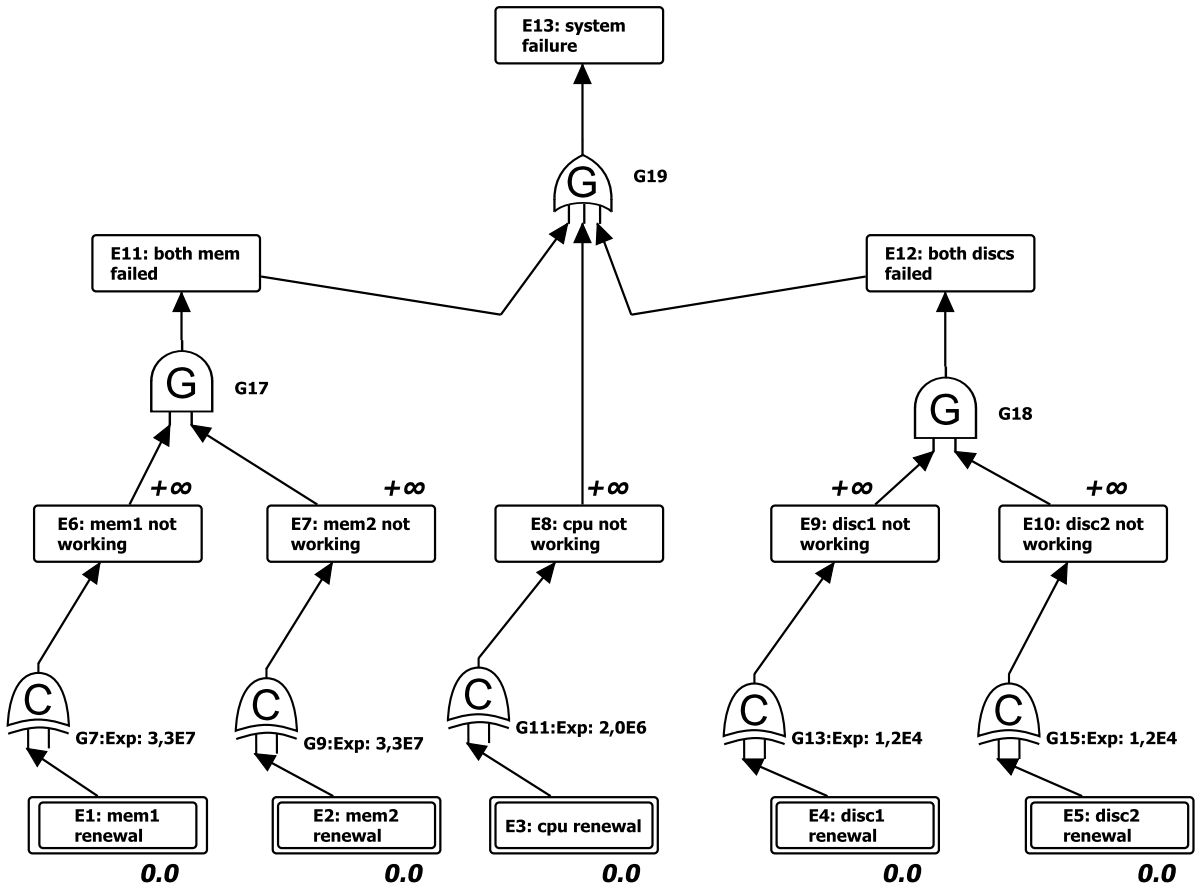
Model będący odwróconym graficznie drzewem jest rozbudowywany metodą dedukcyjną „od góry do dołu" tak, że ostatecznie drzewo niezdatności tworzy graf dwudzielny (rys. 4.1). Istotnie, przy rozbudowie wykorzystuje się naprzemiennie dwa typy węzłów: **zdarzenia** (elementy $E1$-$E8$ na rysunku) i **bramy** ($G1$-$G3$).

Każde zdarzenie, oznaczane prostokątem, niesie częściową informację o stanie niezawodności systemu. Zdarzeniem może być awaria jednego ($E1$) lub grupy komponentów ($E6$). Szczególnym typem zdarzenia jest **hazard** ($E8$) będący korzeniem drzewa i oznaczający awarię całego systemu. Zdarzenia mające koła u podstawy ($E1$-$E5$) są natomiast **zdarzeniami podstawowymi**, tj. nie są dalej rozbudowywane, ale w zamian mają przypisane zmienne losowe chwili ich wystąpienia.

Bramy, graficznie i behawioralnie nawiązujące do elementów znanych w układach cyfrowych, symbolizują związki pomiędzy zdarzeniami. W klasycznych drzewach niezdatności ich repertuar jest wąski: dostępne są jedynie bramy AND i OR. Zdarzenie będące na wyjściu bramy AND zachodzi, gdy zachodzą wszystkie zdarzenia będące na jej wejściach. W przypadku bramy OR wystarczy, że zachodzi jedno ze zdarzeń wejściowych, aby zaszło zdarzenie wyjściowe. Ze względu

na problemy z analizą oraz prawidłową budową modeli unika się bram zawierających negację o takich jak NOT, XOR czy NOR.

4.2 Składnia i semantyka PDNZC



Rysunek 4.2: PDNZC systemu komputerowego z redundancją zaprojektowane w narzędziu autorskim

PDNZC zachowują nowatorską koncepcję budowy modelu niezawodności opracowaną dla drzew niezdatności. Jakkolwiek model wciąż jest drzewem i grafem dwudzielnym, to zmienia się zbiór dostępnych bram i zdarzeń. W rezultacie PDNZC mają znacznie wyższą moc opisową niż język podstawowy.

W PDNZC istnieją dwa typy bram: **uogólniające** oraz **przyczynowe** oznaczane w modelach graficznych odpowiednio literami G oraz C (np. $G19$ oraz $G7$ na rys. 4.2).

4.2.1 Bramy uogólniające

Istotną cechą bram uogólniających z punktu widzenia intuicyjności PDNZC jest ich wyraźne podobieństwo do operatorów klasycznej logiki z jednej strony i do znanych w elektronice bram logicznych z drugiej. Przy definicjach rozszerzeń drzew niezdatności zawartych w PDNZC

wykorzystywane będzie oznaczenie:

(T symbolizuje logiczną prawdę, podczas gdy F - fałsz)

$z(t) = T \iff$ zdarzenie z występuje w chwili t

$z(t) = F \iff$ zdarzenie z nie występuje w chwili t

Bazując na powyższym, następująca zależność może zostać wykorzystana do matematycznego opisu całej rodziny bram uogólniających:

$z(t) = T \iff B(x_1(t), ...x_n(t)) = T$

$B -$ formuła boolowska zależna od typu bramy: AND, OR i inne

$x_i -$ zdarzenie wejściowe bramy

$z -$ zdarzenie wyjściowe bramy

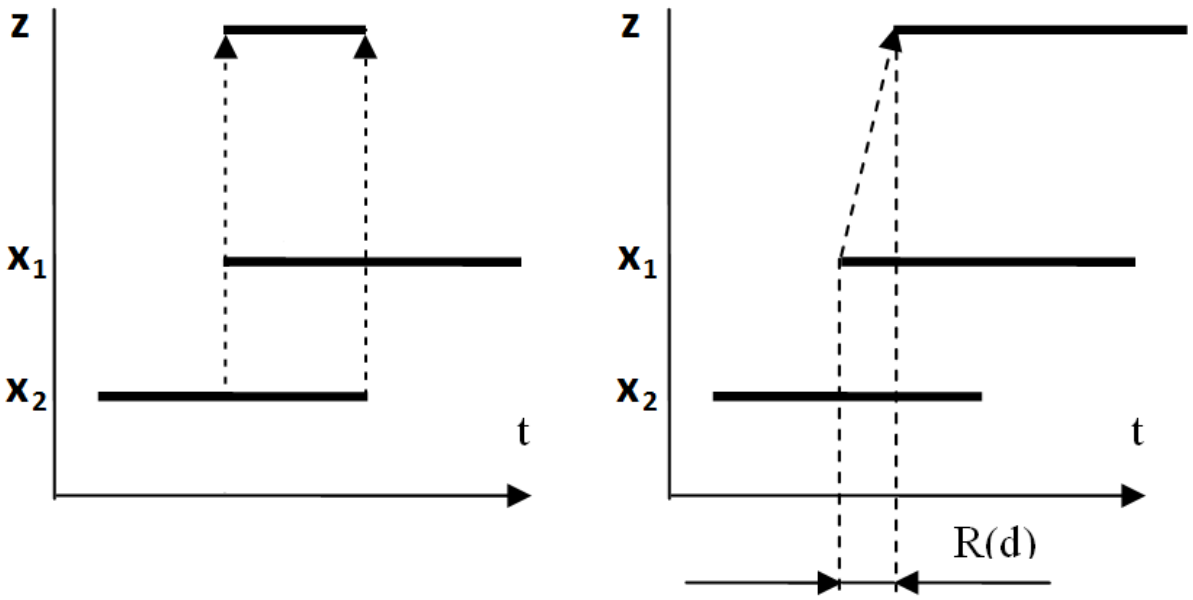
Dla dwuoperatorowej bramy AND:

$$B(x_1(t), x_2(t)) = x_1(t) \wedge x_2(t)$$

a dla trójoperatorowej bramy OR:

$$B(x_1(t), x_2(t), x_3(t)) = x_1(t) \vee x_2(t) \vee x_3(t)$$

Przykładowo, działanie bramy $G17$ z rys. 4.2 należy interpretować następująco. Gdy nastąpi zdarzenie $E6$ oraz $E7$, to z chwilą startu późniejszego rozpocznie się zdarzenie $E11$. Ponadto jak sugeruje model czasowy bramy $GAND$ przedstawiony na rys. 4.3a, kiedy zakończy się którekolwiek ze zdarzeń wejściowych, to zdarzenie wyjściowe zakończy się w tej samej chwili.



(a) Brama uogólniająca AND

(b) Brama przyczynowa AND

Rysunek 4.3: Model czasowy bram GAND i CAND, x_1, x_2 - zdarzenia wejściowe, z - zdarzenie wyjściowe, $R(d)$ - pewna realizacja zmiennej losowej d przypisanej do bramy CAND

4.2.2 Bramy przyczynowe

Porównanie rys. 4.3a z 4.3b pozwala dostrzec dwie zasadnicze różnice w działaniu bram przyczynowych w porównaniu z uogólniającymi. Jeśli zdarzenie wyjściowe bramy CAND wystąpi, to trwa ono nawet jeśli którekolwiek ze zdarzeń wejściowych zostanie zakończone. Ponadto rozpoczęcie zdarzenia wyjściowego jest opóźnione w czasie w stosunku do chwili zajścia obydwu zdarzeń wejściowych o czas odpowiadający realizacji zmiennej losowej d przypisanej do bramy CAND. Można to opisać formalnie w następujący sposób:

$$\begin{aligned} &occur(z) \Rightarrow occur(x) \wedge occur(y) \wedge overlap(xs, xk, ys, yk) \wedge \\ &(\tau(xs) \leqslant \tau(ys) \Rightarrow \tau(zs) = \tau(ys) + R(d)) \wedge \\ &(\tau(xs) > \tau(ys) \Rightarrow \tau(zs) = \tau(xs) + R(d)) \end{aligned}$$

gdzie:

$\tau(xs), \tau(ys), \tau(zs)$ – chwile rozpoczęcia zdarzeń x,y oraz z

$\tau(xe), \tau(ye), \tau(ze)$ – chwile zakończenia zdarzeń x,y oraz z

Predykat *overlap* przyjmuje wartość logiczną prawdy jeśli dwa odcinki czasu $\langle as, ak\rangle$ oraz $\langle bs, bk\rangle$ zachodzą na siebie:

$$overlap(as, ak, bs, bk) \leftrightarrow (ak \geqslant bs \wedge ak \leqslant bk) \vee (bk \geqslant as \wedge bk \leqslant ak)$$

Liczba zmiennych losowych przypisanych do bramy COR lub CXOR jest równa liczbie zdarzeń wejściowych tej bramy. Poniższa formuła jest specyfikacją działania bramy CXOR:

$$\begin{aligned} occur(z) \Rightarrow &(occur(x) \wedge \tau(zs) = \tau(xs) + R(d_x)) \oplus \\ &(occur(y) \wedge \tau(zs) = \tau(ys) + R(d_y)) \end{aligned}$$

Przy pomocy bramy CXOR z jednym wejściem można opisać opóźnienie pomiędzy startem zdarzenia będącego przyczyną zależności czasowej a rozpoczęciem zdarzenia wyjściowego będącego jej konsekwencją. Przykładowo, elementy $G7$, $E1$ i $E6$ na rys. 4.2 opisują czas do uszkodzenia pierwszego modułu pamięci. Po wystąpieniu odnowy ($E1$) inicjowana jest brama $G7$. W ten sposób zdarzenie $E6$ oznaczające awarię pierwszego bloku pamięci startowane jest po liczbie jednostek czasu wynikającej z realizacji zmiennej losowej przypisanej do $G7$.

W myśl powyższego, zdarzenie podstawowe znane ze zwykłych drzew niezdatności można zastąpić bramą CXOR i dwoma zdarzeniami zwykłymi. Zaletą takiego rozwiązania jest możliwość opisu elementów naprawialnych, którego nie można uzyskać przy pomocy zdarzeń podstawowych. Zdarzenia podstawowe ostatecznie usunięto z PDNZC.

W PDNZC zdarzenia na wyjściach bram przyczynowych mają przypisany czas trwania oznaczający czy zdarzenie jest *permanentne* czy *natychmiastowe*. W pierwszym przypadku (przy założeniu braku bramy CNOT opisanej dalej):

$$\tau(xe) = +\infty$$

Dla zdarzeń natychmiastowych natomiast:

$$\tau(xe) = \tau(xs)$$

Zakończenie zdarzeń można modelować również w inny sposób. Brama CNOT pozwala na zakończenie zdarzenia wyjściowego, jeśli zaszło jej jedyne zdarzenie wejściowe:

$$\tau(ze) = r \Rightarrow \tau(xs) + R(d) = r$$

Bramę CNOT wykorzystano na rys. 4.4 do opisu następującej sytuacji. Jeśli nastąpi odnowa bloku pamięci ($E1$) należy natychmiast zakończyć bramą $G8$ zdarzenie oznaczające awarię tego komponentu $E6$.

Definicja zbioru zdarzeń początkowych EI to:

$$x \in EI \iff \tau(xs) = 0$$

Takimi zdarzeniami są $E1 - E5$ z rys. 4.4 oznaczające, że w chwili początkowej następuje całkowita odnowa systemu i jest on sprawny.

4.3 Modelowanie i analiza polityk napraw systemu komputerowego

W obliczu braków tradycyjnych drzew niezdatności w modelowaniu systemów naprawialnych, w pracy [146] wprowadzono nowy element składniowy nazwany skrzynką naprawczą (ang. *repair box*). Zdefiniowanie polityki napraw w tym podejściu odbywa się poprzez połączenie zdarzeń podstawowych ze skrzynką i wskazanie jednej z przewidzianych przez Autorów metody naprawczej. Kolejnym etapem jest transformacja całego drzewa w sieci Petriego i to dopiero w tym języku zawarto opis behawioralny polityki.

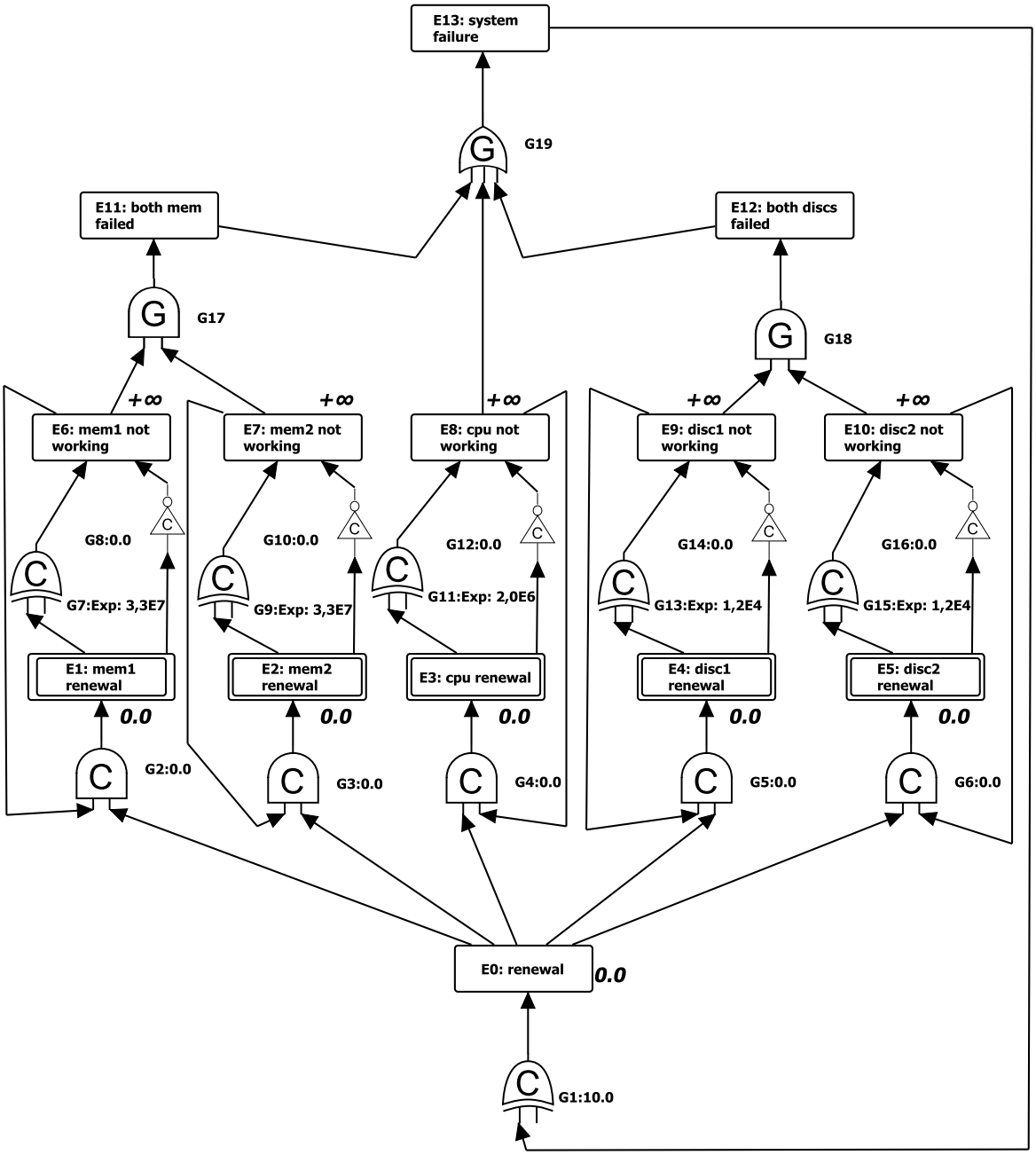
Podstawowym celem rozwoju PDNZC jest umożliwienie opisu zjawisk czasowych w samych drzewach niezdatności bez odwoływania się do metod alternatywnych. Porównanie proponowanej metody ze skrzynkami naprawczymi, wspomagającymi modelowanie pewnej grupy procesów odnowy, dała możliwość oceny siły opisu PDNZC oraz wskazała obszary koniecznych usprawnień, które można również odnieść do problemów niezwiązanych z samymi naprawami.

Założenia wszystkich omówionych poniżej polityk zostały zebrane poniżej.

- Początek konserwacji następuje dopiero, gdy system przestanie działać, czyli po awarii procesora, dwóch dysków lub dwóch pamięci. Nie jest możliwa naprawa pojedynczego dysku czy bloku pamięci bez trwającej awarii systemu.

- W czasie konserwacji system jest wyłączony, a wznowienie następuje dopiero, gdy wszystkie uszkodzone komponenty zostaną naprawione.

- Uszkodzenia elementów, które wydarzą się po zainicjowaniu naprawy, są włączane w jej bieg i nie muszą czekać na rozpoczęcie kolejnej odnowy.



Rysunek 4.4: PDNZC opisujące globalną politykę odnowy

4.3.1 Globalna naprawa systemu

Odnowa globalna znajduje zastosowanie w sytuacjach, gdy czas naprawy można dobrze opisać jedną zmienną losową, a przebieg naprawy nie jest zależny od typu aktualnie uszkodzonych w systemie komponentów. Dobrymi przykładami realizującymi taką politykę są: fizyczną wymiana systemu na nowy - identyczny bądź całkowity restart sprzętu lub oprogramowania.

W myśl powyższego, na samym dole modelu z rys. 4.4 dodano bramę $G1$, której opóźnienie odpowiada czasowi realizacji naprawy. Brama aktywowana jest wystąpieniem zdarzenia $E13$, czyli hazardem. Gdy upłynie właściwy czas zachodzi zdarzenie natychmiastowe odnowy syste-

mu $E0$, które może wpłynąć na bramy $G2 - G6$. Dla każdego z uszkodzonych komponentów, modelowanego jednym z trwających zdarzeń $E6 - E10$, odpowiadająca brama CAND $G2 - G6$ spowoduje rozpoczęcie wyjściowego zdarzenia natychmiastowego. Zatem zdarzenia $E1 - E5$ to odnowy poszczególnych uszkodzonych komponentów. W tym samym momencie wykonają się jeszcze dwie akcje dla każdego uszkodzonego elementu. Na przykładzie bloku pamięci $mem1$ będą to: odpalenie bramy CNOT $G8$ kończącej aktualną awarię reprezentowaną przez $E6$ oraz odpalenie bramy $G7$ zapowiadającej następną awarię odległa w czasie o realizację wykładniczej zmiennej losowej przypisanej do $G7$.

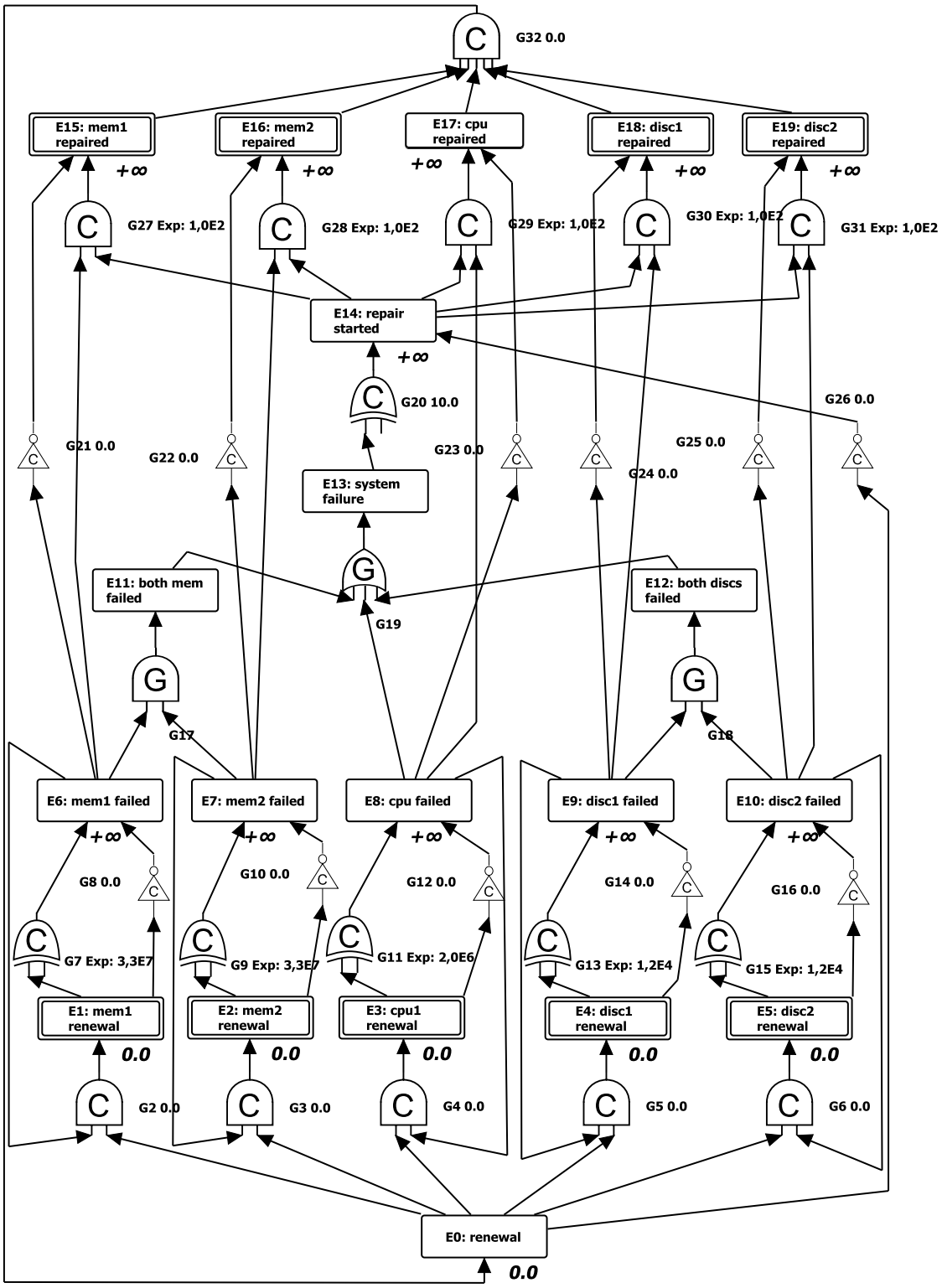
4.3.2 Odnowa systemu komputerowego poprzez równoczesną konserwację wszystkich uszkodzonych komponentów

W polityce równoległej, utożsamianej również z odnową z nieskończoną liczbą konserwatorów, wszystkie uszkodzone komponenty naprawiane są jednocześnie, każdy zgodnie ze swoją własną intensywnością naprawy. W związku z tym, do modelu przedstawionego na rys. 4.5 wprowadzono nowe zdarzenia $E15 - E19$ oznaczające, że dany element został naprawiony. Przylegające bramy CNOT (np. $G21$) kończą zdarzenia natychmiast po awarii danego komponentu.

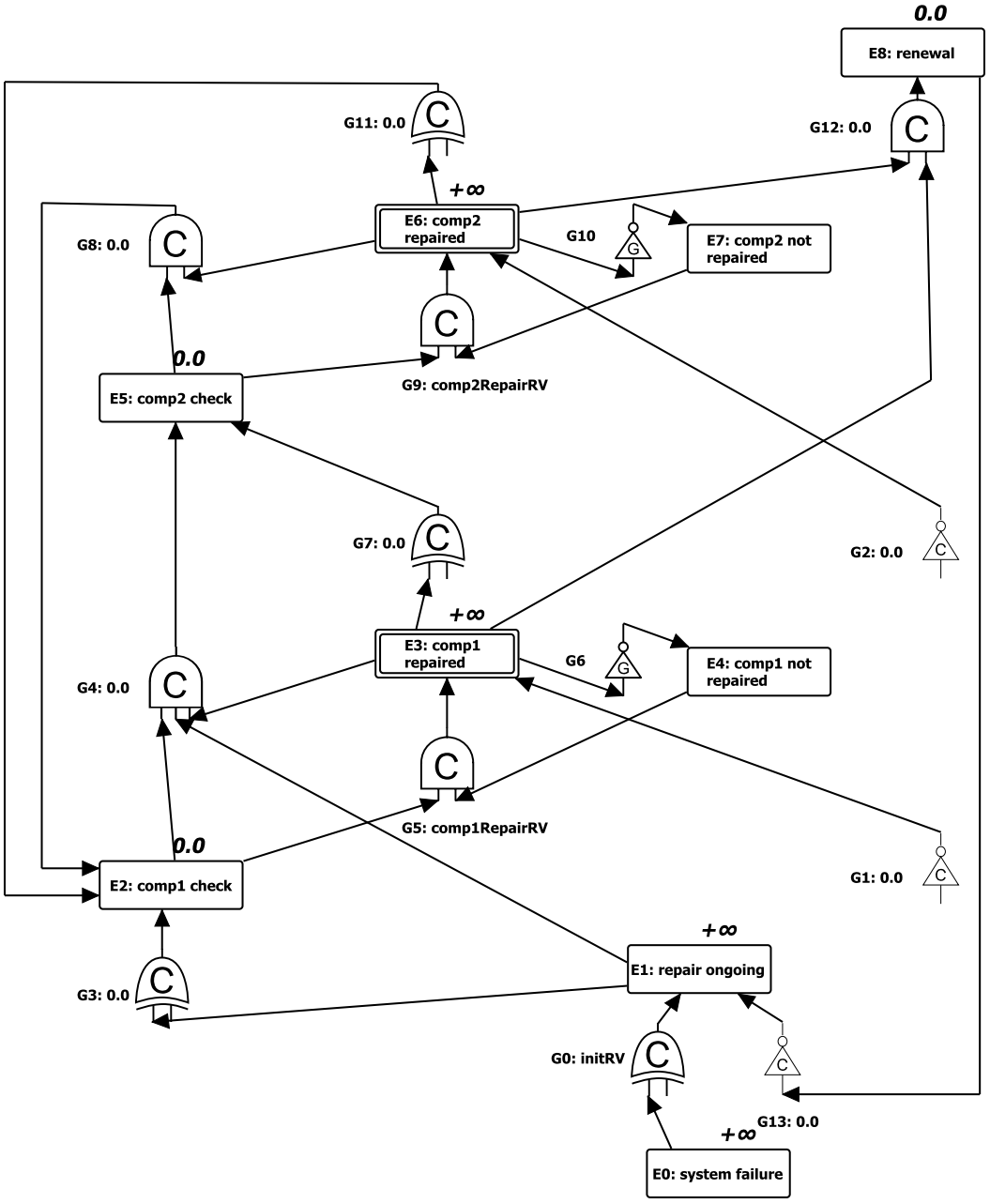
Po wystąpieniu awarii systemu, symbolizowanego rozpoczęciem zdarzenia $E13$, rozpoczyna się inicjalizacja odnowy. Powiązane opóźnienie, wynoszące 10 jednostek czasu, odmierzane jest przez bramę CXOR $G20$ i kończy się startem odnowy modelowanym $E14$. W tej samej chwili czasu każdy uszkodzony komponent (czyli taki, dla którego zachodzi $E6 - E10$) poddawany jest procesowi naprawy. Równoległa odnowa komponentów zachodzi dzięki niezależny bramom CAND $G27 - G31$ mającym przypisane opóźnienie wynikające z elementu, na rzecz którego brama działa. Zakończenie wszystkich elementarnych napraw synchronizowane jest bramą $G32$. Kiedy wszystkie komponenty są sprawne, to rozpoczyna się odnowa systemu $E0$. Następujący opis jest podobny do polityki globalnej. Wszystkie uszkodzone komponenty są jednocześnie odnawiane dzięki bramom CAND aktywującym zdarzenia $E1 - E5$, wtedy i tylko wtedy gdy dany komponent był uszkodzony. To pozwala na zakończenie przy pomocy bram CNOT zdarzeń oznaczających uszkodzenie komponentu i zapowiedź kolejnych awarii bramami CXOR. Ostatnim krokiem naprawy jest zakończenie zdarzenia $E14$ przez bramę $G26$, co oznacza, że konserwacja została zakończona.

4.3.3 Odnowa systemu poprzez sekwencyjną naprawę komponentów

Sekwencyjna polityka naprawy to taka, w której jeden konserwator naprawia kolejno wszystkie uszkodzone elementy aż cały system zostanie odnowiony. Szkielet takiego modelu dla dwóch komponentów, będący jednocześnie szablonem transformacji systemu nienaprawialnego, przedstawiono na rys. 4.6. Bramy CNOT $G1$ i $G2$ powinny zostać połączone ze zdarzeniami uszkodzenia odpowiadających komponentów. Ich wykonanie spowoduje konieczność naprawy określonego komponentu symbolizowaną przez $E3$ i $E6$. W związku z redundancją systemu uszkodzenie pojedynczego komponentu nie musi powodować jeszcze rozpoczęcia naprawy.



Rysunek 4.5: PDNZC opisujące równoległą politykę odnowy



Rysunek 4.6: Szkielet polityki szeregowej

Wraz z upływem czasu, gdy system zostanie uszkodzony ($E0$) brama $G0$ wygeneruje opóźnienie związane z zainicjowaniem naprawy. Wynikające zdarzenie $E1$ będzie trwało do końca naprawy. Następnie bramą $G3$ inicjowany jest sekwencyjny proces sprawdzenia lub naprawy każdego kolejnego elementu.

Elementy $E2 - E4$ oraz $G4 - G7$ opisują przegląd pierwszego komponentu. Zdarzenia $E3$ i $E4$ są wzajemnie komplementarne - rozpoczęcie i zakończenie pierwszego powoduje odpowiednio zakończenie i rozpoczęcie drugiego przez bramę GNOT. W wyniku diagnostyki komponentu przez konserwatora ($E2$) możliwe są dwie sytuacje rozpoznawane przez bramy $G4$ oraz $G5$.

Ignorując chwilowo łuk $E1 - G4$ można zauważyć, że $G4$ modeluje sytuację, w której kom-

ponent jest aktualnie sprawny, czyli nie był uszkodzony lub został wcześniej naprawiony. Wtedy $G4$ powoduje natychmiastowe przejście do diagnostyki drugiego elementu, czyli startuje $E5$.

Alternatywnie, jeśli komponent jest aktualnie uszkodzony, czyli występuje zdarzenie $E4$, konserwator naprawia element, co modeluje brama $G5$ i przypisany do niej czas *comp1RepairRV*. Wystąpienie zdarzenia $E3$ oznacza zakończenie naprawy i daje podstawy do rozpoczęcie diagnostyki kolejnego komponentu ($E5$).

Elementy $E5 - E7$ oraz $G8 - G11$ to powtórzenie procesu dla drugiego komponentu. Zwraca jednak uwagę sprzężenie zwrotne $G8$ - $E2$ powodujące, że naprawą objęte zostaną nowe uszkodzenia powstałe w trakcie samego procesu odnowy. Jeśli wszystkie elementy są już sprawne powstanie pętla zdarzeń $E2$, $E5$,..., która kończy się w następujący sposób.

Brama $G12$ rozpozna sytuację, gdy wszystkie komponenty są aktualnie sprawne. Wystąpienie $E8$ służy dwóm celom. Po pierwsze, powinien rozpocząć się proces resetu poszczególnych komponentów podobnie jak na rys. 4.5. Po drugie, następuje zakończenie naprawy, której trwanie symbolizowane jest zdarzeniem $E1$. Pominięty poprzednio łuk do $G4$ spowoduje teraz zakończenie cyklu przeglądów naprawionych już elementów.

4.3.4 Badania symulacyjne i weryfikacja wyników

W pracy [89] opisano narzędzie, w którym zaprojektowane zostały omawiane polityki napraw, oraz przy pomocy którego przeprowadzono obliczenia symulacyjne. Przyjęte parametry niezawodnościowe (tab. 4.1) były spójne z pracą [146], co pozwala na weryfikację modeli.

Tabela 4.1: Parametry niezawodnościowe komponentów systemu

Komponent	Intensywność [1/h]	
	Awaria	Naprawa
procesor	5×10^{-7}	1×10^{-2}
dysk twardy	8×10^{-5}	1×10^{-2}
moduł pamięci	3×10^{-8}	1×10^{-2}

Eksperyment polega zatem na translacji modelu polityki w kod wykonywalny symulujący zachowanie systemu. Na podstawie pomiaru czasu wystąpienia hazardu, czyli zdarzenia $E13$, wyznaczony został współczynnik niedostępności:

$$NA = \frac{\text{łączny czas występowania zdarzenia } E13}{\text{czas symulowany}}$$

Wyniki symulacji dla dwóch przypadków polityki globalnej, równoległej oraz umieszczono w tab. 4.2. Niewielkie błędy względne umieszczone w ostatniej kolumnie potwierdzają tożsamość modeli PDNZC i opartych o skrzynki naprawcze.

4.4 Moc ekspresji PDNZC

Kilkanaście przypadków modelowania z użyciem proponowanego języka pozwala na podsumowanie w kontekście postawionej tezy spostrzeżeń na temat siły wyrazu PDNZC.

Tabela 4.2: Porównanie wyników symulacji polity napraw

Polityka	Współczynnik niedostępności NA		
	[146]	PDNZC	Błąd [%]
polityka globalna $G1$: 10	$5{,}37\times10^{-4}$	$5{,}36\times10^{-4}$	0,2
polityka równoległa	$8{,}517\times10^{-3}$	$8{,}516\times10^{-3}$	0,02
polityka szeregowa	$1{,}11147\times10^{-2}$	$1{,}10927\times10^{-2}$	0,2
polityka globalna $G1$: 250	$1{,}3253\times10^{-2}$	$1{,}3227\times10^{-2}$	0,2

- Jak pokazano w pracy [116] PDNZC wystarczają do wyrażenia sieci PERT używanych do szacowania ryzyka nieterminowej realizacji projektów.
- W [116] udało się również wyrazić najważniejsze bramy DFT: bramę priorytetową, zimnej rezerwy oraz zależności funkcyjnej.
- W niniejszym rozdziale wykazano, że polityka globalna, równoległa oraz szeregowa mogą być wyrażone w samych drzewach niezdatności, bez odwoływania się do sieci Petriego. Tym samym inżynierowie korzystający z drzew niezdatności nie są ograniczeni do wariantów przewidzianych przez skrzynkę naprawczą i mogą swobodnie adaptować politykę.
- W PDNZC udało się wyrazić nietrywialny i istotny praktycznie problem koordynacji zabezpieczeń odległościowych, co opisano w rozdziale 5.
- Dwie cechy bram przyczynowych: niezależność zakończenia wyjścia od zakończenia wejścia oraz przesunięcie w czasie są względem siebie ortogonalne. Bazując na tym spostrzeżeniu w pracy [156] zdefiniowano bramę uogólniającą z opóźnieniem, dla której wskazano również obszar zastosowań. Taką bramę również można wyrazić przy pomocy PDNZC.
- Dla drzew NDNZC opracowano dwa typy bram przyczynowych [115]. Pierwszy wymaga nakładania się zajść zdarzeń wejściowych i jest spójny z bramami omawianymi w niniejszej rozprawie, których symbole graficzne posiadają literę C. Konstrukcja języka PDNZC umożliwia dodanie bram bez nakładania.

Nie ulega zatem wątpliwości, że PDNZC to wyraźny postęp w stosunku do klasycznych drzew niezdatności, których nie można zastosować w żadnym z wymienionych przypadków.

Należy zaznaczyć, że część prób modelowania zakończyła się jednak niepowodzeniem. W PDNZC brakuje elementu pozwalającego na rozstrzyganie wyścigów pomiędzy procesami. Tym samym opis niektórych polityk napraw, opartych o liczbę konserwatorów większą od 1, ale mniejszą od liczby komponentów, nie jest możliwy. Podobny problem występuje przy próbie opisu modeli kolejkowych o skończonej liczbie serwerów. PDNZC ma wyraźnie ograniczone możliwości modelowania synchronizacji procesów równoległych wynikające z braku elementów jednocześnie kończących i rozpoczynających zdarzenia. Próżno szukać takiego elementu wśród tradycyjnych bram, gdyż klasyczne bramy mają tylko jedno zdarzenie wyjściowe. Powyższe problemy zostaną zaadresowane w rozdziale 6.

Rozdział 5

Koordynacja opóźnień wielokrotnej redundancji gorącej zabezpieczeń odległościowych w elektroenergetyce z wykorzystaniem PDNZC

Zabezpieczenia odległościowe są powszechnie stosowane w elektroenergetyce do ochrony transmisji mocy. Poprawnie skonfigurowane powodują szybkie rozłączenie fragmentu linii z transmisji, ale w taki sposób, aby wyłączony odcinek był fizycznie jak najmniejszy. Sytuację, w której wyłączony obszar jest nadmiernie duży określa się w niniejszym rozdziale *hazardem*. Koordynacja czasowa zabezpieczeń odległościowych ma na celu dobranie czasów rozłączania linii przez zabezpieczenia, tak aby prawdopodobieństwo hazardu było możliwie małe.

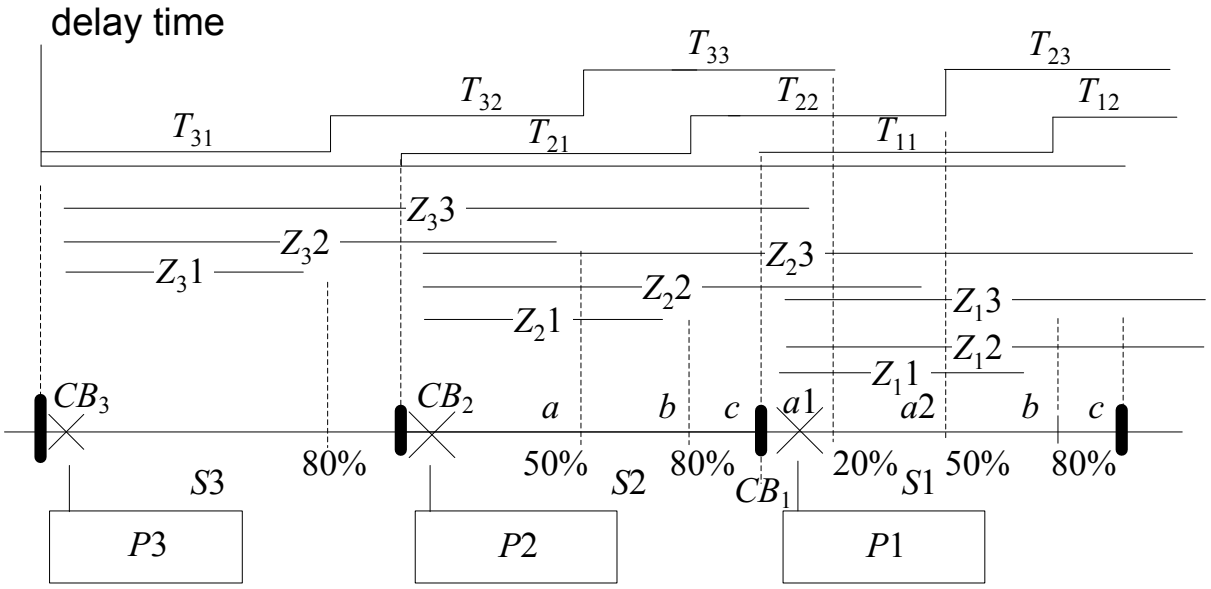
Dotychczasowe zastosowanie PDNZC do tytułowego problemu obejmuje modelowanie, symulację oraz estymację hazardu [8, 9, 92, 97]. W niniejszym rozdziale zostanie opisany język dziedzinowy [51] dedykowany problemowi koordynacji zabezpieczeń oraz algorytm jego transformacji w PDNZC zaimplementowany w języku QVT [124] wspomaganym narzędziem OCL [132]. Wprowadzenie do OCL'a zawarto w dodatku B.

Z powodu wielokrotnej redundancji dobór opóźnień zabezpieczeń odległościowych jest problemem nietrywialnym, a jednocześnie bardzo istotnym praktycznie. Zagadnienie dodatkowo komplikuje jednoczesne wykorzystanie zabezpieczeń różnych typów: jedno- lub wielosystemowych, z członami startowymi lub bez nich. Nie ulega wątpliwości, że istnieje potrzeba szybkiego tworzenia poprawnego modelu niezawodnościowego dla zadanej przez inżyniera konfiguracji systemu elektroenergetycznego. Bezpośrednią odpowiedzią na to wymaganie jest zaprojektowanie dedykowanego języka modelowania, z którego możliwa jest automatyczna synteza modeli PDNZC. Biorąc to pod uwagę można zdefiniować proces analizy niezawodności zabezpieczeń składający się z poniższych kroków.

1. Zdefiniowanie stref zabezpieczeń odległościowych linii transmisyjnej.
2. Opis linii oraz zabezpieczeń w języku dziedzinowym.
3. Wyznaczenie parametrów wejścia i wyjścia zabezpieczeń, na przykład narzędziem EMTP [108].
4. Translacja modelu wyrażonego w języku dziedzinowym w PDNZC.
5. Znalezienie prawdopodobieństwa hazardu metodą symulacyjną lub analityczną.

Etapy 1 i 3 wymagają wiedzy eksperckiej z dziedziny elektroenergetyki i nie podlegają automatyzacji. W niniejszym rozdziale skoncentrowano się na punktach 2 i 4, podczas gdy symulator będący podstawą ostatniego etapu został opisany w [89].

5.1 Struktura i działanie redundantnych zabezpieczeń odległościowych



Rysunek 5.1: Linia elektroenergetyczna z sekcjami (S1-S3), podsekcjami (a1, a2, b,c,d), zabezpieczeniami (P1-P3), strefami (Z) i czasami opóźnień (T) [9]

Linia elektroenergetyczna składa się z sekcji, których granice są wyznaczone przez stacje umieszczone na jej długości. Przypadek analizowany w tym rozdziale to linia o długości 270km podzielona na 3 sekcje: S1, S2 i S3 (rys. 5.1). Z każdą stacją związany jest wyłącznik CB_i sterowany przez zabezpieczenie Pi, realizowane współcześnie jako procesor sygnałowy. Zabezpieczenie działa jednocześnie w 3 strefach oznaczonych symbolami Z_ij, w których i oznacza

numer zabezpieczenia, a j numer strefy. Wykrycie uszkodzenia, np. zwarcia, w jednej ze stref przy spełnieniu opisanych dalej zależności czasowych powoduje wyłączenie transmisji mocy we fragmencie linii na prawo od wyłącznika. Jak wynika z rysunku każda kolejna strefa tego samego zabezpieczenia jest dłuższa i pokrywa większy fragment linii. Przykładowo, $Z_3 1$ pokrywa 80% sekcji S1, podczas gdy $Z_3 2$ całą sekcję S1 oraz 50% długości S2, a $Z_3 3$ pokrywa S1, S2 oraz 20% S3.

Granice działania 9 stref zabezpieczeń wyznaczają podsekcje. Sekcja S2 składa się z podsekcji a, b i c natomiast S3 z a1, a2, b i c. Podsekcja a1 jest chroniona zatem przez 6 stref: 3 sterowane zabezpieczeniem P1, 2 sterowane przez P2 oraz 1 sterowana przez P3. Strefy P1 określa się jako lokalne, gdyż wynikają z obecności zabezpieczenia w sekcji, w której znajduje się a1. Pozostałe to strefy zdalne. Każda z 6 stref może niezależnie spowodować rozłączenie linii przez odpowiadające im zabezpieczenia. Tak wysoki poziom redundancji wynika z awaryjności wyłączników i zabezpieczeń, a także dużych konsekwencji awarii.

Rozłączenie przez strefę lokalną jest optymalne, ponieważ wyłączony z dystrybucji zostaje minimalny fragment linii, czyli sekcja S1. Jeśli jednak rozłączenia dokona jakieś zabezpieczenie zdalne, mocy pozbawiony zostanie nadmierny obszar. W skrajnym przypadku może tego dokonać P1 rozłączając całą linię. Hazardem jest zatem zdarzenie: wyłączenie zdalnego wyłącznika w sytuacji, gdy to wyłącznik lokalny może być wyłączony.

Zabezpieczenie wielosystemowe bez członu startowego działa następująco. Czas, po którym impedancja awarii wchodzi w charakterystykę danej strefy, czyli jest przez nią rozpoznawana, określa się czasem wejścia i naturalnie zależy od odległości do zabezpieczenia. Wszystkie strefy testowane są jednocześnie. Po wykryciu awarii w którejkolwiek strefie zabezpieczenie Pi odczekuje zaprogramowany odcinek czasu T_{ij}, po czym, jeśli impedancja nie wyszła z charakterystyki strefy (czas wyjścia), rozłącza linię. Czasy opóźnień kolejnych stref zabezpieczenia są coraz większe, to znaczy: $T_{i1} < T_{i2} < T_{i3}$, przy czym zazwyczaj $T_{i1} = 0$. Zabezpieczenie wielosystemowe może odmierzać czasy równolegle dla wielu stref i powoduje otwarcie wyłącznika, gdy upłynie pierwszy z nich, a impedancja nie wyjdzie z charakterystyki strefy.

W zabezpieczeniu jednosystemowym z członem startowym jest jeden licznik, który aktywowany jest wejściem impedancji w *strefę startową* typowo obejmującą najdłuższą, trzecią strefę. Dopiero wtedy rozpoczyna się odmierzanie czasu sekwencyjnie dla kolejnych stref. Jeśli awaria została wykryta w chwili τ, to kolejne momenty wyłączenia upływają w chwilach $\tau + T_{i1}$, $\tau + T_{i2}$ oraz $\tau + T_{i3}$ dla stref $Z_i 1$, $Z_i 2$ i $Z_i 3$. Wadą starszych rozwiązań jednosystemowych jest długi czas oczekiwania na reakcję w pierwszej strefie.

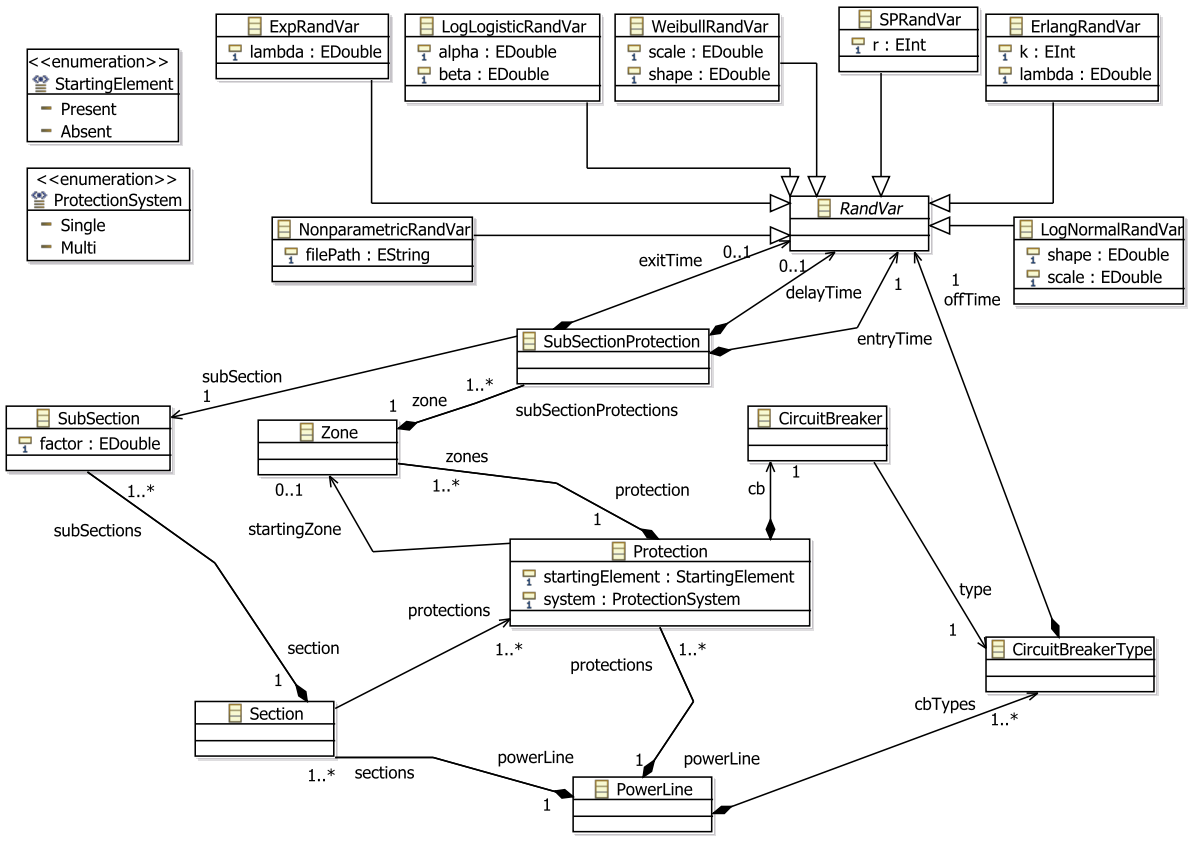
Czasy wejścia i wyjścia, wyznaczane narzędziem EMTP [108], są w rozdziale oznaczane symbolami $T_{iej|kf}$:

- $i \in \{1, 2, 3\}$ numer zabezpieczenia Pi, którego dotyczy czas,
- $e \in \{en, ex\}$ en (lub: ex) oznacza czas wejścia (lub: wyjścia) z charakterystyki impedancji,
- $j \in \{1, 2, 3, S\}$ numer strefy (lub identyfikator strefy startowej) zabezpieczenia Pi przy założeniu, że awaria wydarzyła się w sekcji k, podsekcji f,

- $k \in \{1, 2\}$ numer sekcji,
- $f \in \{a, a1, a2, b, c, d\}$ nazwa podsekcji.

W rozdziale przedstawiona jest transformacja opisu zabezpieczeń w PDNZC, z których można wyznaczyć zależność między opóźnieniami T_{ij} a prawdopodobieństwem hazardu.

5.2 Składnia abstrakcyjna języka do opisu redundancji w zabezpieczeniach



Rysunek 5.2: Model obiektowy języka dziedzinowego koordynacji zabezpieczeń

Zaprojektowany język dziedzinowy składa się z dwu części. Pierwszą jest obiektowa składnia abstrakcyjna języka dziedzinowego. Modele w niej wyrażone stanowią dane wejściowe automatycznej transformacji dziedzinowego modelu zabezpieczeń w PDNZC opisujące niezawodność systemu. Definiowanie modeli w składni abstrakcyjnej, choć możliwe, jest postrzegane przez użytkowników jako niewygodne. W tym celu zaprojektowano drugą część języka nazwaną składnią konkretną, zapisaną w postaci zbliżonej do EBNF i opisaną w podrozdziale 5.4.

Jak widać na dole rys. 5.2 przedstawiającego składnię abstrakcyjną, sekcja (klasa *Section*) jest elementem, z których zbudowana jest linię elektroenergetyczną (*PowerLine*). Na sekcję

składają się podsekcje *SubSection*, a każda z nich posiada atrybut $factor$ oznaczający jaką część sekcji pokrywa.

Równolegle ze strukturą linii modeluje się zabezpieczenia. To referencja *cb* klasy *Protection* wskazuje wyłącznik, którym fragment linii jest rozłączany z transmisji. Klasa $CircuitBreaker$ odpowiada elementom $CB_1 - CB_3$ na rys. 5.1. Ich parametry czasowe określa atrybut $offTime$.

Z atrybutów obiektów klasy *Protection* wynika, czy modelowane zabezpieczenie jest jednoczy wielosystemowe i czy posiada człon rozruchowy. Niezbędne wartości zostały zdefiniowane w typach wyliczeniowych *StartingElement* oraz *ProtectionSystem*. Jeśli atrybut *startingElement* jest równy *Present*, to strefę członu rozruchowego wskazuje referencja *startingZone*. Należy jednak zaznaczyć, że wszystkie strefy, w tym ewentualnie startowa, znajdują się w kolekcji *zones*.

Zawierane przez *Zone* obiekty *SubSectionProtection* są najistotniejsze z perspektywy problemu koordynacji czasowej. To one definiują niezbędne parametry czasowe istotne z perspektywy analizy niezawodności.

Jeśli dane zabezpieczenie podsekcji jest lokalne (tj. nadzorowana podsekcja należy do sekcji, w której zlokalizowane jest nadzorujące zabezpieczenie), do poprawnego opisu wystarczy referencja czasu wejściowego, czyli *entryTime* oraz opóźnienia - *delayTime*. Jeśli jednak zabezpieczenie jest zdalne, konieczny jest również czas wyjścia - *exitTime*.

W rozważanym problemie do definicji czasów wejścia i wyjścia najczęściej znajdują zastosowanie rozkłady Erlanga oraz logarytmiczno-normalny, których klasy są specjalizacjami $RandVar$. Dostępne są inne możliwości opisu, w tym rozkłady nieparametryczne.

5.3 Składnia konkretna języka do opisu redundancji w zabezpieczeniach

Jak zasygnalizowano w podrozdziale 5.2 to składnia konkretna jest faktycznie używana do definicji modelu przez ekspertów i służy do wyprowadzenia składni abstrakcyjnej transformowanej w PDNZC. Modelami składni konkretnej są w uproszczeniu pliki tekstowe.

Składnia konkretna zdefiniowana została przy pomocy rozszerzonej postaci Backusa-Naura składającej się ze zbioru reguł po jednej dla każdej klasy z rys. 5.2. Na przykład regułę dla klasy *PowerLine* zdefiniowano w liniach 1-3 wydruku 5.1.

Gdy parser dopasuje fragment tekstu do reguły, tworzy nowy obiekt klasy powiązanej z regułą oraz ustawia atrybuty i referencje na te, wynikające z fragmentu modelu. W przypadku linii elektroenergetycznej będą to kolejno (wydruk 5.2): nazwa, przynajmniej jeden (z powodu symbolu '+') typ wyłącznika, przynajmniej jedna sekcja i przynajmniej jedno zabezpieczenie. Poszczególne grupy elementów są separowane literałami, w tym przypadku nawiasami klamrowymi. Ze składni abstrakcyjnej parser wnioskuje o typie referencji np. *cbTypes*, a tym samym jest w stanie dotrzeć do reguły gramatycznej, która powinna być spełniona w miejscu odniesienia do referencji, czyli $CircuitBreakerType$.

W przypadku użycia parsera typu „od góry do dołu", począwszy od *PowerLine* kolejne reguły będą dopasowywane w dół aż do uzyskania kompletnego modelu linii elektroenergetycznej

wraz z zabezpieczeniami zaprojektowanymi przez eksperta.

Kiedy w gramatyce odnosi się do atrybutu (np. *name*, linia 1 wydruku 5.1), to po jego nazwie musza występować nawiasy kwadratowe z opcjonalnym typem pomiędzy nawiasami. Kiedy jednak następuje odwołanie do obiektu możliwe są dwie sytuacje. Obecność nawiasów kwadratowych (np. *type*, linia 40) powoduje, że parser wyszukuje wśród istniejących już obiektów bytu o podanej nazwie i to on zostaje przypisany do referencji. Brak nawiasów (*cbTypes*, linia 2) to utworzenie nowego obiektu poprzez analizę odpowiedniej reguły.

Wydruk 5.1: Definicja składni konkretnej języka dziedzinowego

```
1  PowerLine::= "PowerLine" name[] "{"      "CircuitBreakerTypes"
2          "{" cbTypes+ "}" "Sections" "{" sections+ "}"
3          "Protections" "{" protections+ "}" "}";
4
5  CircuitBreakerType::= "CBType" name[] "OffTime" offTime;
6
7  WeibullRandVar::= "Weibull" "{" "scale" ":" scale[FLOAT]
8          "Shape" ":" shape[FLOAT] "}";
9
10 ExpRandVar::= "Exp" "{" "Lambda" ":" lambda[FLOAT] "}";
11
12 SPRandVar::= "SP" "{" "R" ":" r[INT] "}";
13
14 ErlangRandVar::= "Erlang" "{" "K" ":" k[INT]
15         "Lambda" ":" lambda[INT] "}";
16
17 LogNormalRandVar::= "LogNormal" "{" "Scale" ":" scale[FLOAT]
18         "Shape" ":" shape[FLOAT] "}";
19
20 LogLogisticRandVar::= "LogLogistic" "{" "Alpha" ":" alpha[FLOAT]
21         "Beta" ":" beta[FLOAT] "}";
22
23 NonparametricRandVar::= "Nonparametric" "{"
24         "Histogram" ":" filePath[] "}";
25
26 Section::= "Section" name[] "{" subSections+ "}";
27
28 SubSection::= "SubSection" name[] "Factor" factor[INT];
29
30 Protection::= "Protection" name[] "{" cb zones+
31         "System" system[] "StartingElement" startingElement[]
32         ("StartingZone" startingZone[])? "}" ;
33
34 Zone::= "Zone" name[] "{" subSectionProtections+ "}";
35
36 SubSectionProtection::= "SubSection" subSection[]
37         "{" "EntryTime" entryTime ("ExitTime" exitTime)?
38         ("DelayTime" delayTime)? "}" ;
```

```
39
40 CircuitBreaker::= "CircuitBreaker" name[] "Type" type[];
41 }
```

Wydruk 5.2 opisuje linię z zabezpieczeniami z rys. 5.1. W systemie istnieje tylko jeden typ wyłącznika, a więc czas rozłączania jest w każdym zabezpieczeniu określony zmienną w linii 3*.

Linie 5-21 definiują strukturę linii: 3 sekcje oraz wchodzące w ich skład 9 podsekcji. Pozostała część wydruku opisuje kolejno zabezpieczenia P1, P2 i P3, z których P2 jest jednosystemowe, a P1 i P3 wielosystemowe (linie 31, 51, 72). Uwzględniając strefę startową, podsekcja a1 jest łącznie w 7 strefach, dla których należy zdefiniować parametry czasowe. Dla wszystkich stref potrzebne są czasy wejścia. W przypadku lokalnych niezbędne są również opóźnienia, a dla zdalnych dodatkowo czasy wyjścia.

Przykładowo, druga strefa drugiego zabezpieczenia zdalnego $Z_2 2$ ma w podsekcji a1 czasy wejścia i wyjścia opisane rozkładami Erlanga, podczas gdy opóźnienie jest zdefiniowane rozkładem jednopunktowym (linie 39-40).

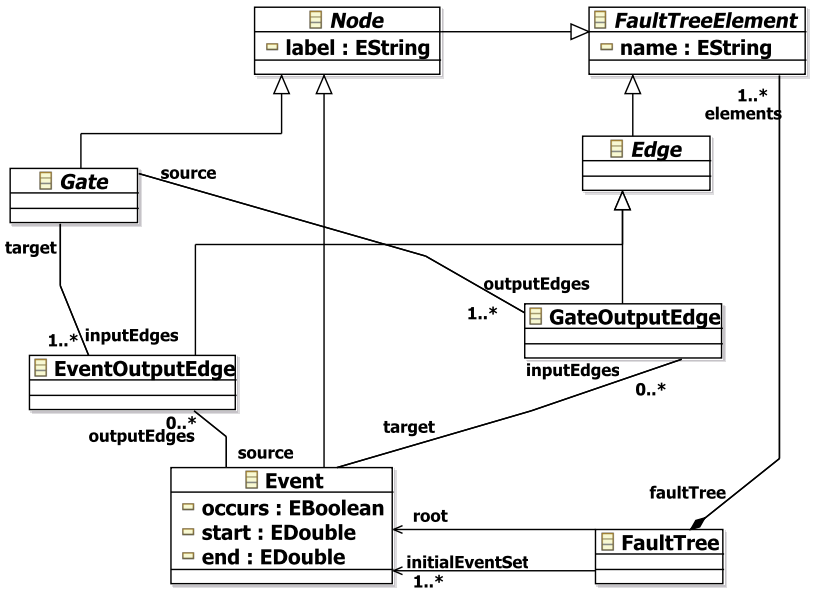
Wydruk 5.2: Model linii elektroenergetycznej wyrażony w składni konkretnej; P1, P3 zabezpieczenia wielosystemowe, P2 - jednosystemowe

```
1  PowerLine powerLine1 {
2  CircuitBreakerTypes {
3      CircuitBreakerType typeA OffTime LogNormal { Scale: 0 Shape: 0 }
4  }
5  Sections {
6      Section S3 {
7              SubSection a Factor 80
8              SubSection b Factor 20
9      }
10     Section S2 {
11             SubSection a Factor 50
12             SubSection b Factor 30
13             SubSection c Factor 20
14     }
15     Section S1 {
16             SubSection a1 Factor 20
17             SubSection a2 Factor 30
18             SubSection b Factor 30
19             SubSection c Factor 20
20     }
21 }
22 Protections {
23     Protection P3 {
24         CircuitBreaker CB3 Type typeA
25         ...
26         Zone Z_33{
27         ...
```

*konkretne wartości atrybutów zostały pominięte dla czytelności

```
28        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
29           ExitTime Erlang { K: 0 Lambda: 0 } DelayTime SP {R: 0} }
30        }
31        System Multi
32        StartingElement Absent
33     }
34     Protection P2 {
35        CircuitBreaker CB2 Type typeA
36        ...
37        Zone Z_22{
38        ...
39        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
40           ExitTime Erlang { K: 0 Lambda: 0 } DelayTime SP {R: 0} }
41        }
42        Zone Z_23{
43        ...
44        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
45           ExitTime Erlang { K: 0 Lambda: 0 } DelayTime SP {R: 0} }
46        }
47        Zone Z_2S {
48        ...
49        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 } }
50        }
51        System Single
52        StartingElement Present
53        StartingZone Z_2S
54     }
55     Protection P1 {
56        CircuitBreaker CB1 Type typeA
57        Zone Z11{
58        ...
59        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
60           DelayTime SP {R: 0}  }
61        }
62        Zone Z_12{
63        ...
64        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
65           DelayTime SP {R: 0}  }
66        }
67        Zone Z_13{
68          ...
69        SubSection a1 {EntryTime Erlang { K: 0 Lambda: 0 }
70           DelayTime SP {R: 0}  }
71        }
72        System Multi
73        StartingElement Absent
74  }}}
```

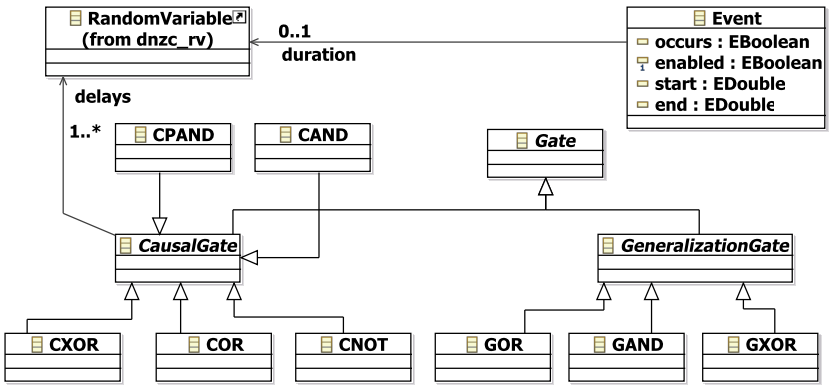
5.4 Obiektowa specyfikacja PDNZC



Rysunek 5.3: Diagram głównych klas języka PDNZC

Rozszerzone drzewo niezdatności, będące grafem dwudzielnym, jest reprezentowane przez obiekty klasy *FaultTree* przedstawionej razem z innymi klasami na rys. 5.3. Klasy *Node* i *Edge* stanowią abstrakcyjne elementy grafu: węzły i krawędzie. Na dwa typy węzłów składają się bramy, czyli obiekty klas pochodnych względem *Gate* oraz zdarzenia, obiekty klasy *Event*. Konwencja modelowania połączeń między węzłami jest następująca. Krawędziom wychodzącym ze zdarzeń i prowadzącym do bram odpowiadają obiekty klasy *EventOutputEdge*, z kolei krawędzie wychodzące z bram i prowadzące do zdarzeń to obiekty *GateOutputEdge*.

Zdarzenie wskazywane referencją *root* to korzeń drzewa oznaczający hazard. Natomiast zbiór zdarzeń w kolekcji *initialEventSet* to zdarzenia zachodzące w chwili początkowej.



Rysunek 5.4: Diagram klas bram uogólniających i przyczynowych

Jak wynika z drugiej części metamodelu przedstawionej na rys. 5.4, klasa *Gate* jest dalej specjalizowana przez *CausalGate* i *GeneralizationGate*. Ponieważ bramy przyczynowe mają

zmienne losowe oznaczające czasy odpalenia, do klasy *CausalGate* dodano kolekcję *delays* w celu przechowywania obiektów *RandomVariable*. Nazwy klas nawiązują do bram i składają się z dwu części. Pierwsza to litera G lub C symbolizująca typ bramy. Pozostała część odnosi się do funkcji logicznej realizowanej przez bramę.

Składnia konkretna dla PDNZC wykorzystana do rysowania modeli graficznych nie zostanie opisana w rozprawie. Omówienie zastosowanej techniki transformacji międzymodelowych do wizualizacji i edycji drzew można znaleźć w [100, 101].

5.5 Transformacja języka dziedzinowego w PDNZC

Bazując na modelu przedstawionym na wydruku 5.2 w niniejszym podrozdziale przedstawiono koncepcję oraz implementację dedukcyjnej metody generacji PDNZC opisującego hazard.

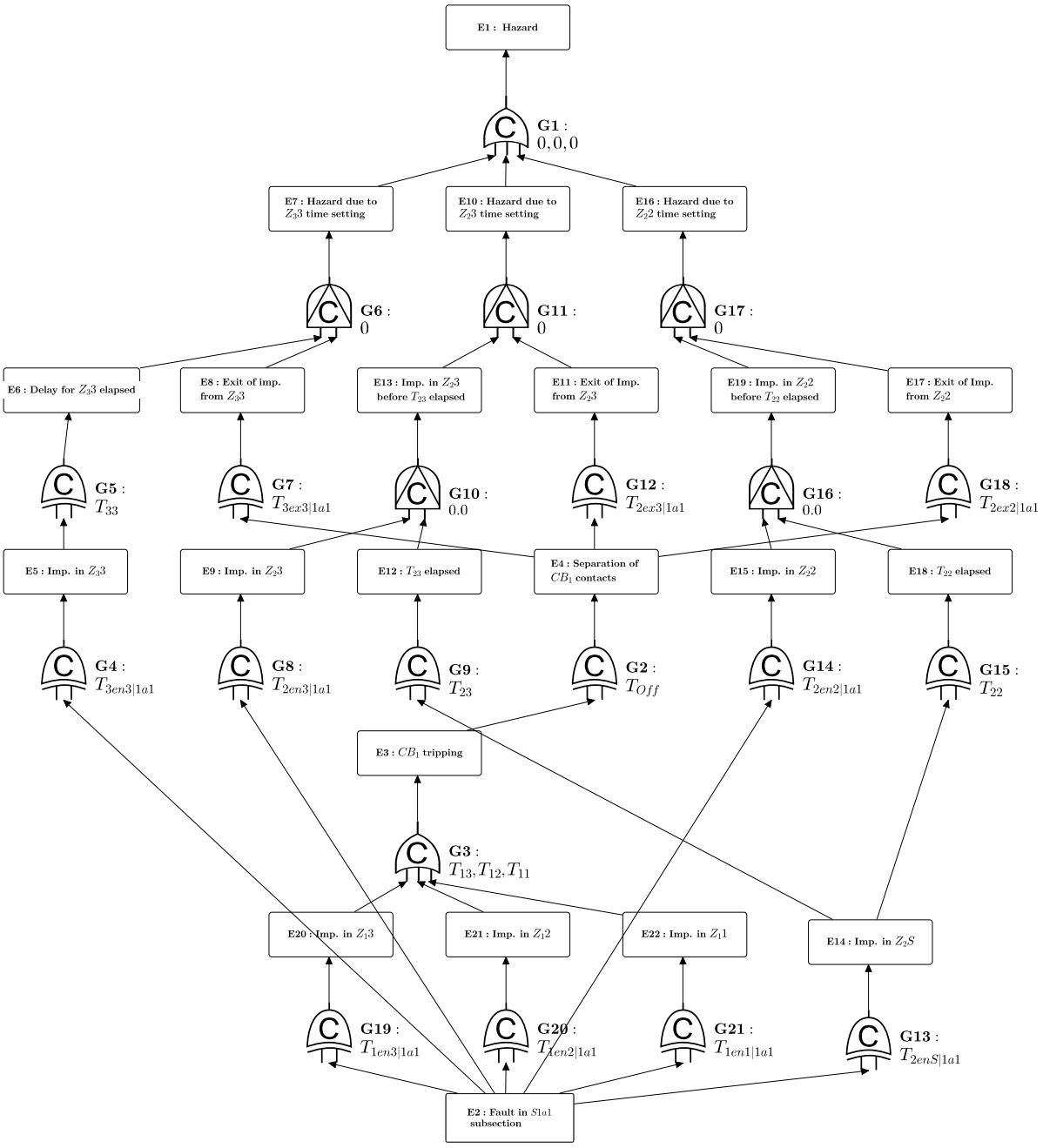
Z fragmentów kodów źródłowych usunięto instrukcje tworzenia krawędzi modelu. Nie pokazano również konstruktorów poszczególnych elementów oraz uproszczono tworzenie zmiennych losowych. Tam, gdzie to możliwe, nazwy zmiennych pasują do tych z rys. 5.5.

5.5.1 Zabezpieczenia zdalne

Jak stwierdzono w 5.1 hazard następuje, gdy nadmierny fragment linii zostaje wyłączony z transmisji, czyli, w odniesieniu do podsekcji a1, aktywowany zostaje wyłącznik CB_2 lub CB_3, podczas gdy CB_1 jest sprawny. To uzasadnia utworzenie elementów $E1$, $G1$ oraz $E2$ w modelu na rys. 5.5. Zdarzenie $E1$ oznaczające hazard staje się korzeniem drzewa i zachodzi jeśli jedna ze stref zdalnych spowoduje rozłączenie linii, stąd brama COR $G1$. Z kolei zdarzenie $E2$ symbolizuje awarię linii i staje się zdarzeniem początkowym. Na wydruku 5.3 zaprezentowano fragment głównego mapowania QVT generującego opisane elementy.

Wydruk 5.3: Główna reguła transformacji języka dziedzinowego w PDNZC

```
1  mapping PowerLine::toFaultTree(in subSection: SubSection): FaultTree {
2      var E1:= new Event("hazard");
3      var G1:= new COR();
4      var E2:= new Event(latexText("Fault in ") +
5              subSection.name + newLine() + latexText(" subsection"));
6      new GateOutputEdge(G1, E1);
7      initialEventSet += E2;
8      root:= E1;
9      ...
10     var subSectionProtections:= dp.objects()[SubSectionProtection]->
11         select(e | e.subSection = subSection)->
12         reject(e | e.isStartingSubSectionProtection())->
13         sortedBy(e|e.oclAsType(SubSectionProtection).zone.name);
14     ...
15 }
```



Rysunek 5.5: Drzewo niezdatności wygenerowane dla podsekcji a1 sekcji S1; P1, P3 zabezpieczenia wielosystemowe, P2 jednosystemowe z członem startowym

Konstrukcja *new GateOutputEdge* oraz *new EventOutputEdge* z parametrami: bramą i zdarzeniem (linia 6) dodają pomiędzy argumentami krawędź w modelu wynikowym. Wywołania konstrukcji pominięto w pozostałych wydrukach dla zwiększenia czytelności. Ponadto linie 10-13 powodują znalezienie kolekcji obiektów *SubSectionProtection* chroniących wskazaną argumentem *subSection* podsekcję. Dla każdego z nich będzie wywołana jedna z poniższych reguł.

Hazard wynika z błędnej nastawy przynajmniej jednej strefy zdalnych, co odpowiada wystąpieniu $E7$, $E10$ lub $E16$. Aby taka sytuacja zaszła w przypadku zabezpieczenia wielosystemowego, opóźnienie strefy musiało upłynąć przed wyjściem impedancji z jej charakterystyki. Taką sytuację można zamodelować bramą priorytetową $G6$, której lewe wejście prowadzi do zdarzenia upłynięcia opóźnienia $E6$, a prawe do wyjścia impedancji $E8$.

Jeśli czas związany z wyjściem awarii z charakterystyki impedancji upłynął, czyli rozpoczęło się $E8$, to znaczy, że separacja kontaktów wyłącznika lokalnego nastąpiła wcześniej o czas zdefiniowany przez zmienną $T_{3ex3|1a1}$, co daje podstawy do utworzenia bramy opóźniającej $G7$ oraz zdarzenia separacji kontaktów $E4$. To zdarzenie, będące częścią wspólną transformacji wszystkich zabezpieczeń zdalnych, jest argumentem nazwanym *contactSep* mapowań (procedur) przedstawionych na wydrukach 5.4 oraz 5.5.

Chwila startu drugiego zdarzenia wejściowego bramy priorytetowej $E6$ wynika z opóźnienia po po wejściu impedancji w strefę $Z_3$3. Stąd elementy $G5$ oraz $E5$. Aby zaszło $E5$, musi upłynąć czas wejścia impedancji w charakterystykę trzeciej strefy P3, czyli $T_{3en3|1a1}$, liczony względem chwili uszkodzenia, co opisano bramą $G4$ i połączeniem z $E2$.

W liniach 3-14 wydruku 5.4 tworzone są opisane bramy i zdarzenia. Całość fragmentu wynikowego mapowania wkomponowuje się pomiędzy argumenty mapowania *lineFault* ($E2$), *contactSep* ($E4$) oraz *topCOR* ($G1$) na rys. 5.5. W liniach 16-21 bramom przypisywane są odpowiednio nazwane parametry czasowe.

Wydruk 5.4: Translacja zdalnych zabezpieczeń wielosystemowych

```
1  mapping SubSectionProtection::remoteProtectionWoSe(in lineFault: Event,
2          in contactSep: Event, inout topCOR: COR) {
3      var partialHazardCPAND:= new CPAND();
4      var partialHazardEv:= new Event("Hazard due to " + self.zone.name
5          + " time setting");
6
7      var delayCXOR:= new CXOR();
8      var delayElapsedEv:= new Event("Delay for "+ self.zone.name + " elapsed");
9
10     var entryCXOR:= new CXOR();
11     var impdInProtectEv:= new Event("Imp. in " + self.zone.name);
12
13     var impdExitCXOR:= new CXOR();
14     var impdExitEv:= new Event("Exit of imp. from " + self.zone.name);
15
16     partialHazardCPAND.delays := Sequence {map noDelayRV()};
17     topCOR.delays += map noDelayRV();
```

```
18      entryCXOR.delays:= Sequence {self.map toEntryTime()};
19      impdExitCXOR.delays:= Sequence {self.map toExitTime()};
20      delayCXOR.delays := Sequence {new RandomVariable("T" +
21          self.zone.protection.protectionNo() + self.zone.zoneNo())};
22      ...
23  }
```

W przypadku zdalnych stref jednosystemowych $Z_2 3$ i $Z_2 2$ hazard zachodzi w następującej sytuacji. Zabezpieczenie jednosystemowe rozpoczyna sekwencyjne sprawdzanie kolejnych swoich stref w określonych chwilach liczonych względem wejścia impedancji w strefę startową. Stąd aby doszło do aktywacji zdalnego wyłącznika, w chwili analizy strefy przez zabezpieczenie (np. $E12$) impedancja awarii musi być widoczna w tej strefie. A zatem musiała w nią wejść ($E9$), ale nie zdążyła wyjść ($E11$). Kaskadowe połączenie bram CPAND $G11$ i $G10$ pozwala na weryfikację, czy sprawdzenie strefy (start $E12$ - $\tau(E12s)$) odbyło się w przedziale zdefiniowanym przez chwile startowe $E9$ i $E11$: $< \tau(E9s), \tau(E11s) >$.

Zdarzenie $E12$ zachodzi T_{23} jednostek czasu po pojawieniu się impedancji uszkodzenia w strefie startowej, co opisują $G9$, $E14$, $G13$ i $E2$. Analogicznie, wejście impedancji w strefę zdalną modelowane jest bramą $G8$ aktywowaną uszkodzeniem $E2$. Natomiast wyjście impedancji ze strefy zdalnej odbywa się dopiero po separacji kontaktów wyłącznika lokalnego, czyli $E4$.

Kod na wydruku 5.5 realizuje opisaną koncepcję. Po utworzeniu podstawowych elementów, tj. bram oraz ich zdarzeń wyjściowych, zwraca uwagę wywołanie mapowania elementu startowego w liniach 22-23. Taka konstrukcja jest niezbędna dla utworzenia tylko jednego, wspólnego dla wszystkich zabezpieczeń zdalnych, zbioru elementów w modelu wyjściowym opisującego wejście impedancji we wspólną strefę rozruchową.

Wydruk 5.5: Translacja zdalnych zabezpieczeń jednosystemowych

```
1  mapping SubSectionProtection::remoteProtectionWSe(in lineFault: Event,
2      in contactSep: Event, inout topCOR: COR)
3  {
4      var partialHazardCPAND:= new CPAND();
5      var partialHazardEv:= new Event("Hazard due to " + self.zone.name +
6          "time setting");
7
8      var impdExitCXOR:= new CXOR();
9      var impdExitEv:= new Event("Exit of Imp. from " + self.zone.name);
10
11     var pand:= new CPAND();
12     var befElapsingEv:= new Event("Imp. in "+ self.zone.name + " before " + "T" +
13         self.zone.protection.protectionNo() + self.zone.zoneNo() + "elapsed");
14
15     var entryCXOR:= new CXOR();
16     var impdInProtectEv:= new Event("Imp. in " + self.zone.name);
17
18     var delayCXOR:= new CXOR();
19     var trElapsedEv:= new Event("T" + self.zone.protection.protectionNo()
```

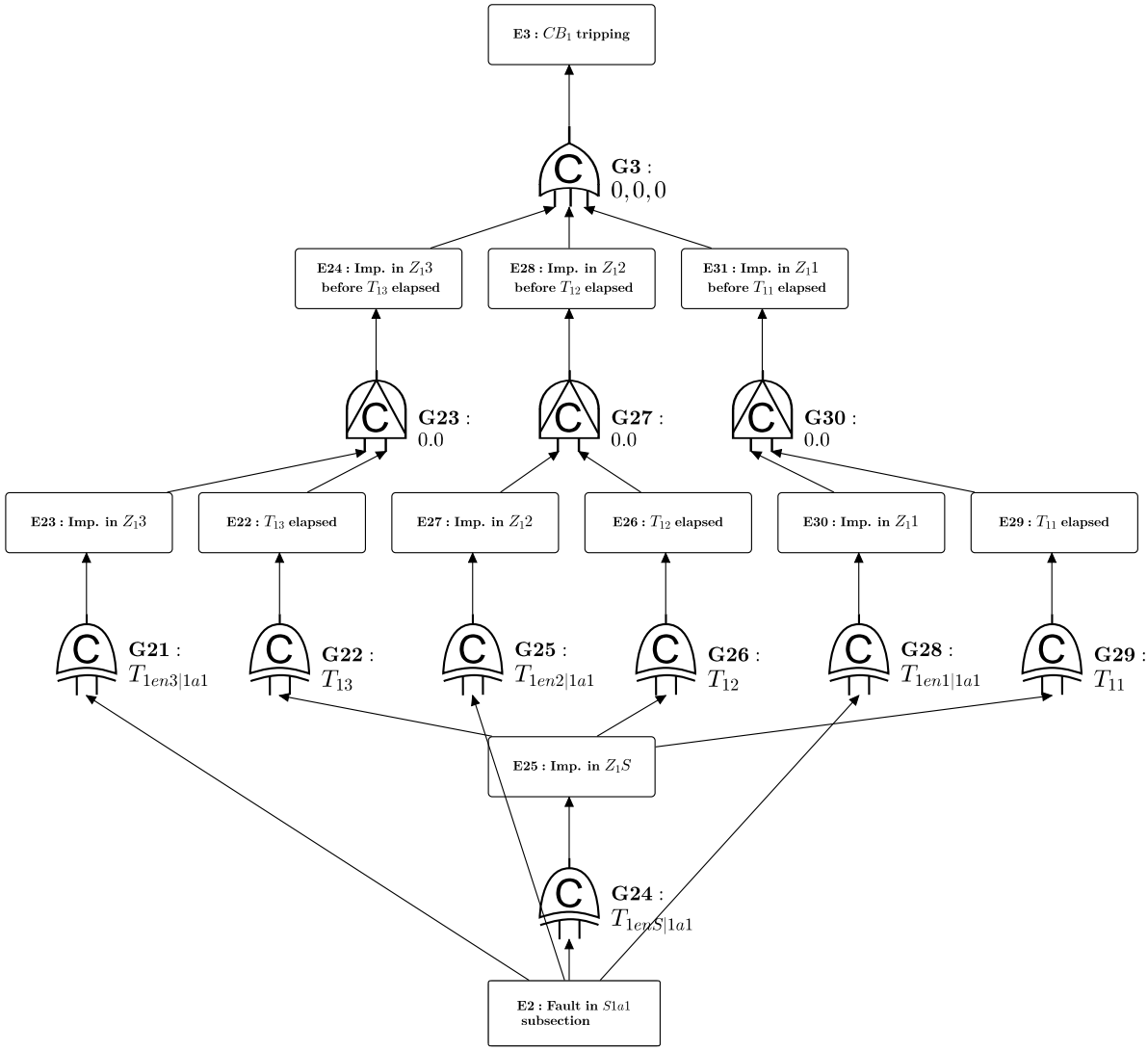
```
20        + self.zone.zoneNo() + " elapsed");
21
22    new EventOutputEdge(self.subSection.map StartingElement(self.zone.protection,
23        lineFault), delayCXOR);
24
25    partialHazardCPAND.delays := Sequence {map noDelayRV()};
26    topCOR.delays += map noDelayRV();
27    entryCXOR.delays:= Sequence {self.map toEntryTime()};
28    impdExitCXOR.delays:= Sequence {self.map toExitTime()};
29    delayCXOR.delays := Sequence {new RandomVariable("T" +
30        self.zone.protection.protectionNo() + self.zone.zoneNo())};
31    ...
32 }
33
34 mapping SubSection::StartingElement(in protection: Protection,
35    in lineFault: Event): Event  {
36
37 init {
38    result := new Event(latexText("Imp. in ") + stroke("Z_" +
39        protection.protectionNo() + "S"));
40  }
41    var faultInZoneS := new CXOR();
42    faultInZoneS.delays := Sequence{protection.startingZone.
43        subSectionProtections->select(e |e.subSection = self)->
44        first().map toSeEntryTime()};
45    ...
46 }
```

5.5.2 Zabezpieczenia lokalne

To aktywacja wyłącznika ($E3$) jest zdarzeniem bezpośrednio poprzedzającym separację jego kontaktów ($E4$). Proces rozłączania uzupełniany jest bramą $G2$ i czasem T_{Off}. Każde z zabezpieczeń lokalnych może dokonać rozłączenia, dlatego poniżej $E3$ generowana jest brama COR ($G3$). Transformacja została umieszczona na wydruku 5.6.

Wydruk 5.6: Mapowanie wyłącznika lokalnego

```
1     // cb - wyłącznik lokalny
2     var E4:= new Event(latexText("Separation of ") + newLine() +
3         stroke(cb.name) + latexText(" contacts"));
4     var G2:= new CXOR();
5     var E3:= new Event(stroke(cb.name) + latexText(" tripping"));
6     var G3:= new COR();
7
8     new GateOutputEdge(G2, E4);
9     new EventOutputEdge(E3, G2);
10    new GateOutputEdge(G3, E3);
```



Rysunek 5.6: Dekompozycja zdarzenia $E3$, gdy P1 jest zabezpieczeniem jednosystemowym z członem startowym

Aby doszło do aktywacji wyłącznika przez wielosystemowe zabezpieczenie lokalne (P1 na rys. 5.5) musi upłynąć czas opóźnienia w przynajmniej jednej strefie. Czas ten jest liczony od momentu wejścia impedancji w tę strefę. W myśl powyższego, algorytm transformacji generuje bramy $G19$-$G21$ oraz zdarzenia $E20$-$E22$ dla każdej strefy lokalnej, a także przypisuje odpowiednie opóźnienia do bramy COR $G3$ (wydruk 5.7).

Wydruk 5.7: Translacja lokalnych zabezpieczeń wielosystemowych

```
1  mapping SubSectionProtection::localProtectionWoSe(in lineFault: Event,
2          inout cbTrippingCOR: COr) {
3     var impdInProtect:= new Event("Imp. in " + self.zone.name);
4     var entryCXOR:= new CXOR();
5
6     entryCXOR.delays:= Sequence {self.map toEntryTime()};
7     cbTrippingCOR.delays += new RandomVariable("T" +
```

```
8         self.zone.protection.protectionNo() + self.zone.zoneNo());
9           ...
10  }
```

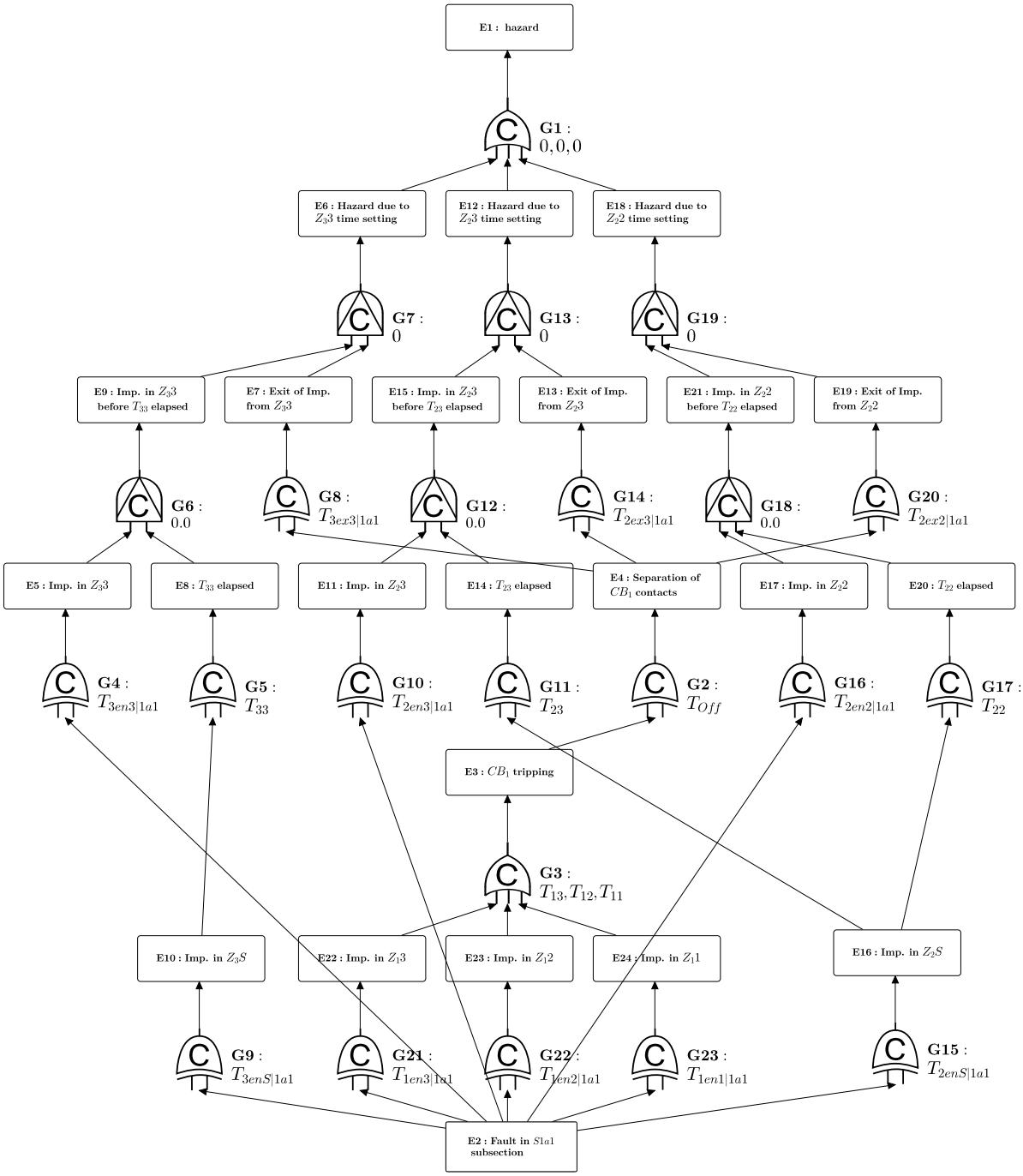
Wariant linii z zabezpieczeniem jednosystemowym z członem rozruchowym jako lokalnym (rys. 5.6) implikuje, że do rozłączenia dochodzi bezpośrednio w chwili sprawdzenia danej strefy przez zabezpieczenie pod warunkiem, że w tej strefie rozpoznawane jest uszkodzenie. To tłumaczy zerowe opóźnienia przy bramie $G3$ oraz przy bramach priorytetowych, np. $G23$. Podobnie jak wcześniej, impedancja jest widoczna po czasie wejścia modelowanym przez bramę $G21$. Natomiast sprawdzanie danej strefy (np. $E22$) odbywa się po opóźnieniu ($G22$) względem znalezienia impedancji w strefie startowej: $E25$ i $G24$.

Wydruk 5.8: Translacja lokalnych zabezpieczeń jednosystemowych

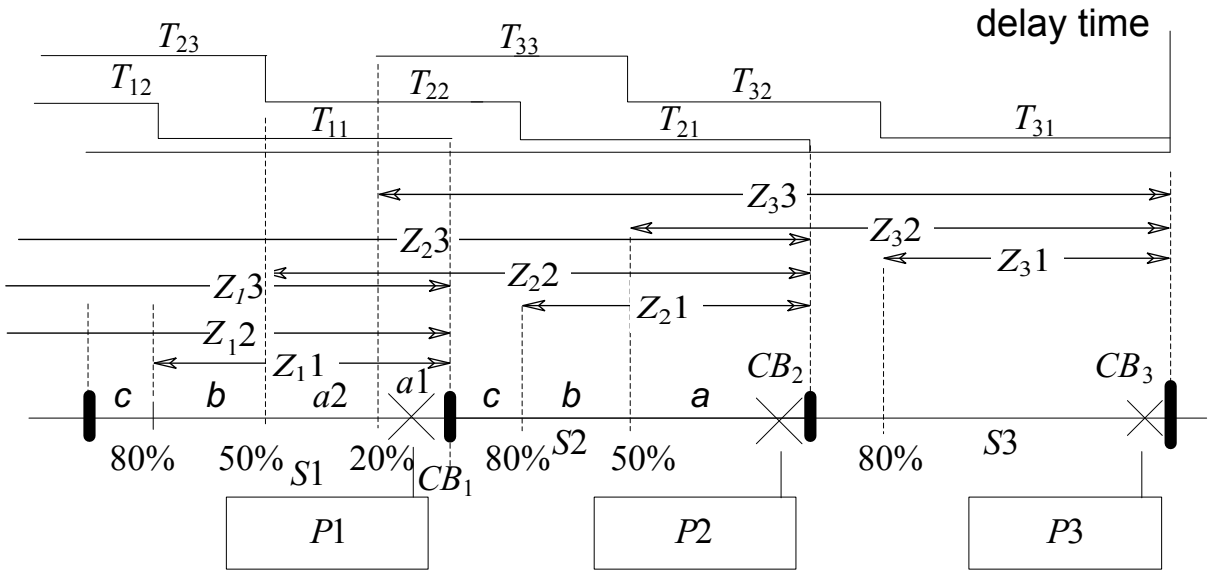
```
1
2     var impInZoneBeforeDelay:= new Event(latexText("Imp. in ") +
3         stroke(self.zone.name)+ newLine() + latexText(" before ") +
4         stroke("T_{" + self.zone. protection.protectionNo() +
5         self.zone.zoneNo() + "}") + latexText(" elapsed"));
6     var order:= new CPAND();
7
8     var delay:= new CXOR();
9     var impdInZone:= new Event(latexText("Imp. in ") + stroke(self.zone.name));
10
11    var delayElapsed:= new Event(stroke("T_{" +
12        self.zone.protection.protectionNo() + self.zone.zoneNo() + "}") +
13        latexText(" elapsed"));
14    var entry:= new CXOR();
15
16    new EventOutputEdge(self.subSection.map
17        StartingElement(self.zone.protection, lineFault), delay);
18
19    cbTripping.delays += new RandomVariable("0");
20    entry.delays:= Sequence {self.map toEntryTime()};
21    delay.delays += new RandomVariable(stroke("T_{" + self.zone.protection.
22        protectionNo() + self.zone.zoneNo()+"}"));
23  ...
24  }
```

5.6 Inne zastosowania translatora

Dzięki automatycznej generacji PDNZC mogą być stosowane nie tylko do ustawiania czasów opóźnień w istniejących liniach elektroenergetycznych, ale również do szacowania wpływu modernizacji całego zabezpieczenia na prawdopodobieństwo hazardu. Przykładowo, wpływ modernizacji zabezpieczenia P3 na prawdopodobieństwo hazardu można znaleźć porównując model z rys. 5.7 z modelem przedstawionym na rys. 5.5.



Rysunek 5.7: Drzewo niezdatności dla podsekcji a1 sekcji S1; P1 zabezpieczenie wielosystemowe, P2, P3 jednosystemowe z członem startowym



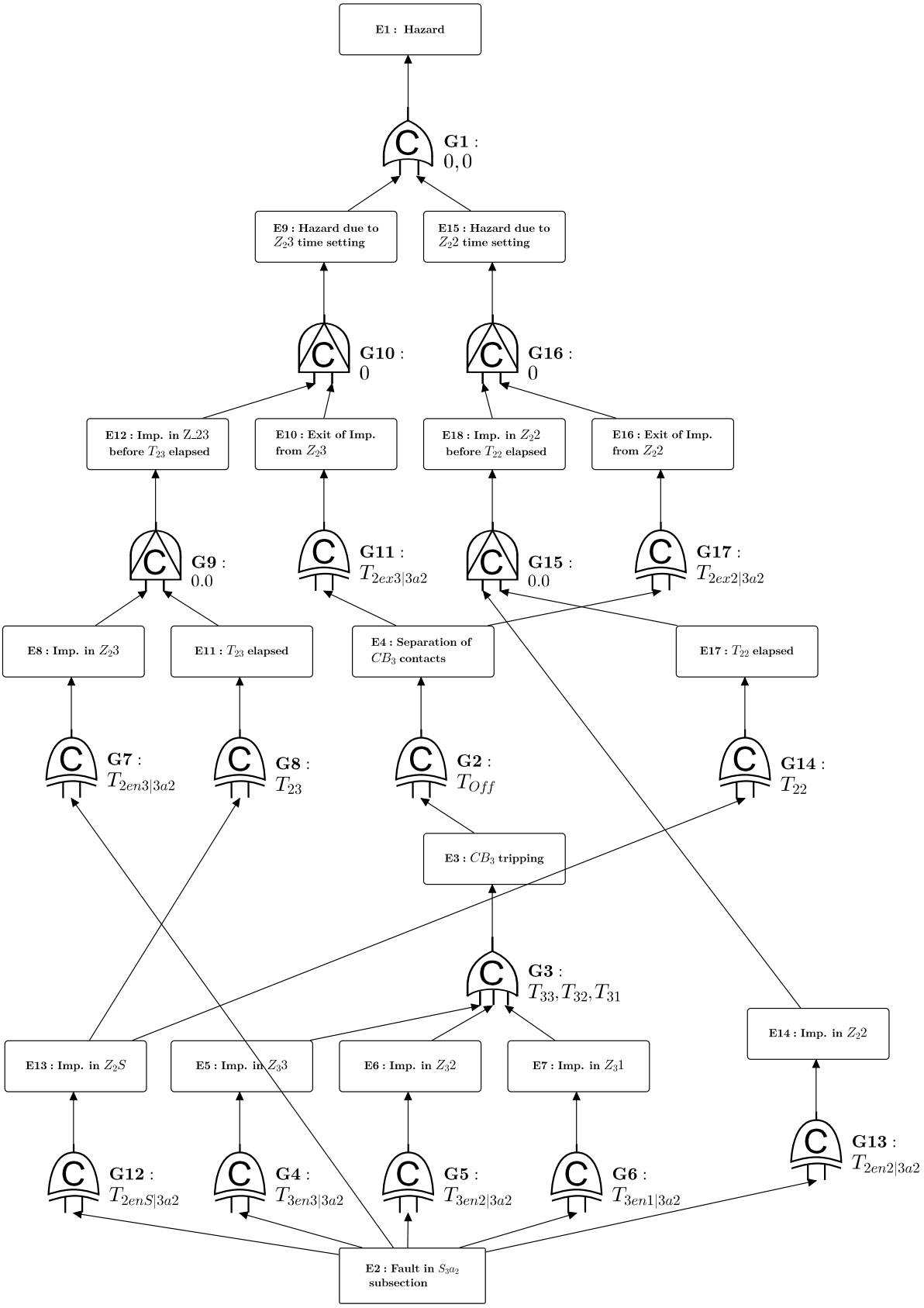
Rysunek 5.8: Linia elektroenergetyczna z zabezpieczeniami umieszczonymi na końcach sekcji

Inne zastosowanie transformacji to porównanie bezpieczeństwa sytuacjach, kiedy zabezpieczenia umieszczone są na początkach lub na końcach sekcji: rys. 5.1 i rys. 5.8. Jak wynika wygenerowanego modelu niezawodności (rys. 5.9) na prawdopodobieństwo hazardu w podsekcji a2 sekcji S3, gdy zabezpieczenia umieszczone są na końcach sekcji mają wpływ jedynie zabezpieczenia P2 i P3.

W opisanych fragmentach kodu założono, że generowane czasy powinny mieć postać zmiennych losowych, co jest jedną z dwu możliwości transformacji. W istocie oprogramowanie można również skonfigurować do generacji NDNZC, w których czas wyrażony jest przedziałami, aby następnie przeprowadzić analizę bezpieczeństwa metodami nieprobabilistycznymi, podobnie jak w [115].

5.7 Omówienie wyników

Szeroki wachlarz modeli zaprezentowany w rozdziałach 4 oraz 5 pozwala uznać, że język PDNZC posiada wysoką i praktycznie użyteczną moc ekspresji. Ponadto analizę niezawodności z wykorzystaniem języka można wesprzeć automatyczną syntezą drzew z jednej strony oraz generacją symulatora z drugiej. W kolejnej części rozprawy, składającej się z rozdziałów 6 oraz 7, omówiono dodatkowe rozszerzenia drzew niezdatności, które pozwolą na zastosowanie w systemach, gdzie PDNZC okazują się niewystarczające.



Rysunek 5.9: Drzewo niezdatności dla podsekcji a2 sekcji S3, gdy zabezpieczenia umieszczone są na końcach linii; P1, P2 zabezpieczenia jednosystemowe, P3 wielosystemowe

Rozdział 6

Etap 2: Włączenie elementów sieci Petriego - Grafy Niezdatności

W rozdziale zostaną zaprezentowane Grafy Niezdatności (GN) będące hybrydą opisanych w rozdziale 4 Probabilistycznych Drzew Niezdatności z Zależnościami Czasowymi oraz Sieci Petriego Wysokiego Poziomu - *ang. High Level Petri Nets* (HLPN) [66]. Problemami adresowanymi przez nowy język są między innymi wyrażenie synchronizacji procesów z zależnościami czasowymi oraz rozstrzyganie konfliktów w dostępie do zasobów współdzielonych. W efekcie GN umożliwią opis przydziału konserwatora, alokacji elementu zapasowego czy zjawisk o naturze systemów kolejkowych.

Opis języka składa się z dwóch części. Pierwszą poświęcono definicji składni i semantyki GN oraz uzupełnieniu brakujących polityk napraw, których nie udało się zamodelować w rozdziale 4 przy pomocy PDNZC. Zwiększenie mocy opisowej drzew niezdatności ma swoje źródło we włączeniu następujących konstrukcji językowych:

- *zdarzeń parametrycznych* przy jednoczesnym zachowaniu zdarzeń dotychczasowych, nazywanych w GN *prostymi* (podrozdział 6.1.1),
- *przejścia natychmiastowego* wzorowanego na HLPN, ale dostosowanego do zdarzeń prostych i parametrycznych (podrozdział 6.1.2),
- *bramy opóźniającej* oraz *generatora zdarzeń* (opisano również w podrozdziale 6.1.2).

W drugiej części (podrozdział 6.3 oraz dodatek A) przedstawiono reguły translacji wybranych bram uogólniających, przyczynowych oraz wariantów przejść w kod wykonywalny umożliwiający symulację zaprojektowanego modelu. Omówiono również przebiegi czasowe wybranych konstrukcji uzyskane symulatorem wygenerowanym z wybranych podmodeli.

Zastosowanie GN do rzeczywistego systemu w dziedzinie transportu publicznego zostanie omówione w rozdziale 7.

6.1 Podstawy Grafów Niezdatności

W modelach PDNZC zastosowanych do systemów z redundancją duża część drzewa jest powtarzana. Przykładem jest fragment $G2, E1, G7, G8, E6$ na rys. 4.4 odpowiadający procesom uszkodzenia oraz odnowy pierwszego z dwu bloków pamięci. Ponieważ obydwa komponenty są tego samego typu, ich parametry niezawodnościowe są takimi samymi zmiennymi losowymi i fragment dla drugiego bloku pamięci jest identyczny z dokładnością do nazw elementów. Poniżej opisane rozszerzenia pozwalają jednak nie tylko na uproszczenie drzewa, ale także na modelowanie problemów nieosiągalnych dla PDNZC.

6.1.1 Zdarzenia proste i parametryczne

Rysunek 6.1: Zdarzenia w GN: x - zdarzenia proste, y - zdarzenie z parametrem p [94]

Aby uniknąć wielokrotnego kopiowania tych samych gałęzi drzewa do GN wprowadzono nową grupę zdarzeń - *zdarzenia parametryczne*, których symbole graficzne zostały porównane z dotychczas stosowanymi - *zdarzeniami prostymi* - na rys. 6.1.

Każdemu zajściu zdarzenia parametrycznego przypisuje się dokładnie jedną, unikatową wartość z dziedziny jego parametru będącego często zbiorem liczb naturalnych. Zajścia zdarzenia y należy stąd charakteryzować określoną wartością parametru p. Niech:

$$occur_{PAR}(y, ys, yk, p)$$

będzie predykatem oznaczającym, że zdarzenie y z wartością parametru p rozpoczęło się w momencie ys, a zakończyło w chwili yk. Równolegle może mieć miejsce wiele tych samych zdarzeń z różnymi wartościami ich parametrów.

Zdarzenia proste x nie posiadają parametrów i są nierozróżnialne. Czwartym argumentem predykatu $occur_{PR}$ będzie kolejny numer trwającego w chwili xs zdarzenia x.

$$occur_{PR}(x, xs, xk, n)$$

Dla każdego zdarzenia prostego x prawdą jest, że:

$$occur_{PR}(x, 0, +\infty, 0)$$

Ponadto jeśli w pewnej chwili ma miejsce zdarzenie n-te, to w tej chwili zachodzi również zdarzenie z indeksem $n-1$:

$$\forall n \geqslant 1 : occur_{PR}(x, s1, k1, n) \rightarrow \exists s2 \leqslant s1 \wedge k2 \geqslant s1 : occur_{PR}(x, s2, k2, n-1)$$

W GN istnieją dwa zbiory zdarzeń początkowych: IE_{PR} oraz IE_{PAR} odpowiednio dla zdarzeń prostych i parametrycznych. W obydwu przypadkach rozpoczęcie musi nastąpić w początkowej chwili analizy.

$$\langle x, p\rangle \in IE_{PAR} \leftrightarrow \exists xk : occur_{PAR}(x, 0, xk, p)$$
$$\langle x, n\rangle \in IE_{PR} \leftrightarrow \exists xk : occur_{PR}(x, 0, xk, n)$$

6.1.2 Semantyka bram i przejść

Z myślą o wielokrotnych zdarzeniach prostych oraz nowej grupie zdarzeń parametrycznych dokonano formalnej definicji semantyki najpopularniejszych w drzewach niezdatności bram, ale także nowych konstrukcji: bramy opóźniającej oraz generatora. Podrozdział kończy formalizacja wprowadzonego również do GN przejścia.

Brama GAND

Niech predykat *overlap* wykorzystywany w definicjach rodziny bram AND definiuje, czy dwa odcinki czasu $\langle as, ak\rangle$ oraz $\langle bs, bk\rangle$ zachodzą na siebie:

$$overlap(as, ak, bs, bk) \leftrightarrow (ak \geqslant bs \wedge ak \leqslant bk) \vee (bk \geqslant as \wedge bk \leqslant ak)$$

Niech x i y będą zdarzeniami wejściowymi bramy GAND a z zdarzeniem wyjściowym.

Jeśli wszystkie zdarzenia są proste, to jeśli zaszło zarówno n-te zdarzenie x oraz n-te zdarzenie y oraz odcinki ich występowania pokrywają się, to zachodzi również n-te zdarzenie wyjściowe bramy GAND. Chwila jego startu zdefiniowana jest późniejszą chwilą startu spośród n-tych zdarzeń x i y, a moment zakończenia to wcześniejsza chwila ich zakończeń.

$$occur_{PR}(x, xs, xk, n) \wedge occur_{PR}(y, ys, yk, n) \wedge overlap(xs, xk, ys, yk) \leftrightarrow$$
$$occur_{PR}(z, max(xs, ys), min(xk, yk), n)$$

Analogicznie, równoległe zajście zdarzeń x i y z parametrem p, oznacza, że w tym czasie zachodzi zdarzenie z z tym samym parametrem.

$$occur_{PAR}(x, xs, xk, p) \wedge occur_{PAR}(y, ys, yk, p) \wedge overlap(xs, xk, ys, yk) \leftrightarrow$$
$$occur_{PAR}(z, max(xs, ys), min(xk, yk), p)$$

Gdyby x było zdarzeniem prostym a y parametrycznym:

$$occur_{PR}(x, xs, xk, n) \wedge occur_{PAR}(y, ys, yk, p) \wedge overlap(xs, xk, ys, yk) \wedge$$
$$count(z, max(xs, ys)) < n \leftrightarrow occur_{PAR}(z, max(xs, ys), min(xk, yk), p)$$

Wartość funkcji *count* to liczba zdarzeń z rozpoczętych przed chwilą t i kończących się nie wcześniej niż t ($\mathcal{N}$ - zbiór liczb naturalnych z zerem):

$$count(z, t) = inf\{n \in \mathcal{N} : (\forall n1)(\forall zs \leqslant t)(\forall zk \geqslant t) : occur_{PR}(z, zs, zk, n1) \wedge n1 \leqslant n\}$$

Brama *GOR*

Wystarczy, aby zachodziło jedno zdarzenie wejściowe bramy GOR, a zdarzenie wyjściowe zachodzi w tym samym czasie. W przypadku, gdy wszystkie zdarzenia są proste własność jest prawdziwa dla każdego kolejnego zdarzenia n. Dla zdarzeń parametrycznych, na wyjściu znajdą się te same wartości parametru p. Niech x_i będzie zdarzeniem wejściowym bramy GOR:

$$\exists i : occur_{PR}(x_i, xs, xk, n) \leftrightarrow occur_{PR}(z, xs, xk, n)$$

$$\exists i : occur_{PAR}(x_i, xs, xk, p) \leftrightarrow occur_{PAR}(z, xs, xk, p)$$

Semantyka bramy GOR i COR, gdy na wejściach są zarówno zdarzenia proste jak i parametryczne, pozostaje niezdefiniowana.

Brama CAND

Jeśli rozpoczęte zostaną obydwa zdarzenia wejściowe, to moment rozpoczęcia zdarzenia wyjściowego określony jest wartością:

$$tg = max(xs, ys) + R(d)$$

Uwzględniając wymagane przez CAND i opisane przy GAND pokrywanie zdarzeń:

$$occur_{PR}(x, xs, xk, n) \wedge occur_{PR}(y, ys, yk, n) \wedge overlap(xs, xk, ys, yk) \rightarrow$$
$$\exists zk : occur_{PR}(z, tg, zk, count(z, tg) + 1)$$

Gdy x, y i z są parametryczne:

$$occur_{PAR}(x, xs, xk, p) \wedge occur_{PAR}(y, ys, yk, p) \wedge overlap(xs, xk, ys, yk) \rightarrow$$
$$\exists zk : occur_{PAR}(z, tg, zk, p)$$

Predykaty *overlap* zdecydowano się dołączyć w definicjach ze względu na powtarzalne oraz wielokrotne zajścia zdarzeń GN. Należy zauważyć, że w poprzedniej wersji języka NDNZC takiego wymagania nie było [117].

Brama Opóźniająca

Modelowanie opóźnienia, czyli rozpoczęcia zdarzenia późniejszego określony czas po starcie zdarzenia wcześniejszego, odbywało się w PDNZC przy pomocy bramy CXOR z jednym wejściem, np. $G12$ z rys. 5.9. Zgodnie z definicją operatora sumy rozłącznej, jest to rozwiązanie prawidłowe, ale można je uprościć. Powszechność takich relacji czasowych jest kolejną przyczyną, dla której w GN wprowadzono *bramę opóźniającą* o następującej semantyce.

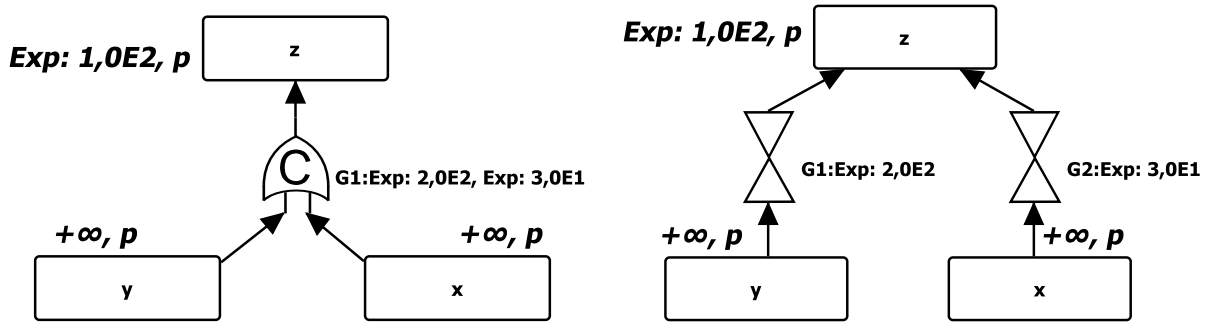
Jeśli R(d) jest opóźnieniem, to parametryczne zdarzenie wyjściowe z zachodzi R(d) po rozpoczęciu parametrycznego x. Obydwa zdarzenia zachodzą z tym samym parametrem.

$$occur_{PAR}(z, zs, zk, p) \rightarrow \exists xs \leqslant zs + R(d), \exists xk : occur_{PAR}(x, xs, xk, p)$$

Dla dwóch zdarzeń prostych połączonych bramą opóźniającą, z każdym startem x następuje rozpoczęcie kolejnego zdarzenia z:

$$occur_{PR}(x, xs, xk, n) \rightarrow \exists zk : occur_{PR}(z, xs + R(d), zk, count(z, xs + R(d_x)) + 1)$$

Brama COR



(a) Oryginalny fragment modelu

(b) COR, gdy y nie zajdzie przed z lub x nie zajdzie przed z

Rysunek 6.2: Interpretacja bramy COR

Gdy połączone bramą COR, to zdarzenie x lub y może rozpocząć z, co przedstawiono na rys. 6.2a. Jednak które z nich tego dokona zostaje rozstrzygnięte dopiero po porównaniu chwil ich startu i realizacji zmiennych losowych przypisanych do wejść bramy.

W prostszej sytuacji, gdy drugie ze zdarzeń (x lub y) nie zostanie rozpoczęte przed rozpoczęciem z przez pierwsze (odpowiednio y lub x), brama COR upraszcza się do modelu przedstawiono na rys. 6.2. W takiej sytuacji pojawia się jedynie opóźnienie pomiędzy zdarzeniem wejściowym i wyjściowym, co opisano dla zdarzeń prostych i parametrycznych poniższymi formułami.

$$\begin{gathered} occur_{PR}(x, xs, xk, n) \wedge \neg\exists ys \in \langle xs, xs + R(d_x)\rangle : occur_{PR}(y, ys, yk, n) \rightarrow \\ \exists zk : occur_{PR}(z, xs + R(d_x), zk, n) \\ occur_{PR}(y, ys, yk, n) \wedge \neg\exists xs \in < ys, ys + R(d_y) > : occur_{PR}(x, xs, xk, n) \rightarrow \\ \exists zk : occur_{PR}(z, xs + R(d_y), zk, n) \end{gathered}$$

$$\begin{gathered} occur_{PAR}(x, xs, xk, p) \wedge \neg\exists ys \in < xs, xs + R(d1) > : occur_{PAR}(y, ys, yk, p) \rightarrow \\ \exists zk : occur_{PAR}(z, xs + R(d1), zk, p) \\ occur_{PAR}(y, ys, yk, p) \wedge \neg\exists xs \in < ys, ys + R(d2) > : occur_{PAR}(x, xs, xk, p) \rightarrow \\ \exists zk : occur_{PAR}(z, xs + R(d2), zk, p) \end{gathered}$$

Jeśli jednak dojdzie do startu drugiego zdarzenia przed rozpoczęciem wyjściowego przez pierwsze, to dochodzi do *efektu minimalizacji* mający dwojakie konsekwencje. Po pierwsze, zdarzenie

wyjściowe rozpoczyna się pod wpływem pierwszego z dwu zdarzeń. Po drugie, późniejsze zdarzenie nie powoduje już startu z, co jest szczególnie istotne, gdy z ma przypisany czas trwania.

$$(occur_{PR}(x, xs, xk, n) \wedge \exists ys \in < xs, xs + R(d_x) > : occur_{PR}(y, ys, yk, n)) \vee$$
$$(occur_{PR}(y, ys, yk, n) \wedge \exists xs \in < ys, ys + R(d_y) > : occur_{PR}(x, xs, xk, n)) \rightarrow$$
$$\exists zk : occur_{PR}(z, min(xs + R(d_x), ys + R(d_y)), zk, n)$$

$$(occur_{PAR}(x, xs, xk, p) \wedge \exists ys \in < xs, xs + R(d_x) > : occur_{PAR}(y, ys, yk, p)) \vee$$
$$(occur_{PAR}(y, ys, yk, p) \wedge \exists xs \in < ys, ys + R(d_y) > : occur_{PAR}(x, xs, xk, p)) \rightarrow$$
$$\exists zk : occur_{PAR}(z, min(xs + R(d_x), ys + R(d_y)), zk, p)$$

Warto zauważyć, że *efekt minimalizacji* ograniczony jest do startem zdarzenia wyjściowego. To powoduje, że jeśli drugie ze zdarzeń wystąpi po z, to spowoduje ponowne jego wystartowanie, jeśli z zdąży się do tego czasu zakończyć.

Brama CPAND

Jeśli równolegle trwają n-te zdarzenia x i y, ale y wystartowało po x, to po upływie czasu wynikającego z realizacji zmiennej losowej d generowane jest kolejne zdarzenie z:

$$occur_{PR}(x, xs, xk, n) \wedge \exists ys \in (xs, xk > : occur_{PR}(y, ys, yk, n) \rightarrow$$
$$\exists zk : occur_{PR}(z, zs + R(d), zk, count(z, ys + R(d)) + 1)$$

Brama CPAND w postaci parametrycznej rozpoczyna zdarzenie wyjściowe kiedy zachodzą zarówno x, jak i y z tym samym parametrem, ale x rozpocznie się przed y:

$$occur_{PAR}(x, xs, xk, p) \wedge \exists ys \in (xs, xk > : occur_{PAR}(y, ys, yk, p) \rightarrow$$
$$\exists zs \leqslant ys + R(d), zk \geqslant ys + R(d) : occur_{PAR}(z, zs, zk, p)$$

Brama CNOT i czas trwania zdarzeń

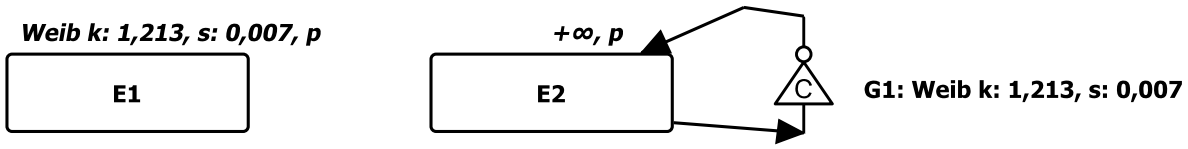
Podczas gdy pozostałe bramy przyczynowe rozpoczynają zdarzenie, CNOT posiada unikatową własność pozwalającą na zakończenie zdarzenia wyjściowego po upływie określonego przypisaną zmienna losową czasu. W przypadku zdarzeń prostych oznacza to, że ostatnie z nich, jeśli istnieje, kończy się w chwili upływu opóźnienia:

$$occur_{PR}(x, xs, xk, p) \rightarrow$$
$$(count(z, xs + R(d)) > 0 \rightarrow \exists zk = xs + R(d) : occur_{PR}(z, zs, zk, count(z, xs + R(d))))$$

Gdy natomiast na wyjściu bramy CNOT jest zdarzenie z parametrem z, to nie jest prawdą, że istnieje taki przedział zachodzenia z z parametrem p, w którym zawierałaby się chwila upływu opóźnienia CNOT:

$$occur_{PAR}(x, xs, xk, p) \rightarrow \neg(\exists zs \leqslant xs + R(d) \wedge zk \geqslant xs + R(d) : occur_{PAR}(z, zs, zk, p)))$$

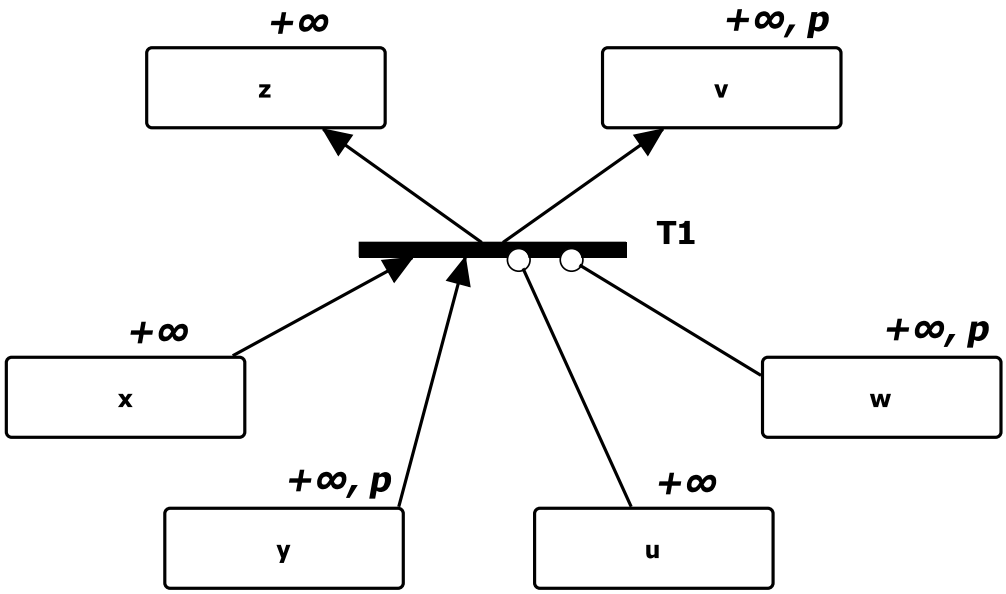
Jeśli zdarzenie jest na wyjściu bramy przyczynowej lub opisanego dalej przejścia, to trwa nieskończoność lub ma przypisany czas trwania będący stałą bądź zmienną losową. Zdarzenie z określonym czasem trwania może być wyrażone jako zdarzenie bez czasu trwania oraz przylegającą brama CNOT według schematu przedstawionego na rys. 6.3.



Rysunek 6.3: Czasu trwania $E1$ wyrażony jako zdarzenie bez czasu $E2$ i przylegająca brama $G1$

Czas trwania zdarzenia będącego na wyjściu bramy uogólniającej zdefiniowany jest natomiast przez tę bramę. Zdarzenie może być na wyjściu maksymalnie jednej bramy uogólniającej. Żadne zdarzenie nie może być z jednej strony na wyjściu bramy uogólniającej a z drugiej strony na wyjściu bramy przyczynowej lub przejścia.

Przejście wzorowane na elemencie sieci Petriego



Rysunek 6.4: Przejście $T1$ ze zdarzeniami wejściowymi: prostym x, parametrycznym y i prostym hamującym w oraz wyjściowymi: prostym z i parametrycznym v

Jeśli warunki odpalenia przejścia są spełnione, to następuje natychmiastowe i atomowe zakończenie zdarzeń wejściowych i rozpoczęcie zdarzeń wyjściowych. W kontekście przejścia istnieją 4 typy zdarzeń wejściowych i 2 typy wyjściowych, co zobrazowano na rys. 6.4.

Aby przejście $T1$ zostało odpalone musi zachodzić przynajmniej jedno zdarzenie x oraz y z pewnym parametrem p. Jeśli na wejściu obecne by były inne zdarzenia parametryczne, to

również musiałyby zachodzić z parametrem p. Jednocześnie, aby $T1$ wystartowało nie może zachodzić u, gdyż jest połączone z $T1$ łukiem hamującym zakończonym okręgiem. Ponadto nie może także zachodzić w z parametrem p. W momencie odpalenia przejścia startowane są kolejny raz wszystkie zdarzenia proste, jak i parametryczne z wcześniej wyznaczonym parametrem p.

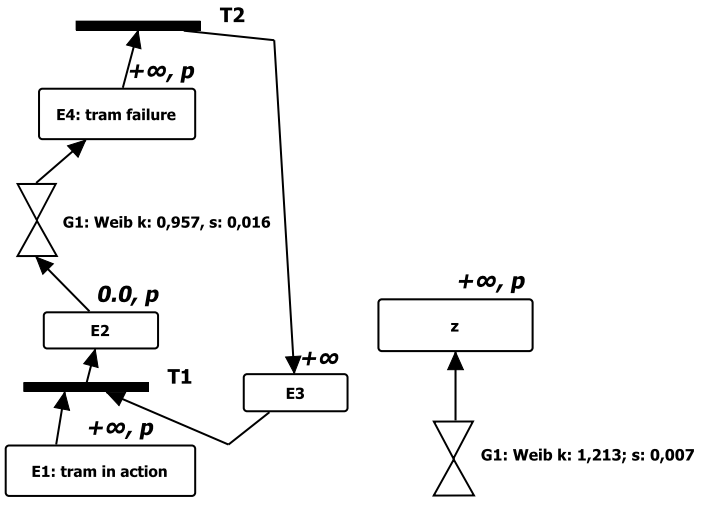
Niech x_i (y_j) oznacza proste (parametryczne) zdarzenia aktywujące. Analogicznie, u_k oraz w_l to proste i parametryczne zdarzenia hamujące przejście, na wyjściu którego znajdują się zdarzenia proste i parametryczne: z_m oraz v_n. Semantyka przejścia zdefiniowana jest następująco.

$$\begin{aligned}
\exists t, \exists p(\forall x_i, \exists xs_i \leqslant t, \exists xk_i \geqslant t : occur_{PR}(x_i, xs_i, xk_i, 1) \wedge \\
\forall y_j \exists ys_j \leqslant t, \exists yk_j \geqslant t : occur_{PAR}(y_j, ys_j, yk_j, p) \wedge \\
\forall u_k \neg(\exists us_k \leqslant t, \exists uk_k \geqslant t : occur_{PR}(u_k, us_k, uk_k, 1)) \wedge \\
\forall w_l \neg(\exists ws_l \leqslant t, \exists wk_l \geqslant t : occur_{PAR}(w_l, ws_l, wk_l, p))) \longrightarrow \\
\forall i : xk_i = t \wedge \forall j : yk_i = t \wedge \\
\forall z_m, \exists zk_m \geqslant t : occur_{PR}(z_m, t, zk_m, count(z_m, t) + 1) \wedge \\
\forall v_n \exists vs_n \leqslant t, \exists vk_n \geqslant t : occur_{PAR}(v_n, vs_n, vk_n, p)
\end{aligned}$$

Poszczególne zbiory mogą być puste, choć na wejściu przejścia musi znajdować się przynajmniej jedno zdarzenie. Ponadto jeśli na wyjściu jest zdarzenie parametryczne, to musi istnieć takie zdarzenie aktywujące o takim samym typie na wejściu.

Brama Generatora Zdarzeń

Często spotykanym zagadnieniem w modelowaniu niezawodności jest potrzeba opisu strumienia uszkodzeń, w którym kolejne awarie podobnych komponentów mają miejsce cyklicznie.



(a) Bez bramy generatora (b) Generator zdarzeń z

Rysunek 6.5: Modelowanie strumienia uszkodzeń - $E1$ zachodzi raz w chwili 0, $E1$ zachodzi w chwili 0 z różnym parametrem dla każdego tramwaju

W modelu należy zatem co pewien odcinek czasu należy wybrać kolejny element z puli i rozpoczynać dla niego określone zdarzenie. Jakkolwiek w języku PDNZC budowa takich modeli była niemożliwa [90], dzięki rozszerzeniom GN modelowanie strumienia uszkodzeń nie jest już problemem. Na rysunkach 6.5b oraz 6.5 przedstawiono dwa sposoby opisu takiego zjawiska w GN na przykładzie uszkodzeń tramwajów opisanych szerzej w rozdziale 7.

Pierwsze podejście, cyklicznie startujące $E4$, opiera się na wprowadzonym do GN przejściu działającym atomowo. W momencie wyboru kolejnego tramwaju do uszkodzenia, $T1$ nie tylko kończy jedno zdarzenie $E1$ i rozpoczyna $E2$, lecz również kończy zdarzenie proste $E3$. Stąd przejście $T1$ nie odpali się aż do ponownego startu $E3$, które, jak wynika z rysunku, zajdzie dopiero, gdy odpali się przejście $T2$. Z kolei odpalenie $T2$ warunkowane jest zachodzeniem $E4$, czyli faktycznym uszkodzeniem tramwaju opóźnionym względem $E2$ o czas interwału zdefiniowany przy bramie $G1$.

Powszechność powyższego schematu modelowania uszkodzeń była motywacją do wprowadzenia w GN przedstawionej na rys. 6.5 bramy generatora. Dla i-tego uszkodzenia opisanego zdarzeniem parametrycznym z:

$$occur_{PAR}(z, R_1(d) + R_2(d) + \ldots + R_i(d), zk_i, i)$$

Dla i-tego uszkodzenia opisanego zdarzeniem prostym z:

$$occur_{PR}(z, R_1(d) + R_2(d) \ldots + R_i(d), zk_i, count(z, R_1(d) + R_2(d) \ldots + R_i(d)) + 1)$$

Jeśli zdarzenie jest na wyjściu generatora, to generator jest jedyną bramą wejściową tego zdarzenia. Opis procesu wynikającego z wielu zmiennych losowych można uzyskać łącząc bramą wiele równoległych par: zdarzenie i jej generator.

6.2 Polityki napraw systemu komputerowego

Istnieją dwa powody dla kontynuacji dyskusji nad politykami napraw rozpoczętej w rozdziale 4. Pierwszy to chęć demonstracji nowych i rozszerzonych elementów GN na przykładzie polityki globalnej, której model zostanie przedstawiony w podrozdziale 6.2.1. Drugi wiąże się z wypełnieniem luki w opisie polityk napraw i ostatecznym zamodelowaniem odnowy opartej o określoną liczbę konserwatorów. Brakującą politykę omówiono w podrozdziale 6.2.2.

6.2.1 Zdarzenia parametryczne w modelu globalnej polityki napraw

Zdarzenia parametryczne można wykorzystać do opisu uszkodzeń i napraw komponentów systemu komputerowego. W takim rozwiązaniu wartości parametrów oznaczają identyfikatory komponentów danego typu.

W nowym modelu GN przedstawionym na rys. 6.6 potrzebne są zatem trzy gałęzie: po jednej dla każdego typu komponentu. Dla części grafu związanej z modułami pamięci i dyskami będą istniały po dwie wartości identyfikatora, a dla procesora jedna. Istnieje zatem 5 zajść zdarzeń

początkowych oznaczających poprawne funkcjonowanie każdego komponentu:

$$IE_{PAR} = \{\langle E1, 1\rangle, \langle E2, 1\rangle, \langle E2, 2\rangle, \langle E3, 1\rangle, \langle E3, 2\rangle\}$$
$$IE_{PR} = \emptyset$$

Po rozpoczęciu powyższych natychmiastowych zdarzeń parametrycznych odpalone mogą być: bramy opóźniające lub bramy negacji. Elementy CNOT zostaną omówione później.

Bramy opóźniające $G4$, $G6$ i $G8$ modelują czas do awarii, po upłynięciu którego rozpoczynane są z odpowiednimi parametrami zdarzenia $E4$, $E5$ i $E6$ oznaczające awarie. Bramy głosujące $G10$ i $G11$ rozpoczną wyjściowe zdarzenia proste, jeśli liczba zajść zdarzeń na ich wejściach wyniesie przynajmniej 2, czyli kiedy pula elementów zapasowych się wyczerpała i dany typ komponentu jest już niedostępny. W systemie istnieje tylko jeden procesor, a więc w jego przypadku brama Vote nie jest potrzebna i zdarzenie $E4$ wchodzi bezpośrednio do $G13$. Brama $G13$ rozpocznie zdarzenie proste $E9$ oznaczające hazard, które z kolei zapoczątkuje naprawę modelowaną przez $G14$. Po jej upływie ($E0$) następuje odnowa uszkodzonych elementów, co jest realizowane bramami CAND mającymi na wejściu zdarzenie parametryczne oraz proste. Zatem jedno zdarzenie $E0$ jest równolegle wykorzystywane przez 3 bramy $G1$, $G2$ oraz $G3$ i wszystkie zajścia $E4$, $E5$ i $E6$. Zamknięcie cyklu uszkodzenia i naprawy dokonują bramy CNOT $G5$, $G7$ i $G9$ natychmiast kończące zdarzenia sygnalizujące poprzednie uszkodzenia.

Zaletą takiego podejścia jest nie tylko czytelniejszy model, ale także brak konieczności rozbudowy drzewa w obliczu dalszego zwiększenia poziomu redundancji w systemie. Niezbędne jest jedynie dodanie kolejnych elementów do zbioru IE_{PAR}.

6.2.2 Model polityki opartej o zasoby

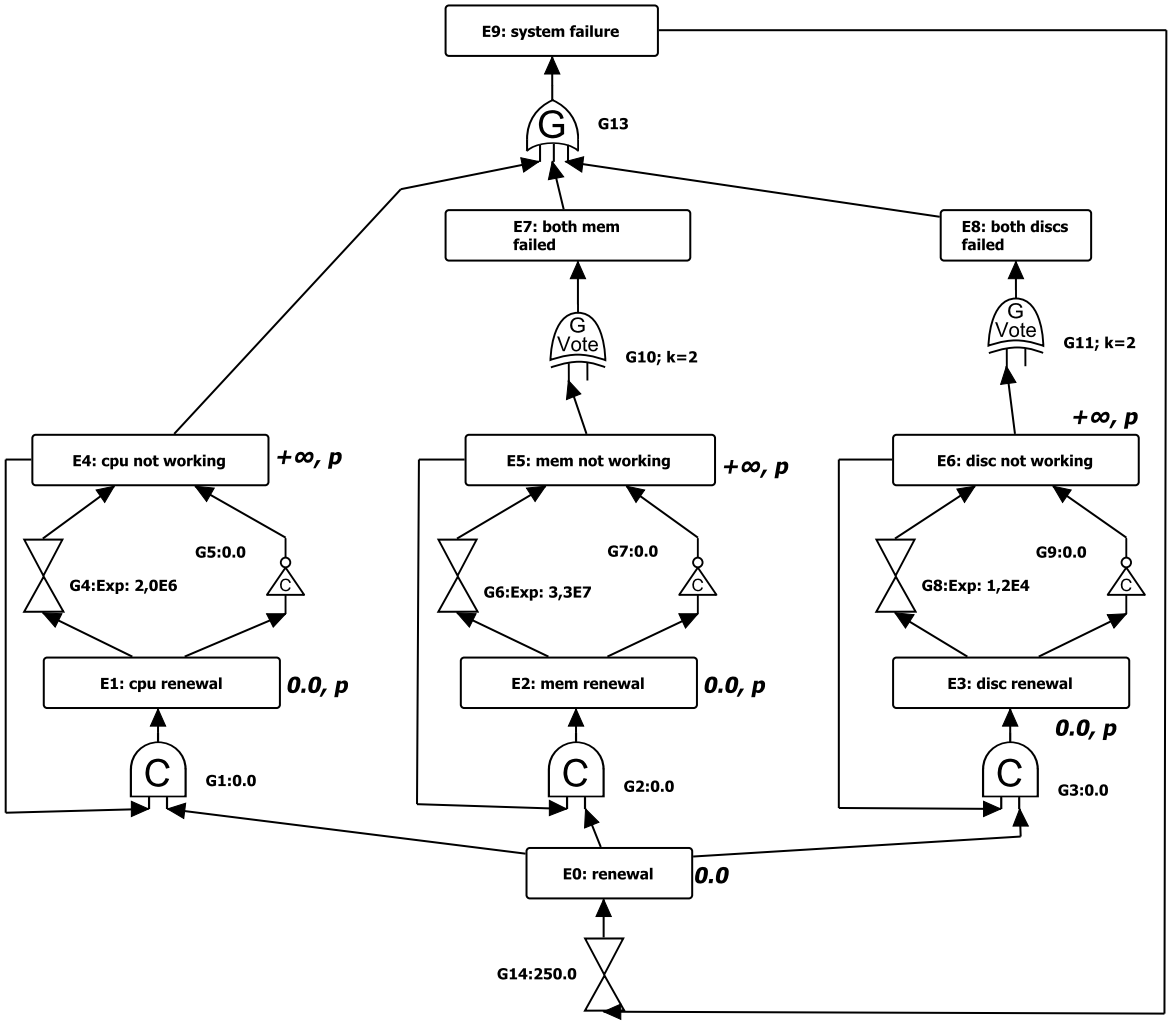
Język PDNZC opisany w poprzednich rozdziałach nie wystarcza do wyrażenia polityki naprawy systemu komputerowego z określoną liczbą konserwatorów, o dostęp do których rywalizują uszkodzone komponenty. Jeśli istnieje k konserwatorów, to maksymalnie k komponentów może być naprawianych równolegle, a rozpoczęcie odnowy kolejnych elementów następuje wraz z zakończeniem poprzednich procesów i udostępnieniem konserwatorów.

Prawidłowy opis takiego schematu wymaga konstrukcji rozstrzygającej konflikt pomiędzy wieloma komponentami jednocześnie żądającymi dostępu do zasobu współdzielonego, czyli konserwatora. Taką funkcję pełni przejście atomowo rozpoczynające i kończące zdarzenia.

Nowością w stosunku do modelu polityki globalnej są zdarzenia $E10 - E23$ oraz bramy $G14 - G24$ przedstawione w górnej części rysunku 6.7. Zdarzenia początkowe są następujące.

$$IE_{PAR} = \{\langle E1, 1\rangle, \langle E2, 1\rangle, \langle E2, 2\rangle, \langle E3, 1\rangle, \langle E3, 2\rangle, \langle E21, 1\rangle, \langle E22, 1\rangle, \langle E22, 2\rangle, \langle E23, 1\rangle, \langle E23, 2\rangle\}$$
$$IE_{PR} = \{\langle E7, 1\rangle, \langle E7, 2\rangle, \ldots, \langle E7, k\rangle\}$$

Proces naprawy aktywowany jest bramą $G14$ poprzez rozpoczęcie zdarzenia prostego $E10$ dokładnie 10 jednostek czasu po wykryciu hazardu. Uzupełnieniem wcześniej opisanego schematu awarii i naprawy komponentów są zdarzenia $E21$-$E23$ wraz z przylegającymi bramami CNOT

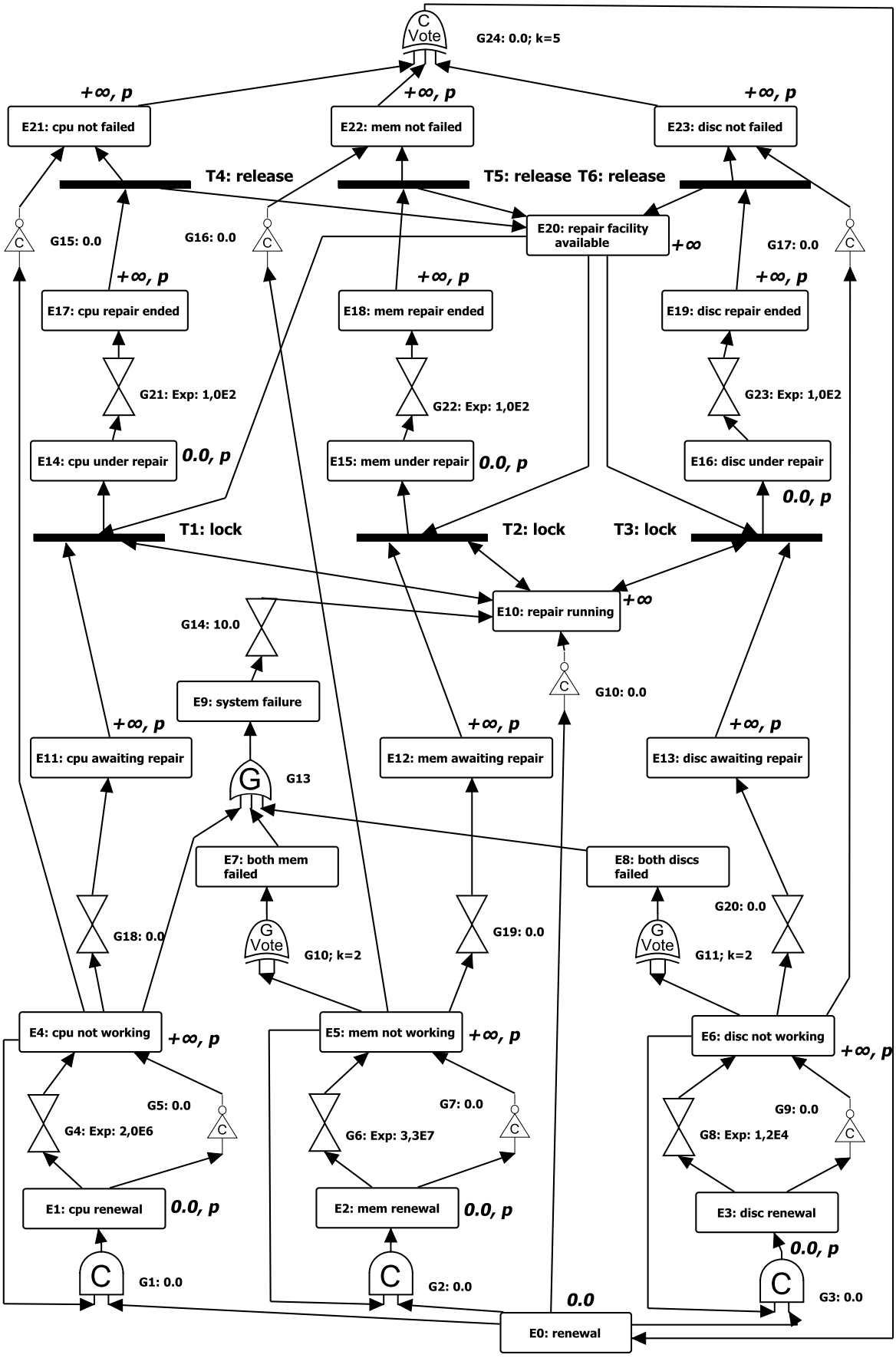


Rysunek 6.6: Uproszczenie modelu polityki globalnej towarzyszące wprowadzeniu zdarzeń parametrycznych: $E1 - E6$ do języka GN

$G15$-$G17$. Dzięki tym bramom, gdy komponent ulegnie uszkodzeniu i rozpoczęte zostanie zdarzenie $E4$-$E6$, nastąpi zablokowanie odnowy aż do momentu, gdy wszystkie komponenty będą sprawne. Brama $G24$ oczekuje 5 zdarzeń na wejściach i dopiero wtedy rozpocznie przy pomocy zdarzenia $E0$ opisaną przy polityce globalnej odnowę oraz dokona zakończenia procesu naprawy - $G10$. Zatem bramy $G14$ i $G10$ odpowiednio rozpoczynają i kończą naprawę całego systemu.

Wraz z awarią komponentu sygnalizowana jest potrzeba jego naprawy: $G18$-$G20$ i zdarzenia z parametrami $E11$-$E13$. Naprawy poszczególnych komponentów powinny kończyć się rozpoczęciem $E21$-$E23$. Omawiany model gwarantuje włączenie do procesu również tych komponentów, które uległy awarii w trakcie przeprowadzanej naprawy.

Liczba zachodzących zdarzeń prostych $E20$ oznacza liczbę wolnych konserwatorów. Jeśli tacy są obecni, oraz rozpoczęta została naprawa systemu $E10$, oraz dany komponent jest uszkodzony, to nastąpi alokacja do niego konserwatora przy pomocy przejścia $T1$-$T3$. Zdarzenia $E14$-$E16$ oznaczające rozpoczęcia naprawy komponentu zostaną wystartowane z parametrami wynikają-



Rysunek 6.7: Polityka odnowy oparta o zasoby z dowolną liczbą konserwatorów wyrażoną liczbą zdarzeń $E20$ zachodzących w chwili początkowej [93]

cymi ze zdarzeń $E11$-$E13$. Przejście zakończy również zdarzenie oznaczające dostępność konserwatora uniemożliwiając przydzielenie tego samego zasobu innym komponentom. Elementy: $E14$-$G21$-$E17$ oraz analogiczne dla pamięci oraz dysków opisują naprawę samego komponentu trwającą czas zdefiniowany przy bramie opóźniającej. Gdy komponent został naprawiony, tzn. wystąpiło $E17$-$E19$ z odpowiednim parametrem, to odpalone zostanie przejście $T4$-$T6$. W efekcie wystartuje $E20$, czyli konserwator będzie dostępny dla kolejnych uszkodzonych komponentów, oraz wystartuje $E21$-$E23$. Jak opisano wcześniej przyczyni się to do natychmiastowej odnowy systemu i rozpoczęcia kolejnego cyklu pracy.

6.3 Automatyczna transformacja grafów w symulator niezawodności

W niniejszym podrozdziale zostanie opisany szkielet transformacji GN zapisanych w języku *Ecore* [160] w kod wykonywalny wyrażony w postaci zbioru reguł wnioskowania systemu JBoss Drools [10]. Szczegóły transformacji zebrano w dodatku A.

6.3.1 Obiektowa specyfikacja GN

Algorytm translacji przetwarza sekwencyjnie każdy element grafu rozumianego zgodnie z metamodelem przedstawionym na rys. 6.8.

Na egzemplarz klasy *FaultGraph* składają się obiekty będące specjalizacjami klasy *FaultGraphElement* czyli krawędzie (*Edge*) oraz węzły (*Node*) grafu. W przeciwieństwie do PDNZC, w GN istnieje tylko jeden typ krawędzi używany do połączenia zarówno bram ze zdarzeniami, jak i przejść ze zdarzeniami w obydwie strony.

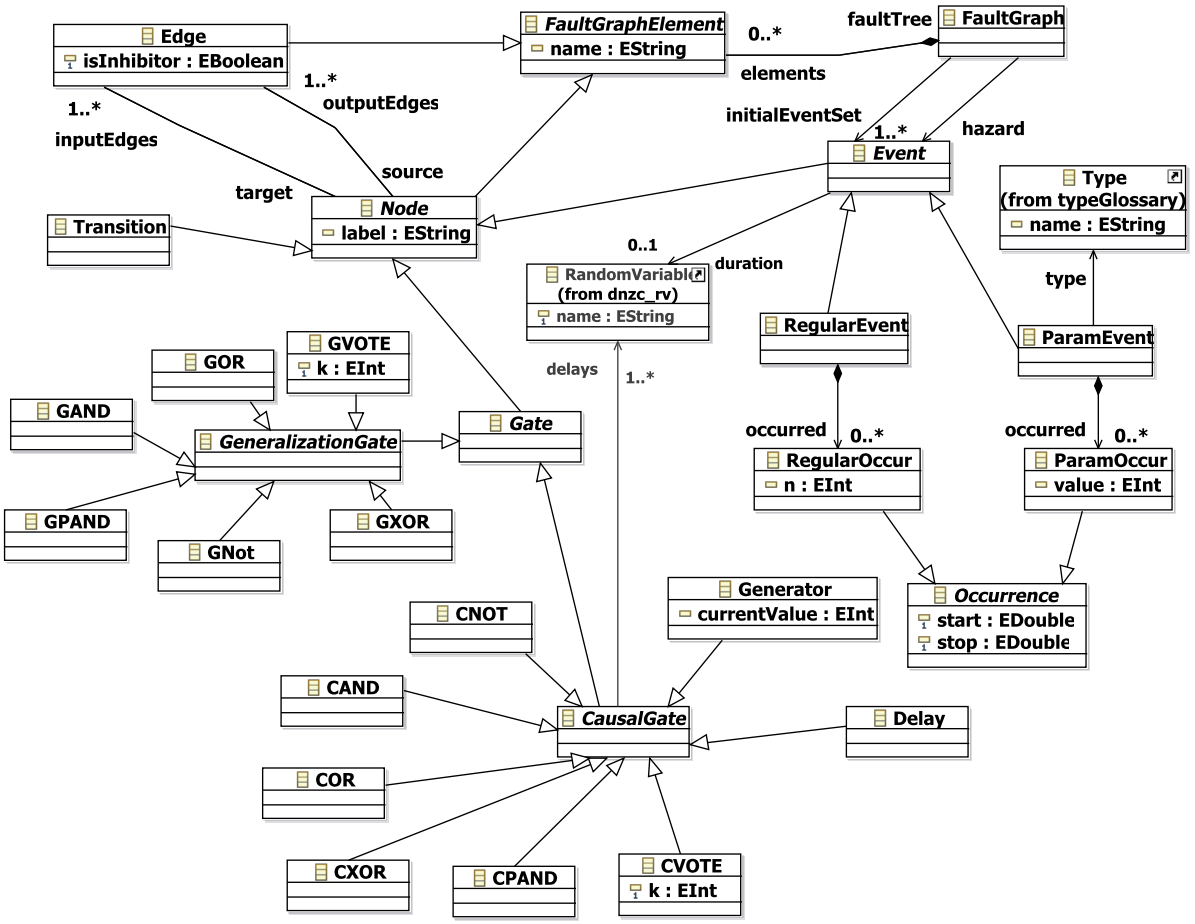
W języku są natomiast obecne trzy typy węzłów: przejścia (klasa *Transition*), bramy (*Gate*) oraz zdarzenia (*Event*). Przejście nie posiada natury czasowej i ma jedynie zdarzenia wejściowe oraz wyjściowe zdefiniowane przez jej klasę podstawową *Node*. W szczególnym przypadku krawędź łącząca zdarzenie z przejściem może być łukiem hamującym, co oznacza się ustawioną flagą *isInhibitor*. Istnieją dwa typy bram: przyczynowe oraz uogólniające. Nowym bramom: opóźniającej oraz generatora odpowiadają klasy *Delay* oraz *Generator*.

Specjalizacje klasy *Event* odpowiadają zdarzeniom prostym *RegularEvent* oraz parametrycznym *ParamEvent*. Każda ma odpowiadający swojemu typowi zbiór zajść: *RegularOccur* oraz *ParamOccur*, które razem z atrybutami klasy podstawowej *start* i *stop* opisują zajście.

Dwa metamodele towarzyszą przedstawionemu szkieletowi języka. Pierwszy, zawierający klasę *RandomVariable* oraz zbiór podklas *ProbabilityDistribution* (rys. 6.9), pozwala na definiowanie zmiennych losowych czasu odpalenia bram oraz czasu trwania zdarzeń. Drugi zawiera zbiór wszystkich typów zdarzeń parametrycznych reprezentowanych przez klasę *Type*.

6.3.2 Koncepcja symulatora

Symulacja GN polega na rozpoczynaniu oraz kończeniu zdarzeń grafu przez zbiór automatycznie wygenerowanych z tego grafu reguł wnioskowania. Każda reguła posiada zbiór przesłanek



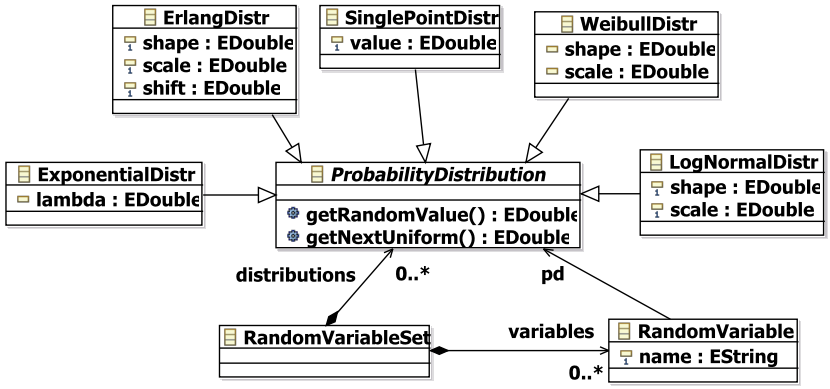
Rysunek 6.8: Obiektowy metamodel GN

nazwanych faktami, a w rezultacie swojego wykonania dokonuje aktualizacji innych faktów, co prowadzi z kolei do wykonania kolejnych reguł. Reguły generowane są z bram i przejść. Faktami są zajścia zdarzeń, czyli obiekty klas *ParamOccur* oraz *RegularOccur*. Zbiór faktów w danej chwili to pamięć robocza.

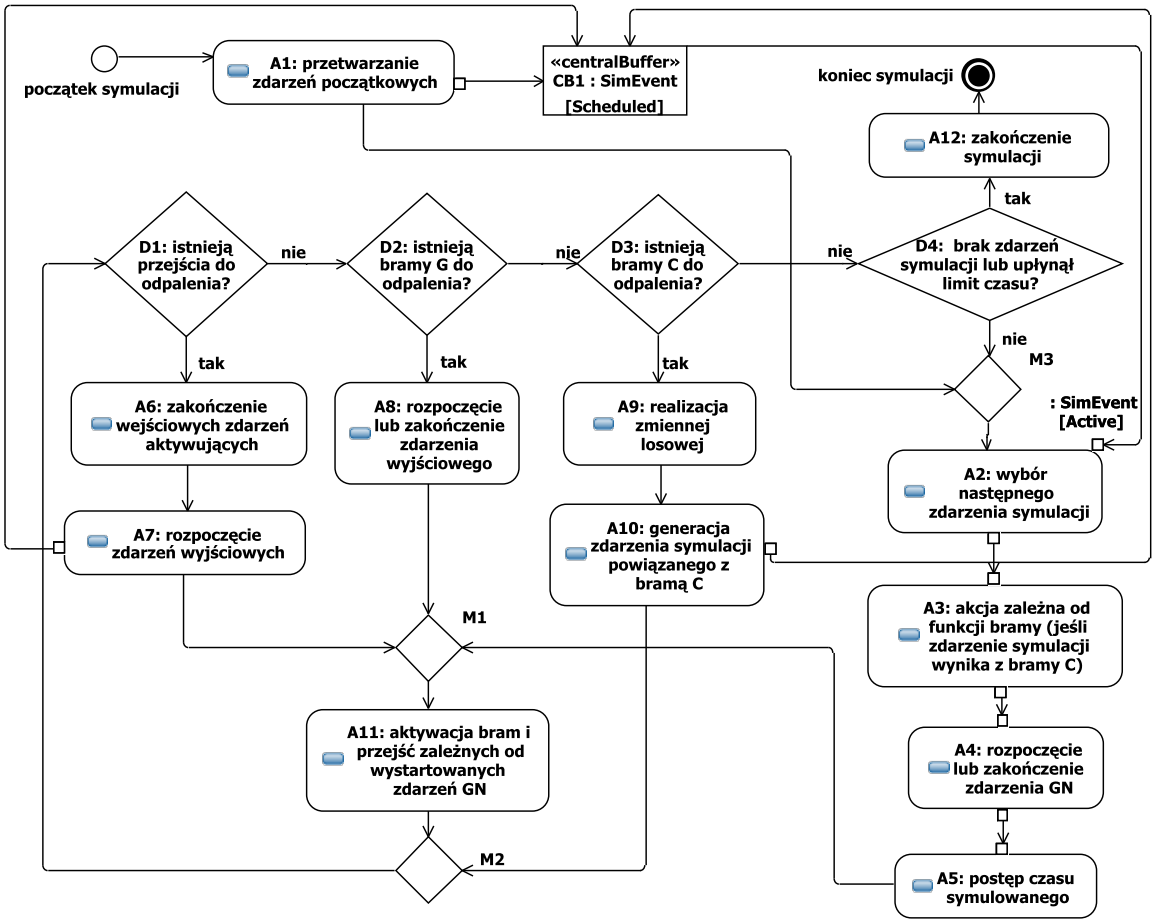
Jak wynika z rys. 6.10 symulacją steruje centralny bufor zdarzeń $CB1$ przechowujący obiekty klasy *SimEvent* i zarządzający prawidłową kolejnością startów i zakończeń zdarzeń GN. Najważniejsze atrybuty obiektów *SimEvent* informują które zdarzenie należy rozpocząć lub zakończyć w określonej chwili czasu. Obiekty w buforze są posortowane rosnąco według tej chwili, a akcja $A2$ zawsze wybiera pierwszy element z bufora.

W chwili początkowej symulacji bufor zasilany jest obiektami *SimEvent* rozpoczynającymi zdarzenia początkowe (akcja $A1$), a później uzupełniany jest wynikami działania bram przyczynowych, które jako jedyne - uwzględniając modelowanie czasu trwania zdarzenia przy pomocy bramy $CNOT$ - mają naturę czasową.

Kolejne zdarzenie symulacji wybierane jest z bufora akcją $A2$. Pomijając akcję $A3$ wyjaśnioną w podrozdziale A.2, zdarzenie symulacji powoduje zawsze rozpoczęcie lub zakończenie jednego zdarzenia ($A4$), w tym aktualizację jego faktu oraz postęp czasu symulowanego ($A5$).



Rysunek 6.9: Zmienne losowe i ich dystrybuanty wspierane przez symulator GN



Rysunek 6.10: Algorytm symulacji GN

Aktualizacja faktu pociąga za sobą przeliczenie reguł sąsiadujących bram oraz przejść ($A11$).

Jeśli wiele elementów ma być odpalonych w tej samej chwili, to o ich kolejności decyduje priorytet. Najwyższy mają przejścia, stąd pierwszy blok decyzyjny $D1$ sprawdza, czy istnieje taki element gotowy do odpalenia. Gdy gotowych do odpalenia przejść jest wiele, kolejność jest niedeterministyczna i dochodzi do wyścigu jak w modelu polityk napraw na rys. 6.7. Reguła odpalenia przejścia dokona zakończenia zdarzeń wejściowych ($A6$) oraz rozpoczęcia zdarzeń wyj-

ściowych ($A7$). Usunięcie oraz dodanie zajść zdarzeń do pamięci roboczej spowoduje ponowne przeliczenie reguł w akcji $A11$.

Jeśli nie ma już przejść do odpalenia, następuje sprawdzenie bram uogólniających - blok decyzyjny $D2$. W akcji $A8$ jest tworzone bądź usuwane zajście zdarzenia wyjściowego, co ponownie prowadzi do $A11$.

Istotą zdarzeń przyczynowych, które rozpatrywane są jako ostatnie w bloku $D3$, jest związany z nimi postęp czasu. Start (lub zakończenie w przypadku bramy CNOT) zdarzenia wyjściowego jest odroczone w czasie związanym ze zmienną losową realizowaną w akcji $A9$. Odroczenie opisywane jest nowym zdarzeniem symulacji dodawanym do bufora $CB1$ i konsekwentnie pomiędzy blokami $D3$ i $M2$ nie dochodzi do aktualizacji żadnego faktu. Nie wykonuje się zatem $A11$. Faktyczne odpalenie bądź zakończenie zdarzenia nastąpi w akcji $A4$, czyli nie tylko w przyszłości, ale również po analizie wszystkich przejść i bram w tamtej chwili czasu. Opisana procedura symulacji bramy przyczynowej wykonywana jest nawet, gdy opóźnienie wynosi zero.

Jeśli nie ma już żadnych zdarzeń symulacji lub przekroczony został limit czasu, symulacja kończy się - $D4$ i $A12$. Szczegóły algorytmu transformacji opisano w dodatku A.

6.4 Modelowanie i symulacja modelu kolejkowego

Na przykładzie prostego systemu telekomunikacyjnego w podrozdziale zostanie pokazany sposób modelowania procesów o naturze cyklicznej przy pomocy Grafów Niezdatności.

W przykładzie populacja użytkowników wykorzystuje zbiór c kanałów do transmisji danych, tak że każde nawiązanie i realizacja połączenia wymaga wyłącznego dostępu do jednego kanału.

Gdy czas pomiędzy kolejnymi połączeniami A a także czas samego połączenia S mają rozkłady wykładnicze, to system jest w istocie systemem kolejkowym typu M/M/c i tym samym istnieje formuła analityczna prawdopodobieństwa oczekiwania. Rosnąca popularność przesyłu pakietowego, w szczególności dostępu do usług społecznościowych, zmusza do odejścia od rozkładów wykładniczych. Dotychczas nie znaleziono jednak rozwiązań dla systemów typu M/G/c oraz G/G/c [85], czyli gdy czas połączenia lub czas pomiędzy połączeniami mają dowolne rozkłady, na przykład rozpatrywane dalej rozkłady Weibulla.

Jednym ze sposobów szacowania prawdopodobieństwa blokowania w modelu G/G/c jest utworzenie nowych, wykładniczych zmiennych zgłoszeń oraz awarii, takich że ich współczynniki skali odpowiadają wartościom średnim oryginalnych zmiennych o rozkładach Weibulla:

$$A'_s = E(A') = E(A) = A_s\Gamma(1 + \frac{1}{A_k})$$

$$S'_s = E(S') = E(S) = S_s\Gamma(1 + \frac{1}{S_k})$$

gdzie:

$A(S)-$zmienna czasu pomiędzy zgłoszeniami (czasu połączenia) o rozkładzie Weibulla

$A_{s|k}(S_{s|k})-$współczynnik skali lub kształtu zmiennej A(S)

$A'(S')-$zmienna czasu pomiędzy zgłoszeniami (czasu połączenia) o rozkładzie wykładniczym

$A'_s(S'_s)-$współczynnik skali zmiennej A'(S')

$\Gamma-$funkcja Gamma

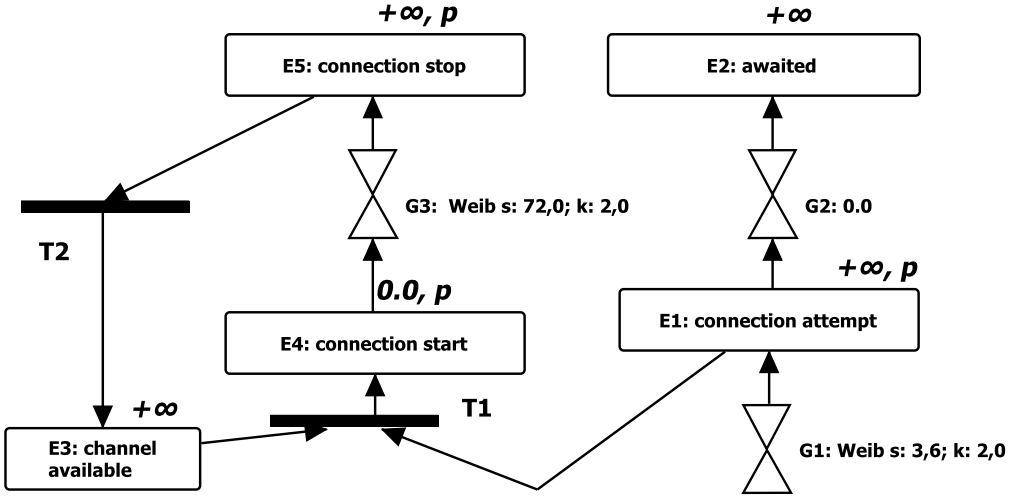
Dla uzyskanego w ten sposób modelu M/M/c opartego o A' i S' współczynnik obciążenia to:

$$\rho = \frac{D'_s}{A'_s}$$

Jeśli $\rho < c$, to prawdopodobieństwo oczekiwania można wyznaczyć z formuły Erlang C:

$$P_{ErlangC} = \frac{\frac{\rho^c}{(c-1)!}}{(c-\rho)\sum_{i=0}^{c-1}\frac{\rho^i}{i!} + \frac{\rho^c}{(c-1)!}}$$

Alternatywą dla estymacji analitycznej jest budowa modelu w języku GN oraz przeprowadzenie badań symulacyjnych przy pomocy opisanej wcześniej transformacji grafu w kod.



Rysunek 6.11: Graf pozwalający na wyznaczenie prawdopodobieństwa oczekiwania w kolejce

Próby nawiązania połączeń opisane są na rys. 6.11 zdarzeniem parametrycznym $E1$, którego zajścia generuje brama generatora $G1$. Jeśli jest dostępny wolny kanał (zachodzi $E3$) to przejście $T1$ zostanie odpalone przed bramą $G2$ i nie dojdzie do rozpoczęcia kolejnego zdarzenia $E2$. Po nawiązaniu połączenia ($E4$) odmierzany jest czas obsługi przez bramę $G3$. Zakończenie połączenia modelowane jest $E5$. Ostatnim elementem jest zwolnienie kanału przejściem $T2$.

Jeśli jednak kanał nie był dostępny w chwili próby uzyskania połączenia, to nie odpali się $T1$ i dojdzie do rozpoczęcia kolejnego zdarzenia $E2$ przez bramę $G2$.

Prawdopodobieństwo oczekiwania można wyznaczyć w symulacji następująco:

$$P_{GN} = \frac{\text{liczba startów zdarzenia } E2}{\text{liczba odpaleń bramy } G1}$$

Błąd względny estymacji względem rozwiązania symulacyjnego określony jest formułą:

$$\delta = \frac{|P_{GN} - P_{ErlangC}|}{P_{GN}} * 100\%$$

W eksperymencie obliczeniowym czas pomiędzy połączeniami A miał rozkład wykładniczy o średniej 72s, a zmianie poddawano współczynnik kształtu czasu połączenia S w przedziale $\langle 1, 2\rangle$ z krokiem 0.2 (tab. 6.1).

Pierwszy wiersz to w istocie przypadek M/M/c, co uzasadnia zbieżność z rozwiązaniem analitycznym i stanowi weryfikację symulatora wygenerowanego przez transformację. Wraz z oddalaniem się czasu obsługi od rozkładu wykładniczego formuła analityczna dostarcza coraz mniej precyzyjne wyniki, choć wzrost błędu jest powolny.

Gdy jednak czas pomiędzy próbami połączeń przestaje być wykładniczy, to błędy rosną gwałtownie i zastosowanie podejścia symulacyjnego staje się koniecznością. W tab. 6.2 zebrano wyniki dla rosnącego współczynnika kształtu zmiennej A.

Dodatkową zaletą podejścia opartego o GN jest możliwość badania innych wariantów modelu, np. wielu równoległych typów usług, z których każda ma inny czas realizacji.

Tabela 6.1: Wyniki symulacji i estymacji modelu M/G/c dla $A_s = 72s, S_s = 3.6s, c = 25$

		Prawdopodobieństwo oczekiwania		
S_k	$E(S)$	P_{GN}	$P_{ErlangC}$	δ
[]	[s]	[]	[]	[%]
1,0	72,00	0,2091	0,2091	<0,01
1,2	67,73	0,1222	0,1243	1,71
1,4	65,62	0,0907	0,0935	3,03
1,6	64,55	0,0771	0,0802	3,99
1,8	64,03	0,0708	0,0743	4,95
2,0	63,81	0,0679	0,0719	5,82

Tabela 6.2: Wyniki symulacji i estymacji modelu G/G/c dla $A_s = 72s, S_s = 3.6s, S_k = 2.0, c = 25$

		Prawdopodobieństwo oczekiwania		
A_k	$E(A)$	P_{GN}	$P_{ErlangC}$	δ
[]	[s]	[]	[]	[%]
1,0	3,60	0,0679	0,0719	5,82
1,2	3,39	0,0822	0,1260	53,29
1,4	3,28	0,0872	0,1656	90,01
1,6	3,23	0,0853	0,1901	122,87
1,8	3,20	0,0795	0,2033	155,78
2,0	3,19	0,0719	0,2091	190,82

Rozdział 7

Modelowanie i symulacja szeregowego systemu z zimną rezerwą funkcjonalną, dostawą uszkodzonego elementu po naprawie oraz rezerwą czasową

W pierwszych dwóch podrozdziałach: 7.1 i 7.2 omówiono z perspektywy niezawodności funkcjonowanie systemu tramwajowego we Wrocławiu oraz przeprowadzono analizę danych rzeczywistych z 2001 roku. Dla zdefiniowanych dalej celów badawczych skonstruowano dwa modele niezawodności. Pierwszy, będący siecią Petriego HLPN [66], przedstawiono w podrozdziale 7.3. Drugi, skonstruowany w języku GN, opisano w podrozdziale 7.4. Wyniki badań symulacyjnych dla drugiego modelu przedyskutowano w podrozdziale 7.5.

7.1 Pojęcie rezerwy czasowej w systemie wsparcia logistycznego

We wrocławskim systemie tramwajowym obsługiwanym przez Miejskie Przedsiębiorstwo Komunikacyjne (MPK) istnieją tramwaje zapasowe będące w rozumieniu niezawodnościowym elementami rezerwy zimnej. Gdy regularny pojazd będący w trasie ulegnie uszkodzeniu, dyspozytor podejmuje decyzję o wysłaniu na miejsce awarii tramwaju zapasowego, który przejmuje funkcję uszkodzonego aż do momentu jego naprawy. Na czas takiej podmiany wpływ mają:

- czas oczekiwania na dostępność któregokolwiek z tramwajów rezerwowych, które w chwili awarii mogą pełnić funkcje innych, wcześniej uszkodzonych tramwajów,
- czas dojazdu dostępnego tramwaju do miejsca awarii określany dalej *czasem wymiany*

Niezależnie od dostępności rezerwy, podstawowy tramwaj jest równolegle naprawiany a następnie doprowadzany z powrotem na trasę, co określa się *dostawą*.

W myśl kontraktu zawartego pomiędzy MPK a Urzędem Miasta Wrocławia awaria tramwaju nie może być odczuwalna przez pasażerów po czasie określanym jako rezerwa czasowa. Zatem wcześniejsza z chwil: wymiany i dostawy powinna zajść przed chwilą awarii powiększoną o rezerwę czasową. Hazard to taka awaria tramwaju, w której ani wymiana, ani dostawa nie zmieściły się w czasie kontraktowym. W niniejszym rozdziale przyjęto, że hazard to awaria *nadsystemu* skupiającego zarówno pojazdy regularne jak i zapasowe, podczas gdy same uszkodzenia tramwajów regularnych to awarie *systemu technicznego*. W dalszej analizie zakłada się również, że tramwaje rezerwowe są niezawodne.

7.2 Analiza danych rzeczywistych o uszkodzeniach

Budowie i symulacjom niezawodności systemu przy pomocy GN przyświecają trzy cele:

1. weryfikacja mocy opisowej GN na przykładzie problemu niedostępnego językowi PDNZC,
2. walidacja modelu poprzez porównanie wyników symulacyjnych z rzeczywistymi,
3. analiza wrażliwości niezawodności systemu na modyfikacje liczby tramwajów rezerwowych.

Dla osiągnięcia powyższych celów analizie poddano rejestr ruchu prowadzonego przez dyspozytornię MPK w 2001 roku, a zaczerpniętego z pracy [182]. Badaniom poddano najliczniejszy podzbiór danych dotyczący roboczego rozkładu jazdy, w którym funkcjonowało 5 tramwajów rezerwowych (symbol n), co należy uznać za wysoki i kosztowny poziom redundancji. Dla osiągnięcia ostatniego celu badania obejmą również zmniejszoną do 3 liczbę tramwajów rezerwowych.

Ze wspomnianego rejestru dni roboczych wyłoniono dalsze 2 podzbiory danych różniące się zmiennymi losowymi czasu pomiędzy uszkodzeniami A, czasu wymiany E oraz czasu dostawy D. Podział uzasadniony jest następująco. Zauważono, że gdy uszkodzenie tramwaju pozwalało na dalszą jazdę, dyspozytor często wstrzymywał zjazd do określonego warunkami logistycznymi momentu. Zatem pierwsza możliwość to przyjęcie chwili awarii za moment zgłoszenia uszkodzenia przez motorniczego. Alternatywą jest interpretacja początku awarii jako chwili zjazdu z trasy.

Kolejnym punktem swobody jest przyjęta rezerwa czasowa oznaczana symbolem R_C, która odpowiada albo minimalnemu czasowi kursu w 2001 roku (41 minut), albo jego połowie. Reasumując uwzględniając dwa poziomy redundancji, dwie możliwości początku awarii oraz dwie rezerwy czasowe otrzymuje się ostatecznie 8 przypadków symulacji zebranych w tab. 7.1. Dla przypadków 1-4 wyniki z wykorzystaniem GN zostaną porównane z rzeczywistymi.

Zmienne losowe *czasu pomiędzy awariami* A systemu technicznego określono rozkładami Weibulla, których gęstość f_a oraz dystrybuanta F_a mają następującą postać:

$$f_a(z_a) = \frac{k_a}{\lambda_a} z_a^{k_a-1} e^{-(z_a)^{k_a}}$$

$$F_a(z_a) = 1 - e^{-z_a^{k_a}}$$

gdzie:

Tabela 7.1: Geneza przypadków testowych

Nr przypadku	Poziom redundancji	Przyjęty moment awarii tramwaju	Przyjęta rezerwa czasowa
1	wysoki	$t_{\text{zgłoszenia}}$	$min(t_{\text{kurs}})/2$
2			$min(t_{\text{kurs}})$
3		t_{zjazdu}	$min(t_{\text{kurs}})/2$
4			$min(t_{kurs})$
5	niski	$t_{\text{zgłoszenia}}$	$min(t_{kurs})/2$
6			$min(t_{kurs})$
7		t_{zjazdu}	$min(t_{\text{kurs}})/2$
8			$min(t_{kurs})$

z_a - zmienna pomocnicza: $z_a \equiv \frac{t}{\lambda_a}$,

λ_a - współczynnik skali,

k_a - współczynnik kształtu, w przypadkach 1,2,5 oraz 6 $k_a = 1$, więc rozkład upraszcza się do typu wykładniczego.

Do zmiennych określających *czas wymiany* E, gdy za chwilę awarii uznaje się moment zgłoszenia dopasowano uogólniony rozkład wartości ekstremalnych:

$$f_e(z_e) = \frac{1}{\lambda_e} exp(-(1 + k_e z_e)^{-1/k_e})(1 + k_e z_e)^{-1-1/k_e}$$

$$F_e(z_e) = exp(-(1 + k_e z_e)^{-1/k_e})$$

gdzie:

z_e - zmienna pomocnicza: $z_e \equiv \frac{t-\gamma_e}{\lambda_e}$,

λ_e - współczynnik skali,

γ_e - współczynnik położenia,

k_e - współczynnik kształtu.

Gdy za chwilę awarii uznaje się moment zjazdu nie udało się znaleźć rozkładu parametrycznego, który wykazałby dopasowanie w teście λ Kołmogorowa na poziomie przynajmniej 0,01. Przyczyną jest częste wstrzymywanie przez dyspozytora zjazdu tramwaju aż przybędzie rezerwa, co widać na rys. 7.2c w dużej masie prawdopodobieństwa dla argumentu równego 0. Stąd zdecydowano się na zastosowanie w symulacji gęstości empirycznej.

Natomiast *czas dostawy* D, w obydwu przypadkach interpretacji początku awarii, najwiarygodniej opisuje czteroparametryczny rozkład Burr'a:

$$f_d(z_d) = \frac{\alpha_d k_d z_d^{\alpha_d - 1}}{\lambda_d (1 + z_d^{\alpha_d})^{k_d + 1}}$$

$$F_d(z_d) = 1 - (1 + z_d^{\alpha_d})^{-k_d}$$

gdzie:

z_d - zmienna pomocnicza: $z_d \equiv \frac{t-\gamma_d}{\lambda_d}$,

k_d - pierwszy współczynnik kształtu,

α_d - drugi współczynnik kształtu,

λ_d - współczynnik skali,

γ_d - współczynnik położenia.

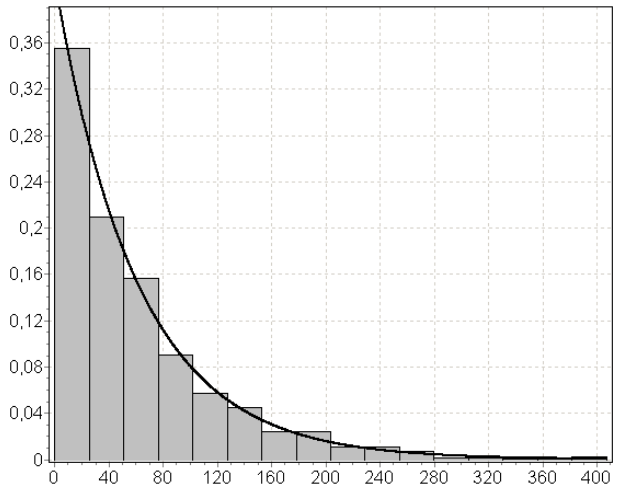
Pełne dane wejściowe symulacji dla każdego z 8 przypadków zebrano w tab. 7.2, a wartości średnie zmiennych losowych umieszczono dodatkowo w tab. 7.3. Opisane dopasowania oraz wykresy kwantyl-kwantyl pokazano na rys. 7.1 oraz 7.2.

Tabela 7.2: Parametry przypadków testowych wyznaczone z danych rzeczywistych

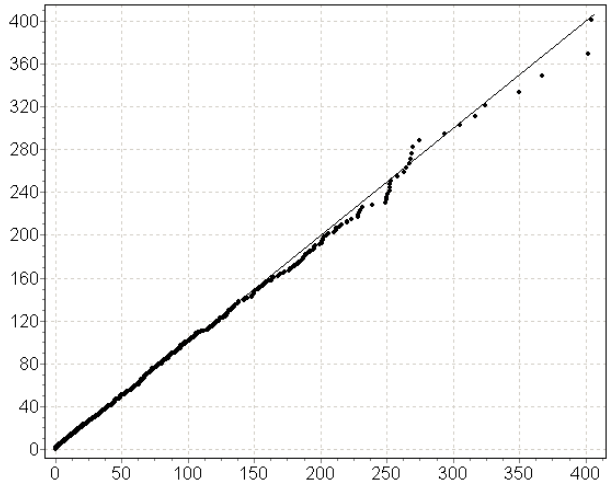
Przypadek	**Awaria**		**Wymiana**			**Dostawa**				**Rezerwa**	
	k_a *[]*	λ_a *[min]*	k_e *[]*	λ_e *[min]*	γ_e *[min]*	k_d *[]*	α_d *[]*	λ_d *[min]*	γ_d *[min]*	R_C *[min]*	n
1	1,00	60,94	0,19	10,85	13,25	0,68	5,31	102,50	-19,09	20,50	5
2	1,00	60,94	0,19	10,85	13,25	0,68	5,31	102,50	-19,09	41,00	5
3	0,98	62,69	*gęstość empiryczna*			0,70	5,59	103,59	-24,53	20,50	5
4	0,98	62,69	*gęstość empiryczna*			0,70	5,59	103,59	-24,53	41,00	5
5	1,00	60,94	0,19	10,85	13,25	0,68	5,31	102,50	-19,09	20,50	3
6	1,00	60,94	0,19	10,85	13,25	0,68	5,31	102,50	-19,09	41,00	3
7	0,98	62,69	*gęstość empiryczna*			0,70	5,59	103,59	-24,53	20,50	3
8	0,98	62,69	*gęstość empiryczna*			0,70	5,59	103,59	-24,53	41,00	3

Tabela 7.3: Wartości średnie zmiennych losowych wykorzystywanych w symulacjach

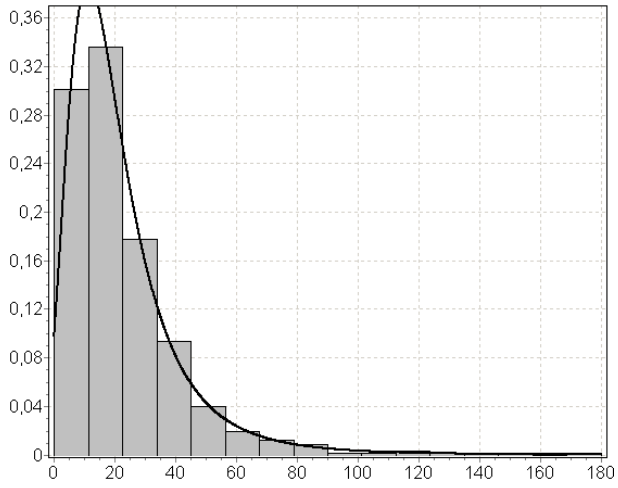
Przypadek	**Awaria**	**Wymiana**	**Dostawa**
	$E(A)$ *[min]*	$E(E)$ *[min]*	$E(D)$ *[min]*
1	60,94	23,38	107,09
2	60,94	23,38	107,09
3	62,91	15,96	100,25
4	62,91	15,96	100,25
5	60,94	23,38	107,09
6	60,94	23,38	107,09
7	62,91	15,96	100,25
8	62,91	15,96	100,25



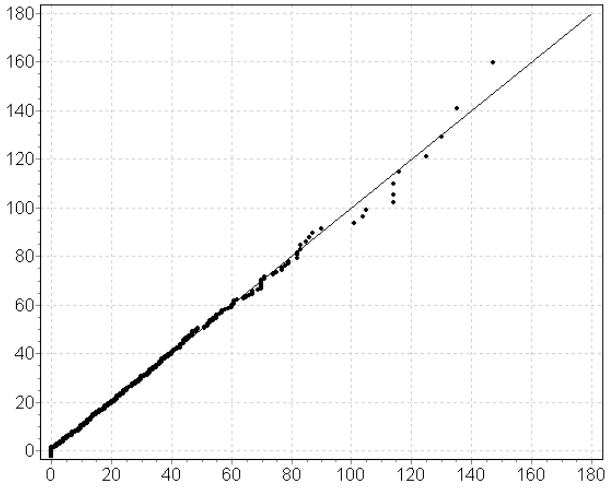
(a) Histogram i estymacja parametryczna zmiennej A



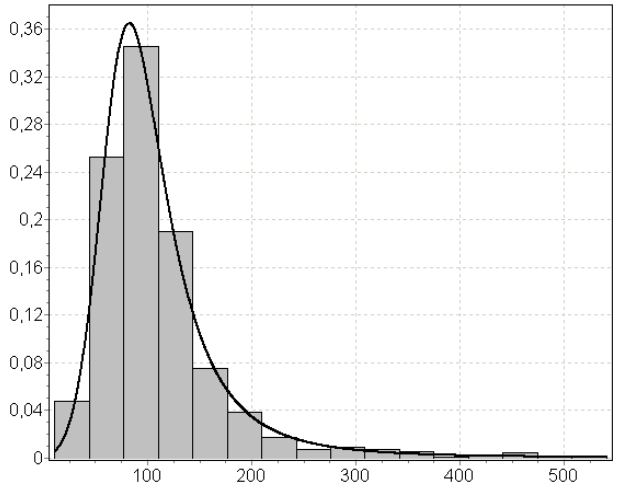
(b) Ocena dopasowania zmiennej A, $p = 0,14$



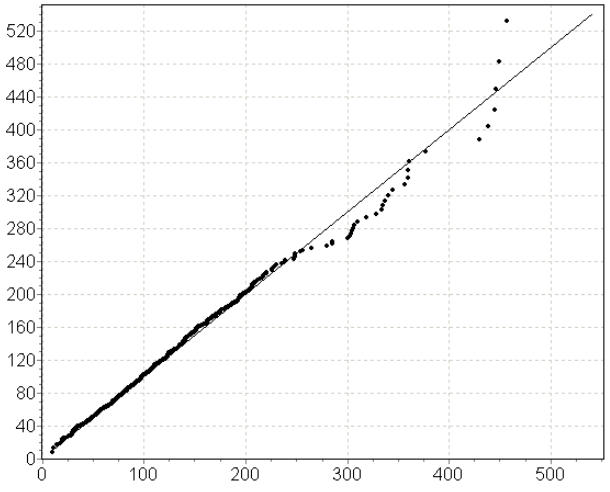
(c) Histogram i estymacja parametryczna wymiany E



(d) Ocena dopasowania wymiany E, $p = 0,31$

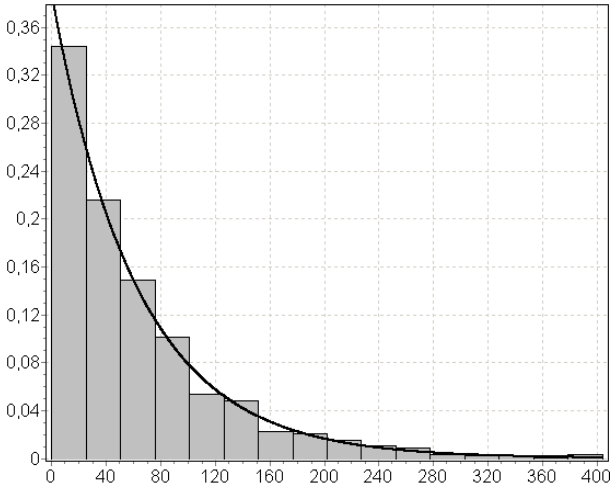


(e) Histogram i estymacja parametryczna dostawy D

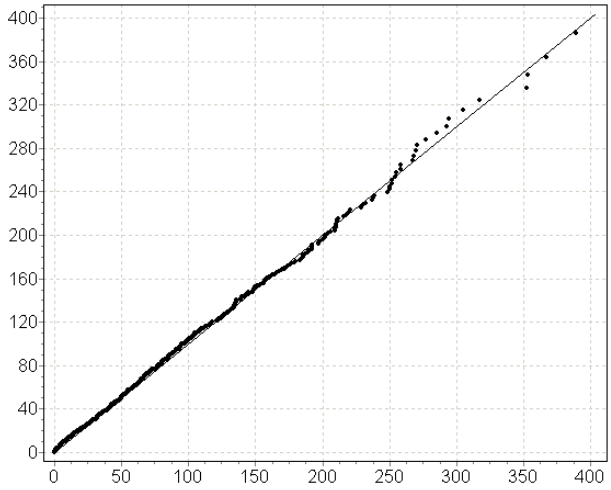


(f) Ocena dopasowania dostawy D, $p = 0,65$

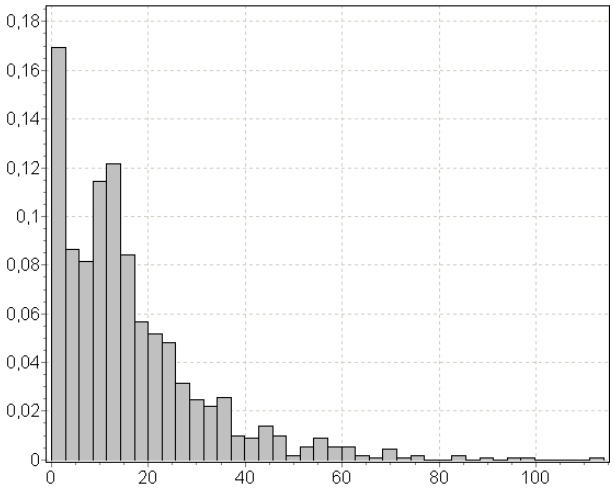
Rysunek 7.1: Wykresy gęstości (lewa kolumna) oraz kwantyl-kwantyl (prawa kolumna) dla: czasu pomiędzy awariami A, czasu wymiany E oraz czasu dostawy D dla przypadków 1,2,5,6. W etykietach podano graniczne wartości akceptacji testów λ Kołmogorowa p.



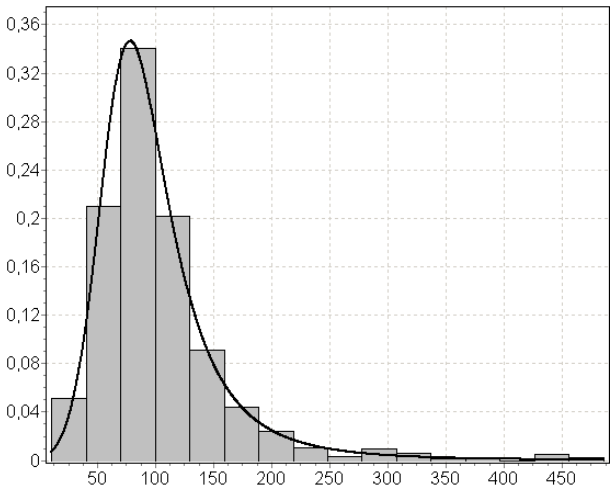
(a) Histogram i estymacja parametryczna awarii A



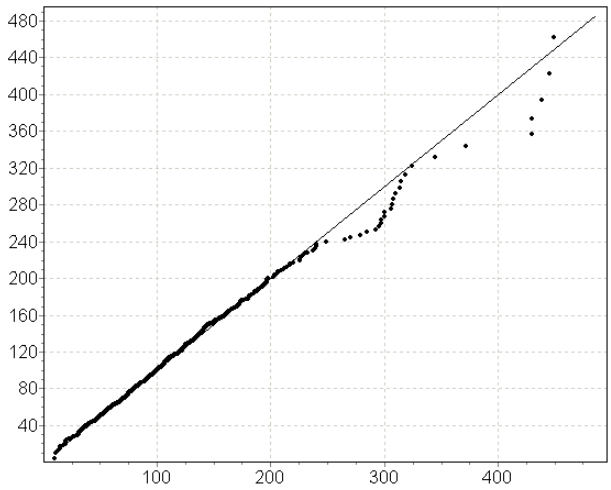
(b) Ocena dopasowania awarii A, $p = 0,025$



(c) Gęstość empiryczna wymiany E, brak dopasowania parametrycznego



(d) Histogram i estymacja parametryczna dostawy D



(e) Ocena dopasowania dostawy D, $p = 0,59$

Rysunek 7.2: Wykresy gęstości (lewa kolumna) oraz kwantyl-kwantyl (prawa kolumna) dla: czasu pomiędzy awariami A, czasu wymiany E oraz czasu dostawy D dla przypadków 3,4,7,8. W etykietach podano graniczne wartości akceptacji testów λ Kołmogorowa p.

7.3 Sieć Petriego wybranego systemu z rezerwą czasową i dostawą

Tradycyjne sieci Petriego, opierające się na jednym typie miejsc i przejść, nie wystarczają do opisu niezawodności systemu tramwajowego. Można wyróżnić dwie grupy problemów:

1. brak możliwości ilościowej specyfikacji czasu opóźnienia przy odpalaniu przejścia, co jest potrzebne przy modelowaniu: czasu pomiędzy awariami, czasu dostawy i wymiany oraz upłynięcia rezerwy czasowej,

2. konflikty przy identyfikacji wielu równolegle biegnących procesów dostawy i wymiany wynikające z nierozróżnialności podstawowych żetonów - przeprowadzenie dostawy powinno kończyć awarię określonego i konkretnego tramwaju.

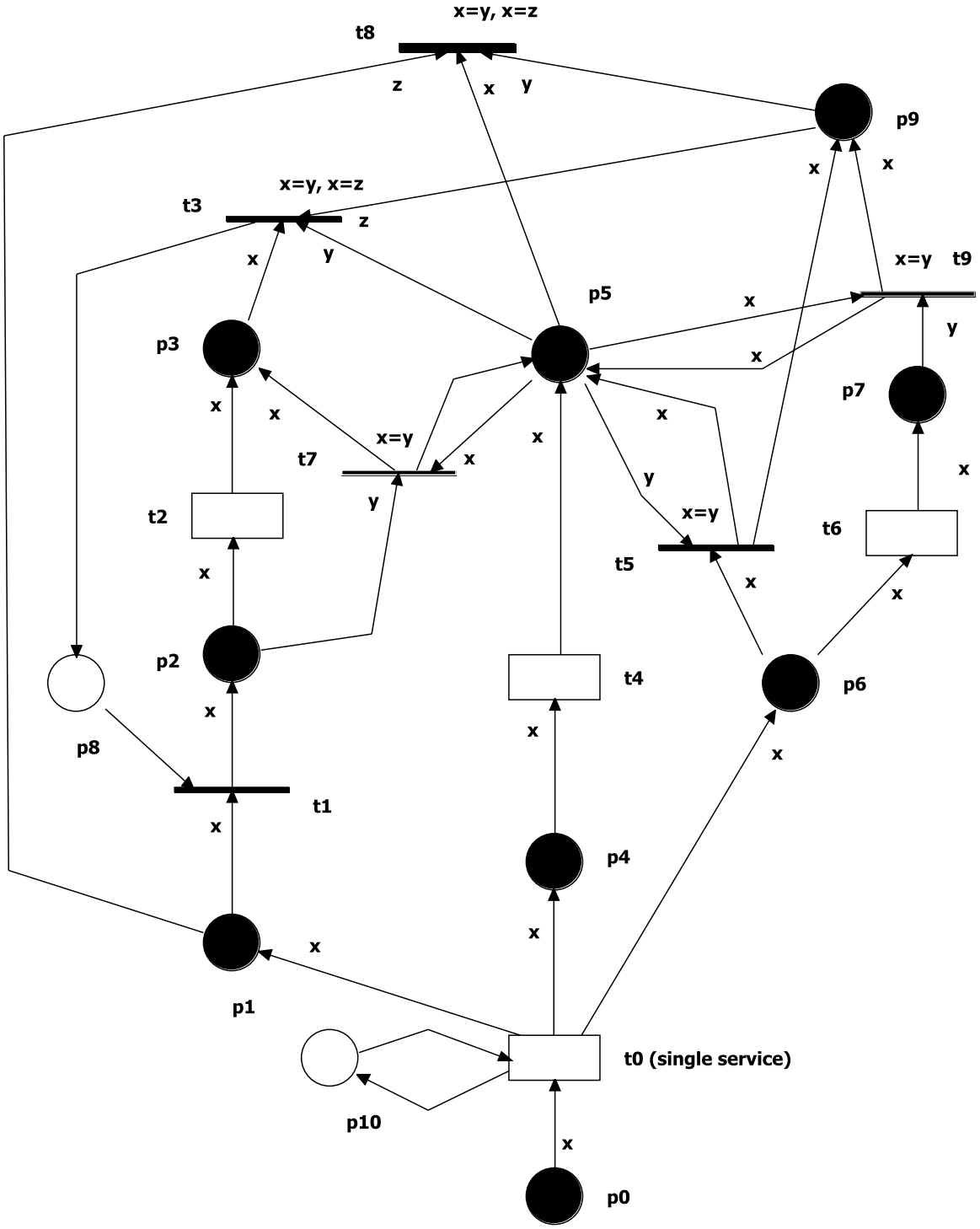
Pierwszą grupę problemów rozwiązano poprzez włączenie zdefiniowanych w pracy [120] przejść czasowych do języka. Takie elementy, w odróżnieniu od dotychczas stosowanych przejść natychmiastowych, mają przypisane zmienne losowe czasu odpalenia, po upłynięciu których żetony wejściowe są atomowo zabierane z miejsc wejściowych, a wstawiane do miejsc wyjściowych. Jeśli zarówno przejście czasowe jak i natychmiastowe jest gotowe do odpalenia, przejście natychmiastowe odpalane jest pierwsze. Symbolem graficznym przejścia natychmiastowego jest pozioma kreska (np. $t1$ na rys. 7.3), a przejścia czasowego - prostokąt bez wypełnienia (np. $t0$).

W kolorowanych sieciach Petriego [71] miejsca mogą mieć przypisany typ nazywany również zbiorem kolorów, a wtedy żetony w nich przechowywane posiadają przypisaną wartość z dziedziny tego typu (czyli kolor). W modelu systemu tramwajowego miejsca kolorowane pozwalają na identyfikację uszkodzonych tramwajów regularnych, podczas gdy miejsca proste służą do opisu nierozróżnialnych tramwajów rezerwowych. Na łukach wchodzących i wychodzących miejsc kolorowanych umieszcza się nazwy zmiennych lokalnych dla przejścia, co pozwala na pobieranie i umieszczanie w przylegających miejscach żetonów o odpowiednich wartościach. Przykładem może być przejście $t9$ na rys. 7.3 odpalane natychmiast, gdy na $p5$ znajdzie się żeton o wartości x równej wartości y podchodzącej z $p7$. Dodatkowo do $p9$ zostanie dodany żeton o tej wartości.

Ostatnie rozszerzenie języka związane jest z przejściem $t0$ na rys. 7.3, które jako jedyne posiada semantykę typu *single service* oznaczającą, że w danej chwili może zachodzić maksymalnie jednej proces odpalania. Zatem $t0$ może być odpalane w danej chwili tylko dla jednego żetonu z miejsca wejściowego $p0$, co spowoduje, że kolejne żetony będą dodawane do miejsc $p1$, $p4$ oraz $p6$ cyklicznie co czas odpalenia przejścia czasowego $t0$. Z kolei semantyka *multiple service* oznacza, że przejście może być odpalane wiele razy równolegle. Przykładowo dostawa zachodzi dla wszystkich uszkodzonych tramwajów jednocześnie, zatem $t4$ posiada semantykę *multiple service*.

Typy miejsc i przejść z modelu na rys. 7.3 wraz z opisem znaczenia umieszczono w tabelach 7.4 oraz 7.5. Znakowanie początkowe modelu obejmuje: żetony proste w miejscu $p8$ symbolizujące dostępność tramwaju rezerwowego oraz żetony parametryczne w miejscu $p0$ oznaczające awarię tramwaju w przyszłości.

Pierwsza zmiana w znakowaniu sieci będzie zatem wynikała z odpalenia $t0$, które, jak już wspomniano, cyklicznie generuje kolejne uszkodzenia. Umieszczenie żetonu w $p1$, $p4$ oraz $p6$



Rysunek 7.3: Sieć Petriego opisująca niezawodność systemu tramwajowego [98]

Tabela 7.4: Definicje miejsc sieci Petriego z rys. 7.3

Nazwa	**Zbiór kolorów**	**Znaczenie**
$p0$	liczby naturalne	żetony w tym miejscu identyfikują swoimi wartościami przyszłe uszkodzenia tramwajów
$p1$	liczby naturalne	występuje uszkodzenie identyfikowane przez żeton
$p2$	liczby naturalne	zachodzi wymiana tramwaju powiązanego z żetonem
$p3$	liczby naturalne	wymiana się zakończyła
$p4$	liczby naturalne	tramwaj związany z wartością żetonu jest naprawiany
$p5$	liczby naturalne	określony tramwaj został naprawiony
$p6$	liczby naturalne	rezerwa czasowa dla określonego uszkodzenia jeszcze nie upłynęła
$p7$	liczby naturalne	upłynęła rezerwa czasowa dla uszkodzonego tramwaju związanego z wartością żetonu
$p8$	brak	tramwaj rezerwowy pozostaje w gotowości
$p9$	liczby naturalne	zaszła dostawa oraz a) jeżeli rezerwa upłynęła: żeton z p7 został usunięty, b) jeżeli rezerwa nie upłynęła, to żeton z p6 został usunięty
$p10$	brak	generowany jest żeton oznaczający kolejne uszkodzenie pewnego tramwaju

Tabela 7.5: Definicje przejść sieci Petriego z rys. 7.3

Nazwa	**Typ przejścia**	**Znaczenie**
$t0$	czasowe	uszkodzenie tramwaju, czas odpalenia wynika z tab. 7.2
$t1$	natychmiastowe	alokacja tramwaju rezerwowego do uszkodzonego
$t2$	czasowe	wymiana tramwaju, czas odpalenia wynika z tab. 7.2
$t3$	natychmiastowe	zakończenie obsługi uszkodzonego tramwaju w sytuacji: dostawa nastąpiła po alokacji tramwaju rezerwowego
$t4$	czasowe	dostawa tramwaju, czas odpalenia wynika z tab. 7.2
$t5$	natychmiastowe	anulowanie odmierzania rezerwy czasowej, gdy nastąpiła już dostawa
$t6$	czasowe	odmierzanie rezerwy czasowej, rozpoczęcie hazardu, czas odpalenia wynika z tab. 7.2
$t7$	natychmiastowe	zakończenie wymiany, gdy pierwsza nastąpiła dostawa
$t8$	natychmiastowe	zakończenie obsługi uszkodzonego tramwaju w sytuacji: dostawa nastąpiła przed alokacją tramwaju rezerwowego
$t9$	natychmiastowe	zakończenie hazardu, gdy nastąpiła dostawa

powoduje rozpoczęcie odpowiednio procesów: oczekiwania na tramwaj rezerwowy, dostawy oraz odmierzania rezerwy czasowej.

Jeśli tramwaj rezerwowy jest dostępny, czyli istnieje jakikolwiek żeton w miejscu $p8$, to następuje alokacja rezerwy poprzez odpalenie przejścia $t1$ i umieszczenie żetonu w $p2$, co rozpoczyna wymianę. Czas wymiany opisywany jest przejściem $t2$, a jej zakończenie symbolizuje żeton w $p3$ o wartości parametru takiej jak uszkodzenie.

Równolegle z alokacją rezerwy oraz wymianą zachodzi dostawa. Powrót tramwaju na trasę oznaczony jest miejscem $p5$, do którego żeton dodawany jest po odpaleniu przejścia czasowego $t4$. Ponadto jeśli dostawa zaszła przed wymianą, to $t7$ przerwie odpalanie $t2$ poprzez usunięcie żetonu z $p2$ oraz oznaczy wymianę za zakończoną dodając żeton do $p3$. Warunek $x = y$ przy $t7$ jest niezbędny dla identyfikacji żetonów oznaczających zakończenie wymiany y i dostawy x tego samego uszkodzonego tramwaju.

Odmierzanie rezerwy czasowej odbywa się podobnym schematem co realizacja dostawy: $t6$ jest przejściem czasowym, po odpaleniu którego żeton umieszczany jest w $p7$, co oznacza trwanie hazardu dla określonego parametrem x uszkodzenia. Dostawa kończy hazard lub powoduje, że hazard dla tego uszkodzenia nie zajdzie. Stąd potrzebne są dwa alternatywne przejścia - $t5$ i $t9$. Pierwsze z nich wykona się, gdy dostawa zaszła przed upłynięciem rezerwy, czyli są żetony w $p5$ i $p6$. Drugie wykona się, gdy przed dostawą doszło do odpalenia $t6$, czyli w $p7$ jest żeton.

Niezależnie które przejście zostanie odpalone $t5$ czy $t9$, efektem będzie umieszczenie żetonu w $p9$ oraz zachowanie żetonu w $p5$. Stąd kandydatami do odpalenia są teraz $t3$ oraz $t8$. Pierwsze z nich wymaga żetonu w $p3$, czyli odpali się, gdy doszło do alokacji rezerwy. W takiej sytuacji $t3$ dodaje żeton do $p8$. Gdy jednak dostawa zaszła przed alokacją tramwaju rezerwowego, czyli wciąż istnieje żeton w $p1$, to przejście $t8$ usunie żetony z $p1$, $p5$ oraz $p9$. W obydwu przypadkach obsługa uszkodzenia zostanie ukończona, a naprawiony tramwaj powróci na trasę.

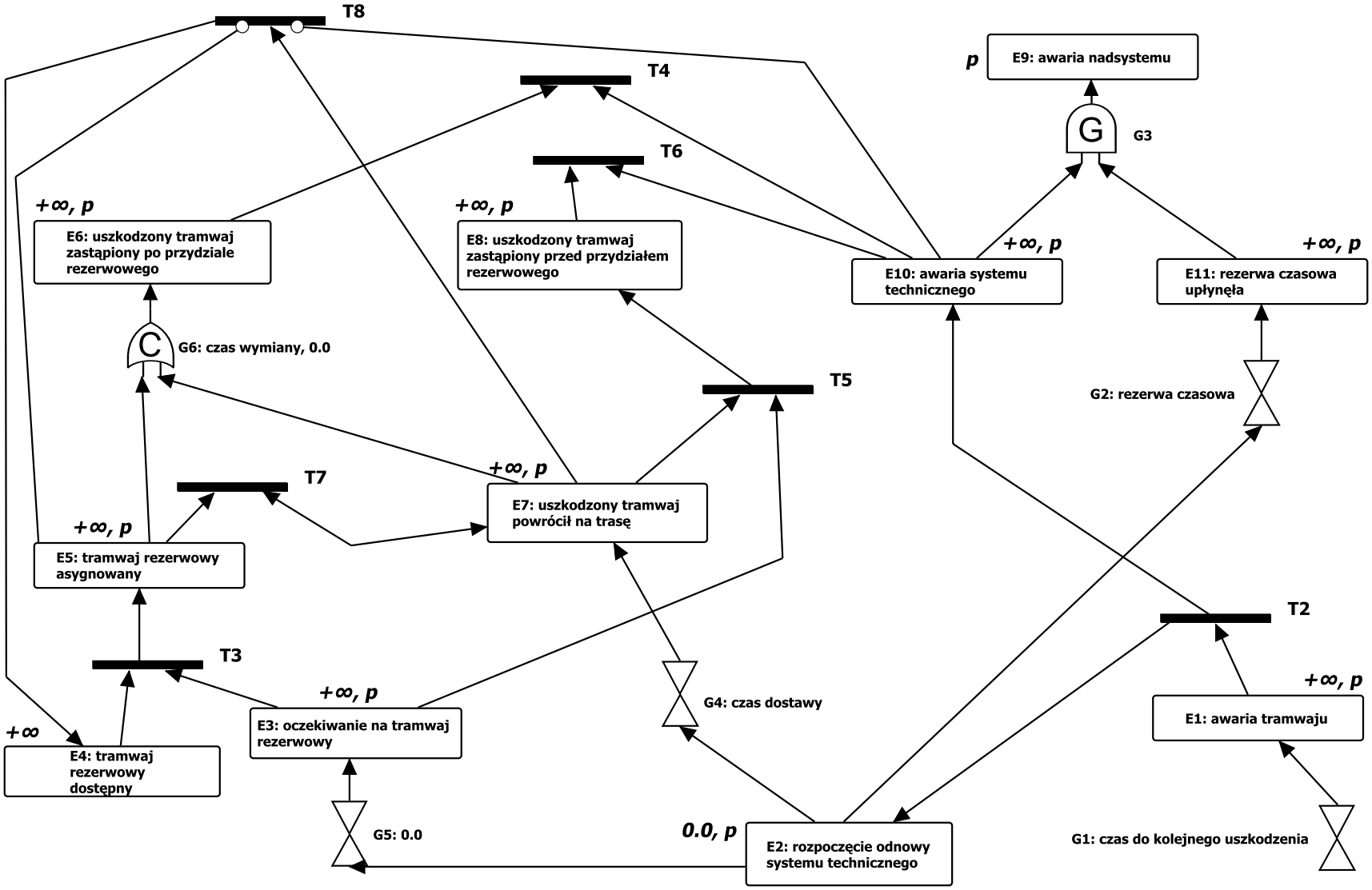
Eksperymenty obliczeniowe przeprowadzone na omówionej sieci udokumentowano w [98].

7.4 Badania symulacyjne GN

W niniejszym podrozdziale omówiono model w języku GN wyrażający niezawodność systemu tramwajowego, podobnie jak opisana powyżej sieć Petriego. Porównanie obydwu podejść przeprowadzono natomiast pod koniec rozdziału.

Awarie tramwajów kursowych są modelowane na rys. 7.4 bramą generatora $G1$ oraz jej zdarzeniem wyjściowym $E1$. W ten sposób kolejne zdarzenia parametryczne $E1$ symbolizujące uszkodzenia tramwajów rozpoczynane są w odstępach czasu wynikających z realizacji zmiennej losowej przypisanej do generatora. Jedynymi zajściami początkowymi w modelu są $E4$ symbolizujące gotowość rezerwy do przejęcia funkcji uszkodzonego tramwaju.

Zdarzenia $E1$ ulegają natychmiastowemu zakończeniu wskutek odpalenia przejścia $T2$, które równolegle rozpoczyna $E10$ oznaczające trwanie awarii systemu technicznego, jak i $E2$ inicjujące procesy obsługi uszkodzonego pojazdu. Z kolei $E2$ rozpoczyna odmierzanie rezerwy czasowej, której symbolem jest brama $G2$.



Rysunek 7.4: Model w języku GN opisujący niezawodność systemu tramwajowego [94]

Jeśli rezerwa czasowa upłynie, tj. rozpocznie się $E11$, a awaria systemu technicznego wciąż trwa, to zachodzi hazard. Takie znaczenie nadaje brama GAND $G3$ oraz zdarzenie oznaczające faktyczny hazard nadsystemu $E9$.

Pomyślne przeprowadzenie dalej opisanych procesów obsługi rozpoczętych zdarzeniem $E2$ zakończy $E10$, czyli - jeśli upłynęła rezerwa czasowa tego tramwaju - zakończy również hazard.

Zainicjowane przez $E2$, dostawa i wymiana przebiegają równolegle. Z jednej strony brama $G4$ odmierza czas dostawy uszkodzonego tramwaju z powrotem na trasę, a z drugiej trwa oczekiwanie na wolny tramwaj rezerwowy ($E3$). Jeśli dostawa zakończy się przed przydziałem rezerwowego (czyli zachodzi $E7$ oraz $E3$), to przejście $T5$ rozpocznie $E8$ kończąc $E7$ i $E3$. Następnie $T6$ zatrzyma wspomniane $E10$.

Jeśli tramwaj rezerwowy stanie się dostępny przed zakończeniem dostawy, czyli zajdzie $E4$, to przejście $T3$ przydzieli rezerwę do uszkodzonego tramwaju i rozpocznie $E5$. To zdarzenie prowadzi do lewego wejścia bramy $G6$, która w tym przypadku zacznie odmierzać czas wymiany uszkodzonego tramwaju na rezerwowy. Jeśli w czasie wymiany zakończy się dostawa, to prawym wejściem bramy COR dojdzie do natychmiastowego rozpoczęcia $E6$ i zakończenia wymiany. W tym przypadku zatem dostawa spowoduje zakończenie $E10$ poprzez przejście $T4$. Należy jednak pamiętać, że doszło do alokacji tramwaju rezerwowego, który teraz musi zostać zwrócony do puli modelowanej przez $E4$. Taką funkcję pełni $T8$, które jednak pozostawało zablokowane łukiem hamującym aż do momentu zakończenia $E10$. Odpalenie $T8$ uwarunkowane jest również trwaniem $E7$, czyli - prawidłowo - nie dojdzie do zwrotu elementu, gdy nie został on wcześniej zaalokowany, gdyż $T5$ w takiej sytuacji kończyłoby $E7$. Łuk hamujący łączący $E5$ z $T8$ powoduje, że $T4$ kończące $E5$ odpali się przed $T8$.

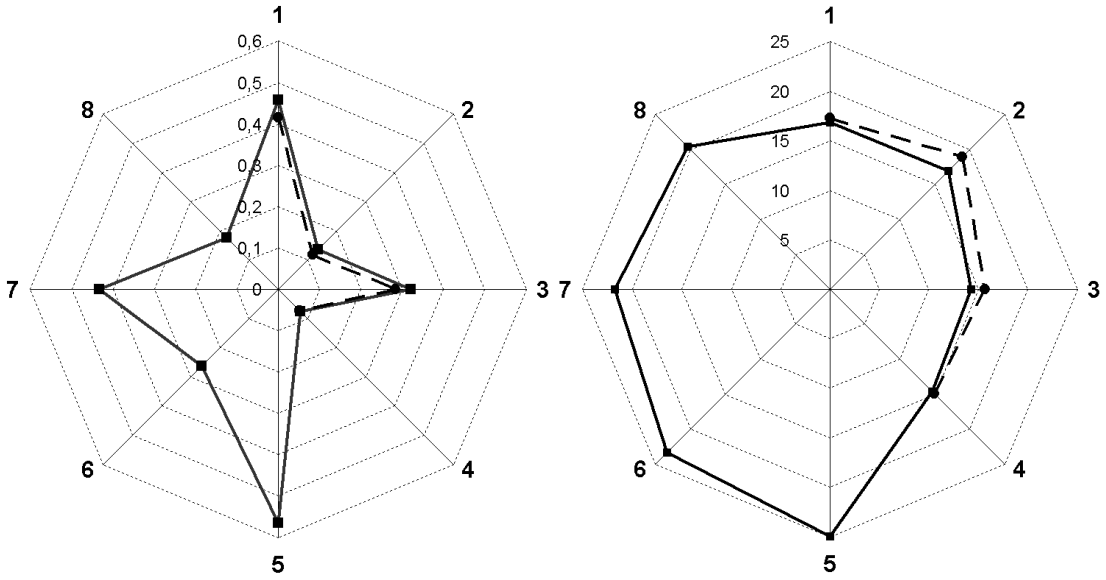
7.5 Omówienie wyników

Dyskusję kompletności i poprawności modelu względem rzeczywistości zawarto w podrozdziale 7.5.1. Natomiast porównania modelu GN z siecią Petriego dokonano w 7.5.2.

7.5.1 Zgodność wyników symulacyjnych z rzeczywistymi

Z wykresu 7.5a wynika wyraźna tendencja do spadku prawdopodobieństwa hazardu wraz ze wzrostem rezerwy czasowej. Dotyczy to obydwu założeń dotyczących początku awarii tramwaju i jest widoczny przy porównaniu przypadku 1 z 2, jak i 3 z 4. Czego również należało oczekiwać, prawdopodobieństwo hazardu wzrosło przy mniejszej liczbie tramwajów rezerwowych (przypadki 5-8), ale wciąż wyniki maleją, gdy rośnie rezerwa czasowa.

Średni czas hazardu (rys. 7.5b) rośnie, gdy maleje liczba pojazdów rezerwowych. Ponadto zmiany w wartościach bezwzględnych dużo bardziej zależą od liczby tramwajów rezerwowych niż od definicji chwili awarii czy wielkości rezerwy czasowej. Niższe wartości średnie czasu hazardu zwracane przez symulator uzasadniają dlaczego histogramy wyników uzyskanych przy pomocy GN na rys. 7.6 mają nieco lżejszy „ogon".



(a) Prawdopodobieństwo hazardu (b) Średni czas hazardu

Rysunek 7.5: Porównanie prawdopodobieństwa hazardu i średniego jego czasu w minutach dla 8 przypadków testowych, linia ciągła - wyniki symulacyjne, linia przerywana - dane rzeczywiste

Maksymalne błędy modelu względem względem danych rzeczywistych dla przypadków 1-4 wynoszą 14,17% dla prawdopodobieństwa hazardu oraz 10,98% dla średniego czasu jego trwania. Średnie wartości tych błędów to odpowiednio 10,7% oraz 6,3%.

Tabela 7.6: Porównanie symulacyjnych prawdopodobieństw hazardu dla zmiennych parametrycznych i empirycznych. W nawiasach błędy względem wyniku rzeczywistego.

	Prawdopodobieństwo hazardu		
Przypadek	Wyniki rzeczywiste	Symulacja	
		Zmienne parametryczne	Zmienne empiryczne
1	0,415	0,457 (10,12%)	0,439 (5,78%)
2	0,120	0,137 (14,17%)	0,129 (7,50%)
3	0,283	0,320 (13,07%)	0,309 (9,19%)
4	0,073	0,077 (5,48%)	0,075 (2,74%)
5		0,565	0,540
6		0,262	0,242
7		0,432	0,419
8		0,177	0,180

Wyniki testów λ Kołmogorowa wskazują, że dla 5 parametrów modelu zmienne losowe zostały bardzo dobrze dopasowane, co liczbowo ujmują przedstawione na rysunkach 7.1 oraz 7.2 duże wartości parametru p. Efektywna estymacja parametryczna czasu wymiany, kiedy za moment awarii przyjmuje się chwilę zgłoszenia okazała się niemożliwa. Gdy było to technicznie możliwe dyspozytor umyślnie pozostawiał uszkodzony tramwaj na trasie efektywnie wydłużając rezerwę czasową. Generuje to dużą masę prawdopodobieństwa zdarzenia, że czas wymiany jest równy 0, a histogram ma dwa wyraźne maksima (rys.7.2c). Żaden analizowany rozkład parametryczny

Tabela 7.7: Porównanie symulacyjnych średnich czasu hazardu dla zmiennych parametrycznych i empirycznych. W nawiasach błędy względem wyniku rzeczywistego.

Prawdopodobieństwo hazardu			
Przypadek	Wyniki rzeczywiste	Symulacja	
		Zmienne parametryczne	Zmienne empiryczne
1	17,30	16,83 (2,72%)	16,64 (3,82%)
2	18,94	16,86 (10,98%)	17,07 (9,87%)
3	15,66	14,25 (9,00%)	15,33 (2,11%)
4	14,93	14,56 (2,48%)	13,90 (6,90%)
5		24,90	24,00
6		23,27	22,67
7		21,70	22,25
8		20,36	19,73

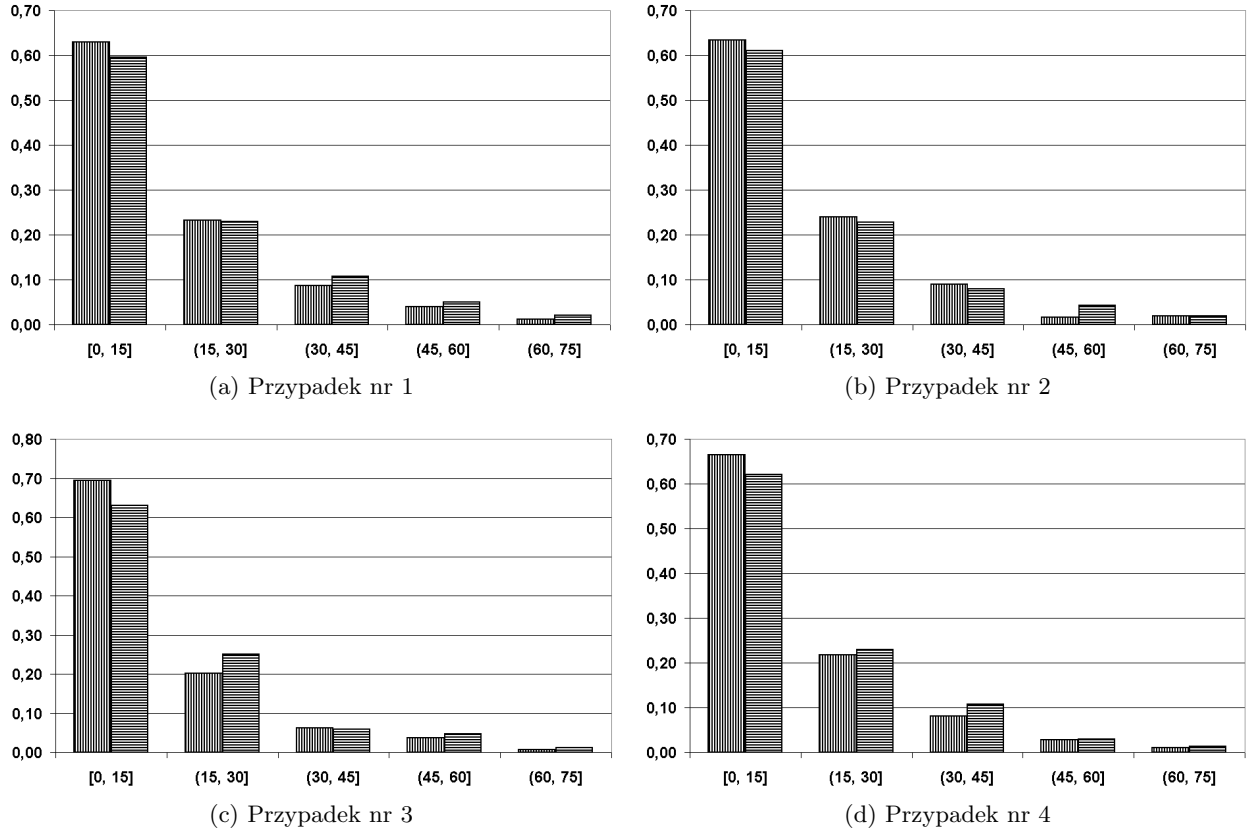
nie osiągnął w teście λ Kołmogorowa akceptowalnej wartości, stąd zdecydowano się na przyjęcie gęstości empirycznej. Powyższe spostrzeżenie uzasadnia jednocześnie dlaczego bezwzględna wartość prawdopodobieństwa hazardu w przypadkach nr 3 i 4 jest niższa niż, odpowiednio, w przypadkach 1 i 2.

Jak wynika z tabel 7.6 oraz 7.7 wyniki symulacji nie są zbieżne z rzeczywistymi, nawet gdy wszystkie zmienne mają postać empiryczną. Niezgodności wynikają zatem albo z błędów przy obliczeniach albo z uproszczeń modelu. Należy podkreślić, że uzyskanie wiarygodnych wyników dla czasu hazardu jest zadaniem dużo trudniejszym niż dla jego prawdopodobieństwa. Przykładowo, w 4. przypadku tylko 7,3% (tab. 7.6) rzeczywistych uszkodzeń mogło zostać poddanych analizie statystycznej. Jeśli rezerwa czasowa byłaby równa 70 min, to posiadane dane rzeczywiste zmuszałyby do liczenia średniej tylko z 14 awarii, co wyklucza eksperyment. To z powyższych przyczyn odrzucono dni wolne w badaniach. Z drugiej strony, problem natury technicznej zaczyna ograniczać obliczenia symulacyjne i numeryczne, gdzie przy dużych rezerwach pojawia się zjawisko zdarzeń rzadkich lub rosną wymagania pamięciowe. Dlatego też wyniki weryfikowano dwoma symulatorami: wygenerowanym z GN według schematu opisanego w dodatku A i niezależnym symulatorem zbudowanym na potrzeby systemu tramwajowego. W ramach weryfikacji z powodzeniem stosowano również dwie opisane w rozdziale 9 metody estymacji, które dostarczają wiarygodnych wyników nawet dla rezerwy czasowej równej 101 minut.

Po dogłębnej analizie danych wejściowych nie ulega wątpliwości, że przyczyny błędów wynikają w dużej mierze z roli czynnika ludzkiego przy podejmowaniu decyzji o zastępowaniu tramwaju, który nie jest uwzględniany w żadnym z modeli. Przykładowo zauważono, że:

1. gdy stwierdzono, że prawdopodobny czas dostawy będzie krótki, wstrzymywano wysłanie rezerwy, co oznacza, że ten tramwaj mógł chwile później obsługiwać inne uszkodzenie,

2. gdy naprawiony tramwaj był już bliski wjechania na trasę, a w pobliżu zaszło inne uszkodzenie dyspozytor wstrzymywał wysłanie tramwaju zapasowego do nowego uszkodzenia aż zwolniła się rezerwa najbliższa geograficznie,



(a) Przypadek nr 1

(b) Przypadek nr 2

(c) Przypadek nr 3

(d) Przypadek nr 4

Rysunek 7.6: Histogramy czasów niezdatności, lewa kolumna wynik symulacyjny, prawa kolumna wynik rzeczywisty, przedziały na osi odciętych w minutach

3. część uszkodzeń, szczególnie porannych, trwała 2-3 minuty, co prawdopodobnie oznaczało błąd w rozruchu lub błąd przejściowy wynikający z zaawansowanego wieku pojazdu,

4. choć ewidencja awarii była w 2001 roku prowadzona w formie elektronicznej, to nie istniał jeszcze system wspomagający dyspozytora w zapisie zdarzeń i w rezultacie w danych historycznych znajdują się czasem przekłamania. Gdy zauważono pomyłkę, to dotknięte dane były usuwane przed analizą statystyczną, jednak pełna korekta danych nie była możliwa.

Wysoka rola czynnika ludzkiego wynika po części z zaawansowanego wieku taboru tramwajowego we Wrocławiu w 2001 roku i konieczności ciągłego nadzoru systemu przez człowieka. Zasadne są zatem badania jakości uzyskanych rozwiązań w zależności od parametrów wejściowych zmiennych losowych, które z pewnością ulegną zmianie po wdrożeniu projektów takich jak Tramwaj+ oraz Inteligentny Transport Miejski mających szansę istotnie poprawić jakość funkcjonowania systemu tramwajowego we Wrocławiu. Dwie metody estymacji niezawodności dla ogólnego systemu z rezerwowaniem czasowym przedstawiono w rozdziale 9.

7.5.2 Porównanie sieci Petriego z modelem GN

Odbiorcy, którym przedstawiono sieć Petriego oraz tożsamy model GN zgodnie twierdzili, że drugi model jest łatwiejszy w odbiorze. Zwracano uwagę na następujące różnice:

1. przejście czasowe $t0$ działa inaczej niż pozostałe przejścia prezentowanej sieci Petriego, a mimo to ma taki sam symbol graficzny, co powodowało, że odbiorcy już na początku popełniali błąd w analizie modelu; problem ten rozwiązano w GN poprzez wprowadzenie bramy generatora o składni nawiązującej do zbliżonej znaczeniowo bramy opóźniającej, ale bez łuków wejściowych,

2. koncepcje „zdarzenia" i jego „zajścia" mocno związane z analizowanym systemem okazały się bliższe odbiorcom niż „miejsce" oraz „żeton" będące jedynie pojęciami języka sieci Petriego: stwierdzenie „zaszło zdarzenie $E9$ oznaczające hazard" bardziej przemawiało do odbiorców niż: „na miejscu $p7$ pojawił się żeton oznaczający hazard",

3. przejście w sieciach Petriego ma bardzo dużą moc wyrazu, która przy opisie prostych zjawisk okazuje się zbędna i nader komplikuje czytanie modelu; przykładem może być przejście $t4$ modelujące jedynie upływ czasu; brama opóźniająca w GN ma jasną interpretację akcentowaną przez symbol klepsydry,

4. przejścia natychmiastowe w sieciach Petriego również mają wiele kontekstów użycia: $t3$ ma charakter logiczny *AND*, podczas gdy $t5$ oraz $t9$ są alternatywne względem siebie, choć realizują tę samą funkcję. Pierwsze z nich odpali się, gdy hazard nie zajdzie, a drugie gdy hazard zaistnieje; GN mają w tym aspekcie dużą przewagę dzięki bramom logicznym, których uznane w inżynierii symbole jednoznacznie identyfikują ich funkcje,

5. choć zarówno bramy przyczynowe jak i uogólniające można wyrazić jako podsieci Petriego, to wykluczenie drugiej grupy szczególnie ujemnie wpływa na odbiór modelu; bramy uogólniające pozwalają na realizacje funkcji logicznych, np. stanu systemu, czego przykładem jest brama $G3$ i jej zdarzenie wyjściowe $E9$,

6. w opisach przejść wykorzystujących kolorowane żetony wejściowe istnieje zazwyczaj wiele symboli na łukach, których wartości są ograniczone wynikami porównań; symbole mają zazwyczaj lakoniczne nazwy i nie odnoszą się w żaden sposób do systemu, a ich mnogość nie tylko uniemożliwia nadanie dłuższych nazw, ale również odwraca uwagę odbiorcy od istoty modelu.

Sieć Petriego sprawiała trudności w analizie nawet Informatykom, którzy wcześniej nie modelowali z użyciem tego języka. Nic zatem dziwnego, że Eksperci z dziedziny transportu wyraźnie optowali za rozwiązaniem bazującym na GN, choć wskazywali również niedoskonałości.

Wspólną wadą modeli jest brak odwołania do struktury systemu technicznego. Aby czytać model niezawodnościowy systemu należy najpierw zrozumieć jego budowę, w czym żaden z obydwu języków nie pomaga. Ponadto gdy zbiory kolorów lub parametry wymagają do opisu więcej niż jednej składowej należy wykorzystać iloczyn kartezjański w ich definicjach (podobnie jak w dodatku A), co dodatkowo komplikuje modele. Problem omówiono głębiej przy konstrukcji kolejnej wersji języka w następnym rozdziale.

Rozdział 8

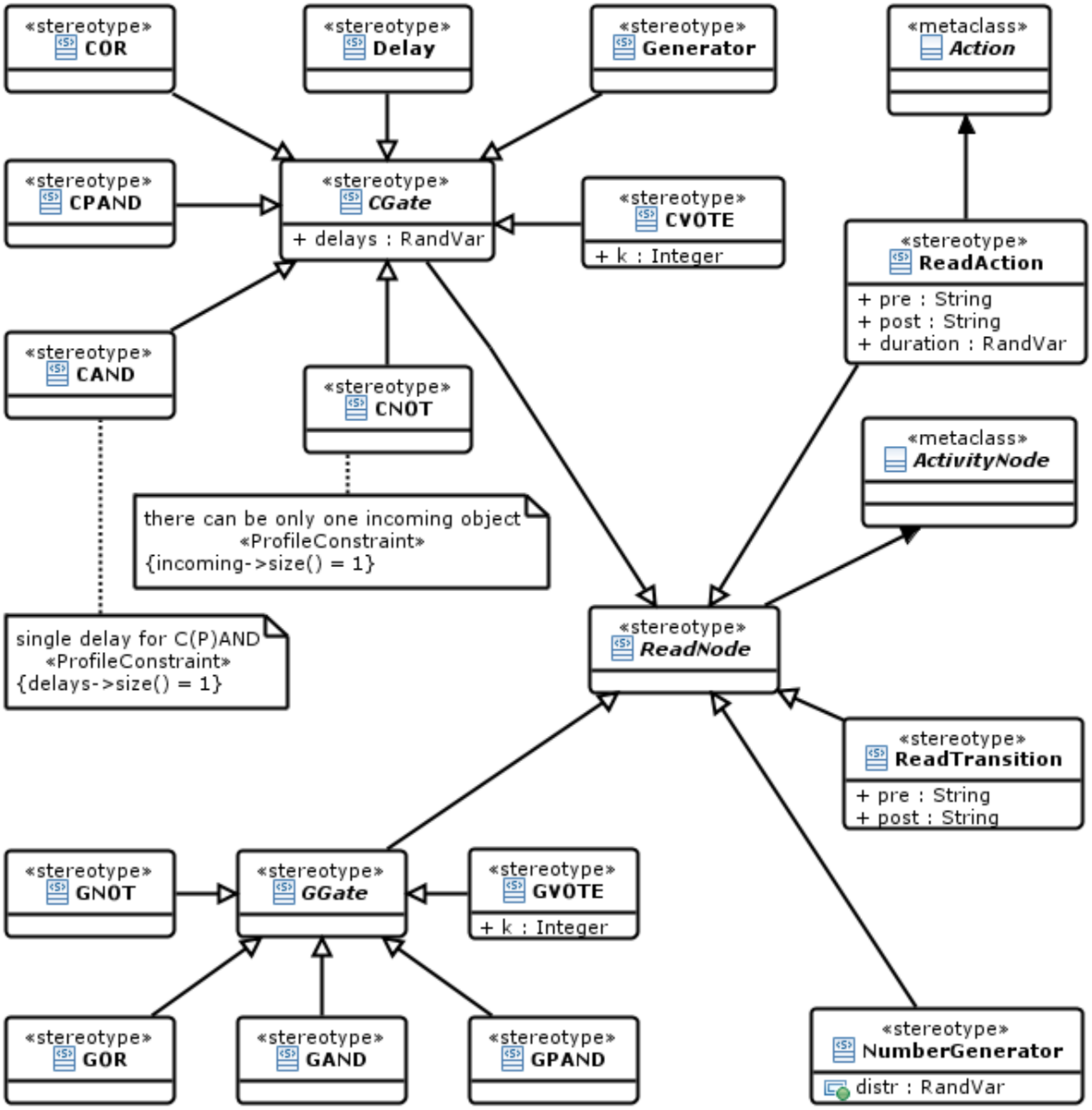
Etap 3: Rozszerzone Niezawodnościowo Diagramy Aktywności

W rozdziale zdefiniowano trzeci zestaw rozszerzeń metody analizy niezawodności systemów z zależnościami czasowymi wywodzącej się z drzew niezdatności. W ostatnim etapie nastąpiła integracja wyników uzyskanych w poprzednich rozdziałach z opracowanym przez OMG narzędziem UML. Celem było takie zwiększenie mocy wyrazu budowanej metody, aby możliwe było precyzyjne wyrażanie procesów z zależnościami czasowymi powszechnie występujących w rozległych systemach technicznych. Spośród dostępnych w UML'u metod opisu dynamiki wybrano diagramy aktywności (DA) będące według [152] jednym z trzech prominentnych języków modelowania procesów biznesowych. Prezentowany język o nazwie Rozszerzone Niezawodnościowo Diagramy Aktywności - *ang. Reliability Enhanced Activity Diagrams* (READ) to zatem rozszerzenie DA w kierunku niezawodności.

W rozdziale omówiono nie tylko definicję składni, ale również ostateczną semantykę translacyjną metody wyrażoną na gruncie innego formalizmu. Aby utrzymać dyskusję w nurcie niezawodności obydwa aspekty zostaną rozpatrzone w kontekście omówionych do tej pory polityk napraw oraz systemu tramwajowego. Zastosowanie READ do zupełnie nowego problemu zaprezentowano w rozdziale 10. Uzupełnieniem opisu języka jest wprowadzenie do wykorzystywanego oprogramowania: CPN Tools oraz Snoopy zamieszczone w dodatku C.

8.1 Składnia READ jako profil UML

Język UML zawiera w sobie opartą na stereotypach, tagach i ograniczeniach metodę swojego rozszerzenia, a zbiór takich logicznie powiązanych elementów nazywa się profilem. Ponieważ ani drzewa niezdatności, ani sieci Petriego nie zawierają w sobie podobnego mechanizmu, a łączenie trzech języków wymaga precyzyjnego opisu składni, zdecydowano się na zastosowane odwrotnego niż w GN podejścia. W READ centralnym językiem jest UML, a elementy drzew niezdatności i sieci Petriego zostały do niego dodane przy pomocy profilu, co ilustruje rys. 8.1.



Rysunek 8.1: Profil UML stanowiący składnię READ [95]

Wszystkie węzły diagramów aktywności są specjalizacjami *ActivityNode*, a więc to tę klasę rozszerza abstrakcyjny stereotyp *ReadNode* będący podstawą całego profilu. Ponieważ każdy stereotyp jest jednocześnie klasą, to można go obiektowo specjalizować. Kolejną warstwę tworzą zatem wciąż abstrakcyjne stereotypy *GGate* oraz *CGate*, które ostatecznie są specjalizowane przez konkretne bramy uogólniające oraz przyczynowe. Bezpośrednimi potomkami *ReadNode* są *ReadTransition* oraz *NumberGenerator* modelujące odpowiednio przejście oraz generator liczb losowych. Przykładem użycia drugiej konstrukcji jest brama $G1$ na rys. 10.3, która po atomowym odpaleniu umieszcza w buforze wyjściowym o typie całkowitoliczbowym wartość wylosowaną według zmiennej reprezentowanej przez *distr*. Ostatni stereotyp *ReadAction* rozszerza natomiast klasę *Action*, gdyż mocno nawiązuje działaniem do swojego odpowiednika z DA.

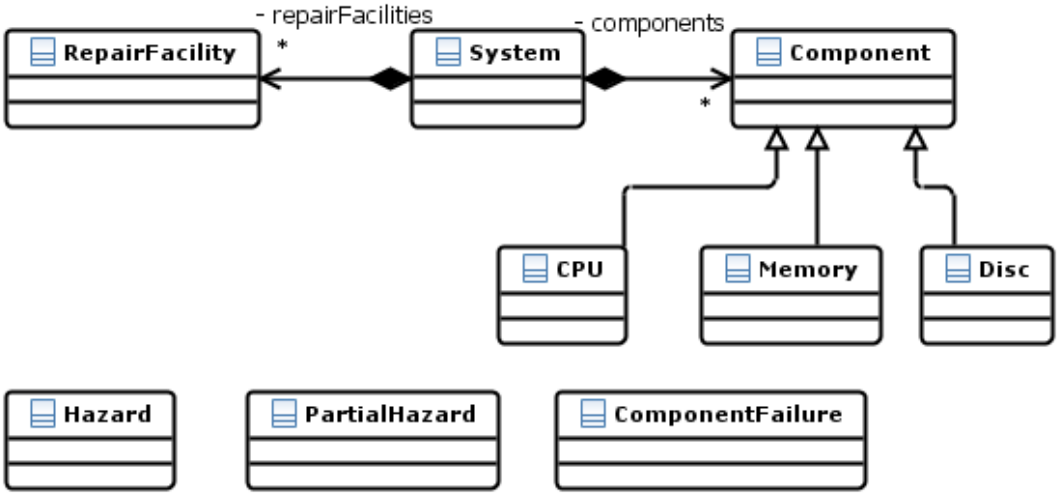
Z powyższego wynika, że po aplikacji profilu do wybranego diagramu aktywności każdy węzeł można oznakować nieabstrakcyjnym stereotypem bramy lub przejścia a akcję również stereotypem *ReadAction*. Kolejnym krokiem w budowie modelu READ jest definicja wartości atrybutów dodanych przez stereotypy, na przykład *delays* dla bram przyczynowych, czyli utworzenie tzw. wartości otagowanych. Wiele spośród omówionych dalej ograniczeń składniowych można wyrazić w języku OCL. Przykładem jest wyrażenie przypisane do bramy CAND weryfikujące, czy takiej bramie przypisano tylko jedną zmienną losową czasu odpalenia.

W kolejnych podrozdziałach omówiono szerzej poszczególne grupy elementów READ.

8.1.1 Elementy READ bazujące na konstrukcjach UML

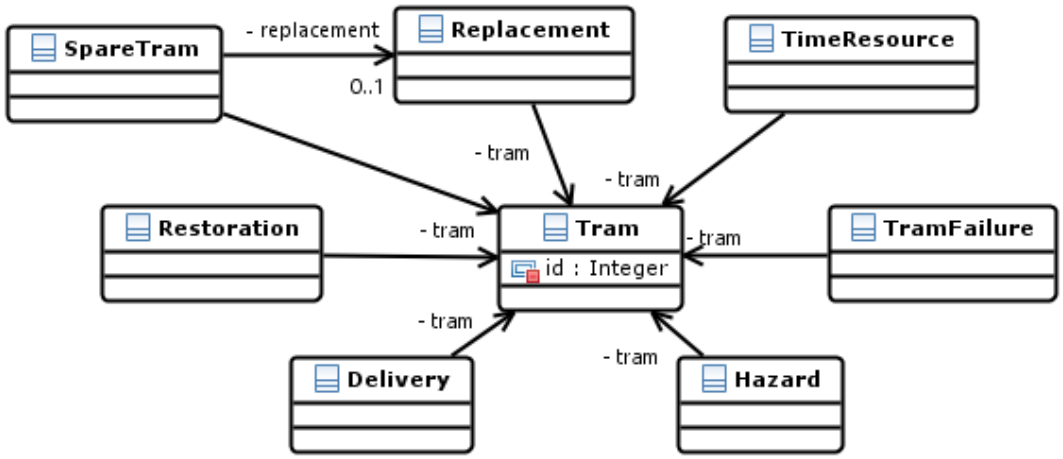
Jednoczesne wykorzystanie wielu typów diagramów UML pozwala na opis tego samego systemu z odmiennych perspektyw. Elementy tych diagramów często odwołują się do siebie, co ułatwia zachowanie i sprawdzanie spójności całego modelu. Podobnie jest w READ, gdzie głównym językiem są DA, ale wspomagającą rolę przyjmują diagramy klas oraz maszyny stanowe.

Istotną zmianą w języku READ w stosunku do wcześniejszych jego wersji jest izolacja opisu struktury systemu od modelu jego niezawodności. Diagram klas będący w READ środkiem wyrazu struktury przejmuje rolę słownika diagramów aktywności dostarczając definicji bytów, ich atrybutów, związków między klasami oraz pośrednio stanów w których mogą znajdować się obiekty. Asocjacje pomiędzy klasami często implikują propagację uszkodzeń, a więc analiza diagramu klas pomaga zrozumieć i rozszerzać diagram aktywności. Dla problemów niezawodności systemów komputerowego oraz tramwajowego diagramami struktury są modele na rys. 8.2 i 8.3.



Rysunek 8.2: Diagram klas naprawialnego systemu komputerowego

Elementem READ łączącym diagramy aktywności, klas oraz stanów jest *bufor centralny*, w którym przechowywane są obiekty określonych klas w jednym ze zbioru możliwych stanów. Akcje i przejścia dodają oraz usuwają obiekty z buforów tym samym aktualizując ich stany. Natomiast bramy tworzą nowe obiekty w odpowiednich stanach umieszczane w buforach wyjściowych. Bufor



Rysunek 8.3: Diagram klas systemu tramwajowego

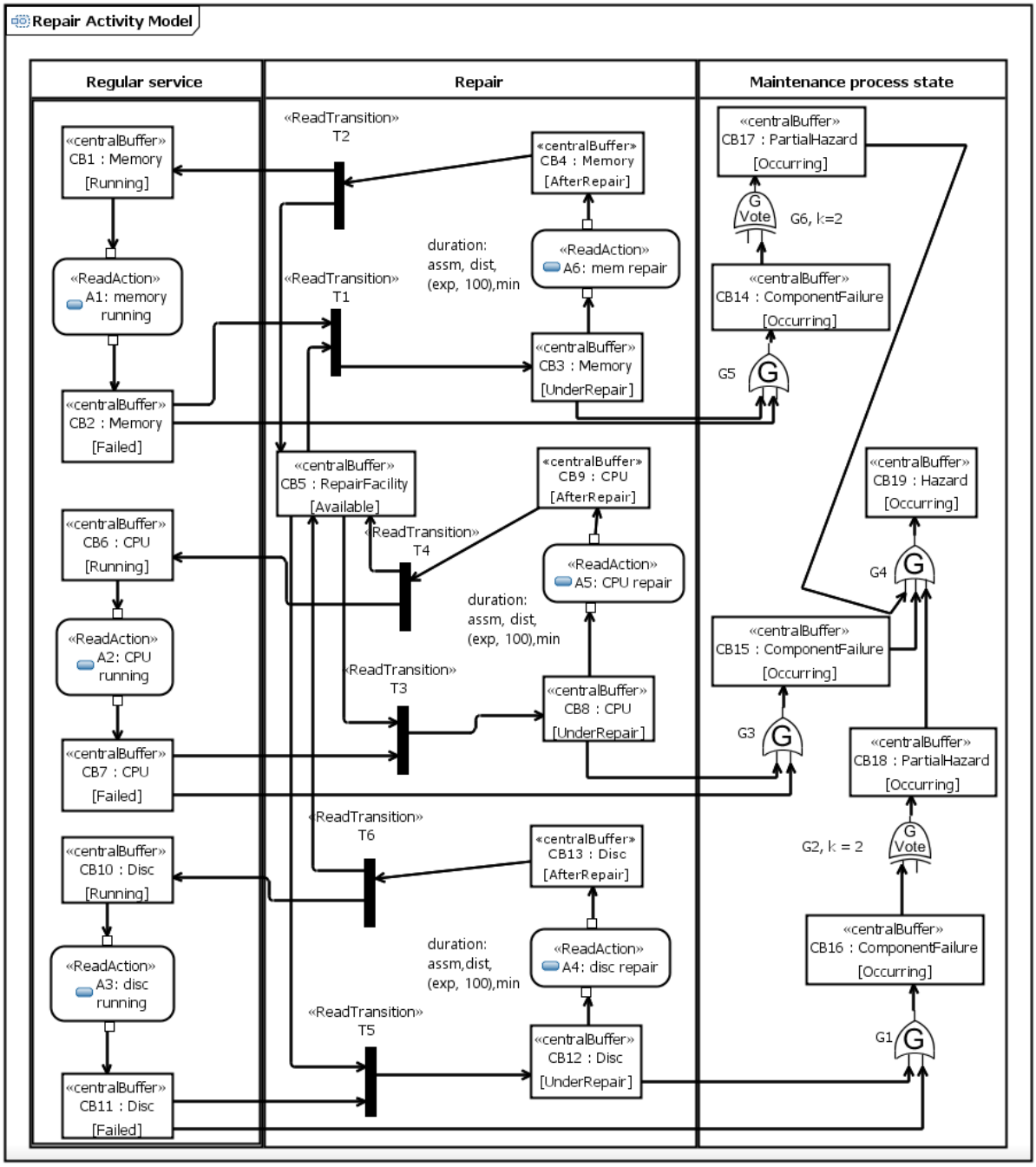
centralny to zatem uogólnienie zdarzenia parametrycznego GN takie, że rolę parametru przyjmuje klasa. Odpowiednikiem wartości parametru jest natomiast obiekt tej klasy. Ponieważ klasy w buforach mogą mieć zarówno atrybuty, jak i referencje do innych klas, logika wykonania akcji, bram i przejść jest bardziej skomplikowana niż w GN i zostanie omówiona w 8.2.

Maszyny stanowe mają w READ wąskie zastosowanie i ograniczają się do gromadzenia wszystkich możliwych stanów obiektów pewnej klasy. Istnieje zatem jedna maszyna dla każdej klasy występującej w buforze. Wiązaniem pomiędzy klasą a maszyną stanową jest referencja $classifierBehaviour$ elementu $Class$ metamodelu UML.

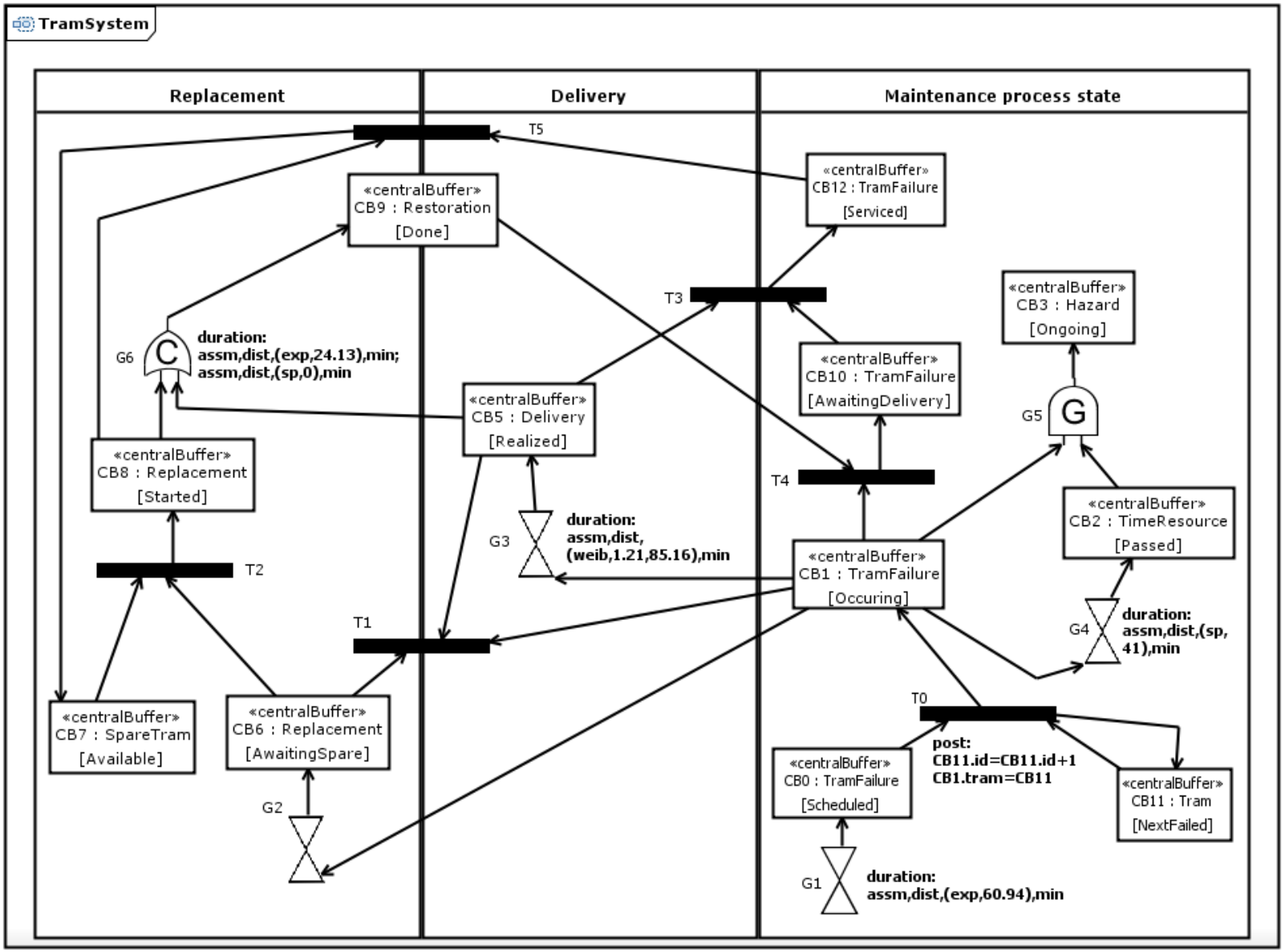
Ograniczeniem składniowym w READ jest założenie, że para $\langle klasa, stan \rangle$ jednoznacznie identyfikuje bufor centralny w modelu. W przeciwnym wypadku stan obiektu nie reprezentowałby etapu procesu eksploatacji, w którym uczestniczy ten obiekt. Z podobnych względów wprowadzono ograniczenie, że bufor centralny w READ ma przypisany dokładnie jeden stan, choć oryginalnie specyfikacja UML dopuszcza ich wiele.

Kolejnym elementem READ o silnych korzeniach w DA jest akcja. Stereotyp *ReadAction* dodaje w stosunku do swojego elementu macierzystego model czasowy opisany zmienną *duration*, a także warunki wstępne oraz efekty umieszczone w atrybutach *pre* i *post*. Klasa $RandVar$ zostanie omówiona w podrozdziale 8.1.4, a język wyrażeń w 8.1.5.

Konstrukcje READ są logicznie pogrupowane w pionowe tory aktywności w zależności od funkcji jakie pełnią. W ramach modelu polityk napraw (rys. 8.4) wyróżnia się tory: *Regular service*, *Repair* oraz *Maintenance process state*. Choć sama klasyfikacja nie wnosi niczego do semantyki READ, tory są konstrukcją poprawiającą czytelność i pozwalającą na szybsze przyswojenie opisywanego procesu. Niektóre elementy mogą należeć do dwóch torów, na przykład $T1$ lub $CB9$ na rys. 8.5, co symbolizuje związki pomiędzy torami na etapie procesu odpowiadającemu wykonaniu tych elementów.



Rysunek 8.4: Model w języku READ polityk napraw



Rysunek 8.5: Model w języku READ niezawodności systemu tramwajowego

8.1.2 Przejście wzorowane na sieciach Petriego

Podstawowy język DA przewiduje dwie metody podziału oraz scalania przepływów. W *podejściu jawnym* wykorzystuje się elementy $Fork$ i $Join$, pomiędzy którymi ścieżki wykonują się równolegle. Alternatywą jest *metoda niejawna*, którą można rozpoznać po akcji z wieloma krawędziami wyjściowymi (rozgałęzienie) oraz po akcji z wieloma krawędziami wejściowymi (scalenie). Zarówno metoda jawna, jak i niejawna są atomowe.

W niezawodności systemów czasu rzeczywistego często wymagane jest wyrażenie zależności pomiędzy danymi a sterowaniem. Niestety specyfikacja UML nie przewiduje mieszania sterowania z przepływem danych dla elementów $Fork$ i $Join$. Stąd w READ zdecydowano się na wprowadzenie nowej konstrukcji wzorowanej na przejściu znanym z sieci Petriego i zweryfikowanej na gruncie GN. Dla odróżnienia nowej konstrukcji od przejścia obecnego w diagramach stanów, element nazwano $ReadTransition$. Natomiast ze względu na podobieństwo w działaniu, symbol graficzny $Fork$, $Join$ i $ReadTransition$ pozostaje taki sam.

W UML nie istnieje obecnie składnia konkretna języka wyrażeń i akcji, co powoduje, że stosowanie pary $Fork/Join$ z atrybutami $guard$ i $effect$ wymaga uprzedniego projektu gramatyki. W rozdziale 8.1.5 zaproponowano język wyrażeń wykorzystywany w atrybutach *pre* i *post* przejść. Pierwszy z nich opisuje zależności pomiędzy obiektami w wejściowych buforach centralnych, a drugi specyfikuje własności obiektów w buforach wyjściowych, co demonstruje przejście $T0$ na rys. 8.5.

Ostatecznie zatem przejście w READ ma moc ekspresji znacznie wyższą niż w GN, gdyż nie ogranicza się do sytuacji, gdy wartości parametrów są sobie równe. Ponadto dostępność w wyrażeniach wszystkich atrybutów klas buforów wejściowych, a także atrybutów klas obiektów powiązanych asocjacjami powoduje, że przejścia READ są znacznie bardziej elastyczne niż w GN, gdyż te drugie opierają się tylko na pojedynczym parametrze. To właśnie dzięki zastąpieniu zdarzeń klasami oraz wprowadzeniu języka opisu wyrażeń możliwe było zastosowanie READ do dokładnego opisu procesu eksploatacji opisanego w rozdziale 10.

8.1.3 Bramy drzew niezdatności

Pomimo że symbole graficzne poszczególnych bram pozostają niezmienione, to konsekwencją usunięcia zdarzeń jest zmiana składni połączeń bram z resztą modelu. W READ na wejściach i wyjściach bram znajdują się bufory centralne obiektów niekoniecznie tych samych typów. Przykładami bram uogólniających są $G5$ i $G6$ na rys. 8.4 a przyczynowych $G2$ i $G6$ na rys. 8.5.

Dodanie akcji do READ nie podważyło pozycji bramy opóźniającej i w konsekwencji obydwa elementy są dostępne dla modelującego. Konstrukcje jednak nie są równoważne i nie można ich stosować zamiennie. Brama opóźniająca, jako element o naturze przyczynowej, nie usuwa obiektu wejściowego, co ma miejsce w przypadku akcji.

Jednym ze wzorców zastosowań bram uogólniających w READ jest utworzenie osobnego toru do pokazania stanu procesu eksploatacji. Przykładami są ostatnie na prawo tory obydwu przykładowych modeli.

Obok bramy może istnieć wyrażenie opisujące wiązanie pomiędzy obiektami wejściowymi i wyjściowymi, które w takiej sytuacji nadpisuje wiązanie domyślne, co omówiono w podrozdziale 8.2. Tym samym przy budowie modelu READ należy pamiętać, że nazwy atrybutów klas mają znaczenie podczas odpalania przejść i bram. Aby atrybuty różnych klas należały do wiązania podczas odpalania, to należy im nadać taką samą nazwę. Odwrotnie, nie należy nazywać atrybutów identycznie jeśli nie powinny należeć do wiązania domyślnego, a jeśli jest to nieuniknione, konieczne jest nadpisanie wiązania domyślnego omówionymi dalej blokami *pre* i *post*.

Gdy czas odpalania bramy opóźniającej nie został określony, to przyjmuje się odpalanie natychmiastowe, czyli opóźnienie jest zdefiniowane rozkładem jednopunktowym o wartości 0.

8.1.4 Model czasowy

W PDNZC oraz GN stosowano dedykowany model opisu czasu pozwalający na wyrażenie jedynie podstawowych gęstości zmiennych losowych. Wynikało to wprost z definicji składni tych języków, gdzie istnieje klasa dedykowana każdej przewidzianej gęstości parametrycznej. Zatem z jednej strony repertuar zmiennych był ograniczony wachlarzem dostępnych klas i przewidzianych parametrów, a z drugiej strony metodą analizy, na przykład wsparciem symulatora.

Dzięki odwołaniu się do uznanego w inżynierii systemowej profilu Marte [172] w READ zostało usunięte pierwsze ograniczenie. W myśl standardu czas definiuje się łańcuchem znaków o określonej gramatyce przechowywanym w obiektach klasy $NFP\_Duration$. Klasa $RandVar$ (rys. 8.1) jest specjalizacją $NFP\_Duration$ i przykładowo może definiować dystrybuantę ($dist$) zmiennej o założonym ($assm$) rozkładzie wykładniczym (exp) o średniej 100000 minut:

$$'assm','dist',('exp',100000),min$$

Zaletami włączenia profilu Marte do READ są:

- zgodność z uznanym standardem i uproszczenie metamodelu READ w zakresie definicji zmiennych losowych,
- możliwość opisu bardziej skomplikowanych zmiennych, takich jak przedstawione w rozdziale 10,
- specyfikacja jednostek czasu,
- określenie źródła definicji zmiennej: dane historyczne, dane historyczne z uwzględnieniem opinii eksperta, założenie itp.

8.1.5 Składnia języka wyrażeń

Ponieważ UML nie zawiera definicji składni języka opisu akcji, w ramach prac nad READ zaprojektowano gramatykę dla opisów warunków odpalania przejść, bram i akcji. Użyto przy tym narzędzia XText [67], stąd uzyskana gramatyka przypomina notację EBNF (wydruk 8.1).

Opis warunków wstępnych, końcowych lub czasu odpalania rozpoczyna się odpowiednio regułą *PreBlock*, *PostBlock* lub *DurationBlock*.

Blok *pre* to ciąg wyrażeń binarnych, z których każde ma dwa argumenty $ValueExpression$ połączone operatorem binarnym (linia 11). Wyrażenie z wartością to z kolei termy połączone operatorami arytmetycznymi dodawania lub odejmowania (linie 20 i 36). Aby zachować poprawną kolejność działań, podwyrażenia z mnożeniem i dzieleniem są w regule niższego poziomu $Term$.

Elementami składowymi termów są obiekty $Factor$ i mogą nimi być: stałe, łańcuchy znaków *rv* opisujące zmienne w formacie Marte (p. 8.1.4), zmienne globalne, wyrażenia $ValueExpression$ ujęte w nawiasy lub nawigacje do atrybutów obiektu - *AttributeAccessExpression*.

Blok *post* to ciąg przypisań. Gramatyka każdego przewiduje nawigację do miejsca przypisania po modelu obiektowym, znak przypisania oraz omówione wyrażenie z wartością (linia 14).

Reguła *DurationBlock* jest zdefiniowana przez wyrażenie z wartością dopuszczające opis zmiennej losowej w formacie Marte.

Wydruk 8.1: Gramatyka języka wyrażeń READ

```
1  PreBlock :
2      "pre:" pre+=BinaryExpression*;
3
4  PostBlock :
5      "post:" post+= Statement*;
6
7  DurationBlock :
8      "duration:" v=ValueExpression;
9
10 BinaryExpression :
11     lhs = ValueExpression op=BINARY_OPERATOR rhs = ValueExpression;
12
13 Statement :
14     l=AttributeAccessExpression "=" s=ValueExpression;
15
16 AttributeAccessExpression :
17     object=ID ("." members+=ID)*;
18
19 ValueExpression :
20     terms += Term (ops+=ARITH_OP_TERM terms+=Term)*;
21
22 Term :
23     factors+=Factor (ops+=ARITH_OP_FACTOR factors+=Factor)*;
24
25 Factor :
26      INT | Float | "true" | "false" | rv=STRING | g=GlobalVar |
27          "(" v=ValueExpression ")" | m=AttributeAccessExpression;
28
29 GlobalVar :
30     "now";
31
```

```
32 BINARY_OPERATOR :
33     "<" | "<=" | "=" | ">=" | ">" | "<>";
34
35 ARITH_OP_TERM :
36     "+" | "-" ;
37
38 ARITH_OP_FACTOR :
39     "*" | "/" | "div" | "mod";
```

8.2 Semantyka READ

Analizując specyfikację UML [130,131] można z łatwością dostrzec, że natura działania diagramów aktywności jest bardzo mocno związana z sieciami Petriego. Choć specyfikacja nie zawiera formalnej definicji DA, opis w języku naturalnym znaczenia wielu elementów, na przykład akcji [131] (s. 320), nawiązuje wprost do przepływu żetonów i mechanizmu odpalania przejścia w sieciach Petriego.

Badania wielu procesów eksploatacji wykazały potrzebę istnienia w języku konstrukcji o atomowym działaniu do rozstrzygania konfliktów w dostępie do zasobów, na co odpowiedzią w GN i READ jest dodanie elementu bazującego na przejściu sieci Petriego.

Konkludując dwie powyższe myśli, naturalnym wyborem gruntu, na którym odbędzie się definicja semantyki READ są właśnie sieci Petriego.

Należy jednak przyznać, że z powodu złożoności DA definicja semantyki całego języka jest zadaniem czasochłonnym. Problem potęguje istotna zmiana mechanizmu działania DA w drugiej wersji UML, który do tej pory odwoływał się do maszyn stanowych.

W dostępnej literaturze dotyczącej drugiej wersji standardu, to cykl prac [162,165,166] stanowi największą znalezioną kontrybucję zarówno dla specyfikatorów samego UML'a, jak i autorów narzędzi przetwarzających zapisane w nim modele. Prace dostarczają reguł transformacji elementów DA w klasę sieci Petriego używaną w narzędziu CPN Tools jednoznacznie specyfikując działanie i dając możliwość wykonania DA. Co szczególnie istotne w kontekście READ, ostatni artykuł jako jeden z nielicznych dostępnych podejmuje problem formalizacji przepływu obiektów.

Po przeglądzie podstawowych problemów formalizacji READ w kontekście aktualnych wysiłków w definicji DA, w kolejnych sekcjach dokonuje się rozszerzenia cyklu prac o definicje elementów READ.

8.2.1 Problemy napotkane przy opisie READ sieciami Petriego

Istotnym elementem definicji formalnej READ jest określenie wykorzystanej klasy sieci Petriego. Z jednej strony zastosowanie klas i obiektów pociąga za sobą konieczność użycia kolorowanych sieci Petriego. Z drugiej strony READ ma naturę czasową, a zatem wybrana klasa sieci musi umożliwiać opis zjawisk czasowych. Obydwa wymagania rozpatrywane równolegle znacznie zawężają listę narzędzi zebraną w [59].

Ostatecznie zdecydowano się na wykorzystanie narzędzia Snoopy umożliwiającego symulację kolorowanych i stochastycznych sieci Petriego. Choć narzędzie jest agnostyczne dziedzinowo, zostało szeroko zastosowane w modelowaniu procesów biologicznych i chemicznych [151].

Jakkolwiek wiele spośród opisanych dalej modeli można zdefiniować i zasymulować w znanym narzędziu CPN Tools [71] używanym w pracy [165], brakującym elementem wykorzystywanej klasy sieci Petriego są przejścia o naturze stochastycznej. W CPN Tools odpalenie przejścia zawsze trwa czas zerowy i od razu po spełnieniu warunków odpalenia żetony są zabierane z miejsc wejściowych. Model czasowy wprowadzono natomiast do żetonów i to żetonom przy odpaleniu przejścia można przypisać chwile pojawienia się w miejscach wyjściowych. Taka koncepcja uniemożliwia modelowanie wyścigów czasowych stanowiących istotny element procesów eksploatacji.

Istnieją dwie istotne różnice pomiędzy odpalaniem przejścia czasowego a następstwem przyczynowym zdarzeń.

Przejście usuwa żetony z miejsc połączonych zwykłymi krawędziami wejściowymi. Ani bramy przyczynowe, ani uogólniające nie rozpoczynają i nie kończą zdarzeń wejściowych. Warto zauważyć, że łuki dwukierunkowe pomiędzy przejściem i miejscem nie rozwiązują problemu, gdyż faktycznie atomowo usuwają i dodają żetony do określonego miejsca potencjalnie powodując odpalenie innych przejść, dla których rozpatrywane miejsce jest wejściowym. Aby uniknąć niepotrzebnej rozbudowy sieci zdecydowano się na użycie łuków tylko do odczytu, których miejsca uczestniczą w wiązaniu jako wejściowe, ale których żetony nie są modyfikowane. Zakończeniem łuku tylko do odczytu jest wypełniony czarnym kolorem okrąg.

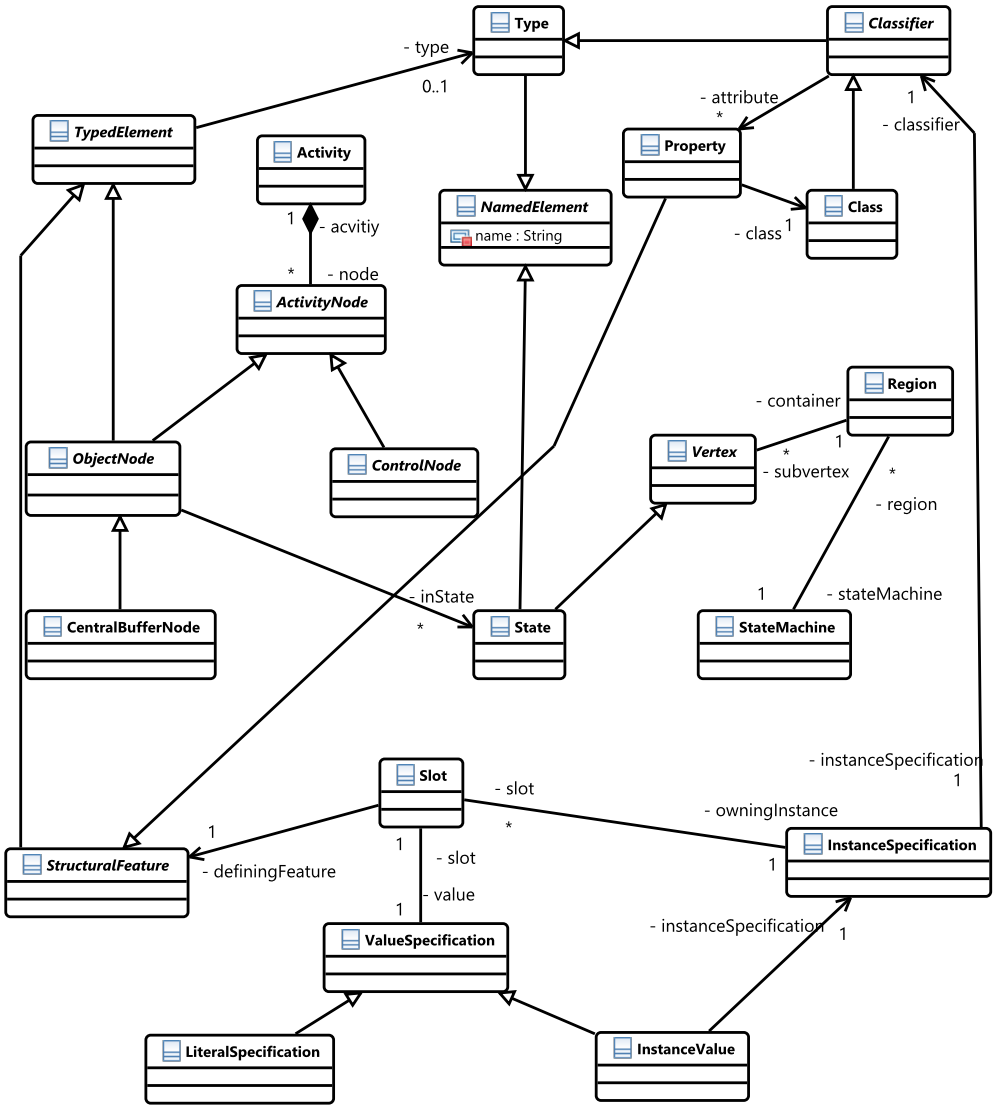
Ponadto w stochastycznych sieciach Petriego usunięcie żetonu będącego w wiązaniu odpalającym przejście czasowe powoduje anulowanie odpalenia tego przejścia. W bramach przyczynowych PDNZC natomiast raz zainicjowane zdarzenie wyjściowe zajdzie nawet, gdy zdarzenia wejściowe ulegną uprzedniemu zakończeniu. Z tego powodu w modelach bram przyczynowych sieci Petriego wprowadzono bufor służący do przechowania wiązań powodujących odpalanie.

Istotnym elementem zapisu READ jako sieci Petriego jest prawidłowy model obiektowy systemu. W żadnej ze znalezionych prac poświęconych formalizacji diagramów aktywności w sieciach Petriego, w tym w [165], nie znaleziono metod opisu związków pomiędzy obiektami, które są istotnymi elementami wielu modeli niezawodnościowych, na przykład systemu lotniczego omówionego w rozdziale 10. We wszystkich omówionych przykładach dokonano zastąpienia referencji atrybutami oznaczającymi identyfikatory skojarzonych obiektów, podobnie jak dokonuje się tego w relacyjnych bazach danych. Przeglądanie grafu obiektów można zatem dokonać poprzez przejścia natychmiastowe łączące żetony po identyfikatorach. Z powyższej konwencji wynika również, że jedno miejsce powinno reprezentować klasę oraz przechowywać zbiór wszystkich jej obiektów, czyli zbiór buforów centralnych tej samej klasy tłumaczy się do tego samego miejsca. Gdyby każdy bufor centralny był osobnym miejscem, to nie znając stanu obiektu wskazywanego przez referencję należałoby przejrzeć wiele miejsc, co znacznie skomplikowałoby model.

Definicje bram typu AND i OR wymagają stwierdzenia kiedy dwa zdarzenia lub dwa obiekty sobie odpowiadają. Takiego problemu nie było w klasycznych drzewach niezdatności, gdzie każde zdarzenie mogło występować maksymalnie jeden raz. W GN, dopuszczających wielokrotne

zachodzenie tych samych zdarzeń, zakładano, że zdarzenia proste są nierozróżnialne i zawsze pozostają sobie równe. Dla zdarzeń parametrycznych porównywana była wartość jedynych parametrów obu zdarzeń. W dalszej części rozdziału omówiono konwencję przyjętą w READ.

8.2.2 Szkielet sieci Petriego odpowiadającej modelowi READ



Rysunek 8.6: Metamodel UML dotyczący READ, opracowanie własne na podstawie [131]

Z metamodelu UML dokonano ekstrakcji najważniejszych z punktu widzenia transformacji READ elementów języka, czego rezultatem jest diagram klas na rys. 8.6 pokazujący związki pomiędzy klasami, obiektami, aktywnościami oraz maszynami stanowymi. Struktura READ modelowana jest przez elementy *Class*, dla których podstawowymi są $Classifier$, *Type* oraz $NamedElement$. Atrybuty reprezentowane są natomiast przez metaklasy *Property* rozszerzające $StructuralFeature$. Klasy łączą się z obiektami na rysunku dwiema drogami. Pierwszą jest asocjacja od $InstanceSpecification$ reprezentującą obiekty do $Classifier$ informującą o typach obiektów. $InstanceSpecification$ jest z kolei w relacji z wieloma bytami *Slot* stanowiącymi

wartości atrybutów obiektu. Stąd, drugą ścieżką łączącą klasy z obiektami jest asocjacja ze *Slot* do $StructuralFeature$. Wartości atrybutów specyfikowane są w bytach $ValueSpecification$, którymi mogą być literały bądź inne obiekty.

Przedstawionymi metaklasami opisującymi aktywność są *Activity* oraz $ActivityNode$, którymi z kolei mogą być węzły związane z obiektami lub sterowaniem. Każdy węzeł przechowujący obiekty, czyli między innymi bufor centralny, jest pochodnym $TypedElement$, czyli posiada referencję do typu, którym może być klasa. Bufory centralne posiadają również referencje do stanów wchodzących w skład regionu danej maszyny stanowej $StateMachine$.

W dalszej części rozdziału wykorzystano opisany w dodatku B język OCL do nawigacji po metamodelu i do tworzenia elementów sieci Petriego. Definicję kolorów wynikowej sieci poprzedza definicja funkcji $Type$ oraz St.

Typy proste reprezentowane są w metamodelu UML przez obiekty typu $Property$. Niech do zbioru $\mathcal{P}$ należą wszystkie takie obiekty transformowanego modelu READ, a funkcja Typ niech przyporządkowuje każdemu z nich dziedzinę przechowywanych wartości:

$$Type : \mathcal{P} \longrightarrow \{Integer, String, Boolean, Float\}$$

Zbiór $\mathcal{S}$ zawiera nazwy wszystkich stanów modelu u wyrażonego w READ:

$$\mathcal{S} = u.objects() \rightarrow select(e \mid e.oclIsKindOf(State)) \rightarrow collect(s \mid s.name)$$

Spośród wszystkich elementów modelu selekcjonowane są te, które są typu *State* lub pochodnego. Tak uzyskana kolekcja stanów przetwarzana jest na zbiór nazw tych stanów.

Ponadto niech funkcja St definiuje zbiory wszystkich stanów, w których mogą się znajdować obiekty każdej z użytych w READ klas, czyli:

$$St : \mathcal{C} \longrightarrow 2^{\mathcal{S}}$$

$$St(c) = u.objects() \rightarrow select(e \mid e.oclIsKindOf(CentralBufferNode)) \rightarrow$$

$$select(cb \mid cb.type = c) \rightarrow$$

$$collect(cb \mid cb.inState \rightarrow at(1)) \rightarrow$$

$$collect(s \mid s.name)$$

Powyższe wyrażenie języka OCL działa następująco. Ze wszystkich elementów modelu u wyrażonego w READ selekcjonowane są bufory centralne, które następnie ogranicza się do tych, w których przechowywane są obiekty klasy będącej argumentem funkcji. Elementy tak powstałej kolekcji przekształca się na pierwszy stan każdego z bufora, gdyż na mocy ograniczeń składniowych bufor ma przypisany jeden i dokładnie jeden stan. Ostateczny zbiór tworzą nazwy uzyskanych stanów.

Rodzina zbiorów kolorów C oraz miejsca P sieci Petriego są uzyskiwane z klas UML - zbiory CK/PK oraz węzłów READ - CN/PN:

$$C = CK \cup CN$$

$$P = PK \cup PN$$

Zbiory CN oraz PN odpowiadające węzłom zostaną omówione w kolejnych podrozdziałach.

Z każdej klasy UML k na drodze produktu kartezjańskiego stanu oraz typów atrybutów uzyskiwany jest jeden kolor wynikowej sieci Petriego $K(k)$:

$$K : \mathcal{K} \longrightarrow CK$$

$$K(k) = St(k) \times \prod_{i=1}^{Attr(k)} Type(k.attribute \rightarrow at(i))$$

$$Attr(k) = k.attribute \rightarrow size()$$

- $\mathcal{K}$ zbiór wszystkich klas modelu wyrażonego w READ,
- $k.attribute \rightarrow at(i)$ wyrażenie OCL wskazujące na atrybut o indeksie i w uporządkowanej kolekcji *attribute* klasy k,
- $Type$ funkcja przekształcająca typ UML w elementarny zbiór kolorów języka Snoopy.

Każdemu buforowi *cb* ze zbioru wszystkich buforów $\mathcal{CB}$ przyporządkowane jest miejsce sieci Petriego o takiej samej nazwie przechowywanej klasy *cb.type* oraz zbiorze kolorów $K(cb.type)$:

$$M : \mathcal{CB} \longrightarrow PK$$

$$M(cb) = p_{K(cb.type)}^{cb.type.name}$$

- $\mathcal{CB}$ zbiór wszystkich buforów centralnych READ,
- $p_{K(cb.type)}^{cb.type.name}$ miejsce o kolorze $K(cb.type)$ i nazwie $cb.type.name$.

Zatem wszystkie bufory danej klasy mapują się do tego samego miejsca, a rozróżnienie stanu będzie się odbywało przez pierwszą składową koloru przechowywanych w nim żetonów.

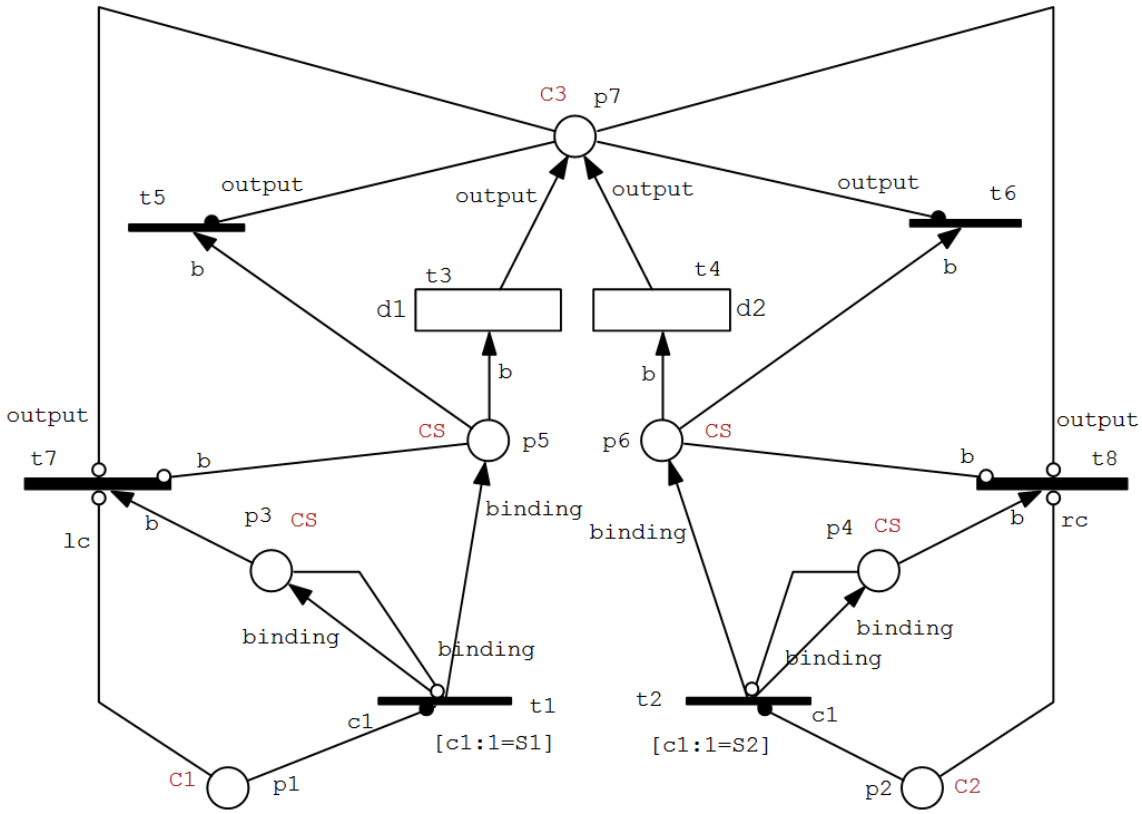
8.2.3 Brama przyczynowa OR

Model na rys. 8.7 reprezentuje wzorzec sieci Petriego dla dowolnej bramy COR połączonej z opisanym w poprzednim paragrafie szkieletem poprzez miejsca $p1$, $p2$ i $p7$. Oznacza to, że:

$$p1, p2, p7 \in PK$$

$$p3, p4, p5, p6 \in PN$$

$$CS \in CN$$



Rysunek 8.7: Wzorzec translacji bramy COR [91]

Wymienione miejsca mogą stanowić na przykład translacje buforów z klasami *Restoration*, *Replacement* i *Delivery* bramy $G6$ z rys. 8.5. Natomiast $S1 - S3$ to stany obiektów w buforach.

Zasadniczym celem bramy COR jest utworzenie obiektu w buforze wyjściowym (przekształconym do $p7$) po upłynięciu czasu od pojawienia się obiektu na jednym z wejść ($p1$ lub $p2$). Ponadto, jeśli w klasie wyjściowej $K3$ znajduje się atrybut o nazwie występującej wśród atrybutów klas wejściowych $K1$ i $K2$, to wartość tego atrybutu w obiekcie wyjściowym zostanie skopiowania z obiektu wejściowego. Analogicznie do opisu w podrozdziale 4.2, bramy przyczynowe READ są aktywowane pojawieniem się obiektu w buforze wejściowym a nie jego obecnością. Oznacza to, że obiekt może zostać usunięty po dodaniu do bufora a przed odpaleniem bramy przyczynowej, a ta brama wciąż zostanie odpalona. W związku z tym sieć Petriego odpowiadająca bramie przyczynowej musi przechowywać wartości kopiowane po pewnym czasie do obiektu wyjściowego. Ciąg takich atrybutów uzyskuje się poprzez usunięcie z $K1$ atrybutów nieistniejących w $K2$:

$$B(K1, K2) = K1.attribute \rightarrow$$

$$reject\Big(e1 \mid not\ \ (K2.attribute \rightarrow exists(e2 \mid e2.name = e1.name))\Big) \rightarrow$$

$$collect(e1 \mid e1.name) \rightarrow sorted()$$

$$k = B(K1, K2) \rightarrow size()$$

Zatem $B(K1, K2)$ to posortowany według kolejności naturalnej ciąg nazw atrybutów wystę-

pujących zarówno w $K1$, jak i $K2$. Kolory zawierające wartości tych atrybutów o typie CS są tworzone przez przejścia $t1$ oraz $t2$ i zapamiętywane dwukrotnie na miejscach odpowiednio $p3$ i $p5$ oraz $p4$ i $p6$. Zbiór kolorów CS jest określony produktem:

$$CS \equiv \prod_{i=1}^{k} Type\Big(K1 \rightarrow attribute \rightarrow selectFirst(e1 \mid e1.name = B(K1, K2)) \rightarrow at(i)\Big)$$

Niech $c1$ oznacza kolor żetonu z $p1/p2$ przy odpalaniu $t1/t2$. Z kolei *binding*, oznaczający wiązanie pomiędzy obiektami wejściowymi a wyjściowymi, to wektor produktu CS wyprowadzony z $c1$:

$$binding \equiv \Big\langle c1 : n_1, \ldots, c1 : n_k \Big\rangle$$

Notacja $c1 : n_1$ oznacza n_1-tą składową produktu $c1$. Wartość n_i to indeks atrybutu $K1$ o nazwie odpowiadającej nazwie i-tego atrybutu z ciągu $B(K1, K2)$:

$$n_i = K1 \rightarrow attribute \rightarrow indexOf\Big(e \mid e.name = B(K1, K2) \rightarrow at(i)\Big)$$

Przejście $t1$ rozpoczyna działanie sieci. Aby $t1$ zostało odpalone, $p1$ musi zawierać żeton, który będzie dostępny w wyrażeniach przejścia poprzez zmienną $c1$. Łuk łączący $p1$ i $t1$ jest tylko do odczytu, gdyż został zakończony wypełnionym okręgiem. Stąd gdy $t1$ zostaje odpalone, $c1$ pozostaje na $p1$. Ponieważ wszystkie bufory centralne są tłumaczone do jednego miejsca, przy przejściu istnieje warunek $c1 : 1 = S1$ oznaczający, że przy odpalaniu będą brane pod uwagę jedynie żetony z bufora wejściowego bramy COR, który przechowuje obiekty w stanie $S1$. Ponadto na miejscu $p3$ nie może istnieć kolor *binding*, gdyż łuk łączący $t1$ i $p3$ jest zakończony pustym okręgiem, więc hamuje odpalanie przejścia. Gdy $t1$ zostanie odpalone, dwa żetony o kolorze *binding* zostaną dodane do miejsc $p3$ i $p5$, co spowoduje zablokowanie kolejnego odpalenia $t1$ dla $c1$. Analogicznie dla $t2$, $p4$ i $p6$.

Przejście czasowe $t3$ (lub $t4$ dla wejścia bramy reprezentowanego przez $p2$) realizuje oznaczoną symbolem $d1$ ($d2$) zmienną losową czasu do utworzenia żetonu o kolorze odpowiadającemu obiektowi wyjściowemu bramy. Po upłynięciu tego czasu do $p7$ zostanie dodany żeton o kolorze *output*, w którym niektóre składowe zostaną skopiowane z $p5$ ($p6$) przez funkcję VD:

$$output \equiv \Big\langle S3, VD(K3, 1), VD(K3, 2), \ldots, VD(K3, K3 \rightarrow attribute \rightarrow size()) \Big\rangle$$

Pierwsza składowa uporządkowanej n-ki to stan skojarzony z buforem wyjściowym bramy COR. Pozostałe składowe VD to albo wartość z $c1$ jeśli atrybut o tym indeksie w $K3$ istnieje również w $B(K1, K2)$, albo domyślna wartość typu tego atrybutu $K3$:

$$VD(K, i) = \begin{cases} b : j & \exists j B(K1, K2) \rightarrow at(j) = K \rightarrow attribute \rightarrow at(i).name \\ \\ K.attribute \rightarrow at(i).type.default & \text{w p. p.} \end{cases}$$

Przejście $t4$ może odpalić się przed $t3$. Jeśli tak się stanie, to żeton na $p7$ spowoduje, że $t5$ usunie żeton z $p5$ tym samym zatrzymując odpalanie $t3$. Dopóki na $p7$ istnieje żeton to $t7$ i $t8$

są zablokowane, czyli żetony na $p3$ i $p4$ blokują $t1$ i $t2$. W rezultacie, dopóki na wyjściu istnieje żeton, to niezależnie co dzieje się na buforach wejściowych, brama jest zablokowana.

Gdy żeton z $p7$ zostanie usunięty, a na wejściu $p1$ wciąż istnieje odpowiadający żeton, to nie dojdzie do ponownego odpalenia bramy. Wynika to z obecności żetonu lc na $p1$ hamującego $t7$, czyli pośrednio również $t1$. Aby doszło do hamowania, na $p1$ musi istnieć żeton lc o kolorze z produktu $C1$, takim że wspólne z $C2$ składowe są równe wartościom z b. Pozostałe składowe nie mają udziału w hamowaniu i oznacza się je symbolem gwiazdki. Zatem:

$$lc \equiv \Big\langle S1, VN(K1,1), VN(K1,2), \ldots, VN(K1, K1 \rightarrow attribute \rightarrow size()) \Big\rangle$$

$$rc \equiv \Big\langle S2, VN(K2,1), VN(K2,2), \ldots, VN(K2, K2 \rightarrow attribute \rightarrow size()) \Big\rangle$$

$$VN(K,i) = \begin{cases} b:j & \exists j B(K1,K2) \rightarrow at(j) = K \rightarrow attribute \rightarrow at(i).name \\ * & \text{w p. p.} \end{cases}$$

Wielokrotne dodawanie i usuwanie żetonów z $p1$ nie wpływa na funkcjonowanie bramy, gdyż pierwsze odpalenie $t1$ spowoduje przechowanie żetonu w $p3$ blokując dalsze odpalenia. Zatem $t3$ i $t4$ zostaną odpalone tylko raz w związku z pierwszym wystąpieniem obiektu na wejściu.

8.2.4 Brama przyczynowa AND

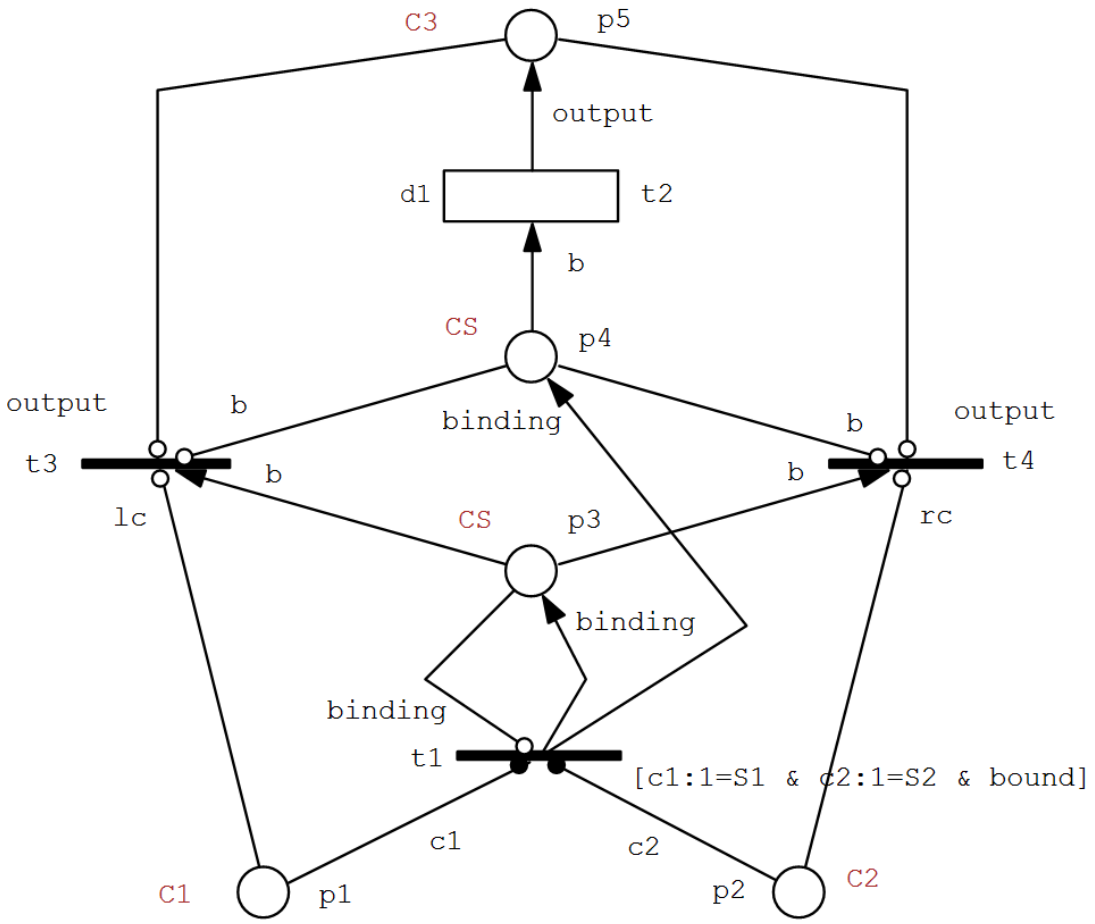
Na rys. 8.8 przedstawiono model bramy CAND, która połączony jest ze szkieletem poprzez miejsca $p1$ i $p2$ odpowiadające buforom wejściowym oraz $p5$ stanowiące translację bufora wyjściowego bramy. Sieć bazuje na oznaczeniach wprowadzonych w poprzednich podrozdziałach.

W przeciwieństwie do COR, CAND generuje obiekt wyjściowy jeśli na obydwu wejściach znajdą się obiekty takie, że wartości wszystkich atrybutów, które występują w obydwu klasach wejściowych jednocześnie są sobie równe. Oznacza to, że na rys. 8.8 zamiast dwóch przejść inicjujących bramę wystarczy jedno, na które jednak nałożony jest dodatkowy warunek równości odpowiednich składowych kolorów żetonów $c1$ i $c2$:

$$bound \equiv c1 : n_1 = c2 : m_1 \quad \& \quad c1 : n_2 = c2 : m_2 \quad \& \quad \ldots \quad \& \quad c1 : n_k = c2 : m_k$$

Skoro przy odpalaniu $t1$ obiekty mają równe wartości wspólnych atrybutów, to nie jest istotne, z którego żetonu zostaną one pobrane przed umieszczeniem w $p3$ i $p4$. Na łukach wykorzystano poprzednio zdefiniowany *binding*. Brama CAND ma przypisaną jedną zmienną losową opóźnienia $d1$, którą realizuje i odmierza przejście czasowe $t2$. W rezultacie na $p5$ pojawia się żeton o kolorze *output* odpowiadający obiektowi klasy $K3$.

Przejścia $t3$ i $t4$ mają funkcje analogiczne do przejść $t7/t8$ w modelu COR. Ich blokada przez łuki hamujące powoduje pozostawienie koloru na $p3$, co z kolei blokuje ponowne wykonanie $t1$. Przed upłynięciem czasu odpalania bramy to kolor na $p4$ hamuje $t3$ i $t4$, natomiast po odpaleniu $t2$ robi to $p5$. Zatem do ponownego odpalenia bramy musi istnieć taka chwila, w której nie ma już powiązanego żetonu na $p5$ oraz nie istnieje on na przynajmniej jednym z wejść.



Rysunek 8.8: Wzorzec translacji bram CAND

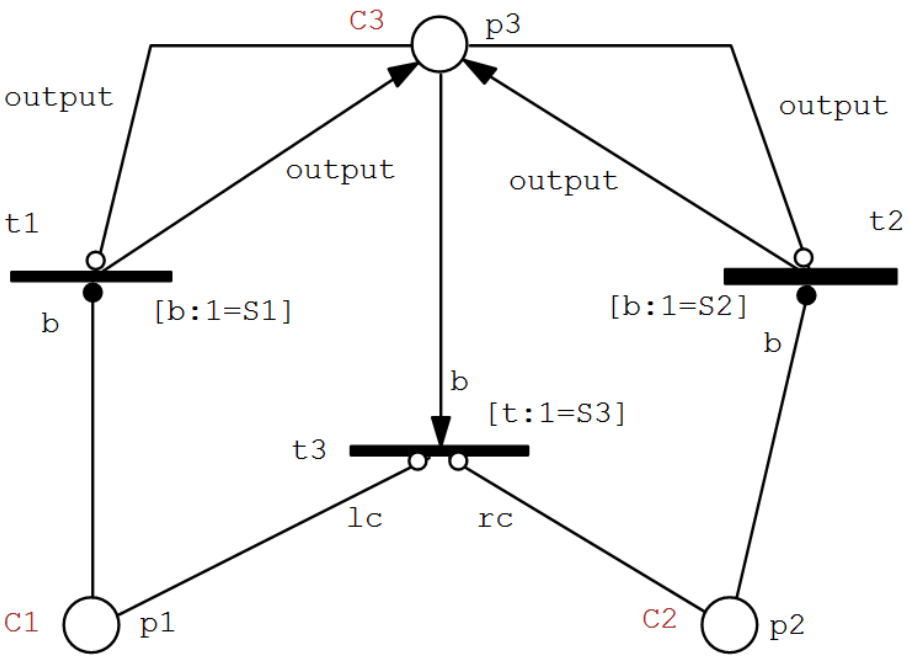
8.2.5 Brama uogólniająca OR

Bramy uogólniające nie posiadają modelu czasowego, co oznacza, że nie istnieje potrzeba tworzenia kopii żetonu wejściowego zawierającego wartości niezbędne dla utworzenia żetonu wyjściowego. Jak wynika ze wzorca przedstawionego na rys. 8.9 pozwoliło to uprościć sieć i w rezultacie jedynymi miejscami modelu są $p1, p2, p3 \in PK$.

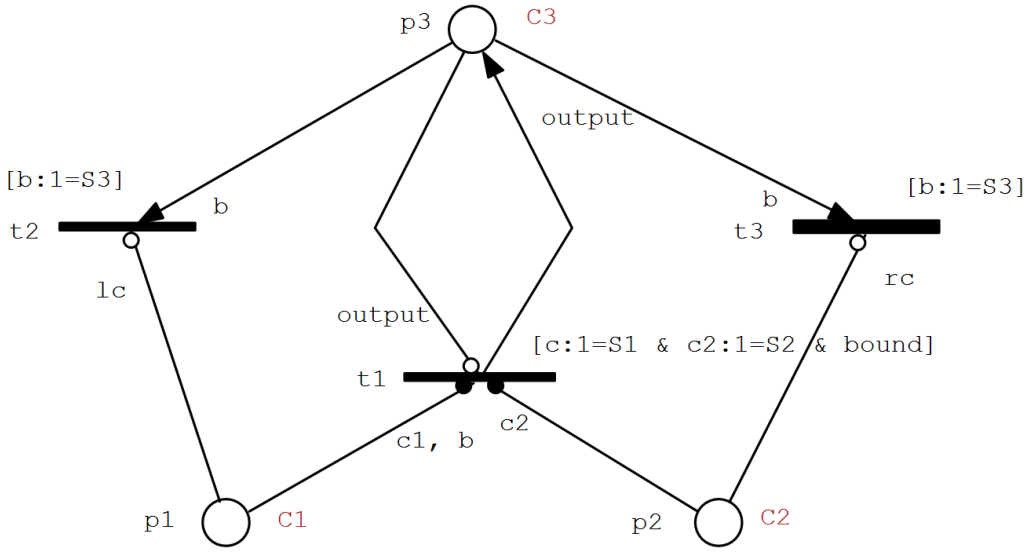
Istnieją dwie możliwości dodania żetonu do miejsca wyjściowego sieci modelującej bramę GOR. Pierwsza, realizowana przez przejście $t1$, wynika z obecności żetonu o kolorze b na $p1$. Jeśli tylko żeton o kolorze *output* nie istnieje na $p3$, to zostanie do niego dodany. Analogiczną funkcję realizuje $t2$ dla miejsca $p2$. Usunięcie żetonu o kolorze *output* z $p3$ możliwe jest przejściem $t3$. Aby do tego doszło ani na $p1$, ani na $p2$ nie mogą już istnieć żetony odpowiednio lc i rc odwołujące się w definicji do składowych żetonu b z miejsca $p3$.

8.2.6 Brama uogólniająca AND

Aby wykorzystać uprzednio zdefiniowaną symbolikę w niniejszym paragrafie zakłada się, że zmienne $c1$ oraz b wskazują ten sam żeton, co oznaczono na rys. 8.10 przecinkiem na łuku od $p1$ do $t1$.



Rysunek 8.9: Wzorzec translacji bram GOR

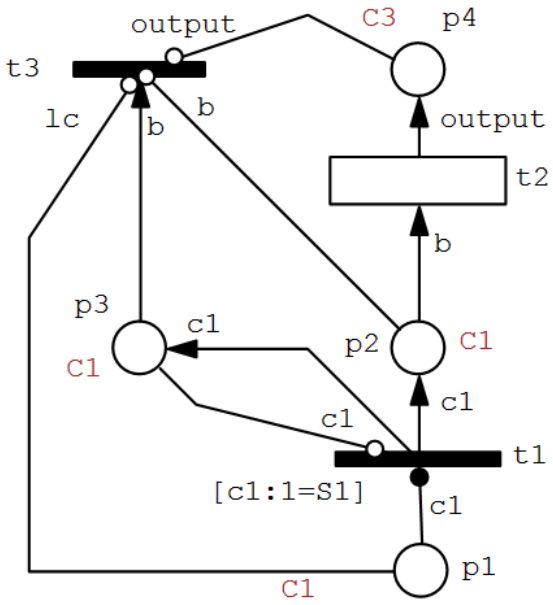


Rysunek 8.10: Wzorzec translacji bram GAND

Jeśli na $p1$ i $p2$ pojawią się żetony $c1/b$ oraz $c2$ o odpowiadających sobie składowych równych, czyli *bound* będzie miało wartość logicznej prawdy, a ponadto pierwsze składowe będą równe $S1$ i $S2$, to dojdzie do odpalenia $t1$ i dodania żetonu o kolorze *output* do miejsca $p3$. Jeśli z któregokolwiek miejsca wejściowego żeton zostanie usunięty, to $t2$ lub $t3$ usunie również utworzony przez $t1$ żeton z $p3$.

8.2.7 Brama opóźniająca

Element tworzy obiekt wyjściowy w zależności od obiektu wejściowego, w przeciwieństwie do akcji jednak, pozostawiając obiekt wejściowy bez zmian. Brama opóźniająca wywodzi się z bram

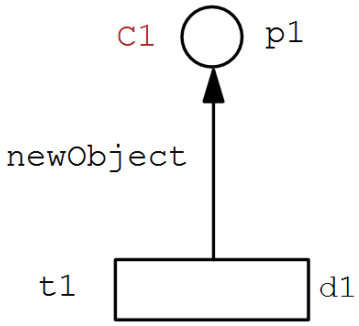


Rysunek 8.11: Wzorzec translacji bram opóźniającej

przyczynowych i dlatego w sieciach tych elementów można odnaleźć podobne grupy elementów.

Miejsce $p1$ na rys. 8.11 odpowiada obiektom wejściowym, a $p4$ obiektom wyjściowym bramy. Rezultatem odpalenia przejścia $t1$ jest przechowanie dwóch kopii żetonu z $p1$: na $p2$ oraz na $p3$. Pierwsza kopia jest niezbędna dla przejścia czasowego odmierzającego opóźnienie $d1$ i usuwającego kolor z $p2$, ale dodającego uprzednio zdefiniowany kolor *output* do $p4$. Druga kopia, w miejscu $p3$, jest potrzebna dla rozpoznania sytuacji: po odmierzeniu opóźnienia obiekt wyjściowy został usunięty z bufora wyjściowego, ale obiekt wejściowy wciąż istnieje na $p1$. Do ponownego odpalenia może dojść dopiero, gdy z bufora wejściowego zostanie usunięty obiekt, co na modelu spowoduje odpalenie przejścia $t3$ usuwającego kolor z $p3$. Tym samym $t1$ zostanie odblokowane.

8.2.8 Cykliczny generator obiektów



Rysunek 8.12: Wzorzec translacji generatora obiektów

Generator, jako brama o naturze przyczynowej, ma za zadanie tworzenie obiektów w swoim jedynym buforze wyjściowym cyklicznie co czas zdefiniowany w pierwszym elemencie kolekcji *delays* stereotypu *CGate*.

Taką funkcję realizuje bezwejściowe przejście czasowe sieci Petriego $t1$ na rys. 8.12. Cykl pracy generatora jest następujący. Po upłynięciu czasu wynikającego z realizacji zmiennej losowej reprezentowanej przez $d1$ następuje utworzenie nowego żetonu o kolorze $newObject$ dodawanego do miejsca $p1$. Definicja nowego koloru to:

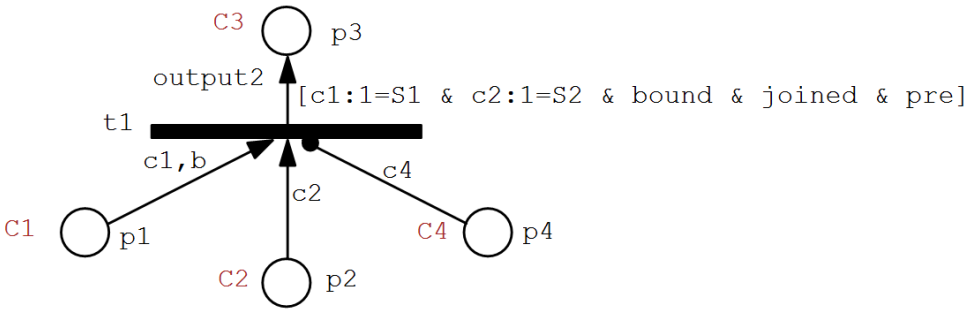
$$newObject \equiv \left\langle S3, D(K3,1), D(K3,2), \ldots, D(K3, K3 \rightarrow attribute \rightarrow size()) \right\rangle$$

$$D(K,i) = K.attribute \rightarrow at(i).type.default$$

Stan $S3$ wynika z parametrów bufora wyjściowego i w przypadku bramy $G1$ na rys. 8.5 jest nim $Scheduled$. Wartości funkcji D to domyślne wartości pól klasy $K3$, której obiekt jest tworzony przez generator. W przykładowym modelu to $TramFailure$.

Po umieszczeniu koloru na $p1$ następuje rozpoczęcie kolejnego cyklu pracy.

8.2.9 Przejście



Rysunek 8.13: Wzorzec translacji przejścia READ

Elementem przypominającym przejście READ jest przejście natychmiastowe sieci Petriego ($t1$ na rys. 8.13). Ponieważ jednak konstrukcja READ może w ogólności zawierać wyrażenia *pre* i *post* wykorzystujące asocjacje, to w odpowiadającej sieci Petriego należy uwzględnić dodatkowe warunki oraz efekty odpalenia. Ich realizacja pociąga za sobą dodanie do sieci wynikowej krawędzi łączących przejście natychmiastowe z miejscami odpowiadającymi klasom na końcach asocjacji. Zatem miejsca wejściowe przejścia to:

$$IP = BP \cup NP$$

- IP zbiór wszystkich miejsc wejściowych przejścia natychmiastowego,
- BP zbiór miejsc wejściowych przejścia wynikających z translacji buforów centralnych, podobnie jak w poprzednich modelach
- NP zbiór miejsc wejściowych przejścia wynikających z translacji wyrażeń *pre* i *post*.

Na rys. 8.13 $BP = \{p1, p2, p3\}$, podczas gdy $NP = \{p4\}$. Wydruk 8.2 ilustruje algorytm wyznaczania zbioru NP.

Wydruk 8.2: Generacja zbioru NP

```
1 //dla każdego obiektu BinaryExpression be znajdującego się w bloku pre/post
2   paths:=be.allChildren()->select(e | e.oclIsTypeOf(AttributeAccessExpression))
3   //dla każdego obiektu AttributeAccessExpression aae z kolekcji paths
4     K:=t.incoming->selectFirst(e | e.source.name=aae.object).source.type
5     //dla każdego oprócz ostatniego elementu member z kolekcji aae.members
6       KN:=K.attributes->selectFirst(e | e.name=member).type
7       CB:=u.objects()->selectFirst(e | e.oclIsTypeOf(CentralBufferNode) and e.type=KN)
8       dodaj typ M(CB) do zbioru NP
9       K:=KN
```

Role predykatów w wyrażeniu strzegącym $t1$ są następujące:

- *bound* podobnie jak w CAND weryfikuje zgodność wartości atrybutów w obiektach wejściowych; brane są pod uwagę kolory z miejsc należących do BP,
- *joined* gwarantuje, że żetony wejściowe przy odpalaniu przejścia odpowiadają powiązanym obiektom,
- *pre* translacja warunku *pre* przejścia READ w składowe kolorów żetonów z miejsc IB.

Przy tworzeniu żetonów wyjściowych w pierwszej kolejności analizowane są wartości wyrażenia *post*. Jeśli *post* ich nie specyfikuje, to wartości mogą być przepisywane ze zbioru wspólnych atrybutów klas wejściowych $B(K1, K2)$. Jeśli atrybut nie należy do $B(K1, K2)$ zostaje mu przypisana wartość domyślna. Stąd zmianie ulega definicja koloru żetonu wyjściowego:

$$output2 \equiv \langle S3, VP(K3, 1), VP(K3, 2), \ldots, VP(K3, K3 \rightarrow attribute \rightarrow size())\rangle$$

Funkcja VP najpierw sprawdzi czy atrybut został ustawiony w części *post* przejścia. Jeśli nie, to zwrócona zostanie poprzednio zdefiniowana wartość $VD(K, i)$:

$$VP(K,i) = \begin{cases} \text{wartość wyrażenia w bloku } post & \text{jeśli atrybut } i \text{ jest specyfikowany w } post \\ VD(K,i) & \text{w p. p.} \end{cases}$$

Dzięki diagramom aktywności dotychczasowe rozszerzenia drzew niezdatności zyskały solidny fundament o jednoznacznej definicji w sieciach Petriego. Moc tak uzyskanego języka zweryfikowano w rozdziale 10, gdzie, podobnie jak w przypadku PDNZC oraz GN, przedstawiono nowy problem, którego dokładna analiza w poprzednich wersjach języka była niemożliwa. Ponadto wraz z trzecim etapem modelowania systemu tramwajowego wyłoniła się nowa metoda obliczeniowa dla systemów z rezerwą czasową omówiona w rozdziale 9.

Rozdział 9

Metody wyznaczania niezawodności systemów z czasem aktywacji rezerwy zimnej, dostawą oraz rezerwą czasową

Aby badania symulacyjne były wiarygodne, to wykorzystywany symulator powinien bardzo wiernie oddawać zachowanie systemu. Ponieważ w skomplikowanym oprogramowaniu należy liczyć się z błędami, to w rzeczywistych zastosowaniach istnieje konieczność wykorzystania metod weryfikacji wyników dostarczanych przez implementacje algorytmów obliczeniowych. Estymacja analityczna jest jedną z takich metod pozwalającą na uniknięcie kosztownej budowy wielu wersji symulatora oraz zmniejszenie nakładu obliczeniowego przy weryfikacji wyników. W niniejszym rozdziale zostaną omówione trzy metody dostarczające w sposób numeryczny prawdopodobieństwa hazardu oraz średniego jego czasu dla omawianego w rozdziale 7 systemu z rezerwą czasową. Prezentowane metody, jak i wyniki obliczeń pochodzą z pracy [99].

Rozsądnym podejściem do analizy niezawodności jest wykorzystanie zarówno wiedzy eksperckiej, jak i danych historycznych przy wyznaczaniu parametrów modelu. Takie badania prowadzą do wyników o wysokim stopniu ufności, jednak są z natury trudniejsze do przeprowadzenia, gdyż parametry zmiennych losowych zwykle nie mają jednoznacznych wartości. Czytelną i praktyczną metodą przedstawiania danych wejściowych w takiej sytuacji są przedziały ufności, gdyż pozwalają na konfrontację wiedzy nawet wielu ekspertów jednocześnie [11]. Niestety symulacja jako metoda z natury powolna okazuje się w wtedy niepraktyczna i ustępuje miejsca szybszym metodom numerycznym. W trakcie konsultacji z Ekspertami w dziedzinie transportu wyznaczono zebrane w tab. 9.1 przedziały parametrów rozkładów Weibulla systemów szynowych, w których w praktyce mieszczą się awaria (zmienna losowa A), wymiana (E) oraz dostawa (D).

Dla poniższych danych przeprowadzono również eksperyment obliczeniowy omówiony w dalszej części rozdziału. Choć w opisach estymacji odwołuje się do systemu tramwajowego, to należy zaznaczyć, że istnieje wiele systemów o podobnej koncepcji rezerwy czasowej i stąd pisane metody mają szersze zastosowanie. Stosowny przegląd literaturowy zamieszczono w pracy [99].

Tabela 9.1: Przedziały parametrów zmiennych losowych zdefiniowane przy udziale Ekspertów

Zmienna	Symbol	$\lambda_{A\|E\|D}$		$k_{A\|E\|D}$	
		min	max	min	max
czas pomiędzy awariami	A	0,007	0,020	0,90	1,10
czas wymiany	E	0,020	0,050	1,00	1,50
czas dostawy	D	0,005	0,010	1,00	1,50

9.1 Metodyka obliczeń

Nawiązując do opisu hazardu w 7.1 hazard H jest zdarzeniem zachodzącym, gdy dostawa ani wymiana nie zostaną ukończone w czasie oznaczonym jako rezerwa czasowa R_C.

Interesujące z punktu widzenia niezawodności parametry hazardu to:

- prawdopodobieństwo hazardu, czyli sytuacji, kiedy uszkodzony pojazd nie zostanie ani naprawiony, ani zastąpiony rezerwowym w ustalonym czasie:

$$Pr(H) = Pr(min(C0 + W, D) > R_C) \tag{9.1}$$

  gdzie $C0$ to czas oczekiwania na rozpoczęcie wymiany (czyli czas oczekiwania na rezerwę).

- średni czas hazardu $E(HT)$ liczony od początku hazardu, czyli od przekroczenia rezerwy czasowej przez awarię do chwili zakończenia wymiany bądź dostawy:

$$E(HT) = E(FT \mid H) = E(min(C0 + W, D) \mid min(C0 + W, D) > R_C) - R_C \tag{9.2}$$

Nieznajomość zmiennej $C0$, wynikającej z czasu trwania poprzednich awarii, uniemożliwia dokładne wyznaczenie powyższych parametrów. Prawdopodobieństwo hazardu zostanie wyznaczone w zależności od dwu wykluczających się zdarzeń: element zapasowy jest dostępny, gdy dochodzi do uszkodzenia (zachodzi AV) oraz element zapasowy jest niedostępny, gdy dochodzi do uszkodzenia (NA). Stąd zgodnie z twierdzeniem o prawdopodobieństwie całkowitym:

$$Pr(H) = Pr(FT > R_C) = Pr(H \mid AV)Pr(AV) + Pr(H \mid NA)Pr(NA) \tag{9.3}$$

System z rezerwą czasową można w dużym uproszczeniu postrzegać jako systemem kolejkowy, w którym zgłoszenia to kolejne awarie, podczas gdy proces obsługi to dostawa elementów naprawionych [96]. Spostrzeżenie pozwoli na wyznaczenie prawdopodobieństw zdarzeń NA i AV, które okażą się niezależne od czasu wymiany.

Gdyby zmienne A i D miały rozkłady wykładnicze, to naturalnym wyborem byłby model kolejkowy $M/M/\infty$, dla którego istnieje rozwiązanie dokładne. W rozważanym przypadku, tj. gdy zmienne mogą mieć również rozkłady Weibulla, najlepiej pasuje $G/G/\infty$, w którym czas obsługi może mieć dowolny rozkład, jednak ten nie posiada obecnie rozwiązania analitycznego i dlatego został odrzucony w dalszych oszacowaniach. W takiej sytuacji podjęto decyzję o przybliżeniu kolejki elementów oczekujących na rezerwę modelem $M/G/\infty$ w następujący sposób.

Niech A' będzie zmienną wykładniczą odpowiadającą napływowi zgłoszeń, taką że:

$$E(A') = E(A)$$

Współczynnik obciążenia modelu $M/G/\infty$ można wtedy wyznaczyć korzystając ze wzorów na wartość oczekiwaną zmiennej o rozkładzie Weibulla:

$$\rho = \frac{E(D)}{E(A')} = \frac{E(D)}{E(A)} = \frac{\lambda_A \Gamma(1 + \frac{1}{k_D})}{\lambda_D \Gamma(1 + \frac{1}{k_A})}$$

gdzie Γ to funkcja Gamma, a zmienne k i λ to parametry rozkładów Weibulla zmiennych losowych A i D. Zmienna n oznacza liczbę wszystkich elementów rezerwowych. Model stanie się podstawą do wyznaczenia prawdopodobieństwa P_{na} zdarzenia: w systemie aproksymowanym modelem kolejkowym $A'/D/\infty$ jest przynajmniej n zgłoszeń. W tym celu należy wyznaczyć prawdopodobieństwa: P_0 - system jest pusty oraz P_k - w systemie jest dokładnie k uszkodzeń.

$$P_0 = e^{-\rho} \tag{9.4}$$

$$P_k = P_0 \frac{\rho^k}{k!} \tag{9.5}$$

Jeżeli element rezerwowy jest niedostępny w chwili awarii, to system jest w stanie o indeksie n lub wyższym. Zatem szukane prawdopodobieństwa to:

$$Pr(NA) = \sum_{k=n}^{\infty} P_k = 1 - \sum_{k=0}^{n-1} P_k \tag{9.6}$$

$$Pr(AV) = 1 - Pr(NA) \tag{9.7}$$

Przy obliczeniach średniego czasu hazardu skorzystano z twierdzenia o prawdopodobieństwie całkowitym w wariancie dla warunkowych zmiennych losowych:

$$E(HT) = E(FT \mid H) - R_C \tag{9.8}$$

$$= E(FT \mid H \cap AV)Pr(AV \mid H) + E(FT \mid H \cap NA)Pr(NA \mid H) - R_C \tag{9.9}$$

Analizując hazard bardziej naturalne jest poszukiwanie zdarzeń $H \mid AV$ oraz $H \mid NA$ zamiast odpowiednio $AV \mid H$ oraz $NA \mid H$, dlatego wykorzystano twierdzenie Bayesa do odwrócenia zależności:

$$E(HT) = E(FT \mid AV \cap H)\frac{Pr(H \mid AV)Pr(AV)}{Pr(H)} +$$

$$E(FT \mid NA \cap H)\frac{Pr(H \mid NA)Pr(NA)}{Pr(H)} - R_C$$

W sytuacji, gdy element rezerwowy jest dostępny, wymiana rozpoczynana jest natychmiast:

$$\text{rezerwa jest dostępna} \longrightarrow C = 0 \longrightarrow Pr(H \mid AV) = Pr(min(E, D) > R_C) \tag{9.10}$$

Obliczenie powyższego prawdopodobieństwa poprzedza wyznaczenie dystrybuanty zmiennej losowej $min(E, D)$:

$$F_{min(E,D)}(t) = F_E(t) + F_D(t) - F_E(t)F_D(t) \tag{9.11}$$

Korzystając z definicji dystrybuanty i równania 9.10:

$$Pr(H \mid AV) = 1 - F_{min(E,D)}(R_C) \tag{9.12}$$

Podobnie dla średniego czasu hazardu:

$$\text{rezerwa jest dostępna} \longrightarrow C = 0 \longrightarrow E(FT \mid H \cap AV) = E(min(E, D) \mid min(E, D) > R_C)$$

Korzystając z wyznaczonego już w 9.12 $Pr(H \mid AV)$:

$$E(FT \mid H \cap AV) = \frac{\int_{R_C}^{+\infty} t f_{min(E,D)}(t)dt}{1 - F_{min(E,D)}(R_C)}$$

Gęstość zmiennej $f_{min(E,D)}$ otrzymuje się z definicji:

$$f_{min(E,D)}(t) = \frac{d}{dt} F_{min(E,D)}(t)$$

$$\text{rezerwa jest niedostępna} \longrightarrow Pr(H \mid NA) = Pr(min(C + E, D) > R_C) \tag{9.13}$$

$$f_E(t) = \lambda_E k_E (t\lambda_E)^{k-1} e^{-(t\lambda_e)^k}$$

Do końca podrozdziału zakłada się, że zmienna C, oznaczająca czas oczekiwania w przypadku, gdy element rezerwowy jest niedostępny w chwili awarii, jest znana. Obliczenie gęstości sumy zmiennych losowych wymaga realizacji operacji splotu, co w dziedzinie liczb rzeczywistych definiuje się następująco:

$$f_{(C+E)}(t) = \int_0^t f_C(\tau) f_E(t - \tau)d\tau$$

Aby jednak uniknąć wysokiej złożoności obliczeniowej splot liczony jest w dziedzinie liczb zespolonych według następującego schematu opisanego na przykładzie zmiennej E. W pierwszym kroku gęstość ciągłej zmiennej jest dyskretyzowana:

$$e_q = \lambda_E k_E \frac{qR}{Q} \lambda_E^{k-1} e^{-\frac{qR}{Q}\lambda_e^k}$$

W powyższym wzorze Q to całkowita liczba próbek, a q to numer próbki, dla której pobierana jest wartość gęstości. Stosunek $\frac{q}{Q}$ przebiega zasięg próbkowania wyznaczany jest na podstawie zmiennej C oraz stałej ϵ oznaczającej maksymalny błąd całkowania gęstości zmiennej C:

$$\int_R^{\infty} f_C(t)dt < \epsilon$$

Następnie gęstość poddano Szybkiej Transformacie Fouriera. Spośród dostępnych w literaturze schematów wybrano algorytm Cooley'a–Tukey'iego [33] wymagający, aby Q było potęgą liczby 2. Algorytm pozwala na uzyskanie zespolonych reprezentacji gęstości (k - numer próbki):

$$C^{(k)} = \sum_{q=0}^{Q-1} c_q exp\left\{-i2\pi k\frac{q}{Q}\right\}$$

$$E^{(k)} = \sum_{q=0}^{Q-1} e_q exp\left\{-i2\pi k\frac{q}{Q}\right\}$$

Splot w dziedzinie liczb zespolonych jest operacją znacznie prostszą niż w dziedzinie liczb rzeczywistych. Do wyznaczenia zespolonej funkcji sumy wystarczy obliczyć iloczyny odpowiadających sobie składowych sumowanych zmiennych rzeczywistych:

$$EC^{(k)} = E^{(k)}C^{(k)}$$

Gęstość zmiennej $C + E$ w dziedzinie rzeczywistej otrzymuje się po zastosowaniu odwrotnej transformaty Fouriera:

$$f^{(k)}_{(C+E)} = \frac{1}{Q}\sum_{q=0}^{Q-1} EC^{(q)} exp\left\{i2\pi k\frac{q}{Q}\right\}$$

Całkując dyskretnie w odpowiednim zakresie powyższy wynik otrzymuje się dystrybuantę $C+E$ w postaci dyskretnej:

$$F^{(k)}_{C+E} = \frac{Q}{R}\sum_{q=1}^{k} f^{(q)}_{(C+E)}$$

Z kolei dystrybuantę zmiennej $min(C+E, D)$ można znaleźć zgodnie ze schematem opisanym w równaniu 9.11. Niech K oznacza numer próbki dystrybuanty odpowiadającą rezerwie czasowej:

$$K = \frac{R_C}{R}Q$$

Prawdopodobieństwo hazardu, gdy element zapasowy jest niedostępny wyznacza się ostatecznie z definicji dystrybuanty oraz równania 9.13:

$$Pr(H \mid NA) = 1 - F^{K}_{min(C+E,D)}$$

Średni czas hazardu przy braku elementu rezerwowego spełnia zależność:

$$\text{rezerwa jest niedostępna} \longrightarrow E(FT \mid H \cap NA) = E(min(C+E, D) \mid min(C+E, D) > R_C)$$

Stąd posiadając dyskretną dystrybuantę $min(C + E, D)$ i posługując się wzorem na wartość oczekiwaną warunkowej zmiennej losowej:

$$E(FT \mid H \cap NA) = \frac{\frac{R}{Q}\sum_{q=K}^{+\infty}\frac{qR}{Q}f^{(q)}_{min(C+E,D)}}{1 - F^{(K)}_{min(C+E,D)}}$$

Czynnik $\frac{R}{Q}$ to wielkość ziarna pomiędzy dwiema próbkami, a $\frac{qR}{Q}$ to liczba rzeczywista mniejsza bądź równa R oznaczająca rzeczywistą współrzędną próbki. W następnych podrozdziałach zostaną opisane trzy metody znalezienia brakującej zmiennej C - a konkretniej - jej gęstości niezbędnej do wykonania operacji splotu.

9.2 Metoda Ia: oszacowanie oparte o rozkłady hipowykładnicze

Aby wyznaczyć prawdopodobieństwo hazardu oraz średni jego czas, gdy brakuje elementu rezerwowego w chwili uszkodzenia (czyli $Pr(H \mid NA)$ oraz $E(FT \mid H \cap NA)$), potrzebna jest zmienna C oznaczająca czas oczekiwania na rezerwę w takiej sytuacji. Założeniem w niniejszym podrozdziale jest, że czas dostawy uszkodzonego pojazdu z powrotem na trasę jest aproksymowany rozkładem wykładniczym $D' = Exp(\lambda)$ o intensywności spełniającej zależność:

$$\lambda_{D'} = \frac{1}{E(D)} \tag{9.14}$$

$$F_{D'}(t) = 1 - e^{-\lambda_{D'} t} \tag{9.15}$$

$$f_{D'}(t) = \lambda_{D'} e^{-\lambda_{D'} t} \tag{9.16}$$

Gdy pewien pojazd ulega awarii oraz nie ma dostępnej rezerwy, to zajmuje kolejne miejsce w kolejce po element rezerwowy. Niech j oznacza liczbę uszkodzeń (zastępowanych lub oczekujących na rezerwę) w systemie przed rozpatrywanym uszkodzeniem, które zajmuje w związku z tym pozycję $j+1$. Podczas gdy awaria dodaje kolejne elementy do kolejki pojazdów oczekujących, dostawa, działająca dla wszystkich uszkodzonych pojazdów równocześnie i niezależnie, usuwa z niej elementy. Stąd jeśli pewien pojazd zostanie naprawiony i dostarczony z powrotem na trasę, to jest usuwany z kolejki, a wszystkie elementy na prawo przesuwane są o jedno miejsce w lewo i ich indeksy zmniejszają się o 1. Pojazd znajdujący się na pozycji $j+1$ przesunie się na pozycję j, gdy zakończy się jeden z j procesów dostawy. Czas do pierwszego przesunięcia to:

$$min(\underbrace{Exp\{\lambda\}, Exp\{\lambda\}, \ldots, Exp\{\lambda\}}_{j}) \tag{9.17}$$

Korzystając z własności funkcji minimum wykładniczych zmiennych losowych:

$$min(Exp\{\lambda_1\}, \ldots, Exp\{\lambda_n\}) = Exp\{\lambda_1 + \ldots + \lambda_n\}$$

można uprościć wyrażenie 9.17 do postaci:

$$Exp\{j\lambda\} = Exp\{\frac{j}{E(D)}\} \tag{9.18}$$

Analogicznie przesunięcie z pozycji j na $j-1$ określa zmienna $Exp\{\frac{j-1}{E(D)}\}$. Zmienna C określa czas oczekiwania na element rezerwowy, a więc czas do zajęcia pozycji n. Jeśli zatem pojazd początkowo zajął pozycję $j+1$, to całkowity czas oczekiwania na rezerwę określa zmienna C_j:

$$C_j = Exp\{\frac{j}{E(D)}\} + Exp\{\frac{j-1}{E(D)}\} + \ldots + Exp\{\frac{n}{E(D)}\} \tag{9.19}$$

Gęstość sumy wykładniczych zmiennych losowych określa gęstość rozkładu hipowykładniczego, ale pod warunkiem, że żadne dwie intensywności sumowanych zmiennych nie są sobie równe. Równanie 9.19 jest sumą $j-n+1$ wykładniczych zmiennych losowych, spośród każda ma

inną intensywność, a zatem założenie rozkładu hipowykładniczego jest spełnione. Jeśli Hyp_m oznacza zmienną losową o rozkładzie hipowykładniczym i m współczynnikach intensywności $\lambda_1, \lambda_2, \ldots, \lambda_m$ to:

$$C_j = Hyp_{j-n+1}\{\frac{j}{E(D)}, \frac{j-1}{E(D)}, \ldots, \frac{n}{E(D)}\}$$

Gęstość Hyp_m jest zdefiniowana następująco:

$$f_{Hyp_m}(t) = \sum_{i=1}^{m} \lambda_i e^{-\lambda_i t} \prod_{j,j\neq i}^{m} \frac{\lambda_j}{\lambda_j - \lambda_i} \tag{9.20}$$

Powyższe wyprowadzenie zakładało, że w systemie istniało j uszkodzeń w chwili awarii kolejnego. Prawdopodobieństwo P_j takiej sytuacji można uzyskać z omówionego modelu $M/G/\infty$ (równania 9.4 oraz 9.5):

$$P_j = \frac{p^j}{j!} e^{-\rho}$$

Ostatecznie zatem, szukana zmienna C ma postać:

$$C = \sum_{j=n}^{\infty} C_j P_j = \sum_{j=n}^{\infty} Hyp_{(j-n+1)}\{\frac{j}{E(D)}, \frac{j-1}{E(D)}, \ldots, \frac{n}{E(D)}\}P_j \tag{9.21}$$

$$f_C = \frac{\sum_{j=n}^{\infty} f_{Hyp_{(j-n+1)}}\{\frac{j}{E(D)}, \frac{j-1}{E(D)}, \ldots, \frac{n}{E(D)}\}P_j}{\sum_{j=n}^{\infty} P_j}$$

□

9.3 Metoda Ib: oszacowanie z użyciem statystyk pozycyjnych

W niniejszym podrozdziale zaprezentowano alternatywną metodę wyznaczenia zmiennej C przy założeniu, podobnie jak w podrozdziale 9.2, że czas dostawy każdego pojazdu D' jest określony wykładniczą zmienną losową zgodnie z zależnościami 9.14 - 9.16.

Niech uszkodzony pojazd zajmuje pozycję $j+1$ w kolejce. Zatem z każdym z j pojazdów uszkodzonych przed rozpatrywanym, rozpoczynając od uszkodzonego najwcześniej, można powiązać kolejną realizację zmiennej losowej czasu dostawy z ciągu $D_1, D_2, \ldots D_j$. Jeśli posortować elementy tego ciągu, to najmniejsza wartość, czyli zmienna losowa będąca pierwszą statystyką porządkowa określona jest wzorem $D^{(1)} = min(D_1, D_2, \ldots, D_j)$, a ostatnia, j-ta statystyka to $D^{(j)} = max(D_1, D_2, \ldots, D_j)$. W ogólności wzorem na gęstość k-tej statystyki jest:

$$f_{D^{(k)}}(t) = \frac{j!}{(k-1)!(j-k)!} F_{D'}(t)^{k-1}(1 - F_{D'}(t))^{j-k} f_{D'}(t)$$

Rozpatrywane uszkodzenie uzyska natomiast dostęp do rezerwy po zajściu pierwszych $(j-n+1)$ dostaw, czyli po upłynięciu czasu określonego statystyką porządkową o indeksie $(j-n+1)$:

$$C_j = D^{(j-n+1)}$$

której funkcja gęstości ma postać:

$$f_{D^{(j-n+1)}}(t) = \frac{j!}{(j-n)!(n-1)!} F_D(t)^{j-n}(1-F_D(t))^{n-1} f_D(t) \tag{9.22}$$

Zmienna C oraz jej gęstość f_C są ostatecznie równe:

$$C = \sum_{j=n}^{\infty} C_j P_j = \sum_{j=n}^{\infty} D^{(j-n+1)} P_j \tag{9.23}$$

$$f_C = \frac{\sum_{j=n}^{\infty} f_{D^{(j-n+1)}} P_j}{\sum_{j=n}^{\infty} P_j}$$

□

9.4 Metoda Ic: oszacowanie wykorzystujące zmienne wykładnicze

W dalszej części rozdziału zostanie również wykorzystany algorytm obliczeniowy operujący tylko zmiennymi wykładniczymi, w którym czasy pomiędzy awariami, dostawa oraz wymiana maja rozkłady wykładnicze. W literaturze znanych jest wiele takich schematów, ale stosowany w niniejszej pracy będzie bazował na nieznacznie zmodyfikowanej metodzie Ia. W myśl opisu w podrozdziale 9.1, oprócz aproksymacji zmiennych A i D zmiennymi A' i D' o rozkładach wykładniczych, taki sam zabieg zostanie wykonany dla wymiany, czyli:

$$\lambda_{E'} = \frac{1}{E(E)}$$

$$F_{E'}(t) = 1 - e^{-\lambda_{E'} t}$$

$$f_{E'}(t) = \lambda_{E'} e^{-\lambda_{E'} t}$$

Metoda Ic pozwala zatem na ocenę proponowanych metod poprzez porównanie z rozwiązaniami dotychczas stosowanymi.

9.5 Metoda II: oszacowanie przy pomocy rozkładów warunkowych

Zmienne C_j w oszacowaniach Ia oraz Ib były skonstruowane przy założeniu bezpamięciowości dostawy, a więc przy szacowaniu czasu potrzebnego na przesunięcie w kolejce zaniedbywano czas, który już upłynął. Gdy kształt rozkładu Weibulla dostawy jest daleki od 1 takie założenie jest nadmiernym uproszczeniem i w tym podrozdziale zostanie zdefiniowana zmienna C uwzględniająca pamięciowość rozkładu. Podobnie jak w poprzednich podrozdziałach zakłada się, że pojazd ulega uszkodzeniu, gdy w systemie jest już j awarii, czyli zajmuje $j+1$ pozycję. Ponadto symbolem $D \mid a$ oznacza się zmienną losową o rozkładzie:

$$f_{D|a}(t) = \begin{cases} 0 & \text{gdy } t < 0 \\ \dfrac{f_D(t+a)}{\int_a^{\infty} f_D(\tau)d\tau} & \text{gdy } t \geqslant 0. \end{cases}$$

Zatem zmienna $D \mid a$ określa pozostały od chwili a czas dostawy, przy założeniu, że dostawa nie zaszła do chwili a.

Kolejne awarie pojazdów zdarzają się średnio co $E(A)$. Zatem jeśli $j+1$-wsze uszkodzenie zachodzi w chwili t, to wcześniejsze zaszło (statystycznie) w chwili $t-E(A)$. Jeszcze wcześniejsze w chwili $t-2E(A)$, a najstarsze w chwili $t-jE(A)$. Czas oczekiwania na przemieszczenie pojazdu, który początkowo był na pozycji $j+1$, na pozycję j wynosi:

$$F_j^{j+1} = min\{\underbrace{D \mid E(A), D \mid 2E(A), \ldots, D \mid jE(A)}_{j}\}$$

W oszacowaniu czasu na kolejne przesunięcie, czyli na pozycję $j-1$, należy uwzględnić czas, który już upłynął czekając na awans na pozycję j. Średni czas, po którym to następuje to $E(F_j^{j+1})$, a więc:

$$F_{j-1}^{j+1} = min\{\underbrace{D \mid (E(A)+E(F_j^{j+1})), D \mid (2E(A)+E(F_j^{j+1})), \ldots, D \mid ((j-1)E(A)+E(F_j^{j+1}))}_{j-1}\}$$

Uogólniając, jeśli pojazd znajdował się początkowo na $j+1$ pozycji, to czas przesunięcia na dowolną pozycję i jest określony przez:

$$F_i^{j+1} = min\{\underbrace{D \mid (E(A)+\sum_{k=j}^{i+1} E(F_k^{j+1})), \ldots, D \mid (jE(A)-\sum_{k=j}^{i+1} E(F_k^{j+1}))}_{i}\}$$

Uszkodzonemu pojazdowi zostanie przydzielona rezerwa, gdy ten przejdzie na n-tą pozycję. Łączny czas oczekiwania na rezerwę określa zatem suma:

$$C_j = \sum_{i=j}^{n} F_i^{j+1}$$

Zmienna C oraz jej gęstość f_C są ostatecznie równe:

$$C = \sum_{j=n}^{\infty} C_j P_j = \sum_{j=n}^{\infty} P_j \sum_{i=j}^{n} F_i^{j+1}$$

$$f_C = \frac{\sum_{j=n}^{\infty} f_{D_{(j-n+1)}} P_j}{\sum_{j=n}^{\infty} P_j}$$

□

Warto zauważyć, że gdy rozkład dostawy będzie jednak wykładniczy, to bazując na własności braku pamięci można zredukować wszystkie zmienne warunkowe, co spowoduje, że F_i^{j+1} stanie się zmienną wykładniczą. Co za tym idzie, całe rozwiązanie zredukuje się do oszacowania Ia.

9.6 Eksperyment obliczeniowy

W kolejnych 3 podrozdziałach zostaną zidentyfikowane grupy systemów z rezerwą czasową, dla których opisane wcześniej algorytmy są efektywnymi metodami wyznaczania niezawodności.

Grupy systemów będą oznaczane ciągiem symboli składającym się z trzech liter, w którym pierwsza oznacza typ dystrybuanty czasu pomiędzy awariami, druga czasu wymiany, a ostatnia czasu dostawy. Przykładowo grupa systemów $M/W/M$ to taka, w której czas pomiędzy awariami ma rozkład wykładniczy (litera M), czas wymiany ma rozkład Weibulla (W), a dostawę ponownie wyraża rozkład wykładniczy.

Obliczenia w każdym z podrozdziałów przeprowadzono z wysoką starannością, tak że każda wartość była liczona przynajmniej dwa razy. Wyniki symulacji całego systemu uzyskiwano dwoma niezależnymi narzędziami: wolniejszym, będącym produktem transformacji modelu READ w kod C++ oraz szybszym, utworzonym specjalnie na potrzeby eksperymentu. Każda z proponowanych wcześniej metod estymacji została zaimplementowana dwa razy: numerycznie oraz metodą Monte Carlo samej zmiennej C. Zarówno symulatory, jak i implementacje estymacji zweryfikowano pod kątem zbieżności ze swoimi odpowiednikami dla ok. 400 badanych.

9.6.1 Model M/W/M

Dokładność oszacowań metody Ia oraz Ib wynika z dwu istotnych założeń. Pierwsze z nich polega na zastosowaniu kolejki typu $M/G/\infty$ do wyznaczenia $Pr(NA)$, czyli prawdopodobieństwa, że rezerwa nie jest dostępna w chwili uszkodzenia regularnego pojazdu. Istotnym ograniczeniem tego modelu jest wymaganie, aby czas pomiędzy kolejnymi żądaniami był opisany rozkładem wykładniczym. Obecnie nie istnieje rozwiązanie modelu $G/G/\infty$, które pozwoliłoby na relaksację tego podejścia. Drugie założenie związane jest z wykładniczym rozkładem dostawy będącym podstawą do opisu czasu do uzyskania dostępu do rezerwy rozkładem hipowykładniczym lub statystyką pozycyjną. Metoda II nie jest ograniczona wykładniczym rozkładem dostawy.

Pierwsze założenie nie będzie ograniczało jakości estymacji, gdy rozkład czasu pomiędzy awariami będzie wykładniczy. Dodatkowo, jeśli czas dostawy również byłby wykładniczy, to metody Ia i Ib dostarczą dokładnych rozwiązań zmiennej C. Warto zauważyć, że żadna z metod nie ogranicza czasu wymiany, a więc z powyższego wywodu wynika, że model M/W/M powinien być rozwiązywany dokładnie. Celem eksperymentu obliczeniowego w niniejszym podrozdziale jest weryfikacja tego spostrzeżenia.

Czas pomiędzy uszkodzeniami oraz dostawa mają w eksperymencie rozkłady wykładnicze, a wymiana ma rozkład Weibulla o kształcie 1,5. Co więcej, przyjęto 2 możliwości rezerwy czasowej oraz 5 stopni rezerwowania, co łącznie daje 10 przypadków testowych zebranych w tab. 9.2.

Wartości uzyskanego prawdopodobieństwa hazardu oraz średniego jego czasu okazały się zbieżne z estymacjami dla wszystkich przypadków testowych i te wartości umieszczono w ostatnich dwu kolumnach tabeli. Zatem w przypadku modeli systemów z rezerwą czasową typu M/W/M, symulacja okazuje się niepotrzebna, a parametry hazardu można otrzymać metodami analitycznymi. Wynikający z tego zysk w czasie obliczeń podsumowano w tab. 9.3.

Obydwie metody Ia i Ib okazały się wyraźnie szybsze niż silnie zoptymalizowany symulator, ale to druga z nich jest najwydajniejsza dostarczając wyniki w czasie ok. 0,8s. Metoda oparta o statystyki pozycyjne jest szybsza z następującego powodu.

Tabela 9.2: Przypadki testowe dla modelu M/W/M

Nr przypadku	A	E		D	R_C [min]	n	Pr. hazardu $Pr(H)$	Średni czas hazardu $E(HT)$ [min]
	λ_A	k_E	λ_E	λ_D				
1	0,02	1,5	0,05	0,01	41	1	0,53134	62,44
2	0,02	1,5	0,05	0,01	41	2	0,32659	41,83
3	0,02	1,5	0,05	0,01	41	3	0,16704	28,89
4	0,02	1,5	0,05	0,01	41	4	0,08304	19,19
5	0,02	1,5	0,05	0,01	41	5	0,04975	12,27
6	0,02	1,5	0,05	0,01	101	1	0,21212	57,30
7	0,02	1,5	0,05	0,01	101	2	0,08022	38,63
8	0,02	1,5	0,05	0,01	101	3	0,02219	28,74
9	0,02	1,5	0,05	0,01	101	4	0,00483	22,68
10	0,02	1,5	0,05	0,01	101	5	0,00088	18,67

Tabela 9.3: Porównanie czasów obliczeń

Nr przypadku	Czas obliczeń [s]		
	metoda Ia	metoda Ib	Symulator
1	3,28	0,81	75,88
2	2,58	0,81	76,95
3	2,17	0,80	76,66
4	1,69	0,80	75,31
5	1,41	0,78	76,84
6	3,08	0,81	76,24
7	2,55	0,80	75,27
8	2,12	0,80	75,47
9	1,74	0,80	75,38
10	1,42	0,78	75,67

Jakkolwiek zmienna C we wzorze 9.21 wynika z ważonej sumy nieskończonego ciągu zmiennych o rozkładach hipowykładniczych odpowiadającym kolejnym miejscom w kolejce, to implementacja metody Ia kończy obliczenia, gdy iloczyny C_j są bardzo małe i dalsze obliczenia wnoszą bardzo niewiele. Ponieważ czas wyznaczenia wartości gęstości rozkładu hipowykładniczego jest proporcjonalny do kwadratu liczby stopni tego rozkładu (wzór 9.20), to czas potrzebny na obliczenie zmiennej C zależy od trzeciej potęgi maksymalnej głębokości kolejki. Natomiast czas obliczenia statystyki pozycyjnej w metodzie Ib jest proporcjonalny do głębokości, co wynika z operacji potęgowania (wzór 9.22), więc cały algorytm zależy od kwadratu głębokości (9.23).

Z powyższego spostrzeżenia wynika również dlaczego czas obliczeń w metodzie Ia spada, gdy wzrasta liczba pojazdów rezerwowych, co widać osobno w przypadkach 1-5 oraz 6-10. Ponieważ przegląd kolejki rozpoczyna się od miejsca tuż za ostatnim rezerwowym, wymagany nakład obliczeniowy jest mniejszy. Powyższego efektu nie widać w metodzie Ib ponieważ zysk ze skracania kolejki jest konsumowany przez wyższy stopień operacji potęgowania.

9.6.2 Model W/W/M

W ogólności jakość aproksymacji w proponowanym modelu obliczeniowym wynika z poprawności szacowania dwu czynników: $Pr(NA)$ oraz f_C. Model W/W/M to taki, w którym czasy pomiędzy awariami oraz wymiana są opisane rozkładem Weibulla, podczas gdy dostawa ma wciąż rozkład wykładniczy. Zatem jedyną przyczyną błędów w szacowaniu niezawodności systemów o naturze W/W/M metodami Ia, Ib oraz II będzie niedokładność $Pr(NA)$ wynikająca z zastosowania modelu kolejkowego $M/G/\infty$ zamiast $G/G/\infty$. Gęstość zmiennej C jest w takich przypadkach dokładna, gdyż wymienione metody poprawnie wyznaczają czas oczekiwania na rezerwę, gdy dostawa ma rozkład wykładniczy.

Eksperyment obliczeniowy przeprowadzono przy zmieniającym się parametrze kształtu czasu pomiędzy awariami. Dane wejściowe zebrano w tab. 9.4.

Tabela 9.4: Zmienne w testach modelu W/W/M

Zmienna	**Wartości**
A	$\lambda = 0,016$, $k = 0,9 \ldots 1,1$; krok 0,01
E	$\lambda = 0,026$, $k = 1,25$
D	$\lambda = 0,07$
R_C	41 lub 101
n	$1 \ldots 5$

Metoda Ia, Ib oraz II nie różnią się algorytmem wyznaczania $Pr(NA)$, więc dostarczają jednakowe błędy względne przedstawione na rys. 9.1-9.4, a zdefiniowane następująco:

$$\delta_H(k_A, R_C, n) = \frac{est_H(k_A, R_C, n) - sym_H(k_A, R_C, n)}{sym_H(k_A, R_C, n)} * 100\%$$

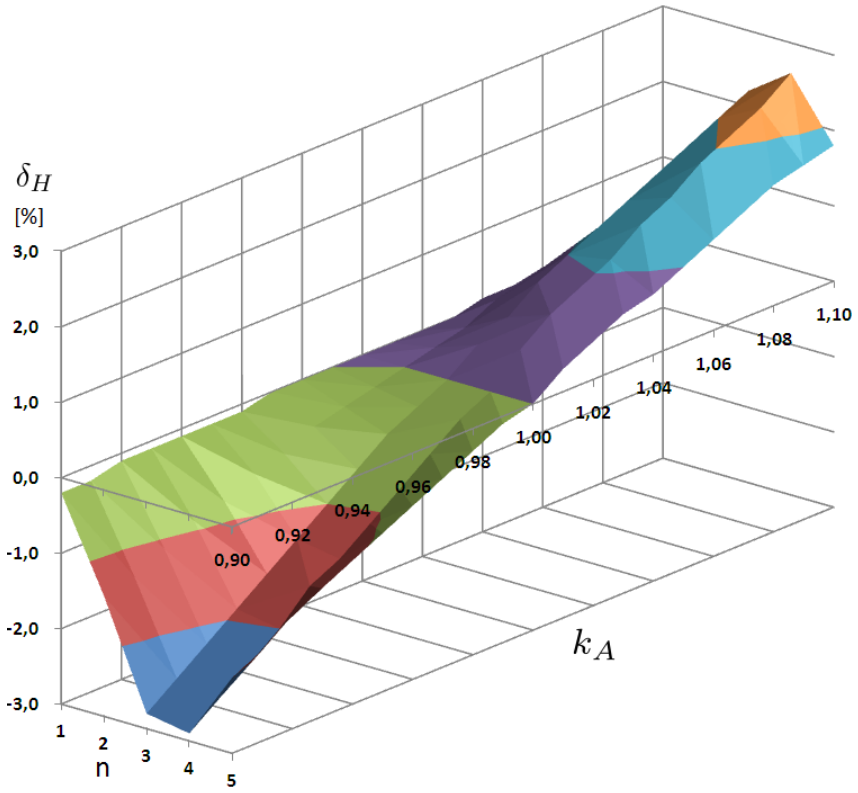
$$\delta_{HT}(k_A, R_C, n) = \frac{est_{HT}(k_A, R_C, n) - sym_{HT}(k_A, R_C, n)}{sym_{HT}(k_A, R_C, n)} * 100\%$$

Gdzie:

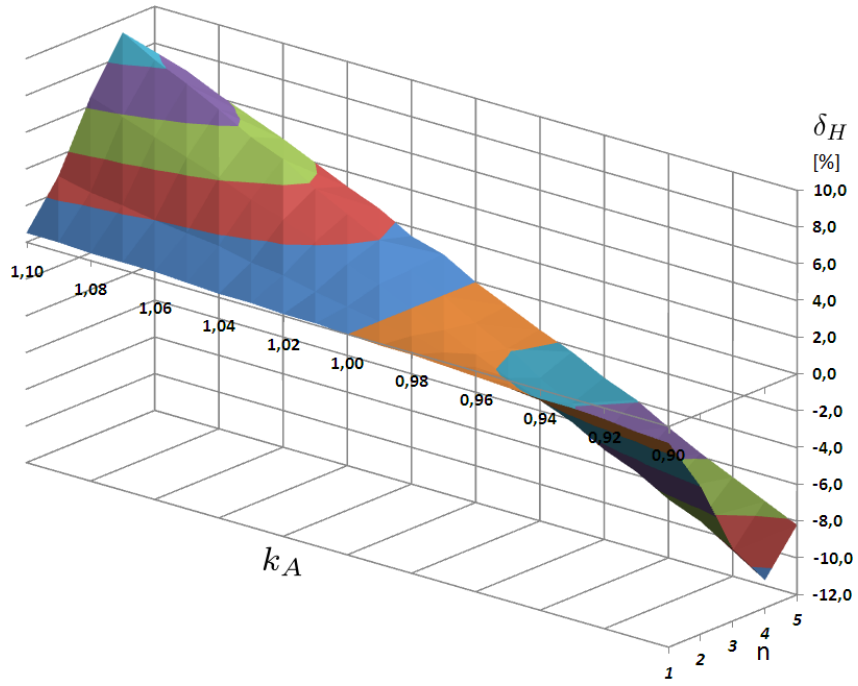
- δ_H, δ_{HT} - błąd względny $Pr(H)$ oraz $E(HT)$ w zależności od kształtu zmiennej A, rezerwy czasowej R_C oraz liczby elementów rezerwowych n,
- est_H, est_{HT} - wyniki estymacji parametrów hazardu,
- sim_H, sim_{HT} - wyniki symulacji parametrów hazardu.

Ponieważ wśród zbioru parametrów kształtu awarii jest wartość $1,0$ w zbiorze testowanych danych istnieją modele M/W/M, które rozwiązywane są dokładnie. Stąd na rysunkach można zaobserwować, że $\delta_H = 0$ oraz $\delta_{HT} = 0$, gdy awaria ma rozkład wykładniczy.

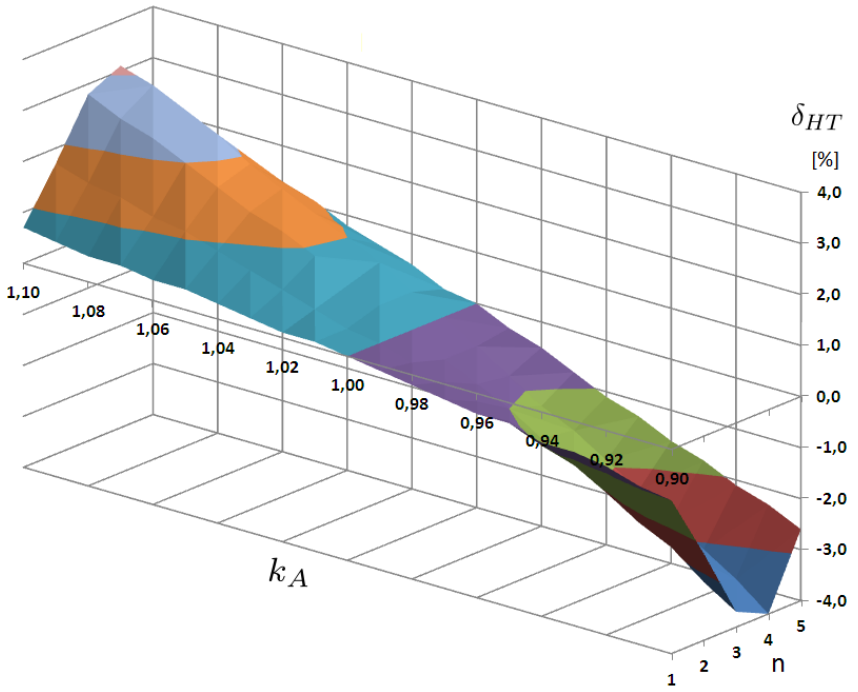
Gdy kształt awarii oddala się od 1, zarówno ku wartościom większym, jak i mniejszym, błędy metody zwiększają się. Warto zauważyć, maksymalny błąd jest mniejszy od 12%, co w większości zastosowań okaże się wystarczające i pozwoli uniknąć czasochłonnej symulacji działania systemu.



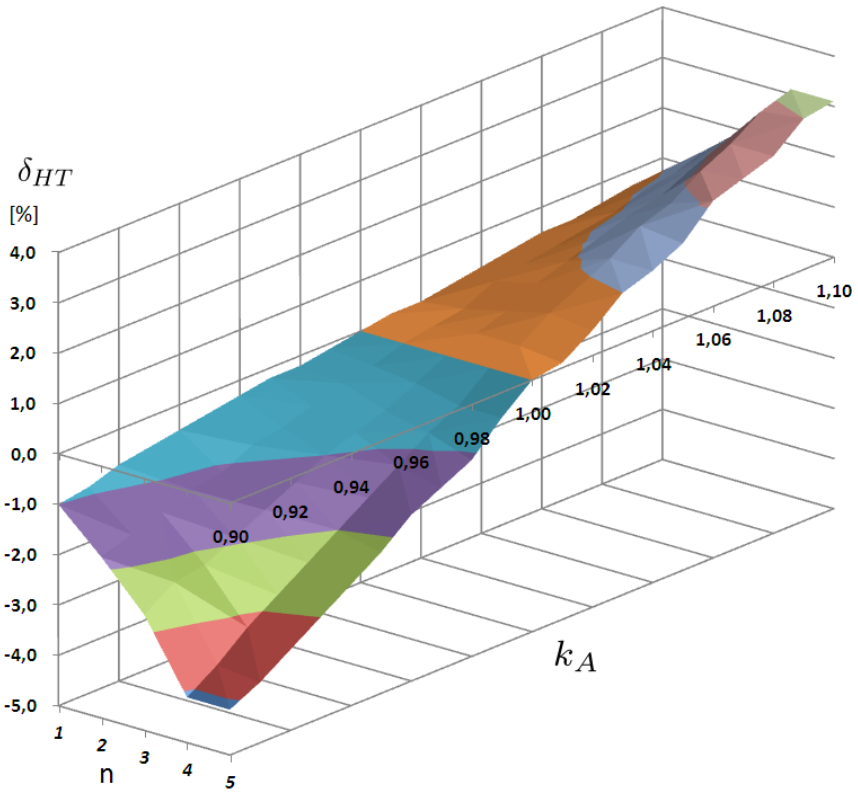
Rysunek 9.1: Błędy względne w szacowaniu $Pr(H)$ w modelu W/W/M, R_C=41 min [99]



Rysunek 9.2: Błędy względne w szacowaniu $Pr(H)$ w modelu W/W/M, R_C=101 min [99]



Rysunek 9.3: Błędy względne w szacowaniu $E(HT)$ w modelu W/W/M, R_C=41 min [99]



Rysunek 9.4: Błędy względne w szacowaniu $E(HT)$ w modelu W/W/M, R_C=101 min [99]

Analizując bardziej precyzyjnie, jeśli czas awarii jest zmienną o rozkładzie Weibulla o tzw. „ciężkim ogonie", czyli $k < 1$, to metody dostarczą rozwiązania niedoszacowane. Przeciwnie, gdy „ogon" jest lekki i $k > 1$, to wartości estymowane będą zbyt duże w stosunku do rezultatów symulacyjnych. Zjawisko wynika z działania modelu $M/G/\infty$, który dostarczy mniejsze $Pr(NA)$ niż w rzeczywistości, gdy awaria ma ciężki „ogon" i większe, gdy „ogon" jest lekki. Zważywszy, że $Pr(H \mid NA)$ jest większe niż $Pr(H \mid AV)$, to, zgodnie ze wzorem 9.3, mniejsze $Pr(NA)$ przyczyni się do zmniejszenia $Pr(H)$ i odwrotnie. Analogicznie dla średniego czasu hazardu.

Zgodnie z przypuszczeniami prawdopodobieństwo hazardu jest monotoniczne względem liczby pojazdów rezerwowych i wyraźnie spada, gdy pula rezerwowa jest liczniejsza, ale błąd względny oszacowań wśród badanych przypadków testowych ma ekstremum, gdy $n = 4$.

9.6.3 Model W/W/W i podsumowanie eksperymentu

W najbardziej ogólnym z rozpatrywanych przypadków czas pomiędzy awariami, wymiana oraz dostawa mają rozkłady Weibulla. Taka konfiguracja wejściowa stanowi największe wyzwanie dla proponowanych metod obliczeniowych.

Analogicznie do poprzedniego eksperymentu w obliczeniach przyjęto maksymalne parametry kształtu: czasu między awariami oraz wymiany. Kształt dostawy natomiast zmieniał się w zakresie $1,0 \ldots 1,5$. Biorąc pod uwagę jedną rezerwę czasową oraz trzy stopnie rezerwowania otrzymuje się łącznie 60 przypadków testowych opisanych w tab. 9.5. Rezultatem obliczeń każdego z nich są błędy względnego prawdopodobieństwa hazardu oraz średniego jego czasu.

Tabela 9.5: Zmienne w testach modelu W/W/W

Zmienna	**Wartości**
A	$\lambda_A = 0,02$, $k_A = 1,1$
E	$\lambda_E = 0,05$, $k_E = 1,5$
D	$\lambda_D = 0,014$, $k_D = 1,0 \ldots 1,5$; krok: 0,1
R_C	41 lub 101
n	$1 \ldots 5$

Na każdym z 6 wykresów rys. 9.5 metoda II okazała się wyraźnie lepsza nie tylko od schematu dedykowanego systemom o zmiennych wykładniczych, ale również od metody wykorzystującej statystyki porządkowe i dopuszczającej niewykładniczy czas wymiany. Dominacja metody II rośnie wraz ze zwiększaniem współczynnika kształtu dostawy.

Dodatkowo, odstępy na wykresach prawdopodobieństwa oraz średniego czasu hazardu pomiędzy liniami punktowanymi a przerywanymi rosną, gdy zwiększa się stopień rezerwowania. Można to wyjaśnić następująco. Gdy n jest większe, to szansa na otrzymanie elementu rezerwowego w chwili awarii również rośnie, czyli $Pr(NA)$ maleje. Tym samym prawdopodobieństwo wyprzedzenia dostawy przez wymianę rośnie, czyli proces wymiany ma większe znaczenie dla niezawodności. Konsekwentnie prawidłowy opis wymiany w modelu obliczeniowym przyczyni się do poprawy jakości oszacowania, czyli linia reprezentująca wyniki modelu Ib obniża się w sto-

sunku do linii metody Ic (np. rys. 9.5e). Przeciwnie, gdy $Pr(NA)$ jest bliskie 1, to zysk z metody opartej o statystyki porządkowe jest niewielki, gdyż prawidłowe szacowanie wymiany ma wtedy marginalne znaczenie. W takiej sytuacji metoda Ib jest nieznacznie lepsza od Ic dedykowanej systemom *M/M/M* (np. rys. 9.5a).

Naturalnie, dodawanie pojazdów rezerwowych zmniejsza prawdopodobieństwo hazardu. Pamiętając o wniosku wysuniętym w poprzednim paragrafie, należy uznać, że proponowane metody są szczególnie przydatne w systemach o wysokiej lub bardzo wysokiej niezawodności. Prawdopodobieństwa hazardu na wykresach 9.5a, 9.5c oraz 9.5e kształtują się na poziomie: $0,002$, $0,0001$ oraz $0,0001$. Stosowanie metody o zmiennych wykładniczych w systemach o wysokim stopniu rezerwowania prowadzi do nadmiernych błędów, które mogą być w dużym stopniu ograniczone poprzez użycie proponowanych metod. Gdy kształt dostawy jest bliski 1, to zarówno metody I, jak i metoda II okażą się odpowiednie. Gdy jednak w badanym systemie kształt dostawy jest daleki od 1, to jedynie metoda II może być rekomendowana, która w takiej sytuacji wyraźnie przewyższy nie tylko podstawową metodę bazującą na zmiennych wykładniczych, ale również bardziej skomplikowaną wykorzystującą statystyki porządkowe.

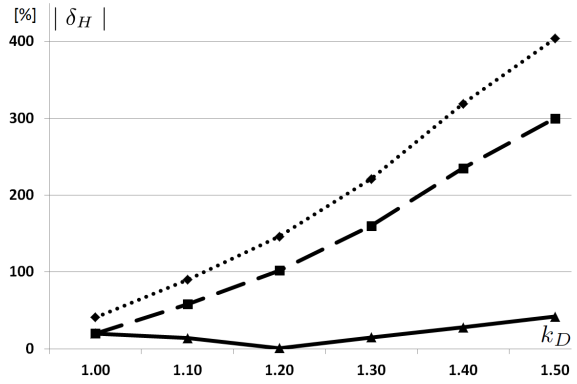
W ramach podsumowania w tab. 9.6 porównano błędy w szacowaniu niezawodności określonej klasy systemu przez każdy z proponowanych algorytmów. Konfrontując jakość oszacowań z czasem potrzebnym na ich uzyskanie (tab. 9.7) można zauważyć, że najlepszymi schematami są: metoda Ib dla wszystkich klas oprócz W/W/W oraz metoda II dla klasy W/W/W.

Tabela 9.6: Jakość estymacji proponowanymi metodami na tle 4 klas systemów z rezerwą czasową

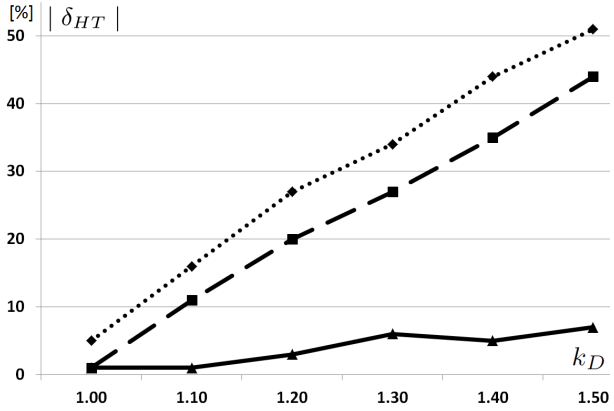
Metoda	**Model**			
	M/M/M	M/W/M	W/W/M	W/W/W
Ic	Wynik dokładny	Błąd umiarkowany	Błąd duży	Błąd bardzo duży
Ia	Wynik dokładny	Wynik dokładny	Błąd niski	Błąd duży
Ib	Wynik dokładny	Wynik dokładny	Błąd niski	Błąd duży
II	Wynik dokładny	Wynik dokładny	Błąd niski	Błąd umiarkowany

Tabela 9.7: Średnie czasy obliczeń dla jednego przypadku testowego

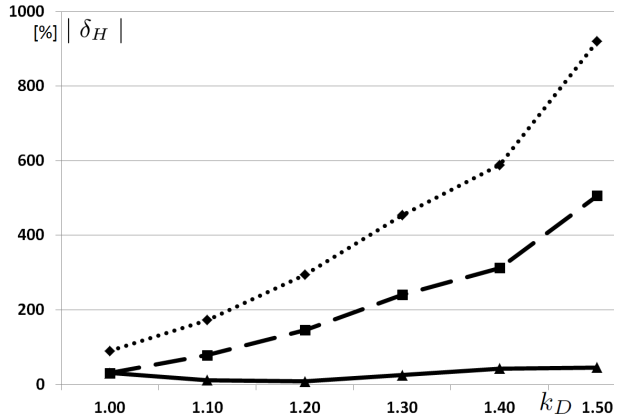
Metoda	**Przeciętny czas obliczeń** [s]
Ib	2,5
Ic	0,8
II	16,1
Symulacja	75,4



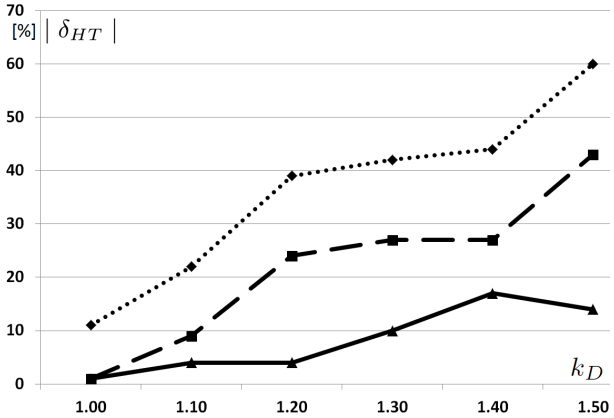
(a) Prawdopodobieństwo hazardu, $n = 3$



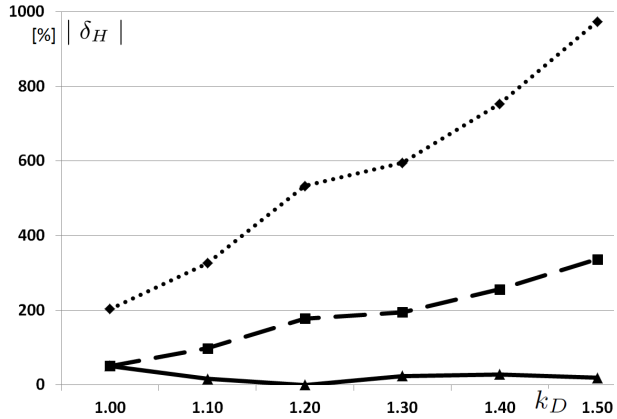
(b) Średni czas hazardu, $n = 3$



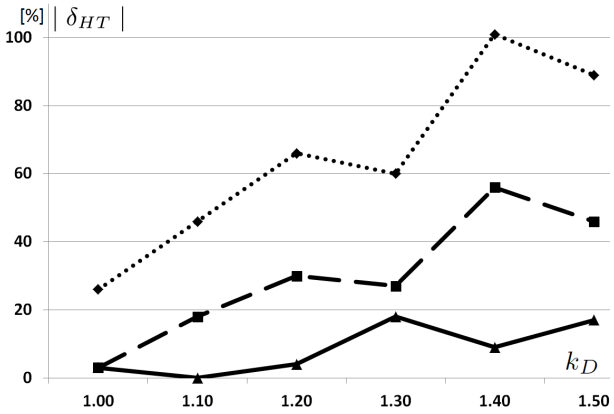
(c) Prawdopodobieństwo hazardu, $n = 4$



(d) Średni czas hazardu, $n = 4$



(e) Prawdopodobieństwo hazardu, $n = 5$



(f) Średni czas hazardu, $n = 5$

Rysunek 9.5: Błąd względny prawdopodobieństwa hazardu (lewa kolumna) oraz średniego czasu hazardu (prawa kolumna) w zależności od współczynnika kształtu dostawy k_D; linia punktowana - metoda Ic, linia przerywana - metoda Ib, linia ciągła - metoda II [99]

Rozdział 10

Analiza i modelowanie niskokosztowych procesów eksploatacji bez rezerwy, z naprawami oraz przeglądami

Zastosowanie PDNZC do precyzyjnego modelowania złożonych procesów eksploatacji jest niemożliwe. Chociaż kolejna ich wersja - Grafy Niezdatności - stanowią duży postęp w mocy ekspresji pozwalając na opis m. in. rezerwy czasowej w systemie tramwajowym, to wciąż nie byłyby w stanie sprostać wymaganiom omówionego w niniejszym rozdziale procesu konserwacji samolotów niskokosztowych linii lotniczych (NLL).

W systemie tramwajowym zakładano, że pojazd może być albo sprawny albo uszkodzony, a stan niezawodnościowy jednego tramwaju nie ma wpływu (poza dostępnością rezerwy) na eksploatację pozostałych. Ponadto optymistycznie zakładano, że rezerwa jest zawsze sprawna. W systemie lotniczym jest inaczej. Choć nie ma dedykowanej puli rezerwowej, to pozostałe samoloty przejmują obowiązki uszkodzonego i tym samym same eksploatowane są intensywniej, co prowadzi do szybszej ich awarii.

Kolejnym założeniem w modelu systemu tramwajowego jest brak bezpośredniego wpływu położenia geograficznego uszkodzonego tramwaju na dalszą jego eksploatację oraz niezdatność całego systemu. Istnieje jedynie pośredni, wyrażony przez wyznaczoną z danych rzeczywistych zmienną, wpływ położenia na szybkość wymiany. Brakuje natomiast możliwości badania wpływu przemieszczenia stacjonowania rezerwy na niezawodność systemu. Dla kontrastu, w systemie NLL uszkodzenie samolotu w bazie, gdzie często stacjonują inne statki jest prostsze w obsłudze niż uszkodzenie na lotnisku kursowym i wygenerowane opóźnienia będą statystycznie mniejsze.

Celem niniejszego rozdziału jest demonstracja mocy ekspresji języka READ w konfrontacji z problemem specyfikacji eksploatacji samolotów NLL o znacznie większych wymaganiach opisu niż inne problemy rozważane do tej pory w rozprawie. Modele READ będą ukierunkowane na znalezienie czasów opóźnień lotów zarówno przy starcie, jak i lądowaniu wśród lotów opóźnionych, czyli na wyznaczenie metryk w literaturze określanych jako *ADD* (ang. *average delay per*

departure) oraz *ADA* (ang. *average delay per arrival*).

Zmienne losowe parametryzujące kolejne podmodele READ wyznaczono w ramach prac nad inną rozprawą doktorską [80], gdzie oparto się o następujące źródła:

1. dane historyczne przewoźnika Wizzair z 2009 roku,
2. statystyki organizacji EUROCONTROL (ang. *European Organisation for the Safety of Air Navigation*) planującej i koordynującej ruch statków powietrznych w Europie,
3. obserwacje w ramach pracy we Wrocławskim Porcie Lotniczym im. Mikołaja Kopernika.

10.1 Niektóre założenia niskokosztowych linii lotniczych

Dla każdego lotu F1 w okresie 29.03.2009 – 25.10.2009 rozumianego jako ⟨ lotnisko startowe A, lotnisko końcowe B, samolot C ⟩ istniał lot F2 ⟨ B, A, C ⟩ występujący po F1. Niech F1 i F2 będą nazwane lotami stowarzyszonymi. Ponadto F2 występuje najczęściej od razu po F1, choć zdarzają się ciągi typu F1 F3 F4 F2, gdzie F3 i F4 to również loty stowarzyszone. W takiej sytuacji F1 i F3 wykonywane są wieczorem, a F4 i F2 kolejnego dnia rano. Celem takich ciągów jest ekonomiczny transport samolotu do lotniska F3, gdzie w nocy odbywa się np. inspekcja. Ciąg lotów tego samego statku w jednym dniu tworzy serię będącą listą dwukierunkową.

Z powyższej obserwacji wynika, że NLL mają infrastrukturę opartą na lotniskach – *bazach*, gdzie stacjonują samoloty w nocy oraz *lotniskach kursowych*. Bazy to zwykle średniej wielkości porty w Europie Centralnej lub Wschodniej. W ciągu dnia samoloty opuszczają bazy i kursują pomiędzy lotniskami małymi oraz większymi, pod wieczór jednak zawsze wracają do którejś z baz, zazwyczaj do tej, z której wyleciały rano. Priorytetem jest ograniczenie czasu przebywania w lotniskach większych do minimum. Konsekwentnie NLL nie latają do lotnisk dużych ze względów ekonomicznych (zamiast Londyn Heathrow lądują np. w Luton, zamiast Frankurt am Main - Frankfurt Hahn, zamiast Paris Charles de Gaulle – Beauvais Tillé), chyba że jest to faktycznie opłacalne - np. ze względu na bazę przesiadkową w Madryt Barajas.

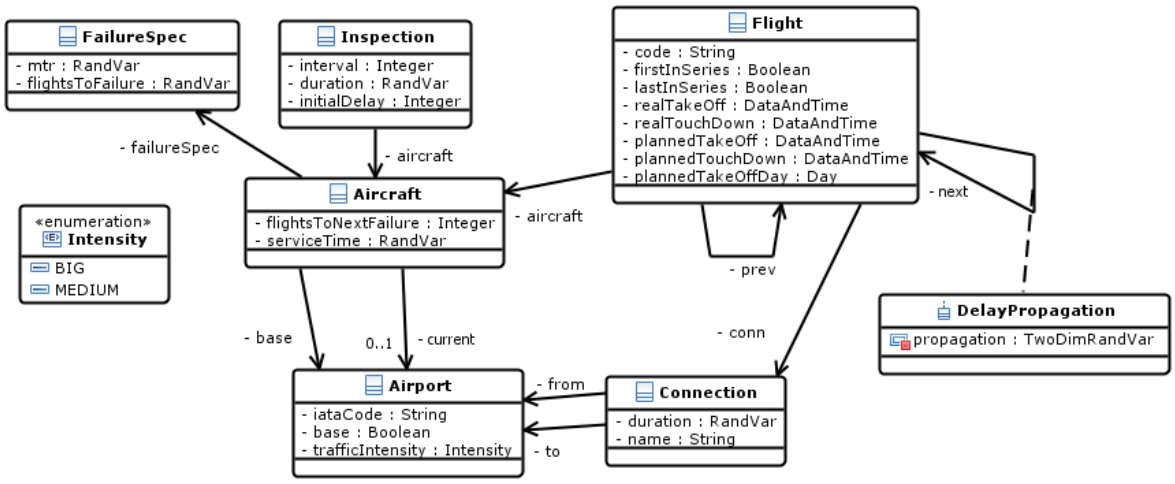
Pule samolotów rezerwowych u przewoźników NLL są rzadkością i pozwolić sobie na nią mogą jedynie najwięksi operatorzy jak Lufthansa. Spośród 33 samolotów obsługiwanych przez Wizzaira w 2009 roku wszystkie były intensywnie eksploatowane, a jedyna dłuższa przerwa w ciągu doby miała miejsce w godzinach ok. 0:00 - 6:00. Krótkie przerwy pomiędzy lotami w ciągu dnia są wykorzystywane na przygotowanie statku do kolejnego lotu i ewentualnie nadrabianie niewielkich opóźnień. Z analogicznych względów przewoźnicy NLL unikają lotów bez pasażerów ograniczając je do minimum.

10.2 Model strukturalny systemu niskokosztowych linii lotniczych

Istotną zmianą w języku READ w stosunku do wcześniejszych jego wersji jest izolacja opisu struktury systemu od modelu jego niezawodności. Diagram klas będący w READ środkiem wyrazu struktury przejmuje rolę słownika diagramu aktywności dostarczając definicji bytów,

ich atrybutów, związków między klasami oraz pośrednio stanów w których mogą znajdować się obiekty. To naturalne, że asocjacje pomiędzy klasami często implikują zachowanie systemu wskutek awarii, więc rozpatrzenie diagramu klas pomaga zrozumieć i rozszerzać diagram READ.

Przykładem takiego opisu jest model na rys. 10.1 będący częścią omawianego w rozdziale zastosowania READ do problemu NLL.



Rysunek 10.1: Diagram klas systemu NLL

Centralną klasą jest *Aircraft* opisująca atrybuty i relacje samolotów z pozostałymi elementami systemu. Cechami każdego samolotu są: wyznaczony z danych rzeczywistych czas obsługi pomiędzy lotami nazwany *serviceTime* oraz liczba lotów pozostałych do kolejnego uszkodzenia reprezentowana przez atrybut $flightsToNextFailure$, którego wartość jest zmniejszana po każdym wykonanym locie. Samolot ma ponadto ma swój port-bazę, w której najczęściej stacjonuje nocą (atrybut *base*), a ponadto, jeśli aktualnie nie wykonuje lotu, to ma przyporządkowane lotnisko aktualnego stacjonowania *current*. Obiekty klasy *Airport* mają przypisaną intensywność ruchu $TrafficIntensity$ - średnią lub dużą.

Klasa *Flight* modeluje lot. Każdy z nich opisywany jest planowanym i rzeczywistym czasem startu oraz lądowania. Loty uporządkowane są na liście dwukierunkowej według chwil planowanego ich startu, co realizują asocjacje *next* oraz *prev*. Lotnisko końcowe określonego lotu jest zawsze zgodne z lotniskiem początkowym kolejnego. Taka struktura jest konieczna z dwu powodów. Po pierwsze, w sytuacji dużego opóźnienia kilka lotów tego samego samolotu i z tego samego lotniska może być gotowych do realizacji, co wymaga ich uporządkowania. Po drugie, jak wynika z opisu modelu lotów przedstawionego w 10.3, opóźnienie pomiędzy kolejnymi lotami jest zależne od typu lotniska, na którym ląduje samolot między tymi lotami. Stąd prawidłowym miejscem do specyfikacji takiego opóźnienia jest klasa asocjacji *next* nazwana *DelayPropagation*. Jedynym jej atrybutem jest dwuwymiarowa zmienna losowa omówiona w dalszej części rozdziału. Pierwszy z lotów każdej listy ma atrybut $firstInSeries$ o wartości odpowiadającej logicznej prawdzie, podczas gdy ostatni z lotów ma ustawioną flagę $lastInSeries$.

Każdy lot jest realizacją połączenia (klasa *Connection*) definiującego port startowy, docelowy

oraz planowy czas trwania lotu. Obiekty *Connection* tworzone są na podstawie siatki Wizzair z sezonu letniego 2009 roku obejmującej 160 połączeń pomiędzy 64 lotniskami. Łącznie w tym czasie wykonano 32000 lotów.

Kolejnym po lotach procesem, któremu poddawane są samoloty jest wykrycie i naprawa uszkodzeń. Stąd obiekt $Aircraft$ posiada referencję do specyfikacji uszkodzeń wynikającej z typu modelowanego samolotu. To w obiektach $FlightSpec$ można odnaleźć zmienną losową $flightsToFailure$, która generuje kolejne wartości $flightsToNextFailure$ powiązanego obiektu $Aircraft$. Atrybut mtr to zmienna losowa czasu naprawy samolotu przez ekipę naprawczą.

Każda inspekcja przeprowadzana na samolocie modelowana jest poprzez osobny obiekt *Inspection* posiadający referencję do samolotu będącego jej przedmiotem. W niniejszym rozdziale ograniczono się do tzn. inspekcji czasowych przeprowadzanych co określony interwał na każdym samolocie w celu wykrycia powstałych uszkodzeń. Atrybutami klas $Inspection$ są *interval* będący stałą określającą liczbę minut pomiędzy dwoma przeglądami oraz *duration* stanowiący zmienną losową czasu trwania przeglądu. Istotnym elementem jest również atrybut *initialDelay* pozwalający na synchronizację pierwszych inspekcji różnych samolotów.

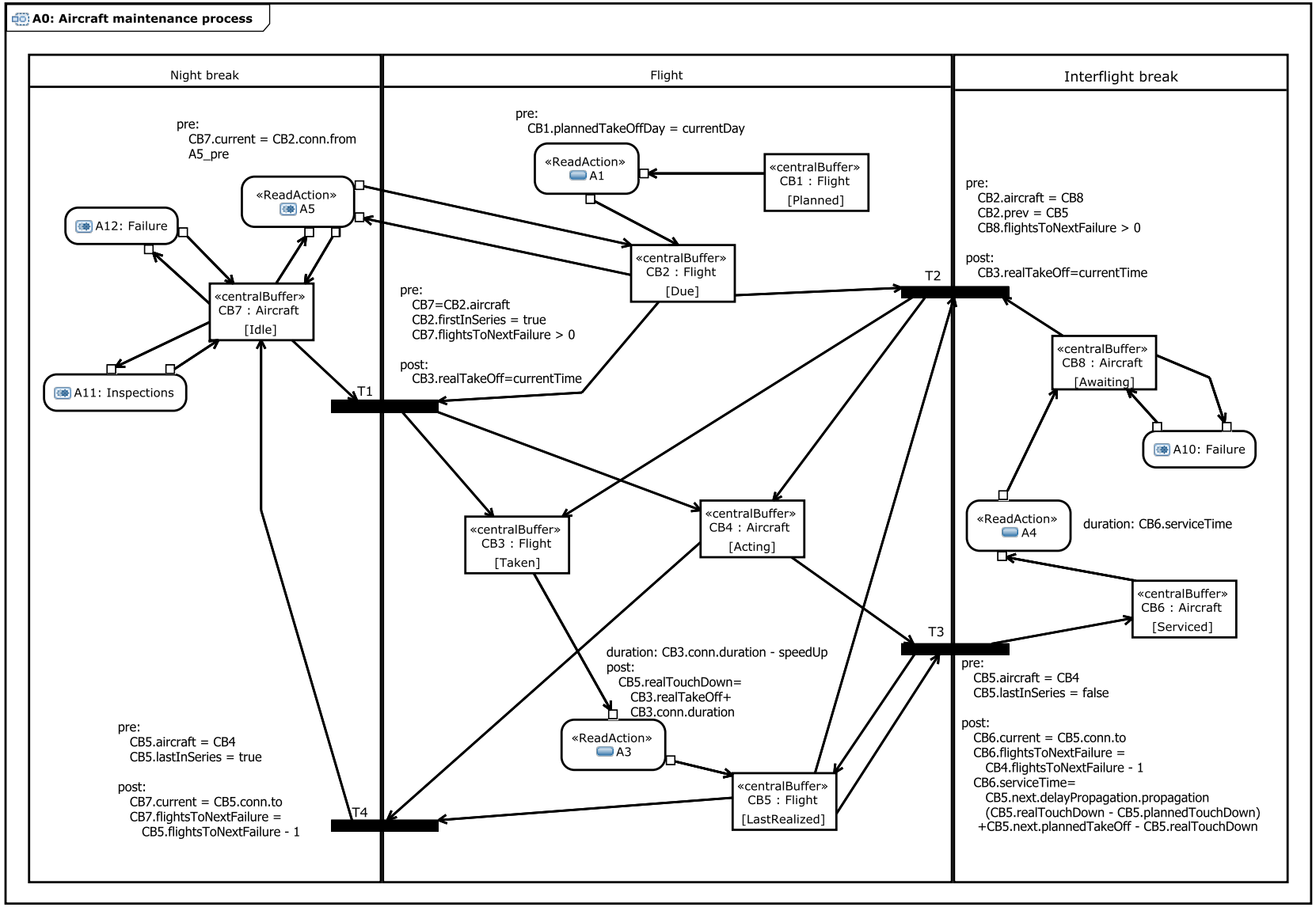
10.3 Główny model procesu eksploatacji

Przewoźnik Wizzair nie posiada samolotów rezerwowych, a wszystkie statki są w ciągłym użytkowaniu, stąd proces eksploatacji można podzielić na 3 stałe elementy najwyższego poziomu. Przepływ obiektów pomiędzy torami *Night break*, *Flight* oraz *Interflight break* na rys. 10.2 opisuje interakcje tych podstawowych etapów. Poprzez akcje $CallBehaviour$, czyli $A10$, $A11$ i A12, wywoływane są zachowania niższego poziomu, takie jak detekcja i obsługa uszkodzeń (omówiono w podrozdziale 10.4), inspekcje samolotu (10.5), czy algorytm zastępowania lotów (10.6).

Elementarny proces głównego toru: start - lot - lądowanie należy uzupełnić wiarygodnym modelem ruchu uwzględniającym takie elementy jak: wpływ opóźnień na starcie na opóźnienie przy lądowaniu, zjawisko zwiększania opóźnień na dużych lotniskach, jak i polecenia wież kontrolnych dotyczące zmiany trasy.

W chwili początkowej analizy w buforze $CB1$ znajdują się obiekty symbolizujące wszystkie loty do wykonania przez przewoźnika przy pomocy zbioru samolotów umieszczonych w $CB7$. Gdy nadejdzie odpowiedni dzień, wszystkie loty serii tego dnia zostaną przeniesione przez akcję $A1$ do bufora $CB2$, gdzie będą oczekiwały na wykonanie. Dzięki zastosowaniu profilu Marte do opisu czasu, zmienne losowe zawierają jednostki, a więc czas trwania akcji może być wyrażony również w dniach.

Przejścia $T1 - T4$ odpowiadają za przenoszenie obiektów typu $Aircraft$ pomiędzy torami, czyli modelują chwile rozpoczęcia i zakończenia lotu, przerwy nocnej i przerwy między lotami określonego samolotu. Początkowo wszystkie samoloty znajdują się w buforze $CB7$ i będąc w stanie $Idle$ są gotowe do podjęcia pierwszego lotu danego dnia, gdy zajdzie chwila jego startu.



Rysunek 10.2: Diagram READ opisujący: loty, przerwy pomiędzy lotami oraz przerwy nocne systemu NLL [82]

Rolą przejścia $T1$ jest rozpoczęcie pierwszego takiego lotu, czyli usunięcie skojarzonych obiektów z buforów $CB2$ i $CB7$ oraz dodanie ich do $CB3$ i $CB4$. Aby doszło do odpalenia, samolot i lot muszą stanowić parę, co uzasadnia zapis $CB7 = CB2.aircraft$ w warunkach *pre* przejścia $T1$. Ponadto lot musi być oznaczony jako pierwszy w danym dniu ($CB2.firstInSeries = true$) oraz samolot nie może być uszkodzony ($CB7.flightsToNextFailure > 0$). Zatem zarówno samolot, jak i lot zmienią swój stan: z odpowiednio *Idle* na *Acting* oraz z *Due* na *Taken*, a w obiekcie lotu zostanie zapisany faktyczny czas startu. W efekcie wykonania $T1$ pierwszy lot nowego dnia zostanie podjęty przez samolot.

Uzupełnione bufory $CB3$ oraz $CB4$ są wejściowymi dla konstrukcji $T3$, $T4$ oraz $A3$, lecz obydwa przejścia czekają na obiekt w $CB5$, czyli pośrednio na wymienioną $A3$. Jest to akcja modelująca wykonanie lotu, stąd czas jej trwania wynika z realizowanego połączenia, czyli $CB3.conn.duration$, choć wpływ ma również możliwość przyspieszenia samolotu na trasie. Przyjmując, że $N(t; x; y)$ to gęstość zmiennej normalnej o średniej x i odchyleniu standardowym y, to wyznaczony z danych rzeczywistych czas przyspieszenia jest średnią ważoną dwóch rozkładów:

$$f_P(t) = 0,73N(t; 9,16; 4,22) + 0,27N(t; 17,68; 2,79),$$

Średnio istnieje zatem ok. 11 minutowa rezerwa czasowa na realizację lotu.

W warunku *post* $A3$ zapisuje się jednocześnie rzeczywisty czas lądowania *CB5.realTouchDown*, który podobnie jak *realTakeOff* służy do wyznaczenia średniego czasu opóźnienia przy starcie ADD i lądowaniu ADA. Akcja przeniesie obiekt z $CB3$ do $CB5$, w następstwie czego lot będzie miał stan *LastRealized*.

Warunki *pre* przejść $T3$ i $T4$ różnią się jedynie testem atrybutu *CB5.lastInSeries*. W żadnym rozkładzie pierwszy lot danego dnia nie jest jednocześnie jego ostatnim, więc w omawianej sytuacji odpali się przejście $T3$. Samolot i lot są ponownie łączone w parę wyrażeniem $CB5.aircraft = CB4$. Samolot zostaje przeniesiony do bufora $CB6$ oznaczającego, że rozpoczęła się obsługa samolotu po wykonanym locie. Ponadto następuje aktualizacja referencji *current*, która teraz wskazuje na lotnisko końcowe połączenia. Ponieważ istnieją następne loty do realizacji z aktualnej serii, należy pozostawić informację, który lot został wykonany ostatnio, co uzasadnia łuk powrotny z $T3$ do $CB5$. Najdłuższą część notatki *post* wyjaśniono poniżej.

Wszystkie lotniska docelowe przewoźnika podzielono na średnie i duże. Na lotniskach średnich, na przykład w porcie Wrocław Strachowice (WRO), samolot dostaje pozwolenie na lądowanie praktycznie od razu, gdy jest do tego gotowy, nawet gdy lot ma spore opóźnienie. Na dużych lotniskach jest inaczej. Jeśli samolot nie zdąży na planowaną chwilę lądowania, to musi oczekiwać w kolejce innych samolotów na zgodę na lądowanie, co zazwyczaj powiększa aktualne opóźnienie. W zależności od kombinacji ⟨ typ lotniska startowego, typ lotniska docelowego ⟩ losowana jest zgodnie z dalej omówionymi 4 gęstościami liczba minut dodawana (nawet gdy wylosowana wartość jest ujemna) do czasu obsługi *serviceTime*.

Wszystkie gęstości propagacji opóźnienia bazują na dwuargumentowych rozkładach normalnych, gdzie pierwszy argument, to wartość aktualnego opóźnienia $y1$, drugi argument $y2$ to

wartość o jaką opóźnienie się powiększa. Taki rozkład normalny ma postać:

$$f(y1, y2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} exp\{g(y1, y2)\}$$

gdzie:

$$g(y1, y2) = \frac{-1}{2(1-\rho^2)}((\frac{y_1 - E(y_1)}{\sigma_1})^2 - 2\sigma\frac{(y_1 - E(y_1))(y_2 - E(y_2))}{\sigma_1\sigma_2} + (\frac{y_2 + E(y_2)}{\sigma_2})^2)$$

Gęstość propagacji opóźnienia jest sumą ważoną kilku takich rozkładów. Jeśli T identyfikuje możliwą kombinację portów: $D-S$, $S-S$, $S-D$ lub $D-D$, a k_T liczbę sumowanych członów, to ostateczna gęstość jest równa:

$$dp^{(T)}(y1, y2) = \sum_{i=1}^{i=k_T} p_i^{(T)} f_i^{(T)}(y1, y2)$$

Parametry poszczególnych sumowanych członów zebrano w tab. 10.1. Dla rozkładów $D-S$ oraz $S-S$ $k_T = 3$, podczas gdy dla $S-D$ oraz $D-D$ $k_T = 2$. Wartość $y1$ w symulacji jest znana,

Tabela 10.1: Parametry rozkładów propagacji opóźnienia

	D-S	S-S	S-D	D-D
		Człon 1		
$E(y1)$	-15,04	-15,64	-3,46	6,75
$E(y2)$	0,24	-4,47	-1,79	10,49
$\sigma 1$	4,42	4,29	5,59	5,68
$\sigma 2$	8,60	4,77	3,56	3,85
ρ	0,25	-0,04	0,43	0,59
p_1	0,20	0,14	0,62	0,69
		Człon 2		
$E(y1)$	-2,13	0,64	15,48	17,65
$E(y2)$	3,64	1,35	11,28	20,79
$\sigma 1$	5,47	5,12	6,16	7,86
$\sigma 2$	7,97	8,30	5,68	6,26
ρ	0,36	0,05	0,87	0,81
p_2	0,50	0,52	0,38	0,31
		Człon 3		
$E(y1)$	33,97	35,38		
$E(y2)$	38,85	27,10		
$\sigma 1$	14,58	16,34		
$\sigma 2$	12,99	18,03		
ρ	0,90	0,34		
p_3	0,30	0,34		

toteż następuje redukcja rozkładu do jednoargumentowego i wyznaczenie funkcji kwantylu nowo powstałego rozkładu normalnego.

To przekształcenie zachodzi w $T3$, gdzie $serviceTime$ jest wyznaczany jako suma odstępu między lotami oraz propagacji opóźnienia.

Po wygenerowaniu zmiennej losowej czasu obsługi samolotu, akcja $A4$ wykorzystuje ją do odmierzenia faktycznego czasu serwisowania. W efekcie samolot jest gotowy do podjęcia kolejnego lotu serii, co symbolizuje obecność jego obiektu w $CB8$. Akcja A10 odpowiada za wywołanie diagramu opisującego uszkodzenia, który zostanie opisany w 10.4.

Start samolotu po przerwie między lotami jest reprezentowany przez $T2$, którego opis, bufory wejściowe i wyjściowe są analogiczne do $T1$. Różnicą jest jedynie wyselekcjonowanie z $CB2$ lotu następnego względem przechowywanego w $CB5$ wyrażeniem $CB2.prev = CB5$, który to ostatecznie zostanie usunięty z $CB5$. Gdy wykonywany lot będzie ostatnim danego dnia, to zamiast $T4$ wykona się $T3$ umieszczając samolot z powrotem na torze *Night break* oraz usuwając lot z $CB5$. Przylegające do $CB7$ akcje zostaną omówione w dalszych podrozdziałach.

10.4 Detekcja i obsługa uszkodzeń

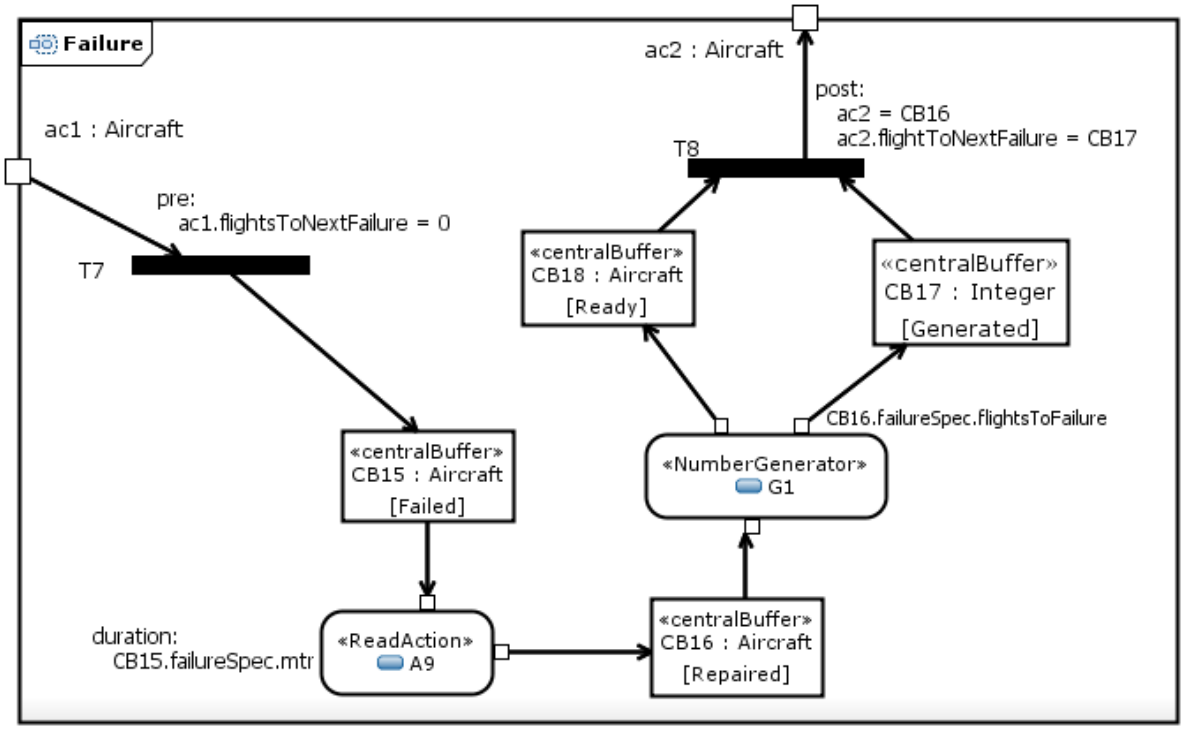
Intensywność eksploatacji samolotu bardzo silnie wpływa na częstotliwość występowania uszkodzeń. Z tego powodu jednostką wyrażającą występowanie awarii w omawianym modelu jest liczba lotów. Z analizy danych rzeczywistych wynika, że liczba lotów do uszkodzenia jest zmienną losową o rozkładzie normalnym, średniej 61 i wariancji 4. Ponieważ Wizzair wykorzystuje tylko jeden typ samolotu, ta zmienna losowa jest wartością atrybutu $flightsToFailure$ wszystkich obiektów $FailureSpec$.

Z kolei licznikiem lotów pozostałych do kolejnego uszkodzenia jest atrybut *flightsToNextFailure* w obiektach klasy $Aircraft$. Konsekwentnie po każdym zrealizowanym locie $T3$ lub $T4$ zmniejsza wartość tego atrybutu, aż wykonana zostanie czynność reprezentowana przez $A10$ lub $A12$, w biegu której licznik zostaje odświeżony (omówiono dalej).

Przy rozpatrywaniu rzeczywistych awarii uwzględniono wszystkie uszkodzenia: zarówno najprostsze, których czas naprawy jest bardzo krótki, jak i te wyłączające samolot na dłuższy okres czasu i tym samym wprowadzające poważne zaburzenia do systemu. Z tego wynika, że sama identyfikacja chwili uszkodzenia jest w modelu niewystarczająca, gdyż niezbędny jest również sposób rozpoznawania typu uszkodzenia. Wszystkie uszkodzenia zidentyfikowane w danych rzeczywistych podzielono na 7 grup różniących się częstotliwością występowania, jak i czasem powrotu do zdatności. Gęstość czasu naprawy w minutach jest ostatecznie równa:

$$f_R(t) = 0,245N(t;51;13) + 0,138N(t;348;19) + 0,118N(t;204;25)$$
$$+0,134N(t;87;16) + 0,193N(t;148;6) + 0,148N(t;69;9)+$$
$$0,024N(t;878;29)$$

Choć w rzeczywistości uszkodzenie może zostać wykryte w trakcie lotu, to czas odnowy w danych rzeczywistych liczony jest zawsze względem rozpoczęcia naprawy, czyli po wykonaniu wszystkich lotów danego dnia lub po obsłudze samolotu pomiędzy lotami. Na głównym diagramie READ odpowiada to akcjom $A10$ i $A12$ wywołującym omówiony poniżej model, gdy statek znajdzie się w stanach odpowiednio $Awaiting$ lub $Idle$. Jak wynika z opisu części *pre*



Rysunek 10.3: Diagram uszkodzeń i napraw [81]

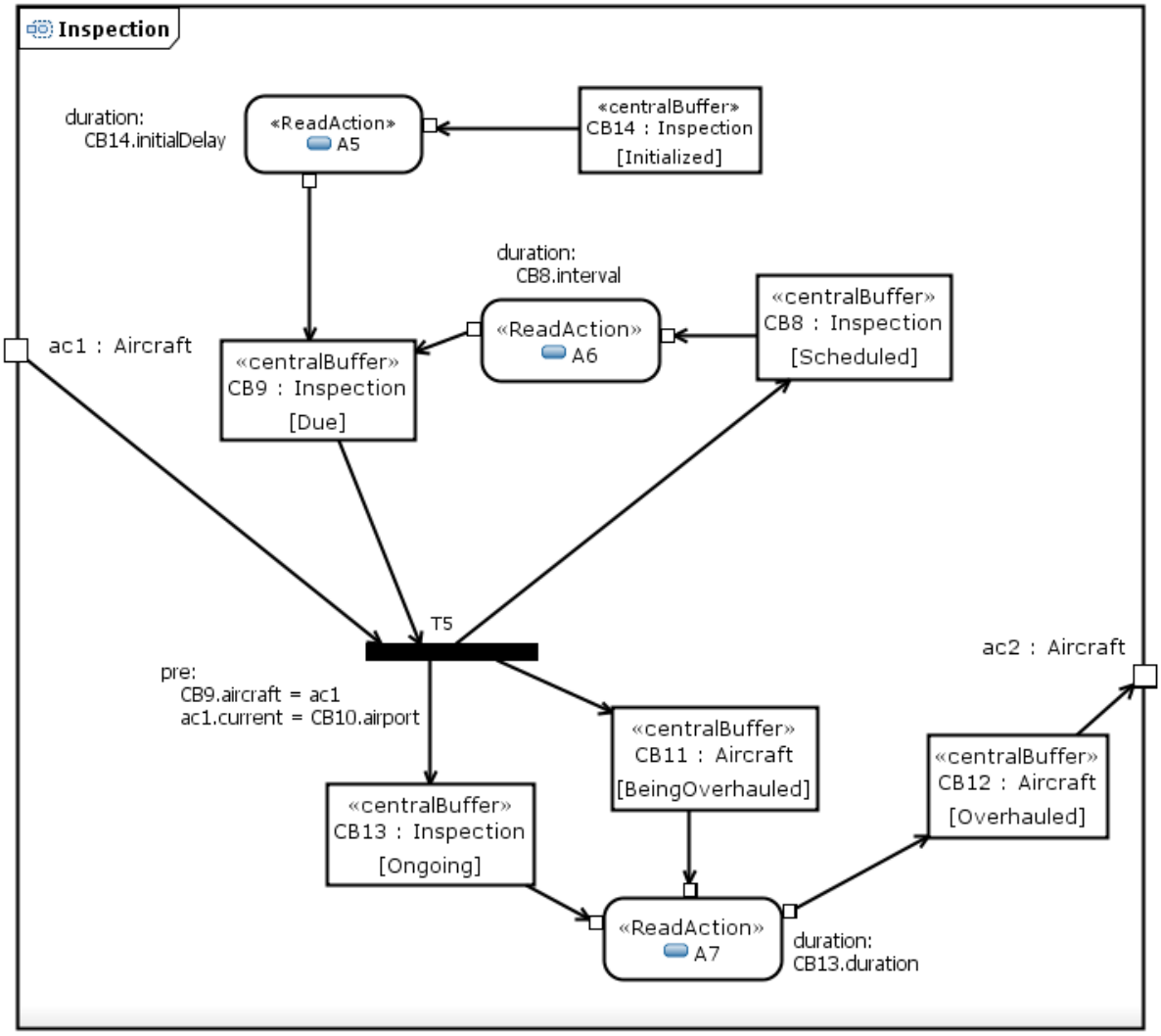
przejścia $T7$, warunkiem wykonania czynności opisanej na rys. 10.3 jest osiągnięcie przez licznik $flightsToNextFailure$ wartości zero. W takiej sytuacji obiekt z $CB7$ lub $CB8$ zostanie przeniesiony do $CB15$, czyli będzie w stanie $Failed$. Kolejnym etapem procesu jest przeprowadzenie naprawy, czemu odpowiada akcja $A9$ realizująca zmienną losową czasu naprawy oraz odmierzającą wylosowany czas według wzoru 10.1. Po naprawie obiekt reprezentujący samolot zostaje umieszczony w buforze $CB16$, a zadaniem pozostałych elementów jest zaplanowanie kolejnej naprawy. Generator $G1$ losuje liczbę lotów do kolejnego uszkodzenia i umieszcza tę wartość w buforze $CB17$. Ostatecznie przejście $T8$ łączy obiekt samolotu z wylosowaną liczbą i zasila $CB7$ lub $CB8$ na diagramie głównym.

10.5 Model inspekcji dziennych

Wszyscy przewoźnicy są zobowiązani regulacjami do przeprowadzania zarówno reaktywnych jak i proaktywnych działań mających na celu diagnostykę i konserwację posiadanych samolotów.

Podczas gdy opisane w poprzednim podrozdziale działania są akcjami głównie reaktywnymi, inspekcje dzienne to czynności proaktywne. Ponieważ ten typ przeglądów wykonywany jest często, pożądane jest jego przeprowadzenie w porze nocnej, tak aby w przypadku braku usterek nie opóźniać lotów kolejnego dnia. Stąd miejscem wywołania diagramu inspekcji przedstawionego na rys. 10.4 jest tor inspekcji nocnej głównego diagramu, a konkretnie akcja $A11$.

Plan inspekcji samolotów jest bardzo istotny dla sprawnego funkcjonowania całego syste-



Rysunek 10.4: Diagram inspekcji

mu. Ekipy naprawcze są w stanie zdiagnozować określoną liczbę samolotów w ciągu jednej nocy, a ponadto inspekcja statku wyklucza jego wykorzystanie do przejęcia opóźnionych lotów. Aby zrealizować plan inspekcji wprowadzono atrybut *initialDelay* oznaczający po jakim czasie względem początku symulacji zachodzi pierwsza inspekcja, czyli przeniesienie obiektu z $CB14$ do $CB9$. Kolejne inspekcje dzienne wykonywane są planowo, czyli co dwa dni.

Elementy $CB8$, $A6$, $CB9$ i $T5$ tworzą cykl, którego częstotliwość pokrywa się z interwałem przeprowadzania inspekcji dziennej jednego samolotu. Rolą cyklu jest generacja kolejnego obiektu w $CB9$ co dwa dni. Gdy odpali się przejście $T5$, czyli gdy rozpocznie się inspekcja, to od razu do bufora $CB8$ zostanie dodany nowy obiekt reprezentujący kolejny przegląd. Następnie akcja $A6$ odmierza interwał, aby w końcu przenieść obiekt inspekcji z $CB8$ do $CB9$.

Z danych rzeczywistych wyznaczono gęstość czasu trwania inspekcji przechowywaną w atrybucie *duration* obiektu *Inspection*. Przyjmując, że $NB(t; x; y; a; b)$ to gęstość rozkładu normalnego o średniej x, odchyleniu standardowym y ograniczonym do przedziału $\langle a, b\rangle$, to gęstość

czasu inspekcji jest równa:

$$f_I(t) = 0,33NB(t;34;14;15;60) + 0,58NB(t;58;15;35;100) + 0,09NB(t;88;13;50;150) \tag{10.1}$$

Każda iteracja cyklu, a konkretnie odpalenie $T5$, uruchamia inspekcję samolotu, którego dotyczy. Ponownie wykorzystano akcję READ, aby po czasie odpowiadającym przeglądowi przenieść obiekt $Aircraft$ ze stanu $BeingOverhauled$ z powrotem do diagramu wyższego poziomu.

10.6 Algorytm zastępowania lotów

Uruchomienie czynności przejmowania opóźnionych lotów przez inny samolot jest uzależnione predykatem $A5\_pre$ na rysunku głównym. Typowo jest to czas minimalnego opóźnienia, po którym rozważa się zastąpienia. Aby wywołać czynność zastępowania przedstawioną na rys. 10.5 dodatkowo musi istnieć na lotnisku początkowym opóźnionego lotu samolot, który w danym dniu wykonał już wszystkie swoje loty, czyli odpowiadający samolotowi obiekt $Aircraft$ znajduje się w buforze $CB7$.

Założeniem algorytmu jest, że po przejęciu opóźnionych lotów wykonujący je samolot skończy ich realizację na lotnisku, na którym powinien stacjonować w nocy, co pozwala uniknąć dodatkowego pustego lotu niezbędnego do podjęcia porannych rejsów.

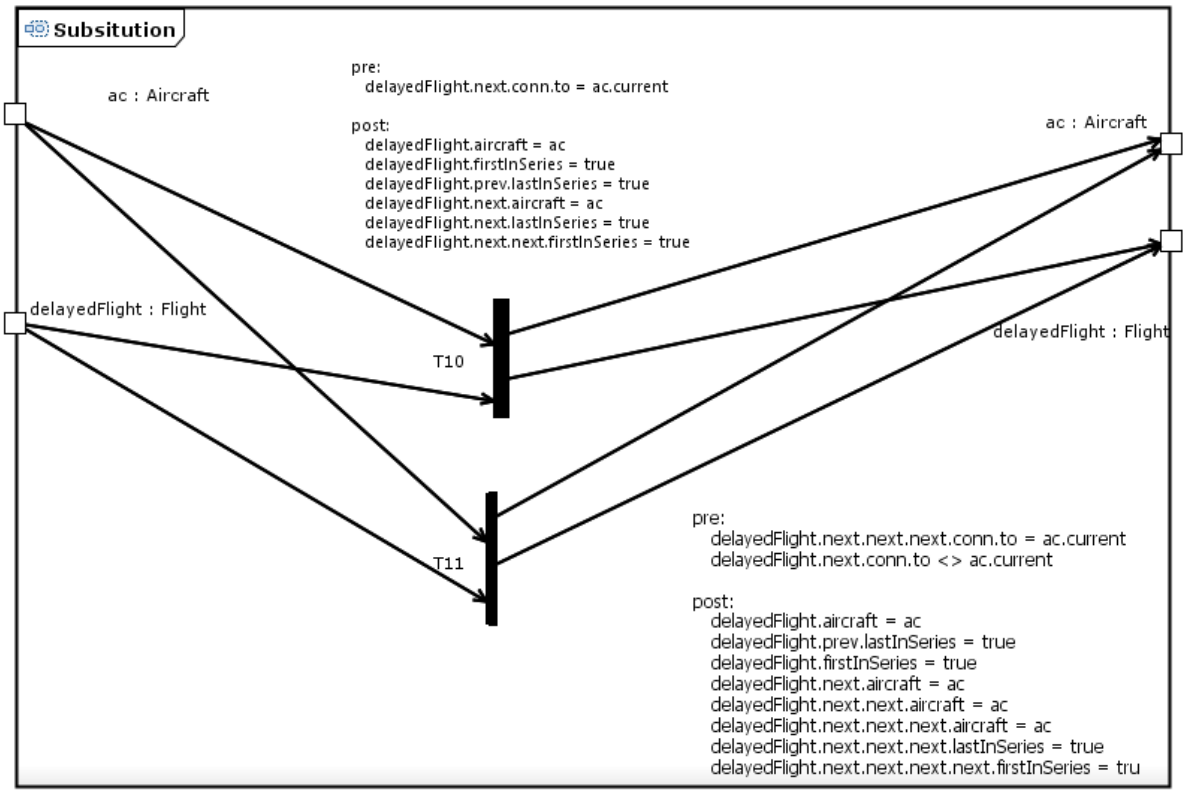
Przejście $T10$ zostanie odpalone, gdy kolejny po reprezentowanym przez $delayedFlight$ locie kończy się na wymaganym lotnisku. W takiej sytuacji przejęta zostanie para lotów: $delayedFlight$ oraz z nim stowarzyszony.

Gdy kolejno wizytowanymi lotniskami są P1, P2, P3 oraz ponownie P1, czyli pomiędzy jedną stowarzyszoną parę lotów wsunięto drugą, to wykona się przejście $T11$. W efekcie wszystkie cztery loty zostaną przejęte przez samolot reprezentowany przez ac.

Zarówno $T10$ jak i $T11$ tworzą nową serię dla powiązanych 2 lub 4 lotów. Wymaga to wyodrębnienia nowej listy serii z istniejącej, czyli, poza ustawieniem atrybutów $firstInSeries$ na $delayedFlight$ oraz $lastInSeries$ na drugim (czwartym - $T11$) po nim, zakończenia wcześniejszej serii oraz ustanowienia nowej na trzecim (piątym - $T11$) po $delayedFlight$.

W rezultacie wykonania akcji $A5$ zmodyfikowany lot zostanie ponownie umieszczony w buforze $CB2$, jednak teraz asygnowany samolot będzie dostępny. Ponieważ modyfikowany lot oraz jeden następny (lub trzy - $T11$) tworzą serię, to wykona się przejście $T1$ i samolot podejmie 2 lub 4 opóźnione loty zgodnie ze schematem opisanym w podrozdziale 10.3.

Podsumowując, w rozdziale wykazano, że język READ ma moc ekspresji wystarczającą do precyzyjnego wyrażenia procesów eksploatacji samolotów niskokosztowych linii lotniczych takich jak wykonanie lotu, inspekcja, naprawa czy algorytm przejęcia lotów. Co niemniej istotne, opracowany model jest hierarchiczny i swoją strukturą odzwierciedla naturalny podział procesów konserwacji, co czyni model czytelnym i intuicyjnym. W głównym modelu READ wykorzystano również tory aktywności dzięki czemu poddiagramy mają swój kontekst wykonania ułatwiający analizę całego modelu.



Rysunek 10.5: Algorytm zastępowania lotów

Rozdział 11

Podsumowanie i osiągnięcia

W dysertacji podjęto problem modelowania niezawodności systemów z zależnościami czasowymi. Praca wpisuje się tym samym w istotny obecnie nurt badań nad metodami analizy systemów czasu rzeczywistego powszechnie towarzyszących działalności człowieka [187, 188] i obejmuje następujące aspekty dyscypliny niezawodności:

1. projektowanie składni i semantyki nowych języków modelowania systemów z zależnościami czasowymi wyrażonymi stochastycznie, w tym algorytmów ich transformacji oraz narzędzi wspomagających modelowanie w myśl paradygmatu MDA,

2. metody analizy niezawodności systemów opisanych proponowanymi językami obejmujące zarówno symulacje Monte Carlo, jak i rozwiązania oraz estymacje analityczne,

3. analiza i modelowanie problemów w dziedzinach informatyki, elektroenergetyki oraz transportu oraz przeprowadzenie obliczeń numerycznych i symulacyjnych dla empirycznej weryfikacji proponowanych metod analizy niezawodności.

11.1 Rozszerzenia drzew niezdatności

Definicja języków modelowania przebiegała w trzech etapach przedyskutowanych w rozdziałach 4, 6 oraz 8. W pierwszym zaprezentowano język PDNZC, którego istotną możliwością jest probabilistyczny opis czasu przy pomocy bram przyczynowych. Następnie, w ramach prac nad GN, do języka wprowadzono przejścia wzorowane na konstrukcji znanej z sieci Petriego oraz dopuszczono parametryzację zdarzeń. Ostatecznie dokonano integracji dotychczasowych rozszerzeń z diagramami aktywności UML, co zwiększyło moc ekspresji metody o możliwość precyzyjnego opisu złożonych procesów obsługi systemów z zależnościami czasowymi. Semantykę finalnej postaci języka nazwanego READ rozstrzygnięto na gruncie sieci Petriego rozbudowując transformację opisaną w [165]. Modelowanie proponowanymi metodami jest wspierane przez nowe narzędzia autorskie oraz rozszerzenia uznanego środowiska IBM Rational Software Architect.

W pracy [84] stwierdzono, że aby metoda analizy niezawodności była postrzegana jako intuicyjna przez inżynierów określonego systemu, to powinna przede wszystkim umożliwiać budowę modeli przypominających swoją strukturą rozpatrywany system. Proponowana w dysertacji metoda READ idzie dużo dalej niż drzewa niezdatności, w których zdarzenia podstawowe symbolizują uszkodzenia komponentów systemu. To diagramy klas i obiektów zapożyczone z języka UML służą do kompleksowego opisu struktury systemu, a każdy obiekt (komponent) może mieć przypisany jeden z wielu stanów niezawodnościowych swojej klasy. Asocjacje i wiązania wykorzystuje się do pokazania związków strukturalnych pomiędzy komponentami. Tak zdefiniowana struktura systemu została wykorzystana we wszystkich diagramach w rozdziale 10 przyczyniając się do zwiększenia intuicyjności budowanych modeli NLL.

Istotną nowością READ w stosunku do HLPN jest możliwość dokładnego opisu samego komponentu systemu bez odwoływania się do iloczynu kartezjańskiego i rachunku zbiorów. W ten sposób uniknięto konieczności tworzenia zbiorów kolorów prowadzącej, co zilustrowano w dodatku C, do odwoływania się do elementów składowych produktów przez nieistotne z punktu widzenia niezawodności indeksy. W READ opis parametrów komponentu zawiera się zbiorze atrybutów jego klasy, z których każdy ma swoją unikatową nazwę, typ i domyślną wartość, co ostatecznie upraszcza modelowanie bogatych w parametry systemów technicznych.

Na przedstawionych w rozdziałach 8 oraz 10 modelach wyraźnie widać, że READ zyskują dużo na czytelności dzięki tzw. „rzeczywistej współbieżności" (ang. true concurrency), w której wszystkie modelowane procesy faktycznie wykonują się równolegle. Jest to cecha, której nie posiadają np. modele Markowa imitujące współbieżność na drodze sekwencyjnego wykonywania kolejnych kroków różnych procesów (semantyka przeplotowa).

11.2 Nowe metody analizy niezawodności

Pierwszą zaproponowaną w rozdziale 6 oraz dodatku A metodą analizy modeli niezawodnościowych jest symulacja Monte Carlo realizowana na drodze automatycznej transformacji modelu w kod wykonywalny symulatora. Składnia abstrakcyjna proponowanego języka zapisana w notacji Ecore stała się podstawą dla algorytmów translacji opisu systemu w zbiór reguł zachowania bram uogólniających, przyczynowych oraz przejść. Projekt systemu regułowego stanowiącego symulator dopełnia definicja pamięci roboczej zawierającej fakty danej chwili, którymi są zajścia zdarzeń prostych i parametrycznych. Dodanie lub usunięcie faktu z pamięci roboczej powoduje aktywację reguły bram i przejść, a te w rezultacie swojego wykonania usuwają istniejące lub dodają nowe fakty aktywując sąsiadujące bramy lub przejścia. Transformację zaimplementowano w języku Xpand, podczas gdy faktyczną platformą przetwarzającą reguły stał się JBoss Drools.

W ramach drugiego aspektu dysertacji znaleziono również metody rozwiązań oraz estymacji niezawodności dla systemów transportowych z rezerwą czasową, a więc takich, w których czas obsługi uszkodzonego elementu jest ograniczony określoną wartością. Bazując na teorii kolejek, statystykach pozycyjnych, rozkładzie hipowykładniczym oraz zmiennych warunkowych w rozdziale 9 zdefiniowano 3 modele obliczeniowe, które porównano z wynikami symulacyjnymi w

kategoriach jakości rozwiązań i czasu obliczeń. Tym samym wykazano, że gdy zmienne losowe awarii i dostawy mają rozkłady wykładnicze, a wymiana jest opisana rozkładem Weibulla, czyli M/W/M w wykorzystywanej notacji, to uzyskane wartości prawdopodobieństwa oraz średniego czasu hazardu są dokładne. Z kolei dla wariantu W/W/M, czyli gdy czas pomiędzy awariami jest relaksowany do postaci Weibulla o parametrze kształtu w przedziale wskazanym przez Ekspertów, metody dostarczają oszacowań o błędzie poniżej 12%. Następnie dzięki przejściu z rozkładu hipowykładniczego na statystyki pozycyjne zaproponowano drugą metodę estymacji wymagającą istotnie mniejszego wysiłku obliczeniowego. Dla najtrudniejszego problemu W/W/W opracowano trzeci schemat estymacji dostarczający wyniki wyraźnie dokładniejsze od dotychczas opracowanych metod.

11.3 Weryfikacja metody w wybranych klasach systemów

Pierwszym pełnym zastosowaniem PDNZC stała się metoda koordynacji zabezpieczeń odległościowych linii elektroenergetycznej. Niedokładna konfiguracja czasów opóźnień zabezpieczeń może prowadzić do hazardu, czyli wyłączenia większego niż niezbędny obszaru dystrybucji mocy. Przy pomocy PDNZC wspartych opisanym w rozdziale 5 językiem dziedzinowym obejmującym obiektową składnię abstrakcyjną oraz tekstową składnię konkretną zautomatyzowano proces obliczeń prawdopodobieństwa hazardu dla zdefiniowanej przez eksperta konfiguracji zabezpieczeń zdalnych i lokalnych linii elektroenergetycznej. Istotą rozwiązania jest translator opisu zabezpieczeń odległościowych linii w drzewa niezdatności zaimplementowany w języku transformacji międzymodelowych QVT i wspierającej drzewa z zależnościami czasowymi wyrażonymi zarówno stochastycznie (PDNZC), jak i w postaci przedziałów (NDNZC). W ten sposób analizę wygenerowanych drzew można przeprowadzić omówioną w rozprawie metodą symulacyjną lub analizą przedziałów [115]. Co więcej, dzięki wsparciu dla dwu typów zabezpieczeń: jednosystemowych z członami startowymi lub wielosystemowych bez elementów rozruchowych prezentowane rozwiązanie pozwala na ocenę poprawy niezawodności po modernizacji zabezpieczeń linii elektroenergetycznej, a także porównanie prawdopodobieństw hazardu, gdy zabezpieczenia umieszczono na początkach lub końcach sekcji.

Pomimo że PDNZC posiadają moc ekspresji wyraźnie wyższą od klasycznych drzew niezdatności, to dopiero GN pozwoliły na zamodelowanie wszystkich wybranych polityk napraw. Kluczowym okazało się włączenie przejścia sieci Petriego w rozdziale 6, które umożliwiło atomową alokację i pozwoliło na opis wyścigu pomiędzy wieloma uszkodzonymi komponentami. Dodatkowo, parametryzacja drzewa efektywnie zaadresowała powtarzalną i redundantną strukturę modeli przyczyniając się do jej uproszczenia. Ostatecznie spośród wielu opracowanych modeli polityk napraw zdecydowano się na umówienie polityk: globalnej, równoległej, szeregowej oraz opartej o zasoby, co umożliwiło porównanie z innym rozszerzeniem proponowanym w [146].

Dla pierwszego z dwu omawianych systemów transportowych - systemu tramwajowego MPK we Wrocławiu, rozpatrzono proces logistyczny oraz powiązane dane historyczne, w wyniku czego zbudowano dwa modele: sieć Petriego oraz graf niezdatności tego samego problemu. W ten

sposób na rzeczywistym przykładzie w sekcji 7.5.2 wykazano, że GN są bardziej intuicyjne niż HLPN, choć te pierwsze wciąż wymagają usprawnień. W obydwu modelach uwzględniono rezultaty analizy statystycznej danych dostarczonych przez przewoźnika i dotyczących uszkodzeń w 2009 roku. Uzyskane wyniki hazardu okazały się odległe o maksymalnie 15% od wartości wyznaczonych z danych rzeczywistych w tym samym okresie, a przyczyny rozbieżności zostały wyjaśnione przy pomocy zgodności testów λ Kołmogorowa i wskazaniu uproszczeń modelu.

Drugim podjętym w rozdziale 10 problemem transportowym było dokładne modelowanie systemu niskokosztowych linii lotniczych wymagającego od języka wysokiej mocy ekspresji. To dzięki rozszerzeniom diagramów aktywności udało się sprostać wymaganiom i formalnie zilustrować poszczególne części eksploatacji samolotów takie jak: lot i przerwy pomiędzy nimi, inspekcje, naprawy czy algorytm zastępowania lotów. Również dzięki READ możliwy był hierarchiczny opis procesu oraz parametrów czasowych określonych jego etapów. Na podstawie ponad 32000 zrealizowanych lotów w sezonie letnim 2009 roku zdefiniowano zmienne losowe liczby lotów do uszkodzenia, czasu trwania naprawy, przyspieszenia czy propagacji opóźnienia zależnej od typu lotniska. Z opinii Ekspertów jednoznacznie wynikało, że to przede wszystkim tory aktywności, opis struktury niezawodnościowej systemu przy pomocy diagramu klasy oraz hierarchiczna budowa modeli ułatwiają analizę i rozbudowę modelu READ.

Empiryczna weryfikacja metody symulacyjnej składała się z kilku etapów. W pierwszym kroku w rozdziale 4 porównano uzyskane przy pomocy PDNZC dostępności systemu komputerowego objętego politykami napraw z rezultatami dostępnymi w literaturze i wynikającymi z wykonania sieci SWN [146]. Intensywnym symulacjom poddano 3 modele w 4 wariantach, a różnice w wynikach poniżej 0.2% zwiększają ufność, że modele PDNZC i SWN są semantycznie tożsame. Drugim etapem weryfikacji było porównanie prawdopodobieństw blokowania modelu GN z wartościami zwracanymi przez formułę Erlang C w rozdziale 6. Przed obliczeniami analitycznymi aproksymowano rozkład Weibulla zmienną wykładniczą, a wynik porównywano z symulacją bazującą na rozkładzie oryginalnym. Tym samym potwierdzono zbieżność rozwiązań, gdy parametr kształtu rozkładu Weibulla był równy jedności i obserwowano rosnące błędy wraz z oddalaniem się od tej wartości.

Już przy pierwszych eksperymentach obliczeniowych z siecią HLPN systemu tramwajowego [98] zauważono, że ze względu na komplikację procesu logistycznego znalezienie jednoznacznych wartości parametrów zmiennych losowych w tym systemie jest kłopotliwe. Z tego powodu badania przeprowadzono dla 8 przypadków będących kombinacjami dwu poziomów redundancji, dwu przyjętych momentów zgłoszenia awarii oraz dwu rezerw czasowych. Jednocześnie, w celu głębszego zbadania systemu tramwajowego, symulacje modelu GN w rozdziale 7 uruchamiano zarówno dla sparametryzowanych zmiennych losowych, jak i ich empirycznych dystrybuant wynikających wprost z danych rzeczywistych. Pomimo uzyskania zadowalającej zgodności wyników symulacyjnych z odpowiadającymi metrykami wyznaczonymi z danych rzeczywistych, w kolejnym eksperymencie zdecydowano się na odmienne uchwycenie zmienności parametrów.

W trakcie konsultacji z Ekspertami dziedzinowymi zaproponowano alternatywną metodę badań dla 8 wymienionych w poprzednim paragrafie przypadków polegająca na niedeterminis-

stycznym zdefiniowaniu parametrów zmiennych A, E oraz D systemu. Taką metodykę przyjęto w rozdziale 9, w którym poszukiwano rozwiązań i estymacji dla uogólnionego systemu tramwajowego. Zapewnienie wysokiej wiarygodności wyników badań, gdy wejściowych zmiennych losowych nie zdefiniowano jednoznacznie, a podano jedynie przedziały ich parametrów, wymagało rozpatrzenia dużej liczby przypadków, a co za tym idzie konieczna była automatyzacja procesu złożonego z kilku równoległych obliczeń. Ponadto poprawny symulator był wciąż niezbędny, ponieważ ocena jakości estymacji była wyznaczana na podstawie odległości od wyniku uznanego za dokładny, którym było właśnie rozwiązanie symulacyjne. Choć dla określonej instancji problemu o parametrach zmiennych z uzgodnionych przedziałów potrzebne jest 5 wyników do wyznaczenia błędów (symulacyjny oraz z 4 oszacowań), to każda wartość była uzyskiwana dwoma implementowanymi w izolacji kalkulatorami. Dwa użyte symulatory to: wygenerowany przez transformację GN oraz dedykowany przetwarzaniu systemów z rezerwą czasową. Z drugiej strony, wartości estymacji każdorazowo dostarczały implementacje: numeryczna oraz symulacyjna dla tego schematu aproksymacyjnego. Po wykonaniu obliczeń dla określonego przypadku testowego niezależny komponent programowy automatycznie weryfikował zgodność odpowiadających sobie kalkulatorów, a ponadto sprawdzano zbieżność rozwiązania opartego o rozkład hipowykładniczy z metodą wykorzystującą statystyki pozycyjne. Redundancja w obliczeniach okazała się niezbędna dla wysunięcia opisanych wcześniej wniosków o jakości metod estymujących.

Konfrontując postawioną w sekcji 3.2 tezę z powyższą analizą dysertacji z trzech przeplatających się perspektyw można potwierdzić, że istnieje język opisu niezawodności bardziej intuicyjny niż sieci Petriego i o wyższej mocy ekspresji niż drzewa niezdatności, i którego metody analizy pozwalają na częściową automatyzację analizy niezawodności systemów w dziedzinach transportu i elektroenergetyki. Poszukiwanym językiem modelowania jest READ, a dla opisanych symulacyjnych i analitycznych metod jego analizy znaleziono rzeczywiste zastosowania w wymienionych dziedzinach.

Bibliografia

[1] L. T. W. Agner, I. W. Soares, P. C. Stadzisz, J. M. Simao. A Brazilian survey on UML and model-driven practices for embedded software development. *Journal of Systems and Software*, przyjęto do druku.

[2] C. André, F. Mallet, R. Simone. Modeling time(s). W: *Model Driven Engineering Languages and Systems*, Tom 4735 *Lecture Notes in Computer Science*, s. 559–573. Springer Berlin Heidelberg, 2007.

[3] E. Andrade, P. Maciel, G. Callou, B. Nogueira. A methodology for mapping SysML activity diagram to time Petri net for requirement validation of embedded real-time systems with energy constraints. W: *Proceedings of the 2009 Third International Conference on Digital Society*, ICDS '09, s. 266–271, Washington, DC, USA, 2009. IEEE Computer Society.

[4] L. Apvrille, J.-P. Courtiat, C. Lohr, P. de Saqui-Sannes. TURTLE: a real-time UML profile supported by a formal validation toolkit. *Software Engineering, IEEE Transactions on*, 30(7):473 – 487, 2004.

[5] Arena. Symulator diagramów aktywności. http://www.arenasimulation.com/.

[6] T. Arpinen, E. Salminen, T. D. Hämäläinen, M. Hännikäinen. MARTE profile extension for modeling dynamic power management of embedded systems. *Journal of Systems Architecture*, 58(5):209 – 219, 2012.

[7] T. Babczyński, M. Kowalski, M. Łukowicz, J. Magott, P. Skrobanek. Safety and reliability models of time-dependent systems. W: *Summer Safety and Reliability Seminars, SSARS 2011*, s. 11–22, Gdańsk-Sopot, Polska, 2011.

[8] T. Babczyński, M. Łukowicz, J. Magott. Selection of zone 3 time delay for backup distance protection using probabilistic fault trees with time dependencies. *Przegląd Elektrotechniczny*, 86(9):208 – 215, 2010.

[9] T. Babczyński, M. Łukowicz, J. Magott. Time coordination of distance protections using probabilistic fault trees with time dependencies. *Power Delivery, IEEE Transactions on*, 25(3):1402 –1409, 2010.

[10] M. Bali. *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing, 2009.

[11] P. Baraldi, M. Compare, E. Zio. Maintenance policy performance assessment in presence of imprecision based on Dempster–Shafer Theory of Evidence. *Information Sciences*, przyjęto do druku.

[12] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, W. Yi. Uppaal implementation secrets. W: *Proc. of 7*th *International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[13] K. Benghazi, J. L. Garrido, M. Noguera, M. V. Hurtado, L. Chung. Extending and formalizing UML 2.0 activity diagrams for the specification of time-constrained business processes. W: *Research Challenges in Information Science (RCIS), 2010 Fourth International Conference on*, s. 93 –100, 2010.

[14] S. Bernardi, F. Flammini, S. Marrone, J. Merseguer, C. Papa, V. Vittorini. Model-driven availability evaluation of railway control systems. W: *Proceedings of the 30th international conference on computer safety, reliability, and security*, SAFE COMP'11, s. 15–28, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] A. Blackler, V. Popovic, D. Mahar. Investigating users' intuitive interaction with complex artefacts. *Applied Ergonomics*, 41(1):72 – 92, 2010.

[16] S. Bloch-Mercier. Optimal restarting distribution after repair for a Markov deteriorating system. *Reliability Engineering and System Safety*, 74(2):181 – 191, 2001.

[17] A. Bobbio, G. Franceschinis, R. Gaeta, L. Portinale. Exploiting Petri nets to support fault tree based dependability analysis. W: *Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on*, s. 146 –155, 1999.

[18] A. Bobbio, G. Franceschinis, R. Gaeta, L. Portinale. Parametric fault tree for the dependability analysis of redundant systems and its high-level Petri net semantics. *Software Engineering, IEEE Transactions on*, 29(3):270 – 287, 2003.

[19] A. Bobbio, L. Portinale, M. Minichino, E. Ciancamerla. Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliability Engineering and System Safety*, 71(3):249 – 260, 2001.

[20] H. Boley. An introduction to object-oriented RuleML. W: *Progress in Artificial Intelligence*, Tom 2902 *Lecture Notes in Computer Science*, s. 4–4. Springer Berlin Heidelberg, 2003.

[21] G. Booch. *Object-oriented analysis and design with applications.* Redwood City: Benjamin Cummings-Wesley, 1993.

[22] Borland Together. http://www.borland.com/products/together/.

[23] T. Bouabana-Tebibel S. H. Rubin. An interleaving semantics for UML 2 interactions using Petri nets. *Information Sciences*, przyjęto do druku.

[24] T. Bouabana-Tebibel, S. H. Rubin, M. Bennama. Formal modeling with SysML. W: *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, s. 340–347. IEEE, 2012.

[25] M. Bouissou J.-L. Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering and System Safety*, 82(2):149 – 163, 2003.

[26] K. Buchacker. Modeling with extended fault trees. W: *High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000*, s. 238 –246, 2000.

[27] L. Cao, B. Ramesh, M. Rossi. Are domain-specific models easier to maintain than UML models? *Software, IEEE*, 26(4):19 –21, 2009.

[28] E. Carneiro, P. Maciel, G. Callou, E. Tavares, B. Nogueira. Mapping SysML state machine diagram to time Petri net for analysis and verification of embedded real-time systems with energy constraints. W: *Advances in Electronics and Micro-electronics, 2008. ENICS '08. International Conference on*, s. 1 –6, 2008.

[29] R. Chanda P. Bhattacharjee. A reliability approach to transmission expansion planning using fuzzy fault-tree model. *Electric Power Systems Research*, 45(2):101 – 108, 1998.

[30] P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[31] D. Codetta-Raiteri. Integrating several formalisms in order to increase fault trees' modeling power. *Reliability Engineering and System Safety*, 96(5):534 – 544, 2011.

[32] S. Cook, G. Jones, S. Kent, A. Wills. *Domain-specific development with Visual Studio DSL Tools*. Addison-Wesley Professional, Boston, 2007.

[33] J. W. Cooley J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[34] J. P. Courtiat, C. A. S. Santos, C. Lohr, B. Outtaj. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. *Computer Communications*, 23(12):1104–1123, 2000.

[35] K. Czarnecki S. Helsen. Classification of model transformation approaches. *Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

[36] M. Dal Cin. Extending UML towards a useful OO-language for modeling dependability features. W: *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. Proceedings. Ninth IEEE International Workshop on*, s. 325 – 330, 2003.

[37] P. David, V. Idasiak, F. Kratz. Reliability study of complex physical systems using SysML. *Reliability Engineering and System Safety*, 95(4):431 – 450, 2010.

[38] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W. Sanders, P. Webster. The Mobius framework and its implementation. *Software Engineering, IEEE Transactions on*, 28(10):956 – 969, 2002.

[39] W. Denson. The history of reliability prediction. *Reliability, IEEE Transactions on*, 47(3):321 –328, 1998.

[40] I. Dokas, D. Karras, D. Panagiotakopoulos. Fault tree analysis and fuzzy expert systems: Early warning and emergency response of landfill operations. *Environmental Modelling and Software*, 24(1):8 – 25, 2009.

[41] J. Dugan, S. Bavuso, M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on*, 41(3):363 –377, 1992.

[42] S. Efftinge. *openArchitectureWare: User guide*. www.openarchitectureware.org, 2008.

[43] C. Eichner, H. Fleischhack, U. Schrimpf, C. Stehno. Compositional semantics for UML 2.0 sequence diagrams using Petri nets.

[44] J. Ellsbrerger, D. Hogrefe, A. Sarma. *SDL, Formal object-oriented language for communicating systems.* Prentice-Hall, 1997.

[45] S. Emadi. Mapping annotated sequence diagram to a Petri net notation for reliability evaluation. W: *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, Tom 3, s. V3–57 –V3–61, 2010.

[46] S. Emadi F. Shams. Transformation of usecase and sequence diagrams to Petri nets. W: *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, Tom 4, s. 399 –403, 2009.

[47] R. Eshuis R. Wieringa. A formal semantics for UML activity diagrams - formalising workflow models. Technical report, 2001.

[48] H. Espinoza, D. Cancila, B. Selic, S. Gérard. Challenges in combining SysML and MARTE for model-based design of embedded systems. W: *Model Driven Architecture - Foundations and Applications*, Tom 5562 *Lecture Notes in Computer Science*, s. 98–113. Springer Berlin Heidelberg, 2009.

[49] Q. Geng, H. Duan, S. Li. Dynamic fault tree analysis approach to safety analysis of civil aircraft. W: *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, s. 1443 –1448, 2011.

[50] M. Gribaudo, D. Codetta-Raiteri, G. Franceschinis. Draw-Net, a customizable multi-formalism, multi-solution tool for the quantitative evaluation of systems. W: *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, s. 257 – 258, 2005.

[51] R. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley, 2009.

[52] M. Grossman, J. E. Aronson, R. V. McCarthy. Does UML make the grade? Insights from the software development community. *Information and Software Technology*, 47(6):383 – 397, 2005.

[53] J. Górski, J. Magott, A. Wardziński. Modelling fault trees using petri nets. W: *The 14th International Conference on Computer Safety, Reliability and Security. SAFE COMP '95*, Belgirate (Italy), 1995.

[54] V. C. Gu, Q. Cao, W. Duan. Unified Modeling Language (UML) IT adoption — A holistic model of organizational capabilities perspective. *Decision Support Systems*, 54(1):257 – 269, 2012.

[55] N. Guelfi A. Mammar. A formal semantics of timed activity diagrams and its PROMELA translation. W: *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, APSEC '05, s. 283–290, Washington, DC, USA, 2005. IEEE Computer Society.

[56] K. H. Han, S. K. Yoo, B. Kim. Qualitative and quantitative analysis of workflows based on the UML activity diagram and Petri net. *WSEAS Trans. Info. Sci. and App.*, 6(7):1249–1258, 2009.

[57] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[58] M. Hause. The SysML modelling language. W: *Fifteenth European Systems Engineering Conference*, 2006.

[59] F. Heitmann D. Moldt. Petri Nets Tools Database, 2012. http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html.

[60] A. Heuer, V. Stricker, C. J. Budnik, S. Konrad, K. Lauenroth, K. Pohl. Defining variability in activity diagrams and Petri nets. *Science of Computer Programming*, przyjęto do druku, 2012.

[61] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, Aug. 1978.

[62] S. H. Houmb K. K. Hansen. Towards a UML profile for Security Assessment. W: *UML'2003, Workshop on Critical Systems Development with UML*, 2003.

[63] D. Hästbacka, T. Vepsäläinen, S. Kuikka. Model-driven development of industrial process control applications. *Journal of Systems and Software*, 84(7):1100 – 1113, 2011.

[64] G. Hura J. Atwood. The use of Petri nets to analyze coherent fault trees. *Reliability, IEEE Transactions on*, 37(5):469 –474, dec 1988.

[65] IBM. Rational Software Architect 8.0, Symulator diagramów aktywności. http://www.ibm.com/.

[66] ISO/IEC 15909-1. *High-level Petri nets: Concepts, definitions and graphical notation*, 2004.

[67] Itemis, www.eclipse.org/text/documentation. *Xtext: Reference documentation.*

[68] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, Apr. 2002.

[69] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. *Object-oriented software engineering - a use case driven approach.* Addison-Wesley, 1992.

[70] Y. Jarraya, A. Soeanu, M. Debbabi, F. Hassaine. Automatic verification and performance analysis of time-constrained SysML activity diagrams. W: *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, ECBS '07, s. 515–522, Washington, DC, USA, 2007. IEEE Computer Society.

[71] K. Jensen, L. M. Kristensen, L. Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9:213–254, 2007.

[72] M. Jiang, M. Groble, A. Neczwid, A. Willey. Modeling real-time communication systems: Practices and experiences in Motorola. *Journal of Visual Languages and Computing*, 17(6):584 – 605, 2006.

[73] F. Jouault I. Kurtev. Transforming models with ATL. W: *Satellite Events at the MoDELS 2005 Conference*, Tom 3844 *Lecture Notes in Computer Science*, s. 128–138, Berlin, 2006. Springer Verlag.

[74] A. Jurado, F. D. Gaspari, V. Vilarrasa, D. Bolster, X. Sanchez-Vila, D. Fernandez-Garcia, D. Tartakovsky. Probabilistic analysis of groundwater-related risks at subsurface excavation sites. *Engineering Geology*, 125:35 – 44, 2012.

[75] B. Kalinowski. *Komputerowe narzędzia szacowania parametrów niezawodnościowych oraz doboru strategii konserwacji rozległego systemu technicznego o strukturze hierarchicznej.* Rozprawa doktorska, Politechnika Wrocławska, Wydział Elektroniki, 2006.

[76] W. Keller M. Modarres. A historical overview of probabilistic risk assessment development and its use in the nuclear power industry: a tribute to the late Professor Norman Carl Rasmussen. *Reliability Engineering and System Safety*, 89(3):271 – 285, 2005.

[77] S. Kelly J. Tolvaven. *Domain-Specific Modeling: Enabling full code generation.* Wiley-IEEE Computer Society Press, New Jersey, 2008.

[78] M. Kessentini, A. Bouchoucha, H. Sahraoui, M. Boukadoum. Example-based sequence diagrams to colored Petri nets transformation using heuristic search. W: *Modelling Foundations and Applications*, Tom 6138 *Lecture Notes in Computer Science*, s. 156–172. Springer Berlin Heidelberg, 2010.

[79] F. I. Khan S. Abbasi. Techniques and methodologies for risk analysis in chemical process industries. *Journal of Loss Prevention in the Process Industries*, 11(4):261 – 277, 1998.

[80] A. Kierzkowski. *Model procesu eksploatacji obiektu technicznego z ograniczeniami czasowymi.* Rozprawa doktorska, Politechnika Wrocławska, Instytut Konstrukcji i Eksploatacji Maszyn, 2012.

[81] A. Kierzkowski, M. Kowalski, J. Magott. Procesy przeglądów i napraw samolotów tanich linii lotniczych. W: *Niezawodność procesów i systemów technicznych, Materiały XL Jubileuszowej Zimowej Szkoły Niezawodności*, Szczyrk, 2012. Wydawnictwo Naukowe Instytutu Technologii Eksploatacji.

[82] A. Kierzkowski, M. Kowalski, J. Magott, T. Nowakowski. Maintenance process optimization for low-cost airlines. W: *Probabilistic Safety Assessment and Management Conference PSAM11/ESREL12*, Helsinki, 2012.

[83] H. Kim, W. Wong, V. Debroy, D. Bae. Bridging the Gap between Fault Trees and UML State Machine Diagrams for Safety Analysis. W: *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, s. 196 –205, 2010.

[84] M. C. Kim P. H. Seong. Reliability graph with general gates: an intuitive and practical method for system reliability analysis. *Reliability Engineering and System Safety*, 78(3):239 – 246, 2002.

[85] J. F. Kingman. The first Erlang century–and the next. *Queueing Systems: Theory and Applications*, 63(1-4):3–12, 2009.

[86] A. Knöpfel, B. Gröne, P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems.* Wiley, Chichester, UK, 2006.

[87] M. Kowalski. Generacja dynamicznych drzew niezdatności dla systemów zdefiniowanych w UML. *Praca magisterska, Politechnika Wrocławska, Instytut Informatyki, Automatyki i Robotyki*, 2009.

[88] M. Kowalski. Simulation research of performance attacks against service oriented architectures. W: *Dependability of networks*, s. 69 – 84, Wrocław, 2010. Oficyna Wydawnicza Politechniki Wrocławskiej.

[89] M. Kowalski. Sterowane modelem narzędzie do analizy probabilistycznych drzew niezdatności z zależnościami czasowymi. W: *Inżynieria oprogramowania w procesach integracji systemów informatycznych*, s. 225 – 232, Gdańsk, 2010. Pomorskie Wydawnictwo Naukowo Techniczne.

[90] M. Kowalski. Engineering the Fault Graph platform for design and simulation of maintenance models. *eInformatica*, przyjęto do druku.

[91] M. Kowalski. Syntax and semantics of reliability enhanced activity diagrams. W: *Proceedings of the European Safety and Reliability Conference, ESREL 2013*, przyjęto do druku.

[92] M. Kowalski J. Magott. Język dziedzinowy do koordynacji czasowej zabezpieczeń odległościowych w elektroenergetyce. W: *Metody wytwarzania i zastosowania systemów czasu rzeczywistego*, s. 199 – 208, Gdańsk, 2010. Pomorskie Wydawnictwo Naukowo Techniczne.

[93] M. Kowalski J. Magott. Conjoining fault trees with Petri nets to model repair policies. W: *Dependable Computer Systems*, Tom 97 *Advances in Intelligent and Soft Computing*, s. 131–143. Springer Berlin / Heidelberg, 2011.

[94] M. Kowalski J. Magott. Grafy niezdatności z zależnościami czasowymi. *Problemy Eksploatacji*, (1):117–128, 2011.

[95] M. Kowalski J. Magott. *Towards a UML profile for maintenance process and reliability analysis*, Tom 97 *Advances in Intelligent and Soft Computing*. 2011.

[96] M. Kowalski J. Magott. Enabling quantitative risk assessment of the real world transport system. W: *Advances in Safety, Reliability and Risk Management - Proceedings of the European Safety and Reliability Conference, ESREL 2011*, s. 1299–1306, 2012.

[97] M. Kowalski J. Magott. Time coordination of heterogeneous distance protections using a domain specific language. *eInformatica*, 6(1):7–26, 2012.

[98] M. Kowalski, J. Magott, T. Nowakowski, S. Werbińska-Wojciechowska. Analysis of a transportation system with the use of Petri nets. *Eksploatacja i Niezawodność*, 49(1):48–62, 2011.

[99] M. Kowalski, J. Magott, T. Nowakowski, S. Werbińska-Wojciechowska. Exact and approximation methods for dependability assessment of tram systems with time resource. *European Journal of Operational Research*, praca zgłoszona do opublikowania.

[100] M. Kowalski K. Wilkosz. A domain specific language in dependability analysis. W: *Proceedings of 2009 4th International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2009*, s. 324–331, 2009.

[101] M. Kowalski K. Wilkosz. Składnia konkretna języka dziedzinowego do analizy niezawodności algorytmu weryfikacji topologii sieci elektroenergetycznej. W: *Od modelu do wdrożenia : Kierunki badań i zastosowań inżynierii oprogramowania*, s. 321–331, 2009.

[102] M. Krysiński G. Anders. Application of time-dependent fault tree models for the analysis of project schedule failure conditions. W: *Communications, 2008. ICC '08. IEEE International Conference on*, s. 5532 –5537, 2008.

[103] M. Kwiatkowska, G. Norman, D. Parker. Quantitative analysis with the probabilistic model checker prism. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, May 2006.

[104] C. Lai, M. Xie, K. Poh, Y. Dai, P. Yang. A model for availability analysis of distributed software-/hardware systems. *Information and Software Technology*, 44(6):343–350, 2002.

[105] Y. Lam H. K. T. Ng. A general model for consecutive-k-out-of-n: F repairable system with exponential distribution and (k-1)-step Markov dependence. *European Journal of Operational Research*, 129(3):663 – 682, 2001.

[106] J. Lara H. Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. W: *Fundamental Approaches to Software Engineering*, Tom 2306 *Lecture Notes in Computer Science*, s. 174–188. Springer Berlin Heidelberg, 2002.

[107] C. Leangsukun, H. Song, L. Shen. Reliability modeling using UML. W: *Proceedings of the International conference on Software Engineering Research and Practice*, 2003.

[108] Leuven Center. *Electromagnetic Transient Program EMTP*, 1987.

[109] A. Lewiński M. Sumiła. Object oriented programming as a method for developing software in rail-traffic-control computer systems. *Archiwum Transportu PAN*, 2(22):220–237, 2011.

[110] A. Lewiński M. Sumiła. The safety assurance method of railway control systems using object oriented languages. *Archiwum Transportu PAN*, 2(22):239–257, 2011.

[111] L. Li, L. Ma, N. Wang, Q. Yang. Modeling method of military aircraft support process based on SysML. W: *Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference on*, s. 1247 –1251, 2011.

[112] J. Liu, X. Lv, Y. Yu, H. Na. A modeling approach of system reliability based on statecharts. *Physics Procedia*, 33:341 – 347, 2012.

[113] M. E. S. Loomis, A. V. Shah, J. E. Rumbaugh. An object modelling technique for conceptual design. W: *ECOOP '87 Proceedings of the European Conference on Object-Oriented Programming*, s. 192–202, 1987.

[114] B. Luciano P. Mauro. Petri nets as semantic domain for diagram notations. *Electronic Notes in Theoretical Computer Science*, 127(2):29–44, 2005.

[115] M. Łukowicz, J. Magott, P. Skrobanek. Selection of minimal tripping times for distance protection using fault trees with time dependencies. *Electric Power Systems Research*, 81:1556 – 1571, 2011.

[116] J. Magott. Moc opisowa drzew niezdatności z zależnościami czasowymi. *Problemy Eksploatacji*, 4:33 – 40, 2009.

[117] J. Magott P. Skrobanek. Method of time Petri net analysis for analysis of fault trees with time dependencies. *Computers and Digital Techniques, IEE Proceedings*, 149(6):257 – 271, 2002.

[118] J. Magott P. Skrobanek. Timing analysis of safety properties using fault trees with time dependencies and timed state-charts. *Reliability Engineering and System Safety*, 97(1):14 – 26, 2012.

[119] J. Magott K. Skudlarski. Estimating the mean completion time of pert networks with exponentially distributed durations of activities. *European Journal of Operational Research*, 71(1):70 – 79, 1993.

[120] M. A. Marsan G. Conte. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2:93–122, 1984.

[121] J. Merseguer, J. Campos, S. Bernardi, S. Donatelli. A compositional semantics for UML state machines aimed at performance evaluation. W: *Proceedings of the Sixth International Workshop on Discrete Event Systems (WODES'02)*, s. 295–, Washington, DC, USA, 2002. IEEE Computer Society.

[122] A. Mettas M. Savva. System reliability analysis: the advantages of using analytical methods to analyze non-repairable systems. W: *Reliability and Maintainability Symposium*, s. 80 –85, 2001.

[123] Z. Micskei H. Waeselynck. The many meanings of UML 2 sequence diagrams: a survey. *Software and Systems Modeling*, 10(4):489–514, 2011.

[124] Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. http://www.omg.org/spec/QVT/1.1/, 2011.

[125] S. Montani, L. Portinale, A. Bobbio, D. Codetta-Raiteri. Radyban: A tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. *Reliability Engineering and System Safety*, 93(7):922 – 932, 2008.

[126] E. F. Moore C. E. Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(43):191–208, 1956.

[127] von Neumann, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, 34:43–99, 1956.

[128] P. P. K. Normalizacyjny. *Bezpieczeństwo funkcjonalne elektrycznych/elektronicznych/programowalnych systemów związanych z bezpieczeństwem - Część 6: Wytyczne do stosowania IEC 61508-2 i IEC 61508-3 PN-EN 61508-6.* Polski Komitet Normalizacyjny, 2007.

[129] Object Management Group. SysML 1.0. http://www.omg.org/spec/SysML/1.3/, 2012.

[130] Object Management Group. Unified Modeling Language (UML) 2.4.1, Infrastructure. http://www.omg.org/spec/UML/2.4.1/Infrastructure, 2012.

[131] Object Management Group. Unified Modeling Language (UML) 2.4.1, Superstructure. http://www.omg.org/spec/UML/2.4.1/Superstructure, 2012.

[132] Object Constraint Language. http://www.omg.org/spec/OCL/2.0/, 2006.

[133] MDA User Guide. www.omg.org/mda/, 2003.

[134] OpenFTA. http://www.openfta.com/.

[135] J. Ouaknine S. Schneider. Timed CSP: A Retrospective. *Essays on Algebraic Process Calculi*, ENTCS 162, 2006.

[136] G. Pai J. Dugan. Automatic synthesis of dynamic fault trees from UML system models. W: *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, s. 243 – 254, 2002.

[137] R. Paige J. Ostroff. A comparison of the Business Object Notation and the Unified Modeling Language. W: *Proc. Second International Conference on the Unified Modeling Language*, UML'99. Springer, 1999.

[138] R. Paige, J. Ostroff, P. Brooke. Principles for modeling language design. *Information and Software Technology*, 42(10):665 – 675, 2000.

[139] R. Pais, L. Gomes, J. Barros. From UML state machines to Petri nets: History attribute translation strategies. W: *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, s. 3776 –3781, 2011.

[140] E. Paté-Cornell R. Dillon. Probabilistic risk analysis for the NASA space shuttle: a brief history and current work. *Reliability Engineering and System Safety*, 74(3):345 – 352, 2001.

[141] C. Petri. *Kommunikation mit automaten*. Rozprawa doktorska, Uniwersytet w Bonn, 1966.

[142] N. Pidgeon M. O'Leary. Man-made disasters: why technology and organizations (sometimes) fail. *Safety Science*, 34(1–3):15 – 30, 2000.

[143] L. Priese H. Wimmel. A uniform approach to true-concurrency and interleaving semantics for Petri nets. *Theoretical Computer Science*, 206(12):219 – 256, 1998.

[144] PTC. Windchill FTA. http://www.ptc.com/products/windchill/fta.

[145] I. Quadri, E. Brosse, I. Gray, N. Matragkas, L. Indrusiak, M. Rossi, A. Bagnato, A. Sadovykh. MADES FP7 EU project: Effective high level SysML/MARTE methodology for real-time and embedded avionics systems. W: *Reconfigurable Communication-centric Systems-on-Chip, 2012 7th International Workshop on*, s. 1 –8, 2012.

[146] D. Raiteri, G. Franceschinis, M. Iacono, V. Vittorini. Repairable fault tree for the automatic evaluation of repair policies. W: *Dependable Systems and Networks, 2004 International Conference on*, s. 659 – 668, 2004.

[147] J. Recker. "Modeling with tools is easier, believe me"—The effects of tool functionality on modeling grammar usage beliefs. *Information Systems*, 37(3):213 – 226, 2012.

[148] R. Robidoux, H. Xu, L. Xing, M. Zhou. Automated modeling of Dynamic Reliability Block Diagrams using Colored Petri nets. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40(2):337 –351, 2010.

[149] G. N. Rodrigues, D. S. Rosenblum, S. Uchitel. Reliability prediction in model-driven development. W: *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems*, MoDELS'05, s. 339–354, Berlin, Heidelberg, 2005. Springer-Verlag.

[150] R. W. S. Rodrigues. Formalising UML activity diagrams using finite state processes. W: *In Reggio et al*, s. 92–98, 2000.

[151] C. Rohr, W. Marwan, M. Heiner:. Snoopy - a unifying Petri net framework to investigate biomolecular networks. *Bioinformatics*, 26(7):974–975, 2010.

[152] N. Russell, A. H. M. T. Hofstede, N. Mulyar. Workflow controlflow patterns: A revised view. Technical report, BPM Center Report BPM-06-22, 2006.

[153] N. Russell, A. H. M. ter Hofstede, D. Edmond, W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. W: *Proceedings of the 25th International Conference on Conceptual Modeling*. Springer, 2005.

[154] J. Saleh K. Marais. Highlights from the early (and pre-) history of reliability engineering. *Reliability Engineering and System Safety*, 91(2):249 – 256, 2006.

[155] I. M. Shaluf, F.-R. Ahmadun, A. R. Shariff. Technological disaster factors. *Journal of Loss Prevention in the Process Industries*, 16(6):513 – 521, 2003.

[156] P. Skrobanek. Zastosowanie bramy uogólniającej XOR z opóźnieniem w analizie drzew niezdatności z zależnościami czasowymi. W: *Metody wytwarzania i zastosowania systemów czasu rzeczywistego*, s. 59 –70, 2010.

[157] J. Smith, M. Schwarzblat, J. Baker. Application of a fault tree construction expert system to an actual nuclear generating station system. *Reliability Engineering and System Safety*, 44(3):237 – 242, 1994.

[158] M. d. S. Soares J. Vrancken. A metamodeling approach to transform UML 2.0 sequence diagrams to Petri nets. W: *Proceedings of the IASTED International Conference on Software Engineering*, SE '08, s. 159–164, Anaheim, CA, USA, 2008. ACTA Press.

[159] T. Staines. Intuitive mapping of UML 2 activity diagrams into Fundamental Modeling Concept Petri net Diagrams and Colored Petri Nets. W: *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*, s. 191 –200, 2008.

[160] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks. *Eclipse Modeling Framework*. Addison-Wesley Professional, trzecie wydanie, 2008.

[161] H. Störrle. Structured nodes in UML 2.0 activities. *Nordic Journal of Computing*, 11(3):279–302, 2004.

[162] H. Störrle. Semantics of control-flow in UML 2.0 activities. W: *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, s. 235 –242, 2004.

[163] H. Störrle. Semantics of exceptions in UML 2.0 activities. W: *Technical Report 0403, Ludwig-Maximilians University Munich, Institut of Computer Science*, 2004.

[164] H. Störrle. Semantics of expansion nodes in UML 2.0 activities. W: *Proc. 2nd Nordic Workshop on UML, Modeling, Methods and Tools (NWUML'04)*, 2004.

[165] H. Störrle. Semantics and verification of data flow in UML 2.0 activities. *Electronic Notes in Theoretical Computer Science*, 127(4):35 – 52, 2005.

[166] H. Störrle J. H. Hausmann. Towards a formal semantics of UML 2.0 activities. W: *Proceedings German Software Engineering Conference, P-64*, s. 117–128, 2005.

[167] K. Sullivan, J. Dugan, D. Coppit. The Galileo fault tree analysis tool. *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, s. 232–336, 1999.

[168] E. Syriani H. Ergin. Operational semantics of UML activity diagram: An application in project management. W: *MoDRE*, s. 1–8. IEEE, 2012.

[169] X. Tan M. Gruninger. Towards axiomatizing the semantics of uml activity diagrams: A situation-calculus perspective. W: *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, Tom 1, s. 324 –327, 2010.

[170] E. Tavares, P. Maciel, B. Silva, M. N. Oliveira, Jr. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. *Information Processing Letters*, 108(2):50–59, Sept. 2008.

[171] P. Terence. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, Dallas, 2007.

[172] UML Profile for Modeling and Analysis of Real-time and Embedded Systems. http://www.omg.org/marte, 2008.

[173] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. http://www.omg.org/spec/QFTP/1.1/, 2008.

[174] UML Profile for Schedulability, Performance, and Time. http://www.omg.org/spec/SPTP/1.1/, 2005.

[175] I. T. Union. Recommendation z.120: Message sequence chart (MSC). ITU-TS, Geneva, 1993.

[176] U.S. Nuclear Regulatory Commission. Washington, D. C. *Fault Tree Handbook: NUREG-0492*, 1981.

[177] M. Vaziri D. Jackson. Some shortcomings of OCL, the object constraint language of UML. *International Conference on the Technology of Object-Oriented Languages and Systems*, s. 555 – 571, 2000.

[178] V. Vitolins A. Kalnins. Semantics of UML 2.0 activity diagram for business modeling by means of virtual machine. W: *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, EDOC '05, s. 181–194, Washington, DC, USA, 2005. IEEE Computer Society.

[179] J. Vrijling. Probabilistic design of water defense systems in the Netherlands. *Reliability Engineering and System Safety*, 74(3):337 – 344, 2001.

[180] M. Wang L. Lu. A transformation method from UML statechart to Petri nets. W: *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, Tom 2, s. 89 –92, 2012.

[181] J. Warmer A. Kleppe. *OCL precyzyjne modelowanie w UML*. Wydawnictwa Naukowo-Techniczne, 2003.

[182] S. Werbińska-Wojciechowska. *Model logistycznego wsparcia systemu eksploatacji środków transportu*. Rozprawa doktorska, Politechnika Wrocławska, 2008.

[183] Y. Xia, J. Chen, M. Zhou, Y. Huang. A Petri-Net-Based Approach to QoS Estimation of Web Service Choreographies. W: *Advances in Web and Network Technologies, and Information Management*, Tom 5731 *Lecture Notes in Computer Science*, s. 113–124. Springer Berlin / Heidelberg, 2009.

[184] Y. Xu. The formal semantics of UML activity diagram based on process algebra. W: *Computer Science and Service System (CSSS), 2011 International Conference on*, s. 2729 –2732, 2011.

[185] Y. Yu, X. Quian, D. Lu, S. Wang, X. Lv, F. Niu. Fault tree and Monte Carlo simulation in passive system reliability analysis. W: *Probabilistic Methods Applied to Power Systems, 2004 International Conference on*, s. 226 –226, 2012.

[186] W. Zamojski. *Teoria i technika niezawodności*. Wydawnictwo Politechniki Wrocławskiej, 1976.

[187] W. Zamojski. Dependability of a discrete transport system. *Journal of Polish Safety and Reliability Association*, 2(3):159–168, 2012.

[188] E. Zio. Reliability engineering: Old problems and new challenges. *Reliability Engineering and System Safety*, 94(2):125 – 141, 2009.

[189] L. Zixian, N. Xin, L. Yiliu, S. Qinglu, W. Yukun. Gastric esophageal surgery risk analysis with a fault tree and Markov integrated model. *Reliability Engineering and System Safety*, 96(12):1591 – 1600, 2011.

Dodatek A

Generacja symulatora niezawodności z modelu wyrażonego w języku GN

W dodatku umieszczono omówienie koncepcji i implementacji algorytmu zaprezentowanego na rys. 6.10. W ten sposób uzupełniono rozdział 6 o szczegóły transformacji grafu niezdatności w kod wykonywalny stanowiący symulator niezawodności modelowanego systemu.

W nomenklaturze MDA [133] model przestawiony na rys. 6.8 może być traktowany jako Model Niezależny od Platformy - *ang. Platform Independent Model* (PIM), podczas gdy generowane wzorce to kod związany z platformą JBoss Drools [10]. Umyślne pominięcie Modelu Zależnego od Platformy - ang. Platform Specific Model (PSM), będącego w klasycznym MDA etapem pośrednim pomiędzy PIM a kodem, wynika z braku standardowego języka modelowania opisu reguł wnioskowania. Może się nim okazać w przyszłości RuleML [20].

W kolejnych podpunktach opisano schematy generacji reguł dla bramy GAND (sekcja A.1), COR (A.2) i przejścia (A.3). Faktycznie użytym językiem tworzenia schematów jest Xpand [51], natomiast wygenerowany kod jest w języku środowiska JBoss Drools.

A.1 Bramy uogólniające

Momenty początku i końca zajścia zdarzenia wyjściowego będącego na wyjściu bramy uogólniającej wynikają z funkcji realizowanej przez tę bramę. Stąd dla każdej bramy uogólniającej w modelu wygenerowane zostaną przynajmniej dwie reguły: jedna rozpoczynająca zdarzenie wyjściowe a druga je kończąca. Reguły realizują akcję $A8$ na rys. 6.10.

Z wydruku A.1 wynika, że w muszą istnieć zajścia wszystkich zdarzeń wejściowych bramy GAND, aby reguła została wykonana (linie 4-7). Jeśli ponadto warunki odpalenia są spełnione: odrębnie dla sytuacji, gdy zdarzenia bramy są proste, odrębnie gdy parametryczne, dochodzi do wykonania konkluzji, czyli części opisanej w liniach 15-17 i odpowiadającej akcji $A8$. Dodanie nowego zajścia w linii 17 spowoduje propagację zajścia zdarzenia w górę drzewa, czyli rozpatrzenie reguł, dla których aktualne zdarzenie wyjściowe jest wejściowym.

Wydruk A.1: Wzorzec reguł rozpoczynających zdarzenie wyjściowe bramy GAND

```
10 «DEFINE GAND_start_template FOR GAND»
11 rule "GAND «name» Start" salience 2 when
12 // fakty:
13 «FOREACH inputEdges AS edge»
14    zdarzenie wejściowe, do którego prowadzi zmienna edge,
15    zajście tego zdarzenia
16 «ENDFOREACH»
17 // jeśli zdarzenie wyjściowe jest parametryczne:
18     fakt, że wszystkie zajścia zdarzeń wejściowych mają ten sam parametr p,
19     fakt, że nie istnieje zajście zdarzenia wyjściowego z parametrem p
20 // jeśli zdarzenie wyjściowe jest proste:
21     fakt, że minimum z liczby wystąpień wszystkich zdarzeń wejściowych jest
22     większy niż liczba zajść zdarzenia wyjściowego
23 then
24    rozpocznij nowe zdarzenie: proste lub parametryczne,
25    w drugim przypadku ustaw parametr na wyznaczony z faktów,
26    dodaj zajście zdarzenia do pamięci roboczej
27 end
28 «ENDDEFINE»
```

Moment zakończenia zajścia zdarzenia bramy GAND wynika z odwrotnej sytuacji. W przypadku zdarzeń parametrycznych musi istnieć taki parametr p, który występuje na wyjściu, ale nie występuje na pewnym zdarzeniu wejściowym (wydruk A.2, linie 4-7). Natomiast dla zdarzeń prostych liczba zajść zdarzenia wyjściowego musi być większa niż minimum z liczby zajść wszystkich wejść (linie 8-10). W efekcie działania reguły usunięte zostaje zajście zdarzenia wyjściowego z pamięci roboczej.

Wydruk A.2: Wzorzec reguł zatrzymujących zdarzenie wyjściowe bramy GAND

```
1  «DEFINE GAND_stop_template FOR GAND»
2  rule "GAND «name» Stop" salience 2 when
3  // fakty:
4     zdarzenie wyjściowe bramy,
5  // jeśli zdarzenie wyjściowe jest parametryczne:
6        zajście zdarzenia wyjściowego z parametrem p,
7        pewne zdarzenie wejściowe, które nie zachodzi z parametrem p,
8  // jeśli zdarzenie wyjściowe jest proste:
9        zdarzenie wejściowe zachodzące mniejszą liczbę razy niż liczba zajść
10        zdarzenia wyjściowego
11 then
12    zakończ zajście zdarzenia wyjściowego,
13    usuń zajście z pamięci roboczej
14 end
15 «ENDDEFINE»
```

Przykładowy rezultat transformacji bramy $G17$ z rys. 4.4 został umieszczony na wydruku A.3.

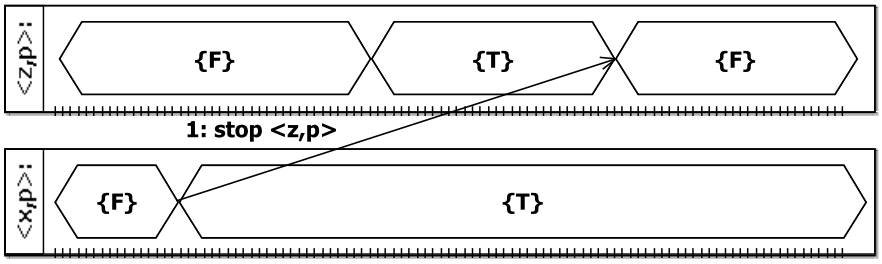
Wydruk A.3: Reguła startująca $E11$ gdy $E6$ i $E7$ zachodzą z wartością $p0$ parametru

```
1  rule "GAND G17 Start" salience 2 when
2     ie0: ParamEvent (name matches ''E6'')
3     t0:  ParamOccur(p0: value, this memberOf ie0.occurred)
4     ie1: ParamEvent (name matches ''E7'')
5     t1:  ParamOccur(p1: value, value == p0, this memberOf ie1.occurred)
6     oe0: ParamEvent(name == ''E11'')
7     not  ParamOccur(value == p0, this memberOf oe0.occurred)
8  then
9     ParamOccur ot0 = faultGraphFactory.createParamOccur(p0);
10    oe0.getOccurred().add(ot0);
11    insert(ot0);
12 end
```

A.2 Bramy przyczynowe

W przeciwieństwie do bram uogólniających, reguły bram przyczynowych nie są aktywowane zajściami zdarzenia wyjściowego. Zgodnie z modelem czasowym jeśli na wyjściu bramy jest zdarzenie proste, to w odpowiedniej chwili w przyszłości zostanie utworzone kolejne zajście. Jeśli natomiast zdarzenie jest parametryczne i w odpowiedniej chwili w przyszłości nie istnieje zajście z tym parametrem, to zostanie ono utworzone. Zdarzenia wyjściowe nie mają więc wpływu na działanie bramy w momencie odpalania.

Konsekwentnie przesłankami reguł tej rodziny elementów GN nie będą ani zdarzenia wyjściowe, ani ich zajścia. Może zajść zatem do sytuacji przedstawionej na przebiegu A.1, w którym brama CNOT zatrzymuje zdarzenie, które rozpoczęło się już po jej odpaleniu.



Rysunek A.1: Możliwe oddziaływanie bramy CNOT na zdarzenie z - zajście z parametrem p następuje po odpaleniu bramy CNOT

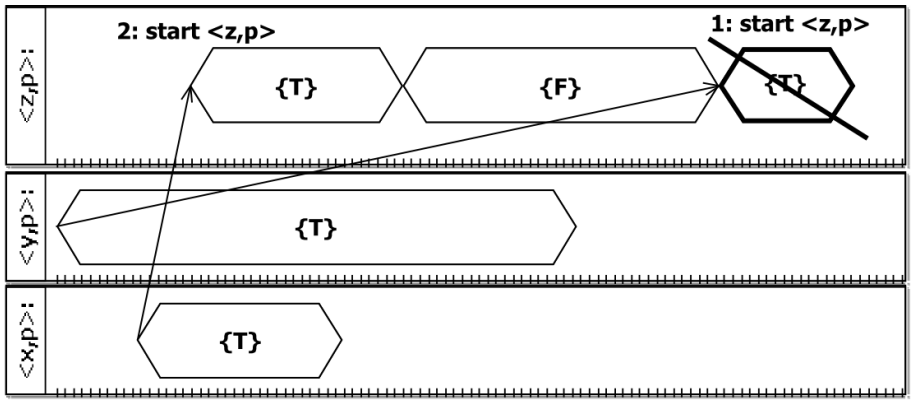
Ponadto ponieważ wiele zdarzeń symulacji może zajść przed aktywacją zdarzenia wyjściowego, faktyczna aktywacja jest odroczona na rys. 6.10 przy pomocy bufora centralnego $CB1$ do momentu, gdy aktualny czas symulacji będzie zgodny z chwilą odpalenia. Aktualizacja zdarzenia wyjściowego przez regułę bramy przyczynowej jest zatem zbędna i konsekwentnie akcja $A11$ zostaje ominięta.

Powyższe uwagi pozwalają zaprojektować schemat generujący reguły bramy COR, który w postaci pseudokodu został przedstawiony na wydruku A.4. Jeśli ma miejsce nowe zdarzenie wejściowe (linie 19-21), to należy zaplanować nowe zajście zdarzenia wyjściowego odległe o czas wynikający z realizacji zmiennej losowej powiązanej z tym zdarzeniem wejściowym (linie 23-26).

Wydruk A.4: Wzorzec reguły odpalającej bramę COR

```
16 «DEFINE COR_start_template FOR COR»
17 «FOREACH inputEdges AS edge»
18 rule "COR «name»«edge.source.name»" salience 1 when
19 // fakty:
20     pewne zdarzenie wejściowe bramy,
21     nowe zajście zdarzenia wejściowego
22 then
23     utwórz nowe zajście zdarzenia prostego lub parametrycznego,
24     znajdź odpowiednią dla wejścia zmienną losową opóźnienia,
25     zrealizuj zmienną losową,
26     dodaj nowe zdarzenie symulacji związane z nowym zajściem do bufora CB1
27 end
28 «ENDFOREACH»
29 «ENDDEFINE»
```

Logika niektórych bram, w tym COR, ma miejsce również w momencie aktywacji zdarzenia wyjściowego. Opisany w rozdziale 6 *efekt minimalizacji* jest realizowany w akcji $A3$ na rys. 6.10 wykonywanej po wybraniu zdarzenia symulacji z bufora, ale przed faktyczną aktywacją zdarzenia. Jeśli zachodzi kolejne zdarzenie wejściowe a inne zdarzenie wejściowe z tym samym parametrem jeszcze trwa, to zdarzenie wyjściowe zostanie odpalone tylko raz we wcześniejszym momencie (rys. A.2). Nie dochodzi więc do drugiego, późniejszego odpalenia bramy. Podobna zależność ma miejsce dla zdarzeń prostych, lecz w tym przypadku wystarczy, że jakiekolwiek zdarzenie zaszło wcześniej i wciąż trwało.



Rysunek A.2: Przebiegi czasowe zdarzeń bramy COR: x, y - zdarzenia wejściowe, z - zdarzenie wyjściowe, p - parametr zajść zdarzeń x, y, z

A.3 Przejścia

Generacja schematu odpalenia przejścia wymaga przejrzenia wchodzącej do tego przejścia krawędzi i wskazywanego przez nią zdarzenia, co odbywa się w liniach 5-21 wydruku A.5. Jak zaznaczono w podrozdziale 6.4, aby doszło do odpalenia przejścia muszą zachodzić wszystkie zdarzenia proste (linia 9) oraz muszą istnieć zajścia wszystkich zdarzeń parametrycznych z tym samym parametrem p (linie 10-14). Ponadto żadne z prostych zdarzeń hamujących nie może mieć miejsca (linie 16-17), podobnie jak nie

może istnieć hamujące zdarzenie parametryczne z parametrem p (linie 18-19). Sprawdzenie powyższych przesłanek odpowiada blokowi decyzyjnemu $D1$.

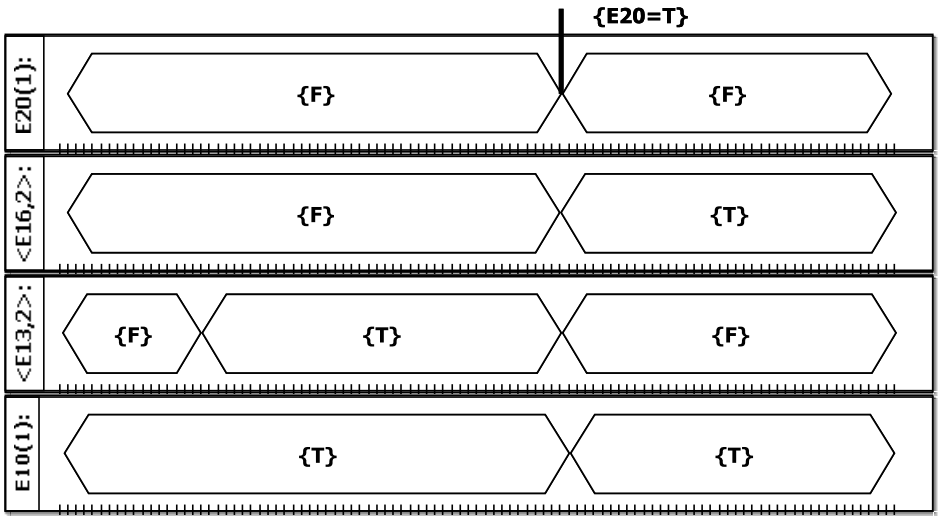
Wydruk A.5: Wzorzec reguły odpalającej przejście

```
1  «DEFINE Transition_start_template FOR Transition-»
2  rule "Transition «name»" salience 3 when
3  // fakty:
4    przejście,
5  «FOREACH inputEdges AS edge-»
6      /* zdarzenie wejściowe przejścia
7      «IF edge.isInhibitor == false -»
8      /* jeśli zdarzenie, do którego prowadzi zmienna edge jest proste:
9           zajście zdarzenia
10        jeśli zdarzenie jest parametryczne:
11          jeśli edge jest pierwszym w sekwencji inputEdges tego przejścia
12            pewne zajście zdarzenia, niech p będzie parametrem tego zajścia
13          jeśli edge nie jest pierwszy w kolekcji inputEdges
14            zajście z parametrem p */
15   «ELSE»
16     /* jeśli zdarzenie jest proste:
17        fakt, że nie istnieje żadne zajście
18      jeśli zdarzenie jest parametryczne:
19        fakt, że nie istnieje zajście z parametrem p */
20   «ENDIF»
21 «ENDFOREACH»
22   then
23 «FOREACH inputEdges AS edge»
24 «IF edge.isInhibitor == false»
25 // jeśli zdarzenie jest parametryczne
26       usuń zajście z wyznaczonym parametrem,
27 // jeśli zdarzenie jest proste
28       usuń jedno zajście,
29     usuń zajście zdarzenia z pamięci roboczej
30 «ENDIF»
31 «ENDFOREACH»
32 «FOREACH outputEdges AS edge»
33 // jeśli zdarzenie jest proste lub
34     jeśli jest parametryczne, ale nie istnieje zajście z parametrem p
35       utwórz nowe zajście tego zdarzenia
36 // jeśli zajście jest parametryczne, przypisz wartość parametru,
37     dodaj zajście do pamięci roboczej,
38 // jeśli zdarzenie ma przypisany czas trwania, to zaplanuj jego zakończenie
39 «ENDFOREACH»
40 end
41 «ENDDEFINE»
```

Podobnie jak w przypadku bram przyczynowych, na odpalenie przejść nie mają wpływu zdarzenia wyjściowe, stąd nie ma ich na liście faktów.

Konkluzja reguły (linie 23-38) przejścia wymaga przeglądu wszystkich krawędzi: zarówno wejściowych jak i wyjściowych. Wszystkie zajścia niehamujących zdarzeń wejściowych są kończone i usuwane z pamięci roboczej. Natomiast w zależności od typu zdarzenia wyjściowego tworzone jest nowe zajście, które następnie dodawane jest do pamięci roboczej.

Przykładem zastosowania przejścia jest alokacja konserwatora wykonywana przez $T3$ na rys. 6.7. Na przebiegu czasowym (rys. A.3) zarejestrowano działanie reguły, kiedy proces naprawy został już zainicjalizowany, czyli $E10$ zachodziło z wartością 1, a drugi dysk został uszkodzony ($E13$ z wartością 2), ale nie było dostępnego konserwatora, czyli nie zachodziło $E20$. Kiedy w końcu doszło do zwolnienia konserwatora, to mógł on być przydzielony dyskowi twardemu. Ta chwila czasu oznaczona jest wyrażeniem $\{E20 = T\}$ na rysunku i oznacza moment, kiedy warunki odpalenia $T3$ były spełnione. Nastąpiło zakończenie $E13$, ale rozpoczęcie $E16$, czyli dysk nie oczekiwał już na naprawę, ale był naprawiany. Konserwator był ponownie niedostępny, czyli nie zachodziło $E10$ z wartością 1.



Rysunek A.3: Przejście T3 z rys. 6.7 atomowo przydziela konserwatora do uszkodzonego dysku

Dodatek B

Wprowadzenie do Języka Ograniczeń Obiektów (OCL)

Wokół graficznego języka UML powstało wiele narzędzi tekstowych, takich jak OCL [132] czy Alloy [68], pozwalających wzbogacić powstałe modele i wyrazić koncepcje niedostępne w podstawowej wersji języka. Do podstawowych zadań najpopularniejszego z nich narzędzia OCL należą:

1. doprecyzowanie modelu UML, aby ten był bezbłędny i jednoznaczny,
2. przeszukiwanie oraz nawigacja po modelu,
3. definicja warunków początkowych i końcowych operacji.

Ze względu na drugi z wymienionych czynników OCL stał się integralną częścią niemal wszystkich systemów transformacji międzymodelowych. W niniejszej rozprawie wariant OCL'a związany z QVTO znalazł zastosowanie przy transformacji języka dziedzinowego w PDNZC oraz w definicji semantyki translacyjnej READ. Celem niniejszego dodatku jest natomiast wprowadzenie do OCL'a w zakresie wykorzystywanym w rozdziałach 5 oraz 8.

Podstawą języka są funkcje rozszerzające zbiór operacji dostępnych dla danej klasy UML'a. Przykładowo funkcja $oclIsTypeOf$ dostępna na każdym obiekcie UML zwraca logiczną prawdę, gdy byt na rzecz którego jest wywoływana jest dokładnie typu będącego argumentem funkcji. Zakładając, że zmienna o odpowiada pewnemu buforowi centralnemu to:

$$o.oclIsTypeOf(CentralBufferNode) = true$$

$$o.oclIsTypeOf(ObjectNode) = false$$

Natomiast funkcja $oclIsKindOf$ zwraca logiczną prawdę, gdy typ badanego obiektu jest zgodny z typem argumentu, lub jest względem niego pochodny. Biorąc pod uwagę, że $CentralBufferNode$ dziedziczy po $ObjectNode$:

$$o.oclIsKindOf(CentralBufferNode) = true$$

$$o.oclIsKindOf(ObjectNode) = true$$

Funkcja *select* służy do ograniczania zbioru według określonego kryterium. Ponieważ wykonywana jest na zbiorze, to symbol kropki przed funkcją zastępuje strzałkę. Jeśli u jest modelem UML, a $u.objects()$

zbiorem wszystkich elementów tego modelu, to wartością następującego wyrażenia jest podzbiór wszystkich elementów $CentralBufferNode$ tego modelu:

$$u.objects() \rightarrow select(e \mid e.oclIsKindOf(CentralBufferNode))$$

Zmienna e jest lokalna dla wyrażenia występującego po znaku | w funkcji *select*.

Z kolekcji można również usuwać elementy, co w poniższym przypadku zaowocuje wszystkimi obiektami pochodnymi $ObjectNode$ które jednocześnie nie są $CentralBufferNode$:

$$u.objects() \rightarrow select(e \mid e.oclIsKindOf(ObjectNode)) \rightarrow$$

$$reject(e \mid e.oclIsTypeOf(CentralBufferNode))$$

Powyższy zbiór jest nieuporządkowany. Kolejne zapytanie go porządkuje i zwraca pierwszy element tak otrzymanej sekwencji:

$$u.objects() \rightarrow select(e \mid e.oclIsKindOf(CentralBufferNode)) \rightarrow asSequence() \rightarrow at(1)$$

Funkcja $selectFirst$ pozwala zapisać powyższe wyrażenie w prostszej postaci:

$$u.objects() \rightarrow selectFirst(e \mid e.oclIsKindOf(CentralBufferNode))$$

Przekształcenie jednego zbioru na drugi, powiązany odbywa się funkcją *collect*. Funkcja $oclAsType$ rzutuje zmienną na określony argumentem typ:

$$u.objects() \rightarrow select(e \mid e.oclIsKindOf(CentralBufferNode)) \rightarrow$$

$$collect(e \mid e.oclAsType(CentralBufferNode).type)$$

Wartością powyższego wyrażenia jest zbiór wszystkich typów występujących w buforach centralnych modelu u.

Funkcja *exists* zwraca logiczną prawdę, jeśli wśród elementów kolekcji istnieje przynajmniej jeden, dla którego wyrażenie będące argumentem *exists* jest prawdziwe. Zatem poniższą konstrukcją można sprawdzić, czy w modelu istnieje element o nazwie $CB1$:

$$u.objects() \rightarrow exists(e \mid e.name = \text{“}CB1'')$$

Operator *not* odwraca logiczną wartość wyrażenia. Zatem jeśli w modelu nie ma elementu o nazwie $CB1$, to wartość poniższego wyrażenia będzie prawdą:

$$not\ (u.objects() \rightarrow exists(e \mid e.name = \text{“}CB1''))$$

Natomiast funkcja *sortedBy* zwraca kolekcję elementów uporządkowaną według porządku naturalnego typu sortowanych elementów:

$$u.objects() \rightarrow select(e \mid e.oclIsKindOf(CentralBufferNode)) \rightarrow$$

$$collect(e \mid e.oclAsType(CentralBufferNode).type) \rightarrow$$

$$sortedBy(e \mid e.oclAsType(NamedElement).name)$$

Funkcja *allChildren* zwraca wszystkie obiekty będące bezpośrednimi i pośrednimi potomkami (w rozumieniu operacji zawierania) obiektu, na rzecz którego jest wywoływana. Jeśli ft odnosi się do obiektu typu $FaultTree$ z rys. 5.3, to $ft.allChildren()$ to zbiór wszystkich elementów modelu PDNZC.

Wyczerpujący opis OCL zawiera pozycja [181].

Dodatek C

Opis wykorzystanych narzędzi projektowania sieci Petriego

Semantyka języka READ jest zdefiniowana w rozdziale 8 poprzez rozszerzenie diagramów aktywności modelowanych w pracy [165] przy pomocy narzędzia CPN Tools. Elementy READ nie dają się jednak w nim zdefiniować ze względu na przyjęty w tym oprogramowaniu model czasowy. W CPN Tools istnieją jedynie przejścia natychmiastowe, odpalane zawsze w pierwszej chwili spełnienia ich warunków. Żetony mogą mieć przypisane zmienne losowe stanowiące podstawę wyznaczenia momentów ich zaistnienia względem chwili odpalenia przejścia. Taka koncepcja ma niższą moc ekspresji niż przejścia czasowe wymagane do translacji READ w HLPN. Z tego powodu zdecydowano się na wykorzystanie języka alternatywnego oprogramowania Snoopy zawierającego przejścia czasowe. Warto zauważyć, że praca [165] nie podejmuje problemu czasu wykonania czynności i nie wykorzystuje modelu czasowego CPN Tools, a zatem prezentowana w niej transformacja może zostać zaimplementowana w narzędziu Snoopy.

Ponieważ w pracy [165] i w rozdziale 8 zakładają się znajomość wykorzystywanych w nich narzędzi, to w bieżącym dodatku zawarto wprowadzenie do oprogramowania CPN Tools oraz Snoopy w zakresie niezbędnym do definicji semantyki READ.

C.1 CPN Tools

Wprowadzenie do CPN Tools nastąpi poprzez rozpatrzenie sieci Petriego zbudowanej w [165] na podstawie diagramu aktywności omawianego w specyfikacji UML. Analizę modelu z rys. C.1 należy rozpocząć od fragmentu kodu po lewej jego stronie zawierającej definicje typów i zmiennych sieci.

Etykietą $ORDER$ oznacza się zbiór kolorów złożonych będący produktem kartezjańskim pięciu typów prostych*. Kolor zamówienia to zatem uporządkowana $5 - ka$. Pierwsza składowa koloru to wartość całkowitoliczbowa odpowiadająca identyfikatorowi zamówienia. Kolejne składowe to cztery flagi logiczne o wartościach $true$ lub $false$ oznaczające czy zamówienie zostało odpowiednio: wypełnione, podsumowane, zapłacone oraz dostarczone. Każde miejsce modelu oznaczone owalem ma przypisany dokładnie jeden zbiór kolorów. Na przykład zbiorem kolorów iN jest INT, a $p1$ - $ORDER$.

Na łukach łączących przejścia z miejscami znajdują się wyrażenia, których wartości to kolory. Wyrażeniami mogą być: zmienne, uporządkowane w nawiasach $n - ki$ oraz funkcje.

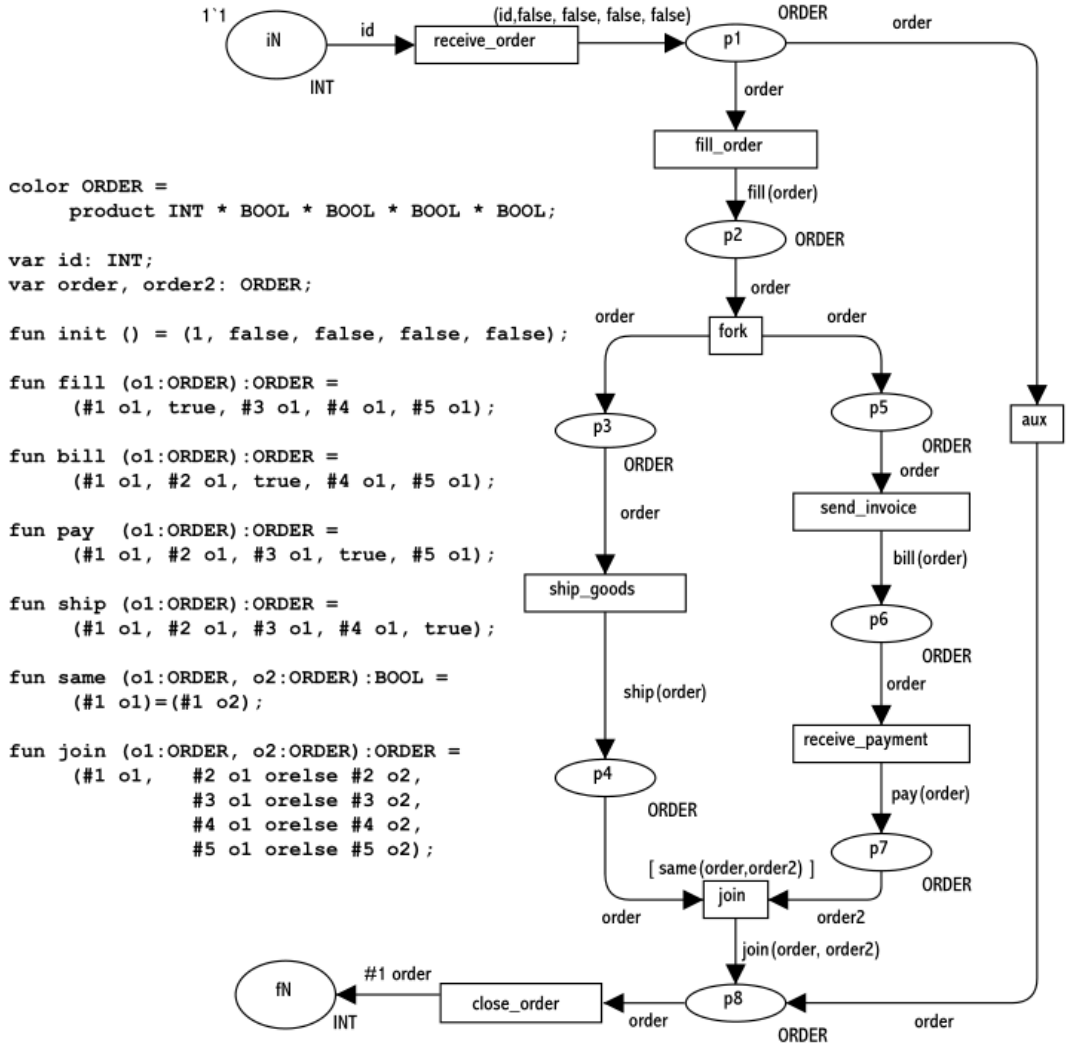
*Począwszy od wersji 3.4 CPN Tools używa się bardziej intuicyjnego literału *colorset* zamiast *color*.

Wszystkie zmienne występujące w sieci muszą zostać zdefiniowane konstrukcjami *var*, w których zmiennym przypisuje się zbiór kolorów. W przykładowym modelu istnieją zmienne *id*, *order* oraz *order*2.

Funkcja *init* zwraca nowy kolor, którego pierwsza składowa jest równa 1 a pozostałe $false$. Z kolei funkcja $fill$ przyjmująca zamówienie jako argument zwraca nowy kolor taki, że składowe o numerach 1, 3, 4, 5 zostają przepisane z argumentu, a druga składowa otrzymuje wartość logiczną *true*. Wyrażenie *#1 o1* oznacza pierwszą składową koloru *o*1.

W CPN Tools istnieją jedynie przejścia natychmiastowe graficznie oznaczane prostokątami z nazwami w środku. Przykładowo, przejście *receive_order* odpala się, gdy w miejscu iN znajdzie się jakiś żeton. Jego wartość będzie symbolizowana zmienną *id*. Z etykiety nad tym przejściem wynika natomiast, jak zmienna *id* jest wykorzystana do utworzenia nowego koloru ze zbioru $ORDER$. Taki kolor, o pierwszej składowej równej wartości z miejsca iN, zostanie umieszczony na miejscu $p1$.

Oznaczenie 1‘1 przy miejscu iN stanowi znakowanie początkowe, które w tym przypadku stanowi jeden żeton całkowitoliczbowy o wartości 1. Gdyby miejsce miało przyporządkowany kolor złożony, to ciąg znaków po apostrofie byłby n-ką uporządkowaną.



Rysunek C.1: Sieć Petriego reprezentująca proces realizacji zamówienia zaczerpnięta z [165]

Oprogramowanie CPN Tools umożliwia również przypisanie żetonom chwili ich zaistnienia. Wyrażenie 1‘1@100 oznacza, że w chwili 100 do miejsca dodany zostanie jeden kolor o wartości 1. Jak wspomniano,

transformacja zaproponowana w [165] nie podejmuje problemu opisu czasu wykonania czynności, a zatem konstrukcja o tej składni nie jest widoczna na rys. C.1.

C.2 Snoopy

W narzędziu Snoopy zbiory kolorów należy zdefiniować również dla typów prostych takich jak liczby rzeczywiste czy wartości dyskretne, których w przeciwieństwie do CPN Tools nie można używać bezpośrednio. Jak wynika z tab. C.1 w modelu bramy CAND używanych jest 7 zbiorów kolorów, z czego pierwsze 4 to zbiory proste, a ostatnie 3 to zbiory złożone będące iloczynami kartezjańskimi wymienionych w ostatniej kolumnie typów prostych.

Każdy kolor o typie *tramId* jest liczbą całkowitą między 0 i 10000 identyfikującą tramwaj, a *tramFailureState* przyjmuje wartość *Occuring*, *Scheduled* lub *AwaitingDelivery*. Z kolei $TramFailure$ to typ złożony będący produktem $tramFailureState$ oraz $tramId$.

Tabela C.1: Zbiory kolorów wykorzystywane w sieci Petriego dla bramy CAND

Nazwa zbioru	**Typ**	**Definicja**
tramId	int	*0-10000*
timeResourceState	enum	*Passed*
tramFailureState	enum	*Occuring, Scheduled, AwaitingDelivery*
hazardState	enum	*Ongoing*
Hazard	product	*hazardState, tramId*
TimeResource	product	*timeResourceState, tramId*
TramFailure	product	*tramFailureState, tramId*

Miejsce reprezentowane jest przez okrąg z dwiema etykietami. Pierwsza to nazwa przyporządkowanego zbioru kolorów, a druga to nazwa miejsca zgodna z formatem pi, gdzie i jest numerem miejsca.

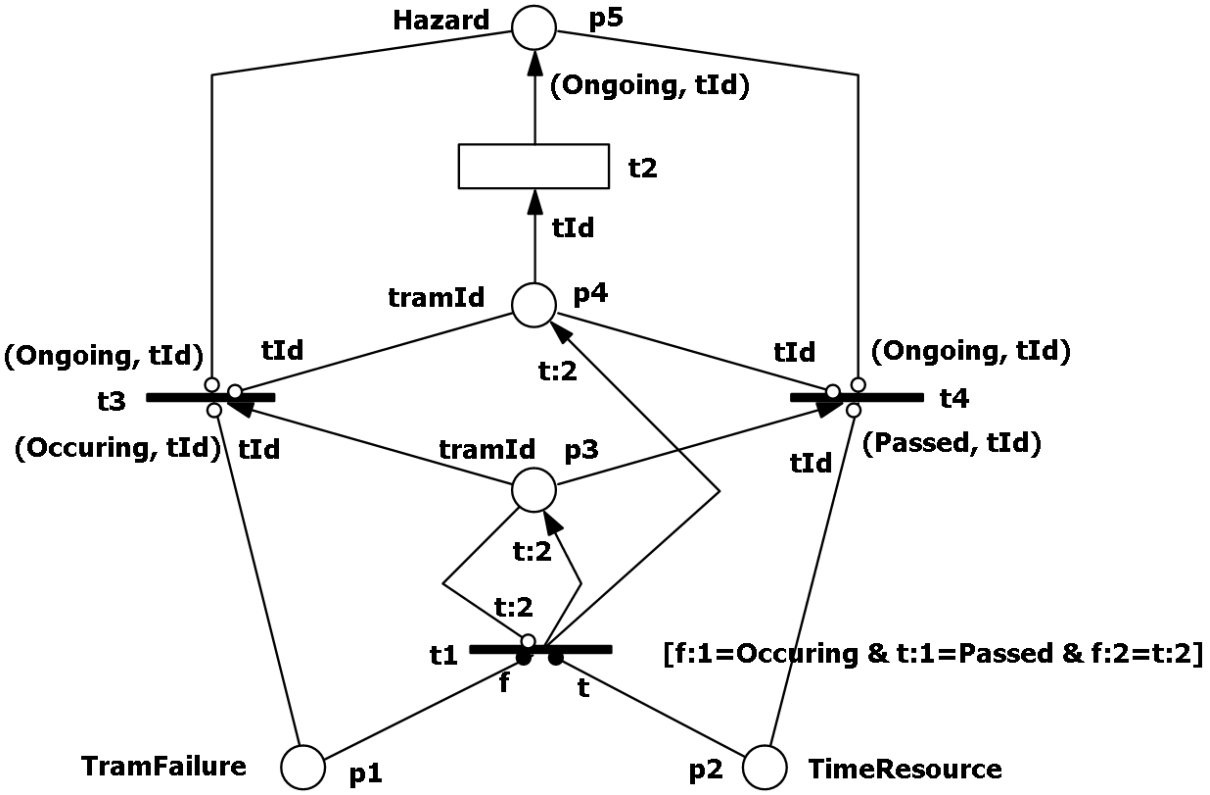
W narzędziu Snoopy istnieją dwa typy przejść: natychmiastowe oznaczone cienką linią oraz czasowe oznaczone niewypełnionym prostokątem (rys. C.2). Przejścia czasowe mają przypisaną zmienną losową oznaczającą czas odpalenia, ale ta zmienna nie jest graficznie pokazana na rysunku.

Wyrażenia na łukach są kolorami definiowanymi m. in. przy pomocy zmiennych, literałów oraz operatora dostępu do składowej koloru. Symbole tId, t, f odpowiadają na rysunku zmiennym o typach odpowiednio $tramId$, $Hazard$, $TimeResource$ oraz $TramFailure$.

Aby odpaliło się przejście $t2$ na $p4$ musi istnieć żeton. Jego wartość jest reprezentowana przez zmienną tId. Przy jej pomocy przejście tworzy nowy żeton o kolorze $(Ongoing, tId)$ dodany do miejsca $p5$.

Odwołanie do określonej składowej złożonego koloru odbywa się po dwukropku, czego przykładem jest $f : 1$ w wyrażeniu strzegącym odpalenie przejścia $t1$. W ten sposób można sprawdzić, czy żeton skojarzony z miejscem $p1$ ma pierwszą składową koloru równą $Occuring$.

Łuki zakończone pustym okręgiem hamują odpalenie przejścia, tzn. jeśli na miejscu wejściowym istnieje żeton o kolorze specyfikowanym przez taki łuk, to przejście nie zostanie odpalone. Łuki zakończone czarnym kołem służą do odczytywania koloru żetonu bez jego usuwania z miejsca wejściowego.



Rysunek C.2: Sieć Petriego specyfikująca działanie bramy CAND