Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Simulation und Graphik

Diplomarbeit

Entwicklung eines erweiterten Proxel-basierten Petri-Netz-Simulators

Verfasser:

Fabian Wickborn

16. November 2004

Betreuer:

Prof. Dr. Graham Horton (Universität Magdeburg)
Dipl.-Inf. Stefan Heller (DaimlerChrysler AG)

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Simulation und Graphik
Universitätsplatz 2
39106 Magdeburg

DaimlerChrysler AG
Abteilung REI/AA
096 / T728
70546 Stuttgart

Fabian Wickborn
Entwicklung eines erweiterten Proxel-basierten Petri-Netz-Simulators
Diplomarbeit, Otto-von-Guericke-Universität
Magdeburg, 2004.

Inhaltsverzeichnis

Abbildungsverzeichnis

Danksagung

Die vorliegende Arbeit wäre nicht entstanden ohne die Hilfe, Unterstützung und auch Ermutigung zahlreicher Personen. Daher möchte ich die Gelegenheit nutzen, ihnen an dieser Stelle zu danken.

Ganz vorne stehen dabei drei Menschen, die mit ihren Ratschlägen, ihren Anregungen, ihrer Kritik und ihrer Freundschaft mir sehr dabei geholfen haben, mich in den letzten Monaten weiterzuentwickeln und zu wachsen: Stefan Heller, Felix Engelhard und Graham Horton.

Stefan Greiner sei herzlichst dafür bedankt, daß er sich als Zweitgutachter für die Diplomarbeit zur Verfügung stellte.

Ich danke weiterhin meinen Eltern Katharina und Lorenz Wickborn, die mir mein Studium ermöglichten und mir so die Gelegenheit gaben, zu werden, was ich sein will. Antje Maurer danke ich für ihr Verständnis und ihren Beistand in Zeiten, in denen Berufliches für mich wichtiger war als Privates.

Ein herzliches Danke auch an Andrea Mayer, Sanja Lazarova-Molnar, Claudia Isensee, Mirko Böttcher, René Chelvier, Thomas Simon und Stephan Finn für Rat, Zuspruch und Freundschaft.

Mein Dank gilt auch erneut dem Team Systemsicherheit der Abteilung REI/AA (ehemals RIC/AS) der DaimlerChrysler AG in Esslingen, welches mich einmal mehr wohlwollend aufnahm und mir eine Arbeit in angenehmer Atmosphäre ermöglichte.

Zu guter Letzt danke ich allen Menschen, denen ich im Studium begegnet bin, von denen ich lernen und mit denen gemeinsam ich Erfahrungen sammeln konnte, allen Freunden, allen Kommilitonen, Professoren und Mitarbeitern an der Fakultät für Informatik der Universität Magdeburg.

„Zufall ist das unberechenbare Geschehen, das sich unserer Vernunft und Absicht entzieht.“

Gebrüder Grimm (Deutsches Wörterbuch)

„Auch das Zufälligste ist nur ein auf entfernterem Wege herangekommenes Notwendiges.“

Arthur Schopenhauer

„Ich glaube nicht an Zufall. Die Menschen, die in der Welt vorwärts kommen, sind die Menschen, die aufstehen und nach denen von ihnen benötigten Zufall Ausschau halten.“

George Bernard Shaw

0. Zusammenfassung

Die *Proxel-basierte Simulation* ist ein junges Verfahren zur Analyse von zeit-diskreten, ereignisorientierten Modellen, welches sich in der Vergangenheit für bestimmte (fest kodierte) Simulationsmodelle als sehr viel effizienter als die diskrete ereignis-basierte Simulation erwiesen hat. In der vorliegenden Diplomarbeit wird zunächst ein allgemeingültiger Proxel-basierter Simulator für erweiterte Stochastische Petri-Netze erstellt in ein bestehendes Werkzeug eingefügt. Zum anderen werden Ideen und experimentelle Ergebnisse für eine Erweiterung präsentiert, durch welche sich die Effizienz der Proxel-basierten Simulation unter bestimmten Umständen weiter verbessern läßt.

Die Arbeit kann den Themengebieten *Zeit-diskrete ereignis-basierte Simulation* und *Stochastische Petri-Netze* zugeordnet werden.

Proxel-based simulation were recently introduced as a method for analysing discrete-event simulation models which is much more efficient than common discrete-event algorithms for certain (hard-coded) models. First, in this Master thesis, a general-purpose implementation for using Proxel-based methods for Stochastic Petri Nets is built. Second, ideas and experimental results for an extension are presented with which the efficiency of the Proxel-based simulation method can be raised further.

The thesis is meant to be part of the topics *Discrete Event Simulation* and *Stochastic Petri Nets*.

1. Einführung

1.1. Über die Arbeit

Die vorliegende Arbeit kann dem Themengebiet Zeit-diskrete, ereignisbasierte Simulation zugeordnet werden. Sie dient der Erlangung des akademischen Grades „Diplom-Informatiker“ und wurde im Rahmen einer industriellen Kooperation mit der Daimler-Chrysler AG angefertigt.

Das Dokument gliedert sich wie folgt: In diesem ersten Kapitel werden die Beweggründe für die Arbeit dargelegt, aus denen sich auch die anschließend beschriebene Aufgabenstellung ergab.

In den Grundlagen (Kapitel 2) wird das bearbeitete Petri-Netz-Werkzeug Expect näher vorgestellt und eine Einführung in Proxel-basierte Simulation gegeben. Das dort zusammengetragenen Wissen ist eine notwendige Voraussetzung für ein umfassendes Verständnis der darauffolgenden Kapitel.

Die allgemeingültige Umsetzung der Proxel-basierten Simulation für Stochastische Petri-Netze wird in Kapitel 3 beschrieben. Den Schwerpunkt bildet hierbei die Abbildung von erweiterten Modellierungsmöglichkeiten sowie der Berechnung von Ergebnisstatistiken auf das Proxel-Verfahren. Desweiteren werden Experimente beschrieben, mit denen die Arbeit validiert wird. Eine vergleichende Messung zur Effizienz des Verfahrens beendet das Kapitel.

Eine Diskussion der Idee, variable Zeitschritte im Proxel-Verfahren einzusetzen, erfolgt in Kapitel 4. Dort werden auch die notwendigen Anpassungen an den Datenstrukturen und dem Algorithmus entwickelt und umgesetzt. Anhand von Beispielmodellen wird ein Vergleich zum bestehenden Verfahren durchgeführt, bei dem Änderungen bei der Effizienz des Verfahrens quantitativ erfasst und dokumentiert werden. Das Kapitel wird mit einer theoretischen Erörterung von Verfahren zur weiteren Automatisierung der Berechnung beendet.

Eine schließende Zusammenfassung der Arbeit, begleitet von einigen Reflexionen durch den Autor beendet als Kapitel 5 den Hauptteil des vorliegenden Dokuments. Dabei wird auch eine Einschätzung weiterer Möglichkeiten bei der zukünftigen Erforschung und der Verwendung Proxel-basierter Simulationsverfahren gegeben.

Der Anhang A beschreibt Details zur Implementierung, die deren Verständnis erleichtern sollen, die aber keine globale Relevanz in dieser Arbeit haben. Er enthält eine Beschreibung der verwendeten Datenstrukturen und erklärt die Entscheidungen, die in Bezug auf Software-Design und Algorithmen getroffen wurden. Mit Hilfe des Anhangs soll sich die geleistete Programmiertätigkeit nachvollziehen lassen.

1.2. Motivation

Bei der Entwicklung neuer Technologien und Produkte in der Industrie gewinnt die Simulation aufgrund der zunehmend komplexer werdenden Struktur von Systemen immer mehr an Bedeutung. Das heißt, es werden Aspekte eines real-existierenden, sich in Planung befindlichen oder gänzlich fiktiven Systems in einem Modell abstrakt nachgebildet, um dieses anschließend durch einen Computer analysieren zu lassen. Eine derartige Abstraktion ist von Vorteil, wenn die Herstellung von realen Prototypen mit hohem finanziellen oder fachlichen Aufwand verbunden oder durch die Art des realen Systems erst gar nicht möglich ist. Die Nachteile, z.B. der Aufwand durch Validierung und Verifikation von Simulationsmodellen, wiegen weniger stark als die Vorteile.

Insbesondere die Aussagengewinnung für die Voraussage von Eigenschaften wie beispielsweise Kosten und Zuverlässigkeit ist für den industriellen Anwender ein interessanter Einsatzpunkt von Simulation. Geeignete Modelle erlauben es dabei, die Entwicklung und das Zusammenspiel von verschiedenen Faktoren (z.B. von existierenden Bauteilen in einem Auto) nachzubilden, um das Verhalten zu studieren und mit den Erkenntnissen Aussagen z.B. über die Lebensdauer von realen Systemen anfertigen zu können. Auf Basis dieser Aussagen lassen sich wiederum Firmenstrategien entwerfen und bewerten. Typische Fragen, die mit den durch die Simulation solcher Modelle gewonnenen Aussagen beantwortet werden, betreffen zum Beispiel die Zuverlässigkeit neuer oder sich noch in der Entwicklung befindender Systeme.

Die Monte-Carlo-Simulation der soeben umschriebenen Kosten- und Zuverlässigkeitsmodelle ist allerdings nicht vollkommen unproblematisch. Denn die Modelle enthalten häufig sogenannte „seltene Ereignisse“, das heißt Zustandsübergänge, die nur mit sehr geringer Wahrscheinlichkeit aufteten. Aufgrund der stochastischen Natur der Simulation und der Abhängigkeit von (Pseudo-)Zufallszahlen kann während der Berechnung nicht garantiert werden, daß das Modell diese unwahrscheinlichen Zustandsübergänge durchlaufen hat. Derartige Modelle mit ereignisbasierten Algorithmen zu analysieren, ist somit sehr aufwendig, da eine sehr hohe Anzahl von Replikationen durchgeführt werden muß, um sicher gehen zu können, daß die seltenen Ereignisse während der Simulation auch eingetreten und somit in den Ergebnissen berücksichtigt sind. Entsprechend groß ist die Analysezeit, welche in Summe durch die Replikationen entsteht.

Theoretisch möglich ist eine analytische Betrachtung der Modelle. Die führt im allgemeinen Fall zu partiellen Differentialgleichungen und scheidet in der Praxis wegen der Komplexität eines entstehenden Gleichungssystems somit für die Anwendung von vornherein aus.

Gleichzeitig zum entstehenden steigenden Bedarf nach Rechenzeit für genaue Ergebnisse und der stetig wachsenden Komplexität der Modelle existiert durch wirtschaftliche Interessen der Wunsch, mit möglichst geringem (zeitlichen) Aufwand an die gewünschten Informationen zu gelangen, um anhand derer Aussagen generieren, Fragen beantworten und Entscheidungen treffen zu können.

Ein neues Verfahren auf der Basis von sogenannten Proxels, welches an der Otto-von-Guericke-Universität Magdeburg entwickelt wurde, verspricht für die Analyse mancher der oben beschriebenen Modelle geringere Laufzeiten und somit einen insgesamt schnel-

leren Arbeitsvorgang bei der Aussagengewinnung durch Simulation.

Das noch junge Verfahren selbst – die ersten Veröffentlichungen datieren gerade einmal zwei Jahre zurück – ist nach wie vor Gegenstand intensiver Forschung und es ist damit zu rechnen, daß weitere Geschwindigkeitsgewinne gegenüber den momentanen Ergebnissen erzielt werden.

In Anbetracht dieser positiven Erfahrungen ist es sinnvoll, Proxel-basierte Simulation weiter zu untersuchen und Anwendern in Industrie und Wirtschaft zugänglich zu machen.

1.3. Aufgabenstellung

Die in dieser Diplomarbeit bearbeitete Aufgabenstellung entsprang einer gegenwärtigen Kooperation des Lehrstuhls für Simulation der Universität Magdeburg und der Daimler-Chrysler AG, in der neue Methoden zur simulativen Analyse von Stochastischen Petri-Netzen auf der Basis von Proxels entwickelt und untersucht werden.

Die Aufgabe selbst bestand nun aus zwei sich ergänzenden Teilen. Der erste Aufgabenteil wurde primär durch den Industriepartner DaimlerChrysler AG vorgegeben. Darin sollten die bis zum aktuellen Zeitpunkt gewonnenen Erkenntnisse und Methoden zur Proxel-basierten Simulation in ein bestehendes Werkzeug zur Analyse von Petri-Netzen, Fehlerbäumen und Zustandsräumen eingefügt werden, um die dort durchzuführenden Analysen und Prognosen schneller durchführen zu können. Die Proxel-basierte Simulation sollte vorerst lediglich für die im Werkzeug behandelten erweiterten Stochastischen Petri-Netze implementiert werden. Der Industriepartner stellte dafür eigene realistische Modelle zur Verfügung und lieferte somit praktische Rahmenbedingungen.

Der zweite Teil der Aufgabenstellung hatte vorwiegend theoretischeren Charakter. Es sollte untersucht werden, ob und wenn ja, wie sehr, sich der Einsatz von lokal angepaßten Zeitschrittweiten während der Simulation positiv auf den Bedarf an Rechenzeit auswirkt. Die gewonnenen Erkenntnisse stellen in jedem Fall einen wichtigen Beitrag zur Erforschung der Proxel-basierten Simulation dar.

1.4. Ziele und Erwartungen

In Analogie zur Aufgabenstellung bestimmten im wesentlichen zwei Ziele diese Diplomarbeit. Das erste Ziel der Arbeit war eine allgemeine Implementierung für die Proxel-basierte Analyse von Stochastischen Petri-Netzen für die Analyse von Simulationsmodellen, die Verwendung in einem existierenden Petri-Netz-Werkzeug finden wird. Da an den Algorithmus die Erwartung gerichtet wird, für einige der bei DaimlerChrysler zu untersuchenden Zuverlässigkeitsmodelle schneller zu sein als die bisher verwendeten Simulationalgorithmen, sollte die Implementierung im Werkzeug für erweiterte Stochastische Petri-Netze verwendet werden. Die stark erweiterte Mächtigkeit für der Modellierung dieses Simulators sollte dabei in vollem Umfang auf die Proxel-Simulation abgebildet werden. An die Implementierung wird die Erwartung gerichtet, insbesondere für einige der beim Industriepartner häufig auftretenden realen Sicherheits- und Qualitätsmodelle schneller Ergebnisse als die diskrete Simulation zu liefern (siehe Kapitel 3).

Bisher existierten noch keine allgemein-gültigen Umsetzungen der Proxel-basierten Simulation und damit auch noch keine praktischen Erfahrungen mit auftretenden Problemen bei ihrer Erstellung. Die angestrebte Implementierung war folglich die erste ihrer Art, in der die Modelle nicht fest in den Algorithmus integriert waren. Ein weiteres Ziel der Arbeit ist daher die Dokumentation der Hindernisse und Problemlösungen während der Umsetzung (siehe Anhang A).

Das zweite Ziel der Arbeit ist das Wissen darüber, ob sich die Effizienz des Algorithmus durch die Verwendung variabler Arbeitsparameter verbessern läßt, und falls ja, mit welchen Verbesserungen gerechnet werden kann. Dazu werden bisher verwendete Annahmen über die Konstanz der performanz-entscheidenden Zeitschrittweite des Algorithmus aufgehoben, um die Analyse zu beschleunigen. Theoretische Vorüberlegungen der Magdeburger Forscher hatten zu der Idee geführt, daß variable, adaptive Zeitschritte sich vorteilhaft auf die Effizienz und die Laufzeit des Algorithmus auswirken würden. In der Diplomarbeit sollten erste Prototypen von Simulatoren unter Einsatz variable Zeitschritte erstellt und anschließend experimentell mit den bestehenden Basisalgorithmen verglichen werden. Es wurde erwartet, daß sich anhand der Ergebnisse quantitative Aussagen über mögliche Veränderungen der Effizienz erstellen lassen. Siehe Kapitel 4.

2. Grundlagen

Die Grundlagen sollen das notwendige Rüstzeug für das Verständnis der darauf folgenden Ausführungen liefern. Dabei wurde aus Gründen der Übersichtlichkeit bewußt darauf verzichtet, Hintergrundwissen übermäßig detailliert darzustellen. Stattdessen weisen Referenzen den interessierten Leser auf weiterführende Literatur hin.

Vorausgesetzt werden allgemeine Kenntnisse über Simulation, Zustandsräume, Erreichbarkeitsgraphen und über Aufbau und Verwendung allgemeiner Stochastischer Petri-Netze, wie sie unter anderem in [Hor] erklärt werden.

Es wird zuerst das Analyse-Werkzeug Expect vorgestellt, dessen erweiterte Mächtigkeit der Modellierung auf die Proxel-basierte Simulation abgebildet werden soll. Anschließend wird anhand von einfachen Beispielen, wie sie in der entsprechenden Literatur gefunden werden, eine Einführung in das Proxel-Verfahren gegeben.

2.1. Das Modellierungs- und Analyse-Werkzeug Expect

Expect (ehemals ein Akronym für **Ex**tended **Pe**tri Net **C**omputing **T**ool) ist ein Softwarewerkzeug, welches primär für die Erstellung und Analyse von Modellen in Form von erweiterten Stochastischen Petri-Netzen, Fehlerbäumen und Zustandsräumen gedacht ist. Expect stellt hierbei die Weiterentwicklung von „PeNeTo“ [HGH02] dar.

Es wurde in der DaimlerChrysler AG für den internen Gebrauch entwickelt. Dabei liegt ein besonderes Augenmerk auf der Analyse von Kosten- und Zuverlässigkeitsmodellen ([Gre04]). Das Werkzeug ist in der Sprache Java geschrieben, wodurch die Sicherstellung der Plattformunabhängigkeit und die schnelle Verbreitung entscheidend erleichtert werden.

Die aktuelle Version umfaßt konkret

- einen graphischen Editor für Petri-Netze, Fehlerbäume und Zustandsräume,
- für Petri-Netze eine manuelle und simulatorgestützte Visualisierung des Tokengames zur Unterstützung der Eingabe und des Verständnisses, einen Simulatorkern zur diskreten ereignis-gesteuerten Simulation, der auch zur parallelen Simulation in der Lage ist [Wic02], einen Erreichbarkeitsgraphen-Erzeuger und -Analysator für die numerische, phasen-verteilte Analyse,
- für Zustandsräume einen Simulator, welcher ebenfalls parallele Simulation zuläßt,
- für Fehlerbäume einen Simulator, der auch nicht-triviale Basic-Events in Form von Petri-Netzen erlaubt [Fin04].

Diese drei Modellklassen sind aus Sicht des Industriepartners besonders gut geeignet, um Kosten- und Zuverlässigkeitsmodelle zu beschreiben und zu analysieren. Deswegen ist eine Verbindung derartiger Editoren und Analyse-Algorithmen in einem einzigen Werkzeug sinnvoll. Weitere einführenden Informationen zu Expect finden sich in [Wic02] und [Gre04].

In dieser Arbeit werden lediglich die erweiterten Stochastischen Petri-Netze betrachtet, da der größte Teil der zu analysierenden Systeme durch Petri-Netz-Modelle abstrahiert wird. Daher erscheint es lohnenswert, insbesondere die Petri-Netz-Simulation weiter zu verbessern und effizienter zu gestalten.

2.2. Erweiterungen der Modellierung von Stochastischen Petri-Netzen

Die durch Expect betrachtete Klasse der Stochastischen Petri-Netze erweitert die Definition aus [Hor] um die nachstehend beschriebenen Eigenschaften.

Für die Parameter der Elemente eines Petri-Netzes können zusätzlich zu konstanten Werten auch Funktionen verwendet werden, die den Parameterwert in Abhängigkeit von verschiedenen Modellzuständen definieren. Zu diesen vorhanden Variablen zählen die Systemzeit des Petri-Netz-Modells und der aktuelle Zustand des Modells, welcher sich in der *Belegung* des Netzes, d.h. der Anzahl von Marken in den Stellen des Petri-Netzes, darstellt (auch *Markierung* des Netzes genannt). Funktionen können für die folgenden Werte verwendet werden: Sämtliche Parameter der Verteilungsfunktion einer Transition , die maximale Tokenanzahl einer Stelle und die Multiplizität einer Kante (unabhängig vom Typ der Kante).

Für der Verteilungen von Transitionen kann eine Zeitskalierung verwendet werden. Dabei wird die verbleibende Zeit bis zum Feuern einer entsprechend konfigurierten Transition in Abhängigkeit von Markierungsveränderungen des Netzes gewichtet. Das Ziel dieser Option ist es, die Schwankung von Verarbeitungszeiten durch wechselnde Verfügbarkeit von Ressourcen einfacher modellierbar zu machen.

Eine einzelne Transition kann mehrere Server (im Vokabular der Warteschlangen-Theorie gesprochen) darstellen. Dies bedeutet, daß die Transition streng genommen eine Repräsentation einer Menge von gleichartigen Transitionen darstellt, die Transition also eine *Multiplizität* besitzt. Gleichartigkeit bedeutet in diesem Fall, daß die selbe Verteilungsfunktion verwendet wird und daß alle repräsentierten Transitionen gleiche Kanten haben (d.h. Kanten mit der selben Multiplizität und denselben angeschlossenen Stellen). Der *Aktivierungsgrad* einer Transition gibt an, wie viele Server gegenwärtig aktiviert sind. Für die Anzahl der repräsentierten Server kann eine obere Schranke angegeben werden (*Multi Server*). Die Anzahl kann auch unbeschränkt sein (*Infinite Server*), so daß immer so viele Server aktiv sind, wie durch die Markierung möglich.

Neben den weit verbreiteten Verdrängungsstrategien *Race Age* und *Race Enable* stehen mit *Race Reset* und *Race Repeat* zwei weitere Strategien zur Verdrängung von Serven zur Verfügung. Race Reset bezeichnet dabei die Eigenschaft einer Transition T, genau jedes

Mal eine neue Zufallszahl für die Feuerzeit zu generieren, wenn eine andere Transition U ($T \neq U$) gefeuert hat. Die Strategie Race Repeat besteht darin, nur dann eine neue Feuerzeit zu berechnen, wenn die Transition selbst gefeuert hat (analog zu Race Age), darüber hinaus aber bei jeder Reaktivierung die ehemals berechnete Zeit erneut komplett zu verwenden (wohingegen bei Race Age die bereits verstrichene Zeit berücksichtigt wird).

Die Rewards (Kostenfunktionen) werden in drei Klassen unterschieden:

1. Nicht-akkumulierte Rewards: Zustands-Kostenfunktionen, die entsprechend des Netzzustandes zeitlich gewichtet berechnet werden.
2. Akkumulierte Rewards: Zustands-Kostenfunktionen, die bei Zustandsübergängen ohne zeitliche Gewichtung inkrementiert werden.
3. Impuls-Rewards: Kostenfunktionen, die beim Feuern einer speziell zugeordneten Transition ohne zeitliche Gewichtung aufsummiert werden.

Alle diese Erweiterungen haben sich bei der Modellierung als hilfreich und vorteilhaft erwiesen, um komplexere Beziehungen und Interaktionen innerhalb der zu analysierenden Systeme auszudrücken. Eine Implementierung dieser Erweiterungen in das Werkzeug stellt somit eine notwendige Voraussetzung dar, diese Vorteile für den Anwender nutzbar zu machen.

Der in dieser Diplomarbeit erstellte Proxel-Simulator sollte alle diese Erweiterungen berücksichtigen und auf die Analyse abbilden können. (siehe dazu Kapitel 3)

2.3. Bisherige Verfahren zur Analyse von Simulationsmodellen

Für die Analyse zeitdiskreter Modelle existiert eine Reihe von Verfahren die, je nach Modellklasse unterschiedlich effizient sind. Das vom Umfang der Modellierungsmöglichkeiten mächtigste Verfahren ist die klassische ereignisbasierte Monte-Carlo-Simulation. Diese setzt auf die Verwendung von Zufallszahlen und macht somit die Ergebnisse selbst zu einer Zufallsvariablen. In der Simulation werden die Abläufe im Modell durch Ereignisroutinen beschrieben, die zu diskreten Zeitpunkten ausgeführt werden. Ein Simulationslauf stellt einen möglichen Pfad im Zustandsraum des System dar. Mehrere Läufe sind nötig, um repräsentative Ergebnisse zu generieren. Diese können zum Beispiel Statistiken wie Mittelwert, Standardabweichung oder Konfidenzintervalle sein.

Die Verwendung von Pseudo-Zufallszahlen bei der Simulation kann unter Umständen jedoch dazu führen, daß sehr hohe Laufzeiten notwendig sind, bevor statistisch zuverlässige Ergebnisse gewonnen werden können. Diese Notwendigkeit ergibt sich aus der stochastischen Natur des Verfahrens, bei dem nicht garantiert werden kann, daß bestimmte Zustandsübergänge durchgeführt werden. Daher wird für jeden Simulationslauf eine andere Folge von Zufallszahlen verwendet. Der für aussagekräftige Ergebnisse nötige Umfang einer solchen Stichprobe von Simulationsläufen ist von der Steifheit des Modells und der Güte der verwendeten Zufallszahlen und auch von den Anforderungen an die Genauigkeit der Ergebnisse des Anwenders abhängig.

2.3.1. Spezialfall Markov-Kette

Für den Spezialfall, daß im Modell lediglich exponentialverteilte Zufallsvariablen für Zustandsübergänge eingesetzt werden (der stochastische Prozess des Modells also als eine *Markov-Kette* ausgedrückt wird), existieren mehrere Analyseverfahren, die zumeist auf dem Lösen von linearen Gleichungssystemen basieren. Sie sind effizienter und genauer als Monte-Carlo-Simulation, da sie auf Zufallszahlen verzichten können. In einer Markov-Kette wird das gesamte System vollständig durch den aktuellen Zustand beschrieben. Die weitere Entwicklung des Modells hängt lediglich von der Gegenwart, nicht aber von seiner Vergangenheit ab. Beschreibt man die Zustandsübergänge im Modell durch die Übergangsraten λ_i (λ_i ist der Parameter der exponential-verteilten Zufallsvariable für den Zustandsübergang i), so erhält man eine zeitdiskrete Markov-Kette (Discrete Time Markov Chain = DTMC).

Die Übergangsraten λ_i der Zustandsübergänge werden durch die *Wahrscheinlichkeitsflußraten* beschrieben. Diese errechnen sich auf der *Ratenfunktion* $h(x)$ (auch *Hazard Rate Function* oder *Instantaneous Rate Function* genannt), welche das Verhältnis zwischen der Dichtefunktion $f(x)$ einer Verteilung und der dazugehörenden Überlebensfunktion $S(x) = 1 - F(x)$ ausdrückt.

$$h(x) = \frac{f(x)}{S(x)} = \frac{f(x)}{1 - F(x)} \tag{2.1}$$

Bei Markov-Ketten sind die Raten λ_i Konstanten – eine Eigenschaft, die sich aus der **Gedächtnislosigkeit** der Exponentialverteilung ergibt: Die Wahrscheinlichkeit, daß ein Ereignis innerhalb eines gegebenen Zeitintervalls in der Zukunft eintritt (bzw. nicht eintritt) ist nicht davon abhängig, wie lange bereits auf das Eintreten gewartet wurde. Mathematisch wird diese Eigenschaft durch die Instantaneous Rate Function $h_{exp}(x)$ der Exponentialverteilung ausgedrückt, die als einzige nicht von der bereits verstrichenen Zeit x abhängig ist.

$$h_{exp}(x) = \frac{f_{exp}(x)}{1 - F_{exp}(x)} = \frac{\lambda e^{-\lambda x}}{1 - (1 - e^{-\lambda x})} = \frac{\lambda e^{-\lambda x}}{e^{-\lambda x}} = \lambda \tag{2.2}$$

2.3.2. Analyse von allgemeinen Verteilungsfunktionen

Die Begrenzung auf eine einzige stochastische Verteilung bei Markov-Ketten ist eine starke Einschränkung der Mächtigkeit der Modellierung. Sollen Modelle mit allgemeinen Übergangsverteilungen untersucht werden, ist aber die durch $h(x)$ bestimmte Wahrscheinlichkeitsflußrate eines Zustandsübergangs nicht länger konstant, sondern von der bereits verstrichenen Zeit x (dem „Alter“ des aktivierten Zustandsübergangs) abhängig. Der Wert x kann zu jedem Zeitpunkt t der Analyse einen Wert zwischen 0 und t einnehmen, da eine Zustandsübergang nicht zwangsläufig schon bei $t = 0$ aktiviert sein muß.

$$\forall t \forall x : 0 \leq x \leq t \tag{2.3}$$

Um den Zustand eines solchen Modells zu beschreiben müssen zusätzlich zu den bisherigen Zustandsvariablen die Alterungs-Informationen der Transitionen (das Alter der jeweiligen Aktivierung) mit gespeichert werden, um die Flußraten bei der Analyse korrekt bestimmen zu können. Diese Alterungs-Informationen sind kontinuierliche Größen.

Eine direkte analytische Herangehensweise an derartige Modelle führt aufgrund der in der Ungleichung 2.3 dargestellten Beziehung dazu, daß der stochastische Prozess der Modelle durch ein System von partiellen Differentialgleichungen beschrieben wird. Dieses Verfahren basiert auf der Verwendung von sogenannten Hilfsvariablen, in denen die bisherigen Aktivierungsdauern von Zustandsübergängen angegeben werden. Für eine solche Analyse bestehen aber in der Praxis weitere Einschränkungen. So ist zum Beispiel maximal ein einziger aktivierter Zustandsübergang mit nicht-exponentialer Verteilungsfunktion für jeden Zeitpunkt der Analyse im Modell zugelassen. [Ger00]

Es ist möglich, die allgemeinen Verteilungsfunktionen durch Phasentyp-Verteilungen als eine Kombination von Exponential-Verteilungen auszudrücken und die Analyse damit auf das Lösen einer Markov-Kette zurückzuführen. Auch dabei sind Hilfsvariablen als zusätzliche Zustandsinformation notwendig, um die Phaseninformationen aufzunehmen. Ebenso wie für die Methode nach German existieren momentan keine allgemeinen Implementierungen. Das Verfahren ist allerdings Gegenstand der Forschung bei der DaimlerChrysler AG und steht somit im Vergleich zum Proxel-basierten Verfahren.

In neueren Veröffentlichungen werden weitere Vor- und Nachteile dieser Herangehensweisen verglichen und mit dem Proxel-Verfahren ein weiteres Denkmuster für die Analyse von Stochastischen Petri-Netzen vorgeschlagen. [Hor02]

2.4. Das Proxel-Verfahren

Das Proxel-basierte Verfahren entspricht nun dem Fluß der Wahrscheinlichkeitsmasse durch die möglichen Zustände des Systems, ähnlich dem transienten Lösen einer DTMC, wobei jedoch die Zugriffe auf die konstanten Raten λ_i durch Aufrufe einer allgemeinen Instantaneous Rate Function ersetzt werden. Der kontinuierliche stochastische Prozess des Petri-Netz-Modells wird dabei diskretisiert.

Für die nachfolgenden Erklärungen wird als Beispiel das in Abbildung 2.1 dargestellte Petri-Netz verwendet. Die Abbildung zeigt zusätzlich den Erreichbarkeitsgraphen des Netzes, dessen Initialzustand durch einen weiteren Kreis gekennzeichnet ist.

Der Name „Proxel“ ist eine Kurzform für „Probability Element“, analog zu „Pixel“ und „Picture Element“. Genau wie das Pixel für bestimmte Bildkoordinaten eine Farbe trägt, so enthält das Proxel für bestimmte Zustandskoordinaten (der *Systemzustand* S) eine *Wahrscheinlichkeit* p. Der Systemzustand setzt sich wiederum aus mehreren Elementen zusammen. Das erste Element ist die bereits erwähnte *Markierung* des Petri-Netzes M, ausgedrückt durch die Anzahl der Tokens in den einzelnen Stellen. Alle möglichen

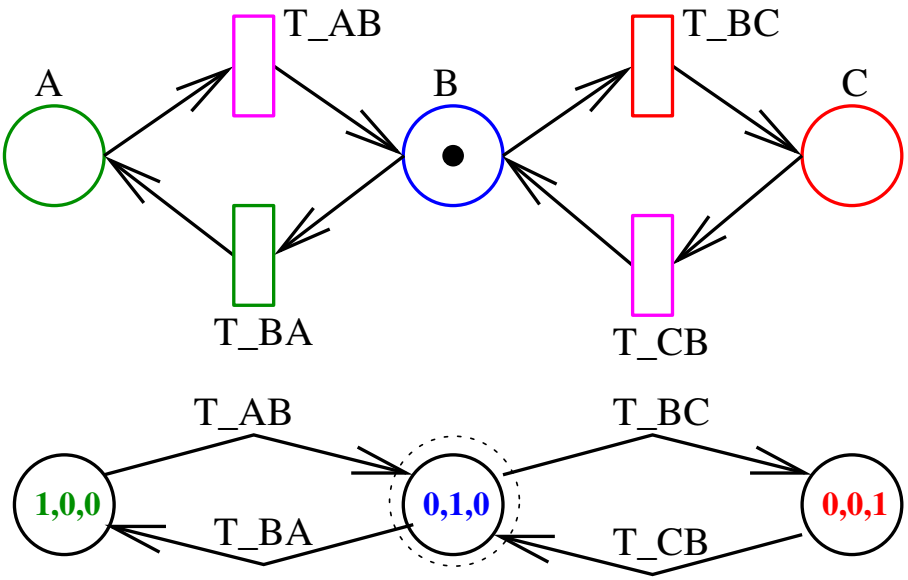

Abbildung 2.1.: Ein einfaches Petri-Netz und sein Erreichbarkeitsgraph

Markierungen eines Petri-Netzes bilden den Zustandraum des Netzes. Insofern hat der Raum aller möglichen Proxels mindestens die Kardinalität des Zustandsraumes.

Da die Wahrscheinlichkeitsflußrate einer Transition vom Alter ihrer Aktivierung abhängig ist, bestimmen diese *Alterungs-Informationen* den Systemzustand mit. Für jede Transition (genauer: für jeden Server) des Petri-Netzes ist die Zeit der Aktivierung Bestandteil der Zustandsinformation. Die Alterungs-Informationen aller Transitionen wird als *Alterungs-Vektor* A des Proxels bezeichnet. Von Relevanz sind die Alterungs-Informationen aller aktivierten Transitionen und aller deaktivierten Transitionen mit Verdrängungsstrategie „Race Age“. Von diesen Werten ist direkt die Wahrscheinlichkeitsflußrate und damit indirekt die Verteilung der Wahrscheinlichkeit im Modell über die Zeit abhängig. Die Elemente des Alterungs-Vektors sind beim Proxel-Verfahren die Hilfsvariablen.

Weiterhin ist die *Systemzeit* t Teil des Systemzustands, da verschiedene Markierung-Alterung-Kombinationen zu verschiedenen Zeitpunkten im Regelfall unterschiedliche Wahrscheinlichkeit haben. Sei p die Anzahl der Stellen im Petri-Netz, $m_0, \ldots, m_p$ die Anzahl der Tokens in den Stellen und $a_0, \ldots, a_c$ die Alterungs-Informaiton der relevanten Transitionen.

$$P = (p, S) \tag{2.4}$$
$$S = (M, A, t) \tag{2.5}$$
$$M = (m_0, m_1, \ldots, m_p), p = \text{Anzahl der Stellen} \tag{2.6}$$
$$A = (a_0, a_1, \ldots, a_c), c = \text{Anzahl aller relevanten Altersinformationen} \tag{2.7}$$

In der Literatur umfasst die Definition eines Proxel zusätzlich auch den Pfad aller Vor-

gängerproxel. Da jedoch Proxels, die nach der obigen Definition als äquivalent bezeichnet werden, ungeachtet ihrer Pfade zusammengefasst werden können, kann die Pfadinformation in der Praxis vernachlässigt werden. Das Proxel speichert somit lediglich die diskrete Markierung, den Zeitpunkt und im Alterungs-Vektor die Werte für die Aktivierungsdauern der einzelnen Transitionen.

Aus der in einem Proxel vorhanden Information kann unter Verwendung der Wahrscheinlichkeitsflußrate der aktivierten Zustandsübergänge bestimmt werden, welche Folgeproxel erreicht werden können. Durch den Algorithmus ergibt sich eine Baumstruktur, welche eine zeit-diskrete Markov-Kette darstellt. Die Wahrscheinlichkeit wird aus dem Initialzustand heraus auf diese Baumstruktur verteilt. Dazu werden die Wahrscheinlichkeiten der Folge-Proxel aus den Flußraten und der Wahrscheinlichkeit des jeweils aktuellen Proxels errechnet. Auf die gewonnenen Folge-Proxels wird der gleiche Algorithmus angewendet ([Hor02], [LMH03a]).

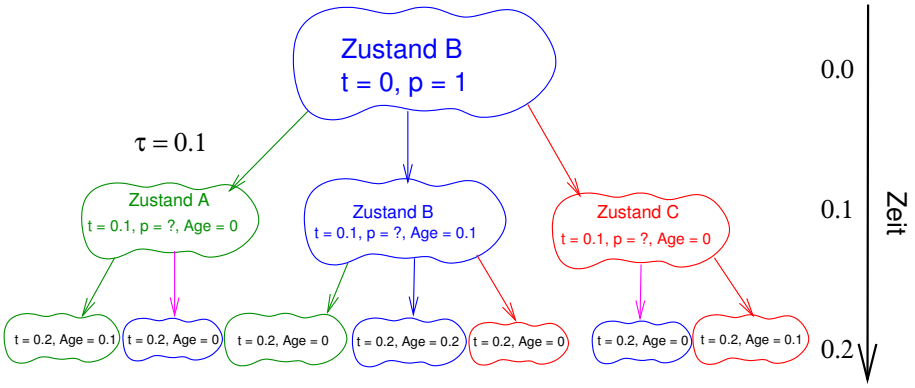

Abbildung 2.2.: Die ersten Schritte einer Proxel-basierten Simulation

Abbildung 2.2 stellt für das oben eingeführte Beispielnetz die ersten zwei Iterationen des Proxel-Algorithmus dar. Aus dem Initialzustand (ein Token in Stelle B, $t = 0$) werden alle möglichen Entwicklungen betrachtet:

- Die Transition T_{AB} feuert, anschliessend befindet sich ein Token in Stelle A.
- Die Transition T_{BC} feuert, anschliessend befindet sich ein Token in Stelle C.
- Es findet kein Zustandsübergang statt, die Markierung verändert sich nicht.

In jedem Fall schreitet die Systemzeit um einen diskreten Zeitschritt (mit der Zeitschrittweite τ voran), nach einer Iteration ist $t = \tau$. Beim Verbleib in der Ausgangsmarkierung muss das Alter der Aktivierung erhöht werden, andernfalls werden jeweils andere Zustandsübergänge mit dem Alter 0 neu aktiviert. Auf jedes erhaltene Proxel wird der gleiche Algorithmus erneut angewendet, so daß nach zwei Zeitschritten der Proxelbaum wie in Abbildung 2.2 dargestellt aussieht. Der Algorithmus terminiert, wenn eine vom Benutzer vorgegebene Endzeit überschritten wird.

2.4.1. Eigenschaften der Proxel-basierten Simulation

Der Proxel-Algorithmus ist eine Diskretisierung dieses kontinuierlichen stochastischen Prozesses unter Verwendung von Hilfsvariablen. τ ist ein Diskretisierungsparameter des Differentialquotienten der partiellen Differentialgleichungen. Der Algorithmus erstellt Proxel für positiv ganzzahlige Vielfache von τ.

Je nach Reihenfolge der Abarbeitung entspricht der Algorithmus einer Breitensuche in der durch die Proxel gebildeten Baumstruktur (es werden zuerst iterativ alle Proxels mit gemeinsamen t verarbeitet) oder einer Tiefensuche (der Algorithmus wird sofort rekursiv auf Folgeproxel angewendet). Die Breitensuche ist von Vorteil, da das System für einen Zeitpunkt $k\tau$ (mit $k \in \mathbb{N}$) komplett durch die Ebene k der Proxel-Baumstruktur beschrieben wird und die vorhergehenden Ebenen nicht mehr relevant sind.

Die durch den Algorithmus vorgenommene Approximation geht von der Annahme aus, daß pro Zeitschritt $k\tau \rightarrow (k+1)\tau$ maximal eine Zustandsänderung stattfindet oder sich der diskrete Zustand des Modells nicht ändert. Daher sollte die Zeitschrittweite τ so klein gewählt sein, daß die Wahrscheinlichkeit, daß es innerhalb von τ zu mehr als einer Zustandsänderung kommt, vernachlässigbar gering ist. Die praktische Verwendbarkeit wird durch die Wahl eines kleinen τ jedoch stark eingeschränkt, da die Größe der erzeugten Proxel-Baumstruktur – und damit die Laufzeit – bei kleiner werdendem τ exponentiell ansteigt. [LMH03a] In Kapitel 4 wird eine Methode vorgestellt, die Anzahl der zu bearbeitenden Proxels zu verringern, indem für die einzelnen Zustandsübergänge unterschiedlich große Zeitschrittweiten verwendet werden.

Ein entscheidener Vorteil des Verfahrens ist, daß der Fehler der Approximation ein Fehler erster Ordnung in Bezug auf τ ist. Dadurch ist es möglich, zwei Proxel-basierte Simulationen mit den Zeitschrittweiten τ_1 und τ_2 ($\tau_1 \neq \tau_2$) durchzuführen, und anhand ihrer Ergebnisse ein drittes Ergebnis für ein theoretisches $\tau_0 = 0$ zu extrapolieren.

Eine Möglichkeit, das exponentielle Wachstum des Baums zu beschränken, besteht darin, einen Schwellwert ε für die Wahrscheinlichkeit neuer Proxel einzuführen. Sei P_{nach} ein Folge-Proxel, das aus P erzeugt wurde. Dann gilt in jedem Fall $P.p_{nach} \leq P.p$. Sollte zusätzlich auch noch $P.p_{nach} < \varepsilon$ gelten, so ist P_{nach} zu verwerfen. Da hiermit der Zustandsraum des Verfahrens stark gestutzt wird, die verworfenen Proxel-Informationen aber nur geringen Anteil an der Lösung haben, können hiermit beträchtliche Einsparungen bei den für den Algorithmus notwendigen Ressourcen verzeichnet werden.

Das Verfahren arbeitet auf dem Erreichbarkeitsgraphen des Petri-Netzes und setzt somit voraus, daß dieser bereits erstellt wurde. Dies stellt eine Einschränkung dar, da die Methode folglich kein Modell mit unendlichem Zustandsraum analysieren kann.

Zeitlose Zustände eines Petri-Netzes stellen für die Methode hingegen keine Einschränkung dar. Die Wahrscheinlichkeit, mit der ein solcher Zustand nach dem Feuern einer zeitbehafteten Transition erreicht wird, wird entsprechend den Gewichtungen der aktivierten zeitlosen Transitionen auf die Folgezustände verteilt. Sollten die Folgezustände ebenfalls zeitlos sein, werden diese analog behandelt. [LMH03b]

Algorithmus 2.1 : Proxel-Algorithmus für Petri-Netze

Eingabe : Anfangsproxel P_0 mit $P_0.t = 0,\ P_0.A = \emptyset,\ P_0.M = M_0$
Eingabe : Zeitschrittweite τ
Eingabe : Endzeit der Simulation T_{max}
Eingabe : Schwellwert ε
Daten : $\mathsf{flussrate}_T$ Beinhaltet einen Wert der Instantaneous Rate Function $h(x)$, welche zur Verteilung der Transition T gehört
Daten : $\mathsf{w'keit}_T$ Folgewahrscheinlichkeit, die sich aus dem aktuellen Proxel durch das Feuern von T ergibt
Daten : wsumme Die Summe der Folgewahrscheinlichkeiten, die sich aus dem aktuellen Proxel ergeben

```
1  step ← 0
2  solange step × τ ≤ T_max tue
3      für jedes Proxel P mit P.t = step tue
4          wsumme ← 0
5          für jede in P aktivierte Transition T tue
6              alter_{P,T} ← in P.A gespeicherte Dauer der Aktivierung von T
7              flussrate_T ← Wahrscheinlichkeitsflußrate von T für alter_{P,T}
8              w'keit_T ← flussrate_T × τ × P.p
9              wenn w'keit_T ≥ ε dann
10                 wsumme ← wsumme + w'keit_T
11         wenn wsumme < 1 dann
12             wenn 1 − wsumme ≥ ε dann
13                 addProxel(1 − wsumme, P.M, altern(P.M, P.A, ∅), P.t + 1)
14             für jede in P aktivierte Transition T tue
15                 wenn w'keit_T ≥ ε dann
16                     addProxel(w'keit_T, folge(P.M, T), altern(P.M, P.A, T), P.t + 1)
17         sonst
18             für jede in P aktivierte Transition T tue
19                 wenn w'keit_T ≥ ε dann
20                     addProxel(w'keit_T / wsumme, folge(P.M, T), altern(P.M, P.A, T), P.t + 1)
21     step ← step +1
```

2.4.2. Allgemeine Proxel-basierte Simulation von Petri-Netzen

Algorithmus 3.1 (Seite 25) berechnet den Proxel-Baum für ein gegebenes Petri-Netz. Das Verfahren basiert auf dem in [Hor02] beschriebenen Algorithmus. Im hier angegebenen Algorithmus wird zusätzlich eine Normierung der Übergangswahrscheinlichkeiten vorgenommen, falls deren Summe größer als 1.0 ist. Dieser Fall kann bei großen Zeitschrittweiten τ eintreten. Der Algorithmus verwendet die Funktionen

- $\mathsf{addProxel}(p, M, A, t)$: Fügt Wahrscheinlichkeit p für das Proxel mit der Markierung M, dem Alterungs-Vektor A zur Zeit t in den Proxel-Baum ein. Falls bisher kein Proxel mit diesen Koordinaten existiert, wird es erstellt. Ansonsten können die Wahrscheinlichkeiten addiert werden. Ist M ein zeitloser Zustand, wird die Wahrscheinlichkeit entsprechend der Gewichtung aller aktivierten zeitlosen Transitionen rekursiv auf die Folgezustände verteilt.
- $\mathsf{folge}(M, T)$: Liefert die diskrete Markierung zurück, die durch das Feuern von T in der Markierung M erreicht wird. Es gilt $\mathsf{folge}(M, \emptyset) = M$.
- $\mathsf{altern}(M, A, T)$: Liefert Hilfsvariablen als Alterungs-Vektor A^* zurück, der basierend auf A entsteht, wenn aus der Markierung M die Transition T feuert.

Von besonderer Bedeutung ist dabei die Berechnung der Hilfsvariablen in der Funktion altern (Seite 15). Sie erstellt aufbauend auf einem bestehenden Vektor einen Alterungs-Vektor für ein neues Proxel. Der Alterungs-Vektor des initialen Proxels zu Beginn der Analyse enthält für alle Transitionen das Alter 0.

Funktion `altern`(M, A, T): Bestimmt einen neuen Alterungs-Vektor

Eingabe : Ausgangsmarkierung M
Eingabe : Ursprünglicher Alterungs-Vektor A
Eingabe : Gefeuerte Transition T (kann $\emptyset$ sein)
Daten : A^* Der neue Alterungs-Vektor
Daten : a_E Die Alterungs-Information für die Transition E

```
1  A* ← ∅
2  für jede Transition E tue
3      wenn E = T dann
4          a*_E ← 0
5      sonst
6          wenn E in folge(M, T) aktiviert dann
7              a*_E ← a_E + 1
8          sonst
9              wenn E vom Typ Race Age dann
10                 a*_E ← a_E
11             sonst
12                 a*_E ← 0
13     Füge a*_E in A* ein
14 return A*
```

2.4.3. Einordnung und Bewertung des Proxel-basierten Verfahrens

Das Proxel-Verfahren ist ein Approximationsverfahren, mit dem der Wahrscheinlichkeitsfluß in einem stochastischen Prozess geschätzt werden kann. Die Methode verwendet Hilfsvariablen zur Speicherung zustandsrelevanter Informationen. Die Güte der Abschätzung ist abhängig von der Größe des Zeitschritts, der beim Verfahren verwendet wird. Der durch die Approximation entstehende Fehler ist erster Ordnung. Diese Eigenschaft wirkt sich positiv bei der Extrapolation von genaueren Ergebnissen aus.

Das Proxelverfahren beseitigt die Notwendigkeit nach Pseudozufälligkeit, wie sie bei der diskreten Simulation verwendet wird. Damit entfällt auch der Bedarf für eine hohe Anzahl von Replikationen zur Gewinnung statistisch verwendbarer Ergebnisse bei steifen Modellen. Hierdurch kann die Laufzeit der Analyse verringert werden.

Diesen Vorteilen steht ein exponentiell wachsender Bedarf nach Speicher und Rechenzeit gegenüber. Die Speicherung der Wahrscheinlichkeit eines Modells in vielen Proxels benötigt mehr Speicher als eine einmalige Beschreibung des Systemzustandes durch die Markierung der Stellen und Feuerzeiten der Transitionen bei der diskreten Simulation.

2.4.4. Eine existierende Anwendung des Proxel-Verfahrens

Ein Petri-Netz, welches bereits Proxel-basiert analysiert wurde, ist ein Garantie-Modell der DaimlerChrysler AG, wie er in [LMH04] beschrieben ist. Abbildung 2.3 zeigt eine

Nachbildung des Modells.

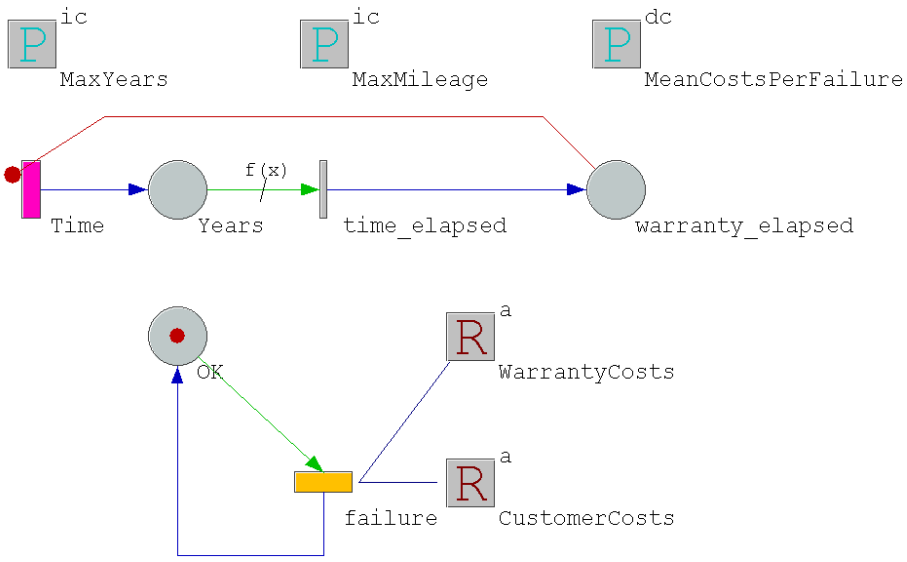

Abbildung 2.3.: Garantiemodell

Das Petri-Netz modelliert den Ausfalls eines Fahrzeugkomponente (Transition `failure`) und die mit der Wiederinstandsetzung anfallenden Kosten. Die Kosten entstehen in Abhängigkeit der Markierung der Stelle `warranty_elapsed` in der Kostenfunktion `WarrantyCosts` bzw. `CustomerCosts` durch das Feuern von `failure`. Die Stelle `warranty_elapsed` wird belegt, wenn in `Years` die im Parameter `MaxYears` enthaltene Markenanzahl erreicht wird. Eine Marke entspricht dem Ablauf eines Jahres. Der Verschleiß des zu untersuchenden Objekts wird dabei über die Transition `Time` beschrieben, welche die Laufleistung des Objekts über jeweils ein Jahr abbildet.

Dieses Modell ist von großer Steifheit, insbesondere wenn die Verteilungen von `failure` und `time` *heavy-tailed* sind. Monte-Carlo-Verfahren erfordern für die Analyse eine hohe Anzahl von Replikationen, [LMH04] nennt Laufzeiten von 20 bis 30 Stunden.

In der selben Quelle wird eine Implementierung der Proxel-basierten Simulation für dieses Modell beschrieben. Der vorgestellte Proxel-Simulator ist in der Lage, Ergebnisse mit vergleichbarer Genauigkeit in wenigen Minuten zu liefern. Er macht sich dazu die in 2.4 beschriebene Eigenschaft zu nutze, um aus mehreren Ergebnissen mittels der Lagrangeschen Formel ein Ergebnis für $\tau = 0$ zu extrapolieren.

Der Erreichbarkeitsgraph des Modells und die damit verbundene Programmlogik sind ein fester Bestandteil des Algorithmus. Dadurch wird auf der einen Seite ein Maximum an Geschwindigkeit bei der Analyse erreicht. Auf der anderen Seite besitzt die Implementierung keine Allgemeingültigkeit. Wird das Modell über einzelne Zahlenwerte für die Parameter der Verteilungsfunktionen hinaus verändert, so sind Anpassungen an der Logik im Programm selbst notwendig. Daraus entsteht die Voraussetzung, daß der Anwender Kenntnisse über Programmierung und die Abläufen der Proxel-basierten Simulation besitzen muß, um Änderungen vornehmen zu können. Für die Verwendung

durch Personal ohne eine entsprechende Ausbildung ist die Implementierung folglich nur äußerst beschränkt verwendbar.

Um die Vorteile der Proxel-basierten Simulation einer breiteren Masse von Anwendern zugänglich zu machen, ist es sinnvoll, einen Simulator zu schaffen, der in der Lage ist, als Eingabe ein allgemeines Modell (zum Beispiel in Form eines Stochastischen Petri-Netzes) entgegenzunehmen, um es automatisiert und für den Anwender weitestgehend transparent zu analysieren. Der zu erwartetende Verlust an Effizienz gegenüber Speziallösungen wie der oben beschriebenen wird durch die Flexibilität der Anwendung gerechtfertigt.

Ein Simulator für die Analyse allgemeiner Modelle existieren in Form von bisher unveröffentlichen Prototypen an der Universität Magdeburg. Diese verlangen als Eingabe einen im Voraus erstellten Erreichbarkeitsgraphen und sind somit ebenfalls nicht in der Lage, Modelle mit unendlichem Erreichbarkeitsgraphen zu analysieren.

Im nächsten Kapitel wird eine Implementierung des Proxel-basierten Simulationsverfahrens vorgestellt, welche allgemeine Stochastische Petri-Netze als Eingabe entgegennimmt und analysiert. Die Methode verzichtet auf eine Vorausberechnung des Zustandsraums und vergrößert die Mächtigkeit des Verfahrens um verschiedene Erweiterungen von Stochastischen Petri-Netzen.

3. Implementierung eines Proxel-Algorithmus in Expect

3.1. Rahmenbedingungen

Das Vorhaben, das bestehende Software-Tool Expect zu erweitern, war mit einigen Rahmenbedingungen verbunden, auf die im folgenden eingegangen werden soll. So war zum einen durch den Industriepartner vorgegeben, daß als Programmiersprache Java zum Einsatz kam. Zum anderen bestand die Anforderung, daß die bestehenden Datenstrukturen zur Darstellung der Petri-Netz-Modelle verwendet werden, ohne sie zu verändern.

Als äußerst vorteilhaft erwies sich, daß in einem schon vorhandenen Erreichbarkeitsgraph-Erzeuger des Tools bereits Algorithmen existierten, um für eine spezielle diskrete Markierung den Aktivierungsgrad der Transitionen und die durch das Feuern von Transitionen erreichbaren Folgezustände zu bestimmen. Diese Algorithmen konnten vollständig wiederverwendet werden, was nicht zuletzt auch für den Industriepartner von Interesse war, da sich der Wartungsaufwand für solche Algorithmen somit auf eine einzelne Implementierung beschränkt.

Die schwerwiegendste Rahmenbedingung ergab sich aus der Aufgabenstellung, den gesamten möglichen Modellierungsumfang des Tools (siehe 2.2) auf die Proxelsimulation abzubilden. Analog zur gleichen Mächtigkeit der Eingabe sollte auch die Ausgabe, d.h. die generierten statistischen Ergebnisse, die gleichen Werte umfassen. Die Tabelle 3.1 stellt dar, welche Ergebnisse für die einzelnen Elementklassen von Petri-Netzen durch Expect berechnet werden.

Elementklasse	**Erzeugte Ergebnisstatistiken**
Stellen	Erwartungswert für die Markenanzahl Wahrscheinlichkeit für die Leere
Transitionen	Erwartungswert für die Feuerrate Wahrscheinlichkeit für die Aktivierung
Rewards	Erwartungswert für die Kostenfunktion

Tabelle 3.1.: Die erzeugten Statistiken der Petri-Netz-Analyse

Der Industriepartner stellte Modelle (unter anderem das bereits in Abschnitt 2.4.4 vorgestellte Garantie-Modell) mitsamt Analyse-Ergebnissen zur Verfügung, um die Validierung und Verifikation der Implementierung zu unterstützen.

3.2. Erweiterungen des Proxel-Algorithmus

Die Proxel-basierte Simulation agiert auf dem Erreichbarkeitsgraphen eines Petri-Netzes. In der Literatur wird empfohlen, den Erreichbarkeitsgraphen vor der eigentlichen Simulation zu erzeugen. Dies führt zu der Annahme, daß der Erreichbarkeitsgraph endlich ist. [LMH03b]

Diese Prämisse kann aufgehoben werden, indem auf die explizite Erzeugung des Erreichbarkeitsgraphen verzichtet wird und stattdessen zur Laufzeit der Simulation für jedes zu untersuchende Proxel die erreichbaren Folgezustände bestimmt werden. Der damit einhergehende Performanceverlust wiegt weniger stark, wenn in Betracht gezogen wird, daß aufgrund der Darstellung von Petri-Netz-Eigenschaften durch Funktionen der Erreichbarkeitsgraph ohnehin nicht nur von der diskreten Markierung des Petri-Netzes abhängig ist. Vielmehr besteht auch eine Abhängigkeit von Variablen wie der Systemuhr der Simulation. Die Folge davon ist, daß das Feuern einer bestimmten Transition bei einer ebenfalls bestimmten diskreten Markierung unterschiedliche Folgezustände haben kann. Dies wiederum bedeutet, daß eine vorangestellte Erzeugung des Erreichbarkeitsgraphen ohnehin keine korrekten Ergebnisse liefert, da sie in Unkenntnis der Systemuhr erfolgt.

3.2.1. Speicherung des Alters einer Transition

Bei einer expliziten Bestimmung der Menge der aktivierten Transitionen (wie oben beschrieben) ist es nicht nötig, den Alterungs-Wert 0 im Alterungs-Vektor zu hinterlegen. Der Alterungs-Vektor kann somit nur Werte $k \cdot \tau,\ k \in \mathbb{N} \setminus 0$ enthalten. Es werden Werte für alle Transitionen gespeichert, welche mindestens mit einem Alter $a \geq \tau$ aktiviert sind oder als Race-Age-Transitionen aus ihrer Aktivierung verdrängt wurden.

Der Alterungs-Vektor der Proxels speichert die Faktoren k als positive Ganzzahlen. Der Vektor hat theoretisch soviele Elemente, wie es zeitbehaftete Transitionen im Petri-Netz gibt. Da nicht immer alle zeitbehafteten Transitionen zeitgleich aktiviert sind, ist der Vektor in den meisten Fällen dünnbesetzt. In der Literatur findet sich die Aussage, daß die Hilfsvariablen im Alterungs-Vektor in unterschiedlichen Markierungen auch unterschiedlichen Transitionen zugeordnet sein können. Ein allgemeiner Simulator muß dann automatisch die kleinstmögliche Anzahl und eine optimale Abbildung von Hilfsvariablen auf die Transitionen vornehmen. [Hor02]

Im Gegensatz dazu läßt sich der Alterungs-Vektor auch als dünnbesetzter Vektor wie in [Knu73] beschrieben speichern und verwenden (vgl. Abbildung 3.1). Als Index des Vektorelements wird dafür die eindeutige ID der Transition verwendet, wie sie durch den Simulator des Industriepartners vorgegeben ist. Der Vektor speichert, wie gehabt, positive Ganzzahlen, aufsteigend sortiert nach der ID der dazugehörenden Transitionen. Damit ist keine vorherige Abbildung der Variablen auf die Transitionen in Abhängigkeit der Markierung mehr notwendig. Jede Transition kann im Alterungs-Vektor höchstens einmal vorkommen.

Es sei $\#_t$ die Anzahl der zeitbehafteten Transitionen im Petri-Netz. Ein Alterungs-Wert kann mit einem Aufwand von $O(\#_t)$ im Alterungs-Vektor gefunden werden. Die

Anzahl der besetzten Elemente im Vektor wird in vielen Fällen deutlich kleiner als $\#_t$ sein, so daß der lineare Aufwand der Suche tatsächlich den *worst case* darstellt.

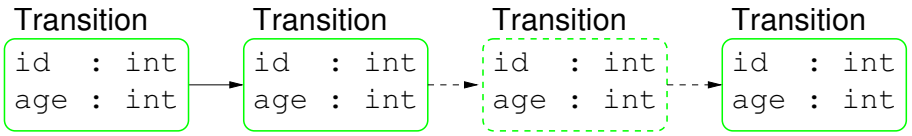

Abbildung 3.1.: Beispiel eines Alterungs-Vektors

Für Transitionen, deren Feuerzeiten einer Exponentialverteilung unterliegen, ist keine Speicherung der Alterungs-Information notwendig, da die Instantaneous Rate Function einer Exponentialverteilung nicht in Abhängigkeit des Alters a steht. Dadurch läßt sich die theoretische Größe des Alterungs-Vektors in der Praxis verringern.

Bei einem GSPN, bei dem allen Transitionen im Petri-Netz Exponentialverteilungen zugeordnet sind, ist die Datenstruktur zur Speicherung der Alters-Informationen folglich leer. Das Proxel-Verfahrenm auf einem GSPN angewendet, entspricht einem stark aufgeblähten Löser für zeit-diskrete Markov-Ketten.

3.2.2. Parameterfunktionen

Die funktionale Abhängigkeit von Parameterwerten während der Proxel-basierten Petri-Netz-Analyse schafft weitere Randbedingungen. So muß zum Beispiel bei der Berechnung der Menge der aktivierten Transitionen und dem Folgezustand im Falle ihres Feuerns sichergestellt werden, daß das Petri-Netz die entsprechende Ausgangsmarkierung und die Systemuhr des Simulators die korrekte Zeit inne hat.

3.2.3. Zeitskalierbarkeit der Transitionen

Die Skalierung einer bereits bestimmten Feuerzeit geschieht ohne Berücksichtigung der Zufallsverteilung, welche der Transition zugeordnet ist. Bei allen Transitionen, für die eine Skalierung aktiviert ist, werden nach jedem Feuern einer beliebigen Transition im Petri-Netz die Feuerzeiten mit einem Faktor gewichtet, der aus einer Funktion der jeweiligen Verteilung bestimmt wird. Diese Gewichtung erfolgt bei der Monte-Carlo-Simulation nach dem Erzeugen einer Zufallsstichprobe der Verteilungsfunktion. Das Alter der Aktivierung ändert sich nicht, lediglich die verbleibende Zeit wird verändert.

Beim Proxel-Verfahren kann die Skalierung folglich nicht angewendet werden, da sie keinen Einfluß auf die Instantaneous Rate Function nimmt. Inwiefern dies die Ergebnisse verfälscht, konnte zum gegenwärtigen Zeitpunkt dieser Arbeit nicht untersucht werden, da noch keine konkreten Implementierungen für Verteilungsskalierungen existierten.

3.2.4. Race Reset und Race Repeat

Die Verdrängungsstrategien Race Reset und Race Repeat lassen sich vollständig auf die Proxel-basierte Simulation abbilden. Für die Verwendung von Race Repeat sind keinerlei Änderungen am bisherigen Algorithmus notwendig. Diese Strategie kann durch den

Algorithmus genau wie Race Repeat gehandhabt werden. Eine derartige Transition wird nach einer Verdrängung mit anschließender Reaktivierung – ebenso wie eine Race Enable Transition – erneut mit einem Alter von 0 beginnen. Daß bei der Monte-Carlo-Simulation wieder die eingangs berechnete Feuerzeit verwendet wird, ist beim Proxel-Verfahren nicht von Belang, da keine Stichproben einer Zufallsverteilung verwendet werden, sondern deterministische Werte einer Funktion.

Für Transitionen mit der Strategie Race Reset sind Anpassungen am Algorithmus notwendig. Solche Transitionen verlieren ihr Alter nach dem Feuern jeder beliebigen Transition. Die Transition kann nur altern, wenn keine Änderung am diskreten Zustand des Petri-Netzes eintritt. Für die Proxel-basierten Verfahren bedeutet dies, daß solche Transitionen nur in der Berechnung des Alterungs-Vektors für ein Verbleiben des aktuellen Zustands berücksichtigt werden müssen. Für jeden anderen Zustandsübergang ist eine Transition mit Race Reset wie eine neu aktivierte Race Enable mit Alter 0 zu behandeln. Da wie in Unterabschnitt 3.2.1 beschrieben ein solches Alter nicht im Alterungs-Vektor gespeichert wird, verringert sich die Größe des Vektors durch Race-Reset-Transitionen noch weiter.

3.3. Multi-Server-Eigenschaften von Transitionen

Die Multiplizität einer Transition erfordert in hohem Umfang Änderungen am Proxel-Algorithmus. Jeder aktivierte Server einer Transition ist wie eine eigenständige Transition mit denselben Vor- und Nachbedingungen (d.h. denselben Kanten, derselben Guardfunktionen, derselben Verteilung etc.) zu behandeln.

Insofern ist für jeden aktivierten Server ein Alterungs-Wert zu speichern. Darüberhinaus können Race-Age-Transitionen (auch parallel zu aktivierten Servern) aus der Aktivierung verdrängte Server besitzen. Diese Server werden im weiteren Verlauf als **pausierte Server** bezeichnet. Auch für solche pausierten Server sind Alterungs-Werte zu speichern. Server einer Transition, die gleichen Alters sind, haben gleiche Flußraten. Dies ergibt sich daraus, daß alle Server dieselbe Instantaneous Rate Function mit demselben Alterungs-Wert aufrufen, um die eigene Flußrate zu bestimmen.

Die Übergangswahrscheinlichkeit zwischen zwei Proxels ergibt sich aus der gesamten Flußrate der Transition, durch deren Feuern der Übergang vollzogen wird. Die Wahrscheinlichkeit des folgenden Proxels ist das Produkt aus der Flußrate und der Wahrscheinlichkeit des Ausgangs-Proxels (siehe Algorithmus 3.1, Zeile 8). Es sei T die Transition, deren gesamte Flußrate bestimmt werden soll, n die Anzahl der aktivierten Server von T, A der aktuell gültige Alterungs-Vektor. Die Gesamtflußrate der Transition ist dann die Summe der Flußraten aller aktivierten Server:

$$rate_T(A) = \sum_{i=1}^{n} h_T(a_{T,i}) \tag{3.1}$$

Alle Server einer Transition T überführen das Modell beim Feuern aus dem Zustand M_0 in denselben Folgezustand $\mathsf{folge}(M_0, T)$. Die Annahme, daß pro Zeitschritt beim

Proxel-Verfahren maximal ein Zustandsübergang stattfindet, darf durch die Multiplizität einer Transition nicht aufgehoben werden. Daher kann pro Zeitschritt maximal ein Server im Modell feuern.

3.3.1. Alterungs-Matrix für die Server und Transitionen

Um die Server-Multiplizität auf die Proxel-basierte Simulation abbilden zu können, muß zunächst die Definition des Alterungs-Vektors eines Proxels erweitert werden. Für eine einzelne Transition muß statt einem einzelnem Wert nunmehr für jeden Server ein Wert gespeichert werden. Bei Race-Age-Transitionen muß zusätzlich unterschieden werden, ob ein Server jeweils aktiviert ist oder pausiert.

Bei der Speicherung die Server einer Transition zu indizieren, ist nicht die günstigste Methode, da die Anzahl der möglichen Server (theoretisch) unendlich sein kann und Server beliebig verdrängt und reaktiviert werden können. Als vorteilhafter erweist es sich, die Server entsprechend ihres Alters zu indizieren. Da mehrere Server vom gleichen Alter sein können, ist die mit dem Alterswert indizierte Information die Anzahl der Server für dieses Alter.

Eine Möglichkeit, die Anforderungen umzusetzen, ist eine Datenstruktur, wie sie in 3.2 dargestellt ist. Dabei werden die Transitionen wie in 3.2.1 beschrieben im Speicher gehalten. Statt auf einen ganzzahligen Alterungs-Wert verweist eine Transition ihrerseits wieder auf zwei dünnbesetzte Vektoren `aktiv` und `pausiert`. Diese enthalten als Elemente Server-Anzahlen, die entsprechend ihrem Alter indiziert sind. Im folgenden wird diese Datenstruktur als die *Alterungs-Matrix* bezeichnet.

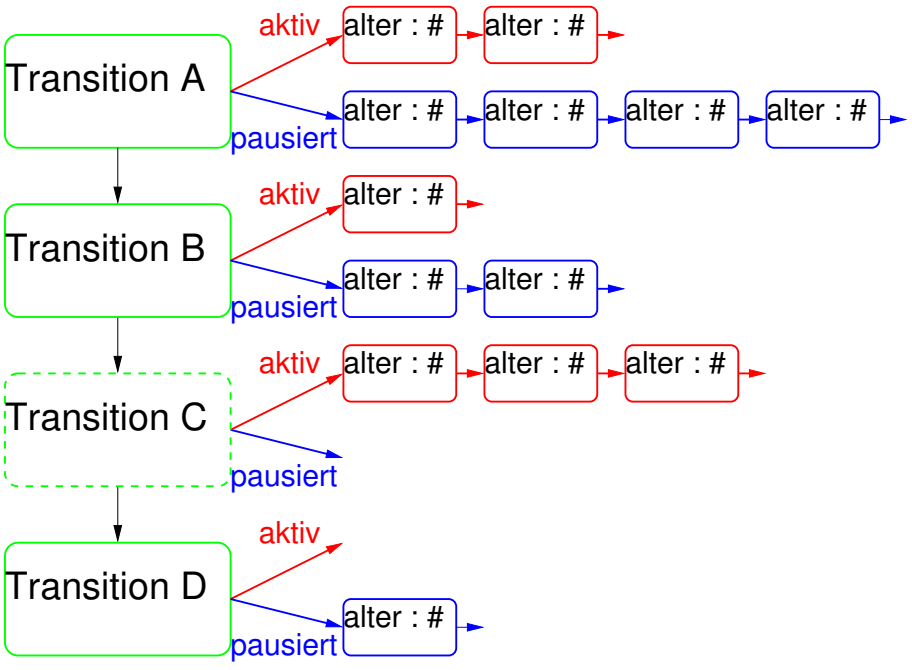

Abbildung 3.2.: Struktur der Alterungs-Informationen

Die Alterungs-Matrix erweist sich für die untersuchten Modelle als hinreichend effizient, um bei der Berechnung auf die Alterungs-Informationen zugreifen zu können. Von Vorteil ist die dynamische Größe der Datenstruktur. Sie ist für jedes Proxel und zu jedem Zeitpunkt der Analyse so groß wie notwendig und so klein wie möglich. Der lineare Aufbau ist jedoch für zukünftige Anwendungen nicht zwangsläufig hinreichend effizient. Weitere Forschung ist notwendig, um geeignetere Strukturen ausfindig zu machen.

3.3.2. Verdrängung und Reaktivierung von Servern

Durch das Feuern einer Transition ändert sich in der Regel der diskrete Zustand im Modell und somit auch der Aktivierungsgrad der Transitionen. Es lassen sich im wesentlichen drei mögliche Entwicklungen des Aktivierungsgrades feststellen:

1. Der Aktivierungsgrad bleibt unverändert. Das Alter aller aktivierten Server erhöht sich dementsprechend um τ. Alle pausierten Server behalten ihr Alter bei.

2. Der Aktivierungsgrad nimmt ab. Die komplette Verdrängung aller Server (Aktivierungsgrad = 0) ist ein Spezialfall davon. Bei Race-Age-Transitionen wird das Alter der verdrängten Server weiterhin gespeichert. Das Alter der aktivierten Server erhöht sich um τ.

3. Der Aktivierungsgrad steigt an. Neu aktivierte Server beginnen mit einem Alter 0. Bei Race-Age-Transitionen werden bevorzugt ehemals verdrängte Server reaktiviert und dann erst bei weiterem Bedarf zustätzlich Server mit dem Alter 0 aktiviert.

Für die Transition, deren Server soeben gefeuert hat, ist der alte Aktivierungsgrad in jedem der obigen Fälle künstlich um Eins zu verringern. Damit wird sichergestellt, daß dieser Server erneut zu altern beginnt, sollte der reale Aktivierungsgrad unverändert bleiben.

Werden Server im wie zweiten Fall verdrängt, so ist die Wahrscheinlichkeit der Verdrängung für jeden einzelnen Server gleich. Bei der Monte-Carlo-Simulation wird durch zufällige Ausfall entschieden, welche Server deaktiviert werden. Durch die Wiederholung der Simulation in mehreren Replikationen werden verschiedene Möglichkeiten der Verdrängung implizit behandelt. Bei der Proxel-basierten Analyse müssen alle möglichen Verdrängungen explizit betrachtet werden. Werden von n aktiven Servern k ($0 \leq k < n$) verdrängt, so gibt es $\binom{n}{k}$ verschiedene Möglichkeiten für eine k-elementige Untermenge der n Server. Ein kombinatorischer Algorithmus, diese $\binom{n}{k}$ Möglichkeiten iterativ zu bestimmen, findet sich in [Knu04]. Bei der Anwendung eines derartigen Algorithmus ergibt sich für jede Transition T_i eine Menge S_i von möglichen Alterungs-Kombinationen der Server dieser Transition. Diese Menge kann auch leer sein, falls keine aktiven oder pausierten Server für eine Transition exisitieren.

Es sei t die Anzahl der Transitionen, die in den Alterungsmatrizen berücksichtigt werden müssen. Um nun die Menge $\mathcal{A}$ aller möglichen Alterungs-Matrizen für das Folge-Proxel zu generieren, müssen unter Zuhilfenahme des Kartesischen Produkts aus diesen Mengen S_i alle möglichen t-Tupel gebildet werden.

$$A \in \mathcal{A} \leftarrow \prod_{i=1}^{t} S_i = S_1 \times \cdots \times S_t := \{(s_1, ..., s_t) | s_i \in S_i\} \tag{3.2}$$

$$|\mathcal{A}| = \prod_{i=1}^{t} |S_i| = \prod_{i=1}^{t} \binom{n_i}{k_i} \tag{3.3}$$

Um nun alle möglichen Folge-Proxel für das Feuern eines Servers zu gewinnen, wird die erreichte diskrete Markierung mit allen möglichen Alterungs-Matrizen aus $\mathcal{A}$ verbunden. Insofern ist die Anzahl der Matrizen gleich der Anzahl der Folge-Proxel. Die Wahrscheinlichkeit für jedes dieser Folge-Proxel ergibt sich aus der Wahrscheinlichkeit des Ausgangs-Proxel, multipliziert mit der Flußrate des Servers h_s dividiert durch die Anzahl der entstehenden Proxel.

$$P'.p = \frac{P.p \cdot h_s(a_s)}{|\mathcal{A}|} \tag{3.4}$$

Existieren für eine Transition mehrere Server gleichen Alters, so erzeugen sie beim Feuern jeweils dieselbe Menge an möglichen Alterungs-Informationen. Dies bedeutet, daß sie auch die gleiche Menge an Proxels erzeugen. Da gleiche Proxels zusammengefaßt werden können, indem ihre Wahrscheinlichkeiten kumuliert werden, lassen sich Server gleichen Alters in einem Durchlauf behandeln. Die Wahrscheinlichkeit der neuen Proxels wird dann entsprechend der Anzahl der äquivalenten Server ($\#_{srv}$) skaliert:

$$P'.p = \frac{P.p \cdot h_s(a_s)}{|\mathcal{A}|} \cdot \#_{srv} \tag{3.5}$$

Analog ist zu verfahren, wenn für eine Race-Age-Transition weniger pausierte Server reaktiviert werden als in der Alterungs-Matrix für diese Transition vorhanden sind. Auch hier besteht für jeden Server eine gleichgroße Wahrscheinlichkeit, daß er reaktiviert wird.

Der nach (3.3) berechnete Wert für die Anzahl der möglichen Alterungs-Matrizen, die durch Verdrängungen entstehen, hat ein massiv exponentielles Wachstum. Theoretisch lassen sich Modelle mit beliebig oft multiplizierten Transitionen auf die Proxel-basierte Simulation abbilden. Für die praktische Verwendbarkeit stellt der hohe Aufwand jedoch eine starke Beeinträchtigung dar, weil die Analyse unter Umständen äußerst hohe Laufzeiten und ein beträchtliche Menge an Arbeitsspeicher erfordert.

3.4. Algorithmus

Im folgenden wird der Algorithmus und einige wichtige Funktionen für die Proxel-basierte Simulation der erweiterten Stochastischen Petri-Netze in Pseudo-Code-Notation angegeben. Die Eingabe und die verwendeten Variablen entsprechen, soweit nicht anders gekennzeichnet, denen aus Abschnitt 2.4.2. Der Übersichtlichkeit halber wurde darauf

verzichtet, die Berechnung der Summe der Übergangswahrscheinlichkeiten und die Normierung explizit aufzuführen. Die Notwendigkeit eines solchen Verfahrens besteht jedoch nach wie vor.

Algorithmus 3.1 : Proxel-Algorithmus für erweiterte Stochastische Petri-Netze

```
1  step ← 0
2  solange step × τ ≤ T_max tue
3      für jedes Proxel P mit P.t = step tue
4          für jede in P aktivierte Transition T tue
5              flussrate_T ← Summe über die Flußraten aller aktiven Server von T
               w'keit_T ← flussrate_T × τ × P.p (dabei wsumme bilden und ggf. normieren)
6              wenn der durch T erreichbare Zustand M_n zeitlos ist dann
7                  zeitlos(M_n, w'keit_T, P)
8              sonst
9                  serverEinzeln(P, T, w'keit_T)
10             wenn wsumme < 1 dann
11                 addProxel(1 − wsumme, P.M, altern(P.M, P.A, ∅), P.t + 1)
12     step ← step +1
```

Prozedur `zeitlos`(M, p, P): Verteile Wahrscheinlichkeit in einem zeitlosen Zustand

```
   Eingabe : Zeitlose Markierung M, Wahrscheinlichkeit p, Ausgangsproxel P
1  Σ ← Summe der Gewichte w_{T_i} der aktiven zeitlosen Transitionen
2  für jede in M aktivierte zeitlose Transition T_i tue
3      w'keit_T ← p × w_{T_i} × Σ^{-1}
4      wenn der durch T_i erreichbare Zustand M_n zeitlos ist dann
5          zeitlos(M_n, w'keit_T, P)
6      sonst
7          T ← Transition, die aus dem Ausgangsproxel heraus feuerte
8          serverEinzeln(P, T, p);
```

Es ist zu beachten, daß sich aufgrund der Verwendung von Alterungs-Matrizen der Aufruf und der Rückgabewert der Funktion altern() verändert haben. Die Funktion erwartet einen zusätzlichen Parameter mit der Information, welches Alter der gefeuerte Server besessen hat. Dies ist notwendig, um bei der Berechnung den korrekten Server zu deaktivieren. Desweiteren erzeugt die Funktion nun keine einzelne Alterungs-Matrix. Stattdessen wird eine Menge von Matrizen erstellt, deren Verwendung jeweils gleichwahrscheinlich ist. Dafür überprüft die Funktion für jede Transition im Petri-Netz die Differenz im Aktivierungsgrad vor und nach dem Zustandsübergang. Entsprechend dem Wert der Differenz werden die Alterungs-Informationen gemäß dem in Abschnitt 3.3.2

Prozedur `serverEinzeln`(P, T, p): Erzeuge neue Proxel

Eingabe : Ausgangsproxel P, gefeuerte zeitbehaftete Transition T, Wahrscheinlichkeit p

Daten : $\mathcal{A}$: Menge von erreichbaren Alterungs-Matrizen

1 **für jeden** *in T aktivierten Server s* **tue**
2 $\quad a_s \leftarrow$ Alter von s aus $P.A$
3 $\quad \mathsf{w'keit}_s \leftarrow \mathsf{flussrate}_s \times \tau \times p$
4 $\quad \mathcal{A} \leftarrow \mathsf{altern}(P.M, P.A, T, s)$
5 $\quad i \leftarrow |\mathcal{A}|$
6 $\quad$ **für jede** *in $\mathcal{A}$ gegebene Alterungs-Matrix A* **tue**
7 $\quad\quad \mathsf{addProxel}(\frac{p}{i},\ P.M,\ A,\ P.t + 1)$

beschriebenen Verfahren verändert und als Rückgabewert verwendet.

3.5. Berechnung der Ergebnisstatistiken

Die von einem Proxel getragene Information ist die approximierte Wahrscheinlichkeit, dmit der sich das Modell zu einem bestimmten Zeitpunkt der Simulation in einem konkreten Zustand (Markierung der Stellen und Alterung der Server) befindet. Im folgenden wird erklärt, wie sich daraus die in Tabelle 3.1 aufgeführten Ergebnisstatistiken berechnen lassen.

Diese Statistiken können jeweils lediglich nur an bestimmten Ausführungspunkten der Simulation erstellt werden. Allgemein lassen sich die Statistiken dabei in drei Klassen einteilen:

1. Werte, die lediglich von der Existenz eines Proxels abhängig sind,
2. Werte, die von der Existenz eines Proxels sowie den aktivierten Transitionen abhängig sind, und
3. Werte, die von der Existenz eines Proxels und dem Feuern einer der aktivierten Transitionen abhängig sind.

Ausschließlich abhängig von der Existenz eines Proxels sind die mittlere Tokenanzahl in einer Stelle Pl_{mean}, die Wahrscheinlichkeit für die Leere einer Stelle Pl_{empty} und der Erwartungswert für eine nicht-akkumulierte Zustandskostenfunktion Re_{nonAcc}.

$$Pl_{mean} = \frac{\tau}{T_{max}} \sum_{P} P.p \cdot P.m_{Pl} \tag{3.6}$$

$$Pl_{empty} = \frac{\tau}{T_{max}} \sum_{P|P.m_{Pl}=0} P.p \tag{3.7}$$

$$Re_{nonAcc} = \frac{\tau}{T_{max}} \sum_{P} P.p \cdot f_{Re} \tag{3.8}$$

Die Aktivierungswahrscheinlichkeit $Tr_{enabled}$ einer Transition Tr ist als einzige Statistik sowohl von der Existenz eines Proxels sowie den dabei aktivierten Transitionen abhängig.

$$Tr_{enabled} = \frac{\tau}{T_{max}} \sum_{P|\,Tr \in P.enabled} P.p \tag{3.9}$$

Die Feuerrate einer Transition Tr_{rate}, der Erwartungswert für eine akkumulierte Kostenfunktion Re_{acc} und der Erwartungswert für eine Impuls-Kostenfunktion Re_{imp} sind von der Wahrscheinlichkeit abhängig, mit der eine Transition feuert.

$$Tr_{rate} = \frac{1}{T_{max}} \sum_{P|\,Tr \in P.enabled} P.p \cdot h_{Tr}(k\tau) \tag{3.10}$$

$$Re_{acc} = \sum_{P} \sum_{Tr \in P.enabled} P.p \cdot h_{Tr}(k\tau) \cdot f_{Re} \tag{3.11}$$

$$Re_{imp} = \frac{1}{T_{max}} \sum_{P|\,Tr \in P.enabled} P.p \cdot h_{Tr}(k\tau) \cdot f_{Re} \tag{3.12}$$

In den oben genannten Termen werden die Bezeichnungen aus Abschnitt 2.4 verwendet. Darüber hinaus bezeichnet

- k den aktuelle Zeitpunktindex,
- $P.m_{Pl}$ die Belegung der Stelle Pl in der diskreten Markierung $P.m$,
- $P.enabled$ die Menge der aktivierten Transitionen im durch P beschriebenen Zustand,
- f_{Re} ein soeben bestimmtes Ergebnis der Kostenfunktion Re,
- h_{Tr} ein soeben bestimmtes Ergebnis der Hazard Rate Function der Transition Tr,
- τ die Zeitschrittweite und
- T_{max} die Systemzeit, bis zu der die Proxelanalyse durchgeführt werden soll.

Der Term $\frac{\tau}{T_{max}}$ gewichtet eine Statistik zeitlich. Die Proxel-basierte Simulation kennt $\frac{T_{max}}{\tau}$ verschiedene Zeitpunkte in der Simulation. Die Summe der Wahrscheinlichkeiten aller Proxels, die zu einem bestimmten Zeitpunkt gehören, muß laut Definition gleich 1 sein. Um eine Ergebnisstatistik für alle betrachteten Zeitpunkte zu generieren, ist es notwendig, diese wie oben dargestellt zu normieren.

Wie gezeigt wurde, können mit der Proxel-basierten Simulation alle statistischen Ergebnisse aus der Tabelle 3.1 berechnet werden.

3.6. Datenstrukturen

Für eine diskrete Markierung des Petri-Netzes existieren in der Regel mehrere Proxels mit unterschiedlichen Alterungs-Matrizen. Es bietet sich an, diese Markierungen nur ein einziges Mal zu speichern, um anschließend eine Referenz aus den dazugehörigen Proxels zu setzen. Theoretisch ließen sich auch dieselben Alterungs-Matrizen durch verschiedene Proxel nutzen, der Einfachheit halber ist dies allerdings nicht implementiert worden.

Ein Proxel referenziert somit auf eine Struktur Markierung und auf eine Matrix Alterung (siehe 3.3.1). Als Nutzlast trägt das Proxel zwei Wahrscheinlichkeitswerte p1 und p2 und die ihnen zugeordneten Zeitwerte t1 und t2. Eine Markierung enthält ihrerseits Referenzen auf alle Proxels, welche für die Markierung gültig sind. Damit lassen sich Proxels für eine bekannte Markierung schneller im Speicher finden.

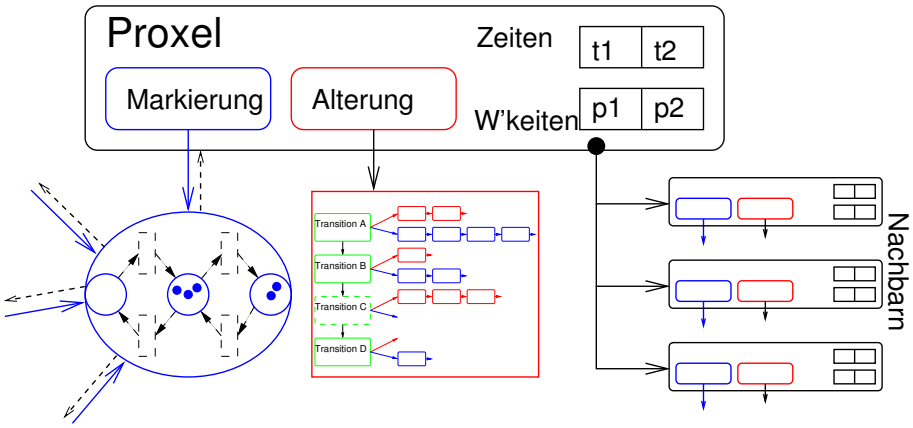

Abbildung 3.3.: Ein Proxel als Datenstruktur

Der größte Flaschenhals des Algorithmus ist die Suche nach bereits existierenden Proxels mit einer speziellen Markierung und einem speziellen Alterungsvektor. Diese Suche muß bei jedem erreichten Proxel erneut durchgeführt werden, um gegebenenfalls Proxels zusammenfassen zu können. Es ist möglich, diese Suche massiv zu beschleunigen, wenn für jedes Proxel eine Assoziation von Transitionen und den über sie erreichten Folge-Proxels erzeugt wird. Die Folge-Proxels brauchen somit nur ein einziges Mal bestimmt werden, ähnlich den Zuständen bei der Erzeugung eines Erreichbarkeitsgraphen. Ein solcher Proxel-basierter Simulator, der jedes Proxel mit seinen Folge-Proxeln verknüpft, ist

gut geeignet, um erst während der Analyse die Erreichbarkeitsinformationen aufzubauen, wie in Abschnitt 3.2 beschrieben.

Für das im Rahmen dieser Arbeit bearbeitete Werkzeug des Industriepartners wurde unter Berücksichtigung der funktionalen Freiheit bei der Modellierung darauf verzichtet, die Erreichbarkeitsinformation nur ein einziges Mal zu erzeugen und infolge wiederzuverwenden. Da die Erreichbarkeit von Zuständen während der Analyse von der Systemuhr abhängig sein kann, ist nicht gewährleistet, daß eine einmalig bestimmte Erreichbarkeitsinformation über die verbleibende Analyse gültig ist.

Stattdessen existiert in der Implementierung ein Caching von Proxel-Übergängen: Jedem Proxel wird für jede aktivierte Transition eine Datenstruktur zur Aufnahme von Proxels zu Verfügung gestellt. In dieser fügt der Algorithmus die jeweiligen Folge-Proxels (die *Nachbarn*) ein. Wird das ursprüngliche Proxel im weiteren Verlauf der Anaylse erneut betrachtet, wird nach Berechnung von Folge-Markierung und -Alterungs-Vektor in der Menge dieser Nachbarn nach einem Proxel mit den entsprechenden Koordinaten gesucht. Es wird davon ausgegangen, daß die Nachbarmenge während der Analyse nur wenige Folge-Proxels (≤ 3) enthalten wird. Bei einer Übereinstimmung kann die Wahrscheinlichkeit schneller kumuliert werden, als dies bei einer Durchsuchung aller existierenden Proxel der Fall wäre. Sollte kein äquivalentes Proxel gefunden werden, wird ein neues erzeugt und initial in die Nachbar-Datenstruktur als Assoziation für die gefeuerte Transition eingefügt.

Diese Art der Speicherung ermöglicht, daß bei der wiederholten Verarbeitung von Proxels auch neu entdeckte Folgezustände berücksichtigt werden können, und ist deswegen für die gegebenen Rahmenbedingungen besser geeignet.

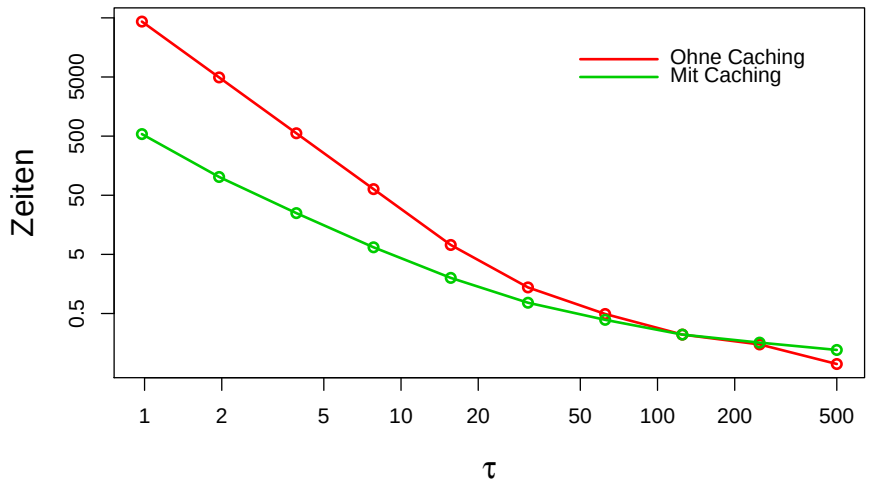

Abbildung 3.4.: Geschwindigkeitsgewinn durch Caching von Folge-Proxels

Die Abbildung 3.4 stellt dar, welcher Geschwindigkeitsvorteil durch solch ein Caching geschaffen wird. Für die gewonnenen Daten wurde das Garantiemodell aus Abschnitt 2.4.4 mit verschiedenen Zeitschrittweiten τ untersucht. Der rote Graph zeigt einen exponentiellen Anstieg der Laufzeit bei sinkendem τ, wie er bei einer Proxel-basierten Simulation ohne Caching auftritt. Der grüne Graph präsentiert die Ergebnisse für das Verfahren mit Caching. Auch dieses Verfahren hat nach wie vor exponentiellen Aufwand, ist aber für aufwendigere Analysen deutlich schneller.

Das Caching ist folglich eine sinnvolle Erweiterung der Datenstrukturen bei der Proxel-basierten Simulation, da hierdurch die Komplexität der Strukturen in bezug auf die Zeitschrittweite verringert werden kann.

3.7. Validierung und Bewertung

Sowohl die Validierung als auch die Verifikation der geleisteten Arbeit wurde vorrangig anhand von Vergleichen der Ergebnisse verschiedener Analysemethoden durchgeführt. Mehrere Petri-Netz-Modelle, von denen einige akademisch-fiktiver Natur und andere vom Industriepartner zur Verfügung gestellt waren, wurden sowohl mittels Monte-Carlo-Simulation als auch Proxel-basiert (und bei Markov-Ketten zusätzlich durch ein entsprechendes Lösungsverfahren) untersucht. Es ergab sich eine Übereinstimmung der Ergebnisse durch die verschiedenen Verfahren, was die Annahme stützt, daß die Implementierung korrekt ist. Im folgenden werden zwei der durchgeführten Versuche im Detail beschrieben.

Die Implementierung wurde zum Beispiel anhand des in [LMH04] vorgestellten Garantie-Modells mit dem in derselben Quellen beschriebenen Speziallöser und einer Monte-Carlo-Simulation verglichen. Dazu wurden jeweils elf Analysen mit jeweils halbierter Zeitschrittweite τ, beginnend bei 500, durchgeführt. Dabei erwies sich die Implementierung der Proxel-basierten Simulation im Werkzeug des Industriepartners für steigende Anforderungen an die Genauigkeit als leistungsfähiger. Der Speziallöser hat ein Modell mit gegebenem Erreichbarkeitsgraphen fest in die Analyse einkodiert. Als C-Programm ist er deutlich schneller als eine gleichwertige Java-Implementierung. Dieser Geschwindigkeitsvorteil macht sich insbesondere bei der Berechnung bei größeren τ bemerkbar. Durch das oben beschriebene Caching wird jedoch ein Geschwindigkeitsvorteil bei aufwendigeren Analysen erreicht.

Abbildung 3.5 präsentiert die resultierenden Laufzeiten und das Ergebnis der Kostenfunktion `WarrantyCosts` für den soeben beschriebenen Versuch. Bei sinkendem τ konvergiert der errechnete Wert für `WarrantyCosts` bei beiden Lösungsverfahren gegen das gleiche, reale Ergebnis. Auffällig ist, daß bei der allgemeinen Implementierung die berrechneten Werte das reale Ergebnis genauer approximieren als beim Speziallöser. Vermutlich ist die unterschiedliche Implementierung von numerischen Standardoperationen für Gleitkommaarithmetik bei der C-Standard-Bibliothek und der Java 2 Platform API hierfür verantwortlich. Die Abweichung wurde deswegen nicht als Grund betrachtet, an der Validität des Algorithmus zu zweifeln.

In der Graphik nicht berücksichtigt ist die Monte-Carlo-Simulation, für welche die

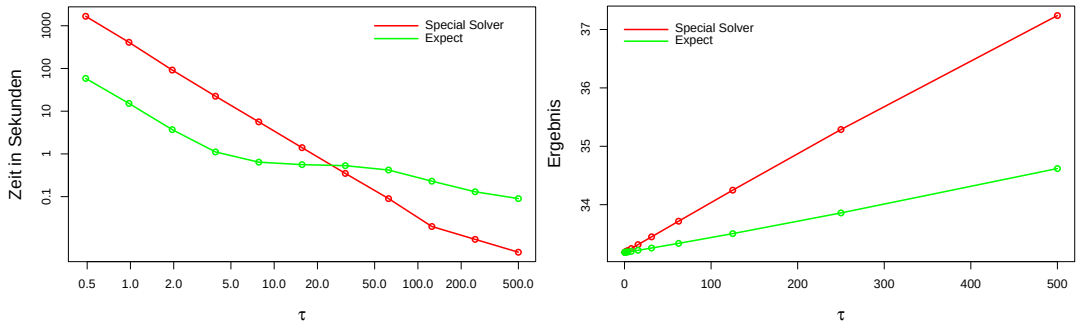

Abbildung 3.5.: Speziallöser und Expect im Vergleich

in [LMH04] angebenen 20 Stunden als Bedarf für die Gesamtlaufzeit der Simulation aber repliziert werden konnten. Die Ergebnisse stimmen mit denen der Proxel-basierten Analysen überein und liefern ebenfalls keinen Grund, die Hypothese über die Korrektheit der Implementierung abzulehnen.

Die Proxel-basierte Simulation ist in der vorliegenden Umsetzung deutlich schneller für das betrachtete Garantiemodell, welches in seiner Struktur und Komplexität repräsentativ für eine Klasse von Modellen ist, wie sie beim Industriepartner für eine Analyse eingesetzt werden. Damit ist mit der vorliegenden Umsetzung das erste Ziel dieser Diplomarbeit als erreicht anzusehen (siehe 1.4). Die Proxel-basierte Analyse bietet dem Industriepartner von nun an die Möglichkeit, derartige Modelle in einem Bruchteil der Zeit zu untersuchen, die vorher für die Analyse notwendig war.

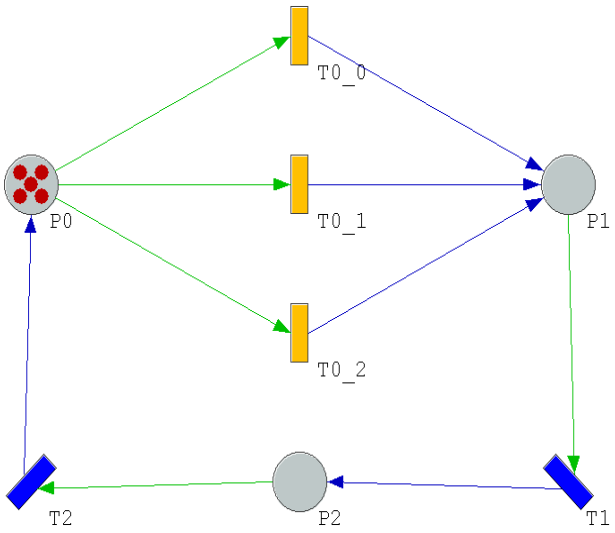

Abbildung 3.6.: Fiktives Modell mit drei äquivalenten Servern

Für den nächsten Versuch wurde ein fiktives Modell geschaffen, welches Multiplizitäten für Transitionen umfaßt. Dabei war die Server-Multiplizität einmal explizit als

Eigenschaft der Transition gegeben und zum anderen implizit durch die Erzeugung von mehreren äquivalenten Transitionen dargestellt. Die zweite Version wird durch die Abbildung 3.6 dargestellt. Die drei Instanzen der Transition `T0` unterliegen jeweils der Weibull-Verteilung ($\alpha = 4, \beta = 4$), die Transitionen `T1` und `T2` werden jeweils durch eine Exponentialverteilung mit $\lambda = 1$ beschrieben.

Untersucht wurde, neben der Äquivalenz der Ergebnisse zur Monte-Carlo-Simulation, inwiefern sich die Zusammenfassung der Transitionen auf die Effizienz des Algorithmus auswirkt. Erwartet wurde, daß die Darstellung der Server-Multiplizität als Parameter der Transition sich bei der Analyse vorteilhaft gegenüber den duplizierten Transitionen erweist, da für die drei Server nur ein Zustandsübergang (statt drei einzelnen Übergängen) zu betrachten ist.

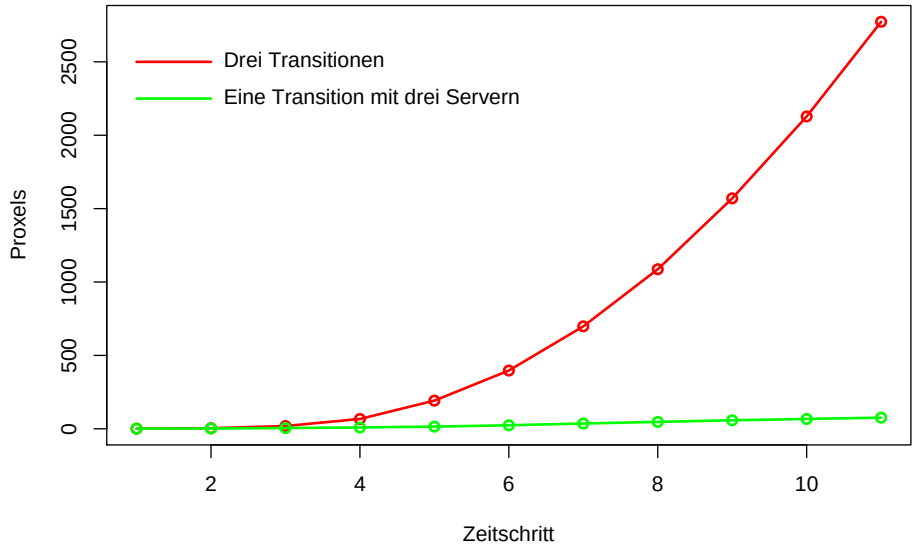

Abbildung 3.7.: Anzahl der Proxels in den ersten Zeitschritten

Die Abbildung 3.7 zeigt die gemessenen Werte für die Proxelanzahl bei der Analyse. Stark auffällig ist, daß viel mehr Proxels bei der Analyse entstehen, wenn die Modellvariante mit drei Transitionen verwendet wird. Der Grund hierfür ist, daß bei drei äquivalenten Transitionen auch dann drei Folgeproxels entstehen, wenn alle Transitionen den gleichen Alterungswert besitzen, da der Algorithmus keinerlei Kenntnis über die Äquivalenz der Transitionen hat. Bei drei Servern in einer einzigen Transitionen können gleichaltrige Server jedoch wie in Abschnitt 3.3 beschrieben gemeinsam betrachtet werden. In diesem Fall wird nur ein Proxel erzeugt.

Die Verwendung von Server-Multiplizitäten ist daher nicht nur eine für den Anwender bequeme Modellierungsart, sondern wirkt sich für das betrachtete Modell auch positiv auf die Effizienz der Proxel-basierten Analyse aus. Diese Vermutung wird Gegenstand weiterer Forschung sein müssen, um allgemeingültige Aussagen treffen zu können.

Durch die vorliegende Implementierung wurde die Proxel-basierte Simulation für den

Industriepartner zugänglich gemacht. Eine derartige Analyse führt für einige der Modelle des Industriepartners zu signifikanten Geschwindigkeitsgewinnen gegenüber den bisherigen Verfahren und ist somit von hohem Nutzen. Das erste Ziel der Diplomarbeit ist damit erreicht. Darüber hinaus konnte nachgewiesen werden, daß sich die Eigenschaft der Server-Multiplizität positiv auf die Effizienz des Algorithmus auswirkt, da hierdurch bei der Analyse weniger Proxels erzeugt werden.

Im folgenden Kapitel wird ein Algorithmus, der variable Zeitschrittweiten verwendet, disktutiert. Dies entspricht dem zweiten Ziel dieser Arbeit, dem Gewinn von Kenntnissen und Erfahrungen über die Effizienzveränderungen eines derart modifizierten Proxel-Verfahrens gegenüber dem bereits existierenden Algorithmus.

4. Variable Zeitschritte im Proxel-Algorithmus

Die Anzahl der Proxels bei der Proxel-basierten Simulation ist exponentiell abhängig von der Anzahl der durchlaufenden Zeitschritte. Damit besteht auch für die Laufzeit und den Speicherbedarf des Algorithmus exponentieller Aufwand. Wird die Anzahl der Proxels, die bei einer Analyse entstehen, verringert, so sinkt auch der Bedarf nach Rechenzeit und Arbeitsspeicher. Eine Möglichkeit, die Anzahl der entstehenden Proxels einzuschränken, ist der in [Hor02] beschriebene Schwellwert, anhand dessen die Wahrscheinlichkeit neuer Proxels gemessen wird. Dabei wird bewußt in Kauf genommen, daß sich gewisse mögliche Entwicklungen des Modells nicht im Ergebnis widerspiegeln, da davon ausgegangen wird, daß sie aufgrund ihrer geringen Beträge keinen wahrnehmbaren Einfluß auf das Ergebnis ausüben.

Einen weiteren Vorschlag zur Beschleunigung des Verfahrens auf Kosten der Güte der Approximation wird in diesem Kapitel vorgestellt. Dabei wird nicht erst nach der Erzeugung von Proxels entschieden, ob diese verwendet werden. Stattdessen werden weniger Proxels erzeugt, von denen dann aber zu erwarten ist, daß sie relevant für das Ergebnis sein werden.

Für dieses Kapitel werden einfache Stochastische Petri-Netze als Modellklasse verwendet, ohne die Erweiterungen aus Kapitel 3. Auf multiplizierte Server wird also verzichtet und die Alterungs-Information in einem Proxel wird durch einen Alterungs-Vektor statt durch eine Matrix ausgedrückt. Desweiteren wird bei der Betrachtung darauf verzichtet, zeitlose Zustände im Petri-Netz noch einmal explizit zu behandeln. Diese können wie gewohnt verarbeitet werden.

4.1. Die zugrundeliegende Idee

Der für die Anzahl der Zeitschritte wichtigste Parameter im Algorithmus ist die Zeitschrittweite τ. In den bisher existierenden Algorithmen wird davon ausgegangen, daß der Zeitschritt für alle Zustandsübergänge und über die gesamte Analysezeit konstant ist.

Die Wahrscheinlichkeit für einen Zustandsübergang w_T durch das Feuern einer Transition T bestimmt sich aus dem Produkt der Flußrate der Transition in Abhängigkeit von der Zeitschrittweite und τ dem Alter ihrer Aktivierung ($h_T(x_T)$).

$$w_T = \tau \times h_T(x_T) \tag{4.1}$$

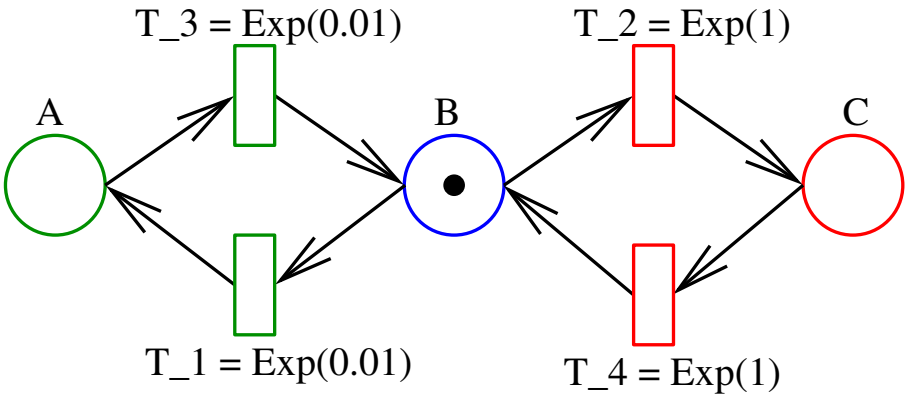

Abbildung 4.1.: Zwei Zustandsübergänge mit sehr unterschiedlichen Raten

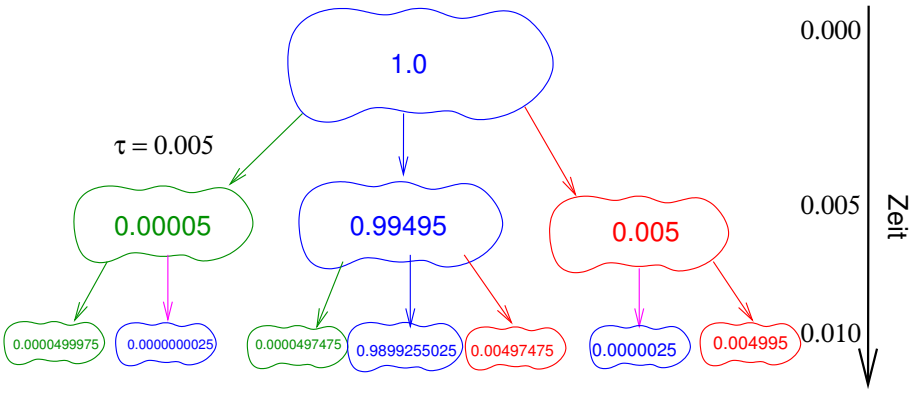

Abbildung 4.2.: Erste Schritte der Proxel-basierten Simulation

Stehen nun zwei Zustandsübergänge mit stark unterschiedlichen Raten in Konkurrenz zueinander, so fließen in die Folgezustände pro Zeitschritt auch stark unterschiedliche Mengen der Wahrscheinlichkeitsmasse des Ausgangsproxels. Dazu ein Beispiel: Zwei Zustandsübergänge T_1 und T_2 unterliegen Exponentialverteilungen mit den Raten $\lambda_1 = 0.01$, beziehungsweise $\lambda_2 = 1$, und stehen in Konkurrenz zueinander. Die ausschließliche Verwendung der Exponentialverteilungen im Beispiel stellt keine Beschränkung der Allgemeinheit dar. Anhand der konstanten Raten lassen sich die nachstehenden Erläuterungen anschaulich gestalten.

Für dieses Beispielmodell werden die Zeitschritte mit der Schrittweite $\tau = 0.005$ vorgenommen, damit eine ausreichend hohe Wahrscheinlichkeit besteht, daß nur ein Zustandswechsel pro Zeitschritt durchgeführt wird. Dabei fließt in den Folgezustand von T_2 100-mal soviel Wahrscheinlichkeitsmasse wie in den Folgezustand von T_1. Dies wird bei jeder Betrachtung der Zustandsübergänge erneut so geschehen. Über T_1 fließt die Wahrscheinlichkeit zu jedem Zeitpunkt um den Faktor 100 langsamer als über T_2. Um T_1 effizient analysieren zu können, wäre eine größere Schrittweite von Vorteil. Diese kann jedoch nicht verwendet werden, ohne die Annahme zu verletzen, daß durch T_2 nicht mehr

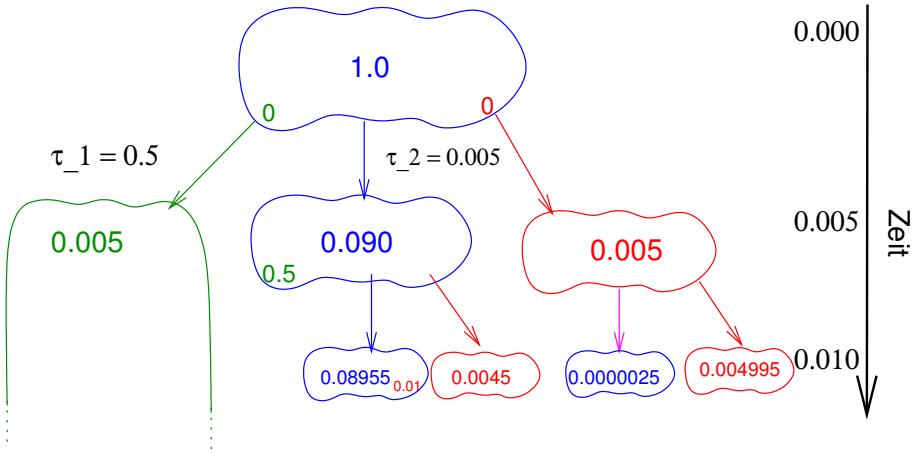

Abbildung 4.3.: Analyse mit unterschiedlichen Zeitschrittweiten

als ein Zustandswechsel pro Zeitschritt stattfindet. Infolge dessen werden durch T_1 auch mehr Proxels erzeugt, als es für eine hinreichend genaue Approximation notwendig wäre. Mehr Proxels bei der Analyse führen wiederum zu einer höheren Laufzeit.

Die Idee ist nun, die Anzahl der erzeugten Proxels und damit den Zeitbedarf zu verringern, indem für den Zustandsübergang T_1 eine größere Zeitschrittweite τ_1 verwendet wird, bei T_2 hingegen weiterhin der ursprüngliche Zeitwert $\tau < \tau_1$ zum Einsatz kommt. Dadurch wird der Wahrscheinlichkeitsfluß über T_1 seltener abgetastet und es entstehen weniger Proxels, bei einer Iteration wird im Gegenzug aber auch mehr Wahrscheinlichkeitsmasse als vorher über T_1 bewegt. Weiterhin repräsentieren die entstehenden Proxels ein größeres Zeitintervall. Dieses Intervall wird im weiteren Verlauf als die *Gültigkeit* des Proxels bezeichnet.

Besteht eine Wahrscheinlichkeit größer 0, daß der aktuelle Zustand nicht verlassen wird, erzeugt der Algorithmus ein entsprechendes Proxel mit gleicher Markierung und inkrementierter Alterungs-Information. Dieses Proxel ist nur solange gültig, wie das am kürzesten gültige der Proxels, die soeben über die entdeckten Zustandswechsel erreicht wurden. Die Weiterverarbeitung des Proxels für das Verweilen im Zustand sei *Wiedervorlage des Zustandes* genannt.

Existiert bei der Wiedervorlage eines Zustandes ein Proxel mit größerer Gültigkeit, so darf der entsprechende Zustandsübergang nicht durchgeführt werden. Am Beispiel: Für die Dauer der Gültigkeit eines über T_1 erzeugten Proxels darf kein weiteres Proxel über den Zustandsübergang erzeugt werden, wenn der Ausgangszustand B erneut vorgelegt wird. Dies gilt nicht, wenn B zwischenzeitlich auf anderem Wege (z.B. über T_2) verlassen und wieder neu betreten wurde. Wenn allerdings das Verweilen in B betrachtet wird, gilt T_1 als gesperrt.

Die Abbildung 4.3 zeigt die entstehenden Proxels der ersten drei Iterationen des Verfahrens für das obige Beispielmodell, wenn für T_1 und T_3 die Schrittweite $\tau_1 = 0.1$ und analog für T_2 und T_4 die Schrittweite $\tau_2 = 0.005$ zum Einsatz kommen.

Der Proxel-basierte Algorithmus bewegt sich aufsteigend durch die Systemzeit der

Analyse. Ein Proxel wird durch den Algorithmus weiterverarbeitet, wenn der Algorithmus zeitlich das Ende der Gültigkeit des Proxels erreicht hat. Streng genommen ist dies schon beim ursprünglichen Algorithmus so, da dort die Gültigkeit aller Proxels von der selben Breite ist, wie die konstanten Zeitschritte .

Proxel-basierte Simulation ist ein Approximationsverfahren für die stochastischen Vorgänge im Modell. Zum einen führt die Annahme, daß nur ein Zustandswechsel pro Zeitschritt möglich ist, dazu, daß die durch mehrere Übergänge bewegte Wahrscheinlichkeitsmasse nicht berücksichtigt wird. Zum anderen ist die Abtastung der Flußratenfunktion den üblichen numerischen Nachteilen von numerischen Integrationsverfahren unterlegen. Ein angepaßter Algorithmus mit variablen Zeitschritten ist diesen Einschränkungen ebenso unterworfen.

Der Algorithmus sollte die gleichen Ergebnisse approximieren, wie die existierende Analyse. Es wird erwartet, daß die vorgenommene Annäherung dabei von geringerer Güte ist, als es bei der Analyse mit dem ursprünglichen Algorithmus unter Verwendung der originalen Schrittweite τ der Fall wäre. Weiterhin wird erwartet, daß die Effizienz des Algorithmus deutlich ansteigt und somit in kürzerer Zeit Analyseergebnisse für die Extrapolation eines korrekten Wertes gewonnen werden können.

4.2. Datenstrukturen

Durch eine entsprechende Anpassung des Algorithmus ändern sich auch die Anforderungen, welche an die Datenstruktur für die Proxels gerichtet werden. Wurden vorher alle Zustandsübergänge aus einem Proxel mit der gleichen Schrittweite durchgeführt, so sind jetzt unterschiedliche Schrittweiten zulässig. Für das Proxel, welches das Verweilen in der Markierung seines Vorgängers darstellt, muß bekannt sein, welche Zustandsübergänge im nächsten Schritt noch erlaubt und welche vorrübergehend gesperrt sind.

Eine Datenstruktur für die Analyse mit variablen Zeitschritten muß in der Lage sein, eine Sperrung der Übergänge in den Proxels abzubilden. Die Dauer der Sperrung ist gleich der des letzen für diesen Übergang verwendeten Zeitschrittes. Die Zeitschrittweite kann durch den Benutzer vorgegeben sein oder (unter anderem) in Abhängigkeit der Flußrate des Zustandsübergangs bestimmt werden (siehe dazu 4.5).

Da die mit der Strategie *Race Enable* verdrängten Transitionen immer mit einem Alter $= 0$ aktiviert werden, gibt die Differenz ihres Alters und der Verweildauer des Proxels an, wie lang die Transition noch gesperrt ist. Bei Transitionen mit der Strategie *Race Age* ist die Verwendung einer weiteren Hilfsvariable notwendig, da das Alter der Aktivierung größer als 0 sein kann und somit nicht ohne weiteres mit der Verweildauer vergleichbar ist. Diese Sperrvariablen werden beim Eintritt in einen neuen Zustand für alle aktivierten Transitionen auf 0 zurückgesetzt, selbst wenn das Alter der Aktivierung ungleich 0 ist.

Desweiteren muß im Proxel die Verweildauer im Zustand gespeichert werden, um sie mit den Sperrvariablen vergleichen zu können. Die Verweildauer wird analog zu den Sperrvariablen nur dann inkrementiert, wenn der Zustand nicht verlassen wird. Das Inkrement der Verweildauer ist gleich dem Minimum der Zeitschrittweiten, die für die Zustandswechsel verwendet wurden. Analog dazu ist das Gültigkeitsende des Proxels

gleich dem kleinsten Gültigkeitsende der anderen Proxels.

Unter Berücksichtigungen dieser Überlegungen ist die nachstehend beschriebene Datenstruktur zur Speicherung der verschiedenen Proxels entworfen worden, welche alle Anforderungen umsetzt.

Es wird davon ausgegangen, daß es ein kleinstes *Basis-*τ gibt, von dem positive ganzzahlige Vielfache gebildet und als Zeitschrittweiten verwendet werden. Jeder Zeitpunkt der Analyse kann als $k\tau$ mit $k \in \mathbb{N}$ ausgedrückt werden. Proxels, deren Gültigkeitsende gleich dem Zeitpunkt $e\tau$ ist, werden in einen sogenannten *Container* C_e eingefügt. Dieser Container ist eine beliebige Datenstruktur zur Aufnahme einer Reihe von Proxels, zum Beispiel der in [Hor02] beschriebene sortierte, dynamische Baum. Der Container wird wiederum in eine verkettete, aufsteigend nach e sortierte Liste L eingefügt, sofern er nicht schon in L vorhanden ist. Das Einfügen ist von linearem Aufwand in Bezug auf die Anzahl der Container in L. Vor dem Verarbeiten der Proxels eines Containers wird dieser aus L entfernt.

Die Datenstruktur erhält drei Invarianten. Zum einen besitzen alle Proxels im Container C_e den Endezeitpunkt $e\tau$, unabhängig vom Zeitpunkt ihrer Erstellung. Diese Proxels werden daher alle in der gleichen Iteration des Algorithmus weiterverarbeitet. Der Faktor e braucht daher für alle Proxels in C_e nur einmal gespeichert zu werden. Die zweite Invariante besteht darin, daß L zu jedem Zeitpunkt $i\tau$ sortiert ist und sich an seinem Anfang der Container mit dem kleinsten Faktor e befindet, für den gilt:

$$e = \min \{\forall e \in \mathbb{N} i\tau < e\tau \vee C_e \neq \emptyset\} \tag{4.2}$$

Dadurch ist der erste Container in L stets derjenige, dessen Proxels als nächste verarbeitet werden müssen, wenn alle Proxels im momentan betrachteten Container C_i verarbeitet worden sind. Dieser Container kann mit konstantem Aufwand aus L entnommen werden. Die dritte Invariante besagt, daß ein Proxel genau dann erzeugt und in L eingefügt wird, wenn die analysierte Zeit im Algorithmus den Anfang der Gültigkeit des neuen Proxels erreicht hat. Folglich sind zu jedem Zeitpunkt alle Proxels in L auch für den gegenwärtigen Zeitpunkt gültig.

Abbildung 4.4 zeigt eine mögliche Entwicklung der Datenstruktur während der Analyse zum Zeitpunkt $(i-1)\tau$. Die Proxels des Containers C_i werden verarbeitet und die Folgeproxels entsprechend ihres Gültigkeitsendes in einen Container L eingefügt.

Die Gültigkeit eines Proxels wird gesichert, indem ein Proxel nur dann in L existiert, wenn der Algorithmus die Anfangszeit erreicht oder überschritten hat, das Ende jedoch nicht. Die Datenstruktur erfüllt somit alle oben formulierten Anforderungen und stellt eine Lösung für die korrekte Speicherung von Proxels bei der Analyse mit variablen Zeitschrittweiten dar.

4.3. Algorithmus

Der Algorithmus 4.1 (Seite 41) erfährt einige notwendige Änderungen gegenüber dem Original. Zum einen ändert sich das Kriterium, mit dem der Algorithmus terminiert,

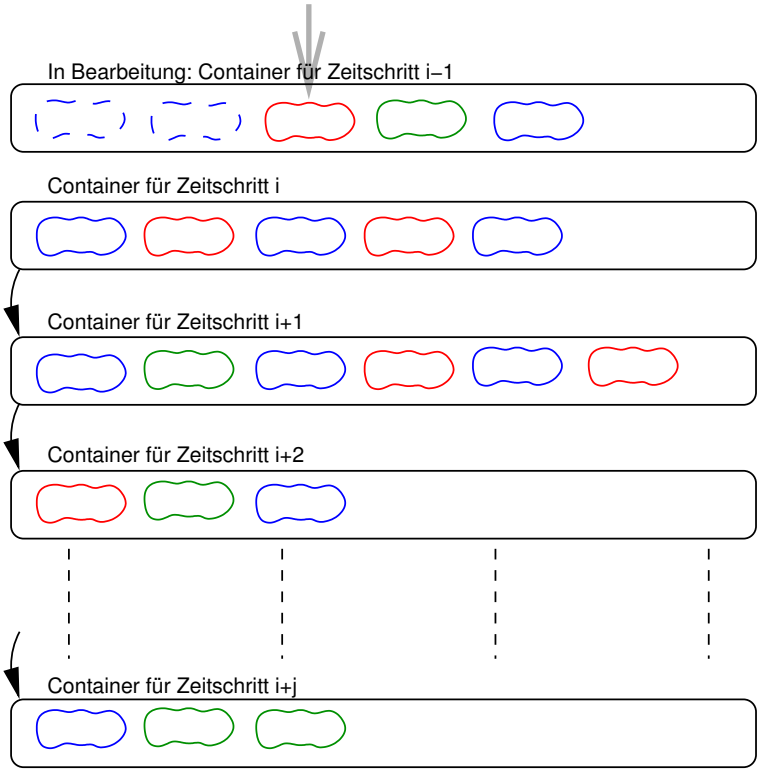

Abbildung 4.4.: Proxels in den Containern einer zeitlich sortierten Liste

marginal. Die Analyse wird weiterhin abgebrochen, wenn der Algorithmus die Endezeit erreicht. Jedoch ist die aktuelle Zeit des Algorithmus nicht explizit durch stepτ gegeben. Stattdessen stellt der Gültigkeitszeitpunkt des Containers, der das erste Element von L bildet, die Systemuhr dar. Ist dieser Wert größer oder gleich T_{max}, wird die Analyse beendet. Alle zu diesem Zeitpunkt in L enthaltenen Proxels bilden das Ergebnis.

Zum anderen ist die Durchführung eines aktivierten Zeitschritts davon abhängig, ob eine Sperrung vorliegt. Dies wird ermittelt, indem die ohne Zustandswechsel verstrichene Zeit mit dem Alter der einzelnen Aktivierungen verglichen wird (siehe Zeile 7). Ist das Alter einer Aktivierung größer als die Verweildauer im Zustand, so wird davon ausgegangen, daß ein Proxel mit längerer Gültigkeit existiert und der Übergang daher gesperrt ist.

Aus Gründen der Übersichtlichkeit wurde darauf verzichtet, die Wahrscheinlichkeitssumme wsumme, die Normierung der Wahrscheinlichkeiten und den Schwellwert ε im Algorithmus erneut explizit aufzuführen. Die Funktion altern nimmt einen zusätzlichen Parameter τ entgegen und erweitert die Alterungs- und Sperr-Informationen um diesen Betrag. Für alle Transitionen wird eine explizite Sperrzeit sperr$_{P,T}$ verwendet. Es sei nochmals erwähnt, daß sich dieser Wert bei *Race Enable*-Transitionen implizit aus dem Alter der Transitionen ergibt.

In Zeile 10 des Algorithmus wird die Breite des zu verwendenden Zeitschrittes τ be-

stimmt. Eine mögliche Implementierung kann aus einer vom Benutzer definierten Abbildung für jede Transition und jedes Alter eine Schrittweite bestimmen. In den Experimenten werden später fest vorgegebene Werte verwendet. Eine Erörterung der Möglichkeiten zur automatischen Anpassung der Schrittweiten wird weiter unten in Abschnitt 4.5 vorgenommen.

4.4. Validierung und Experimente

Der Algorithmus wurde validiert, indem die Analyseergebnisse für Modelle mit denen der bisherigen Proxel-Verfahren verglichen wurden. Die Erwartung war, daß mit dem neuen Verfahren die gleichen Ergebnisse angenähert wederden wie mit dem originalen Algorithmus, allerdings mit einem leicht größeren Fehler. Gleichzeitig wurden während der Analysen Messungen der Effizienz durchgeführt. Gemessen wurde die Anzahl der erzeugten Proxels während eines Analysevorgangs, sowie die Laufzeit.

4.4.1. Modell 1

Das erste analysierte Modell ist das Petri-Netz aus Abbildung 4.1, mit den folgenden Verteilungen für die Transitionen: $T_1 = \text{Exponential}(0.01)$, $T_2 = \text{Exponential}(0.01)$, $T_3 = \text{Exponential}(1)$, $T_4 = \text{Exponential}(1)$. Auch wenn es für die Berechnung nicht notwendig ist, wurden für die Transitionen in allen Analysen explizit Alterungs-Informationen gespeichert. Bei der Analyse mit variablen Zeitschritten wurden T_2 und T_4 mit der Schrittweite $\tau_2 = \tau$ untersucht. Für T_1 und T_3 wurde $\tau_1 = 100\tau$ verwendet. Die Analyse wurde bis $T_{max} = 2$ auf einem Pentium 4 mit 2.7 GHZ und 1 GB Arbeitsspeicher durchgeführt.

Die gemessenen Werte (Tabelle 4.1 und Abbildung 4.5) lassen darauf schließen, daß die Ergebnisse der zwei Algorithmen bei kleiner werdendem τ gegen dieselben Werte konvergieren. Dabei ist auffällig, daß der relative Fehler proportional zur Zeitschrittweite zu sein scheint und für den Zustand A deutlich größer als für B und C ist. Beide Algorithmen approximieren das gleiche Ergebnis. Die Approximation durch den neueren Algorithmus ist, wie erwartet, von geringerer Güte. Die Experimente geben keinen Grund, an der Validität des neuen Algorithmus zu zweifeln.

Nachdem keine Veranlassung besteht, den Algorithmus abzulehnen, soll die Änderung in der Komplexität der Berechnung untersucht werden. Wie in Tabelle 4.2 zu sehen ist, wurden mit dem neuen Algorithmus über ein Drittel weniger Proxels erzeugt. Bei der Laufzeit fällt dieser Vorteil aufgrund der exponentiellen Natur des Algorithmus noch stärker ins Gewicht. Die Verwendung unterschiedlicher Zeitschrittweiten ist bei der Proxel-basierten Simulation für dieses Modell sehr vorteilhaft.

Als nächstes wurde untersucht, welchen Einfluß der Faktor f beim Verhältnis der Schrittweiten $\tau_1 = f\tau_2$ auf den Approximationsfehler und die Effizienz hat. Dazu wurde das obige Modell mehrfach mit $\tau = 0.002$ und verschiedenen Faktoren f untersucht.

In der Abbildung 4.6 sind die gemessenen Werte für die während der Analyse auftretenden Proxels und für die Laufzeit in Abhängkeit von f graphisch aufbereitet. Die Graphen machen deutlich, daß bei ansteigendem f die Proxelanzahl und damit auch die

Algorithmus 4.1 : Proxel-Algorithmus unter Verwendung variabler Zeitschritte

Eingabe : L Datenstruktur nach Abschnitt 4.2, enthält Anfangsproxel

```
1  t ← 0
2  solange L ≠ ∅ ∨ t < T_max tue
3      C ← entferntes erstes Element von L
4      t ← Zeitwert von C für jedes Proxel P ∈ C tue
5          für jede in P aktivierte Transition T tue
6              sperr_{P,T} ← gespeicherte Sperrzeit für T
7              wenn alter_{P,T} ≥ P.a dann
8                  alter_{P,T} ← in P.A gespeicherte Dauer der Aktivierung von T
9                  flussrate_T ← Wahrscheinlichkeitsflußrate von T für alter_{P,T}
10                 τ_T ← weite(T, alter_{P,T})
11                 wsumme ← wsumme + w'keit_T
12                 w'keit_T ← flussrate_T × τ_T × P.p
13                 add(w'keit_T, folge(P.M, T), altern(P.M, P.A, ∅, τ_T), 0, t + τ_T)
14         τ_∅ ← min(τ_T)
15         add(1 − wsumme, P.M, altern(P.M, P.A, ∅, τ_∅), P.a + τ_∅, t + τ_∅)
```

Funktion `add`(p, M, A, a, t): Fügt Proxel in die Datenstruktur ein

Eingabe : Wahrscheinlichkeit p, Markierung M, Alterungs-Vektor A, Verweildauer a, Zeitpunkt der Wiedervorlage t

```
1  wenn C_t ∈ L dann
2      wenn ∃P ∈ C : P.M = M ∨ P.A = A ∨ P.a = a dann
3          P.p ← P.p + p
4      sonst
5          füge neues P = (M, A, p, a) in C_t ein
6  sonst
7      erzeuge C_t
8      füge neues P = (M, A, p, a) in C_t ein
9      füge C_t sortiert in L ein;
```

τ	A		B		C	
	Alt	Neu	Alt	Neu	Alt	Neu
0.002	0.01223	0.01389	0.50169	0.50081	0.48608	0.48530
0.001	0.01223	0.01304	0.50173	0.50130	0.48604	0.48566
0.0005	0.01223	0.01263	0.50175	0.50153	0.48603	0.48584
0.00025	0.01223	0.01243	0.50176	0.50165	0.48602	0.48592

Tabelle 4.1.: Meßwerte für die Ergebnisse

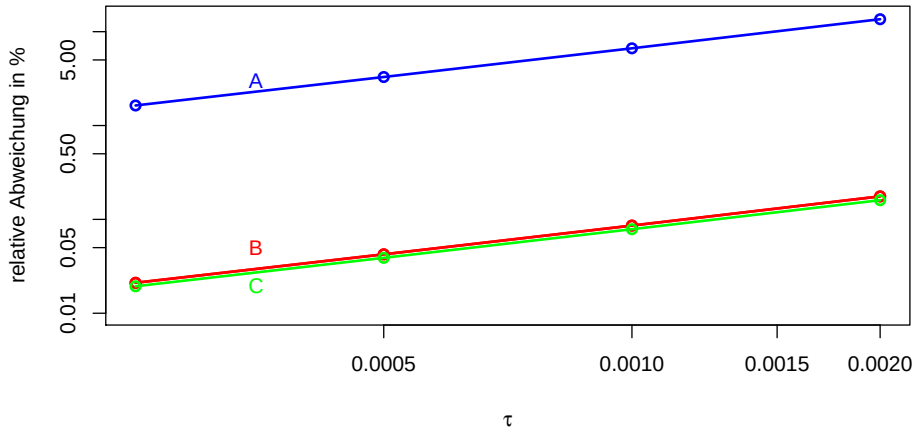

Abbildung 4.5.: Relative Abweichungen der Ergebnisse

Laufzeit gegen einen Wert konvergiert, der nicht unterschritten wird, so sehr f auch anwächst. Dieser Wert ist die Anzahl der Proxels im System, die mit dem Basiszeitschritt τ erzeugt werden.

Betrachtet man nun die relativen Abweichungen der Analyseresultate, der Laufzeit und der Proxelanzahl zu den Werten für $f = 1$ (dargestellt in Abbildung 4.7), so wird deutlich, daß ab $f \approx 20$ nur noch geringere Verbesserungen in der Effizienz möglich sind. Mehr als ein Drittel der Proxels des ursprünglichen Verfahrens kann nicht eingespart werden. Gleichzeitig nimmt der relative Fehler weiter linear zu. Für das Modell stellt $f = 20$ einen guten Kompromiß zwischen maximalem Effizienzgewinn und korrekter Approximation dar. Dieser Wert ist um den Faktor 5 kleiner als das Verhältnis der Flußraten.

4.4.2. Modell 2

Das zweite Modell ist von der gleichen Struktur wie das erste. Verändert wurden die Verteilungen der Transitionen T_2 und T_4, die nun jeweils eine deterministische Feuerzeit $d = 1$ haben. Auch für dieses Modell wurde eine Validierung vorgenommen, die den Algorithmus ebenfalls nicht ablehnte. Von daher sind im folgenden nur die bemerkenswerten relativen Abweichungen präsentiert (siehe Abbildung 4.8).

Durch Verwendung von deterministischen Feuerzeiten entstehen an den entsprechen-

τ	Laufzeit in ms			Proxels		
	Alt	Neu	Abw.	Alt	Neu	Abw.
0.002	3244	2012	37.95%	1504504	1006590	33.09%
0.001	17786	10425	41.39%	6009004	4023080	33.05%
0.0005	108075	61919	42.71%	24018004	16086060	33.02%
0.00025	565459	342486	39.43%	96036004	64332020	33.01%

Tabelle 4.2.: Laufzeit und Proxelanzahl im Vergleich

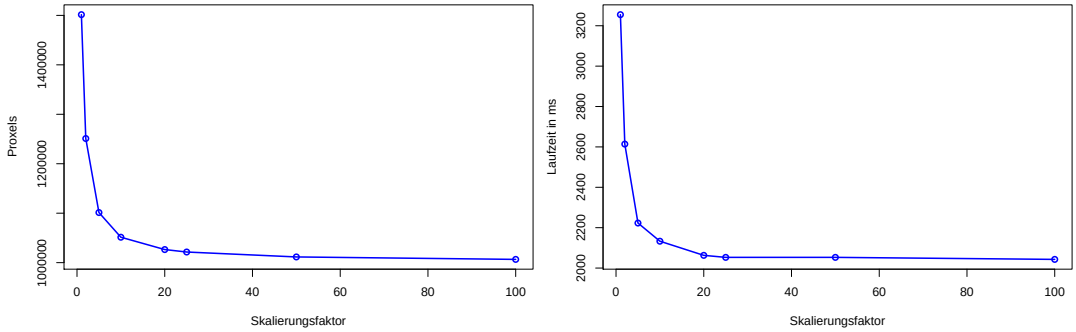

Abbildung 4.6.: Proxels und Laufzeit in Abhängigkeit der Skalierung

den Zustandsübergängen nur dann neue Proxel, wenn sich der Alterungs-Wert der Aktivierung innerhalb einer τ-Umgebung um den deterministschen Wert befindet. Ansonsten werden keine neuen Proxels erzeugt. Es entstehen somit so gut wie alle Proxels über die exponentiellverteilten Transitionen. Für die Analyse des vorliegenden Modells mit unterschiedlichen Zeitschritten bedeutet dies, daß sehr hohe Einsparungen an erzeugten Proxels verzeichnet werden. Es lohnt sich daher, deterministische Feuerzeiten und Verteilungsfunktionen mit beschränktem Träger (zum Beispiel Uniform-Verteilungen) durch Verwendung einer kleineren Schrittweite abzutasten.

Auch beim zweiten Modell ist $f = 20$ eine gute Wahl für ein angemessenes Verhältnis zwischen Approximationsfehlern und Effizienzgewinnen. Es stellt sich für die zukünftige Forschung die Frage, inwiefern dieser Wert von der Struktur des Modells und dem Verhältnis der Schrittweiten abhängig ist.

4.4.3. Fazit

Bei der Proxel-basierten Simulation werden durch die Verwendung von variablen Zeitschrittweiten in einer Analyse Effizienzvorteile erzielt. Die Vorteile werden durch einen zusätzlichen Fehler in der Approximation und eine Vergrößerung des Zustandsraums durch Verwendung zusätzlicher Hilfsvariablen erkauft. Die durchgeführten Experimente legen den Schluß nahe, daß dieser Preis weniger schwer wiegt als der erreichte Geschwindigkeitsvorteil. Die maximal mögliche Beschleunigung hängt von der Modellstruktur und den verwendeten Verteilungsfunktionen ab. Beim Einsatz von Verteilungsfunktionen mit

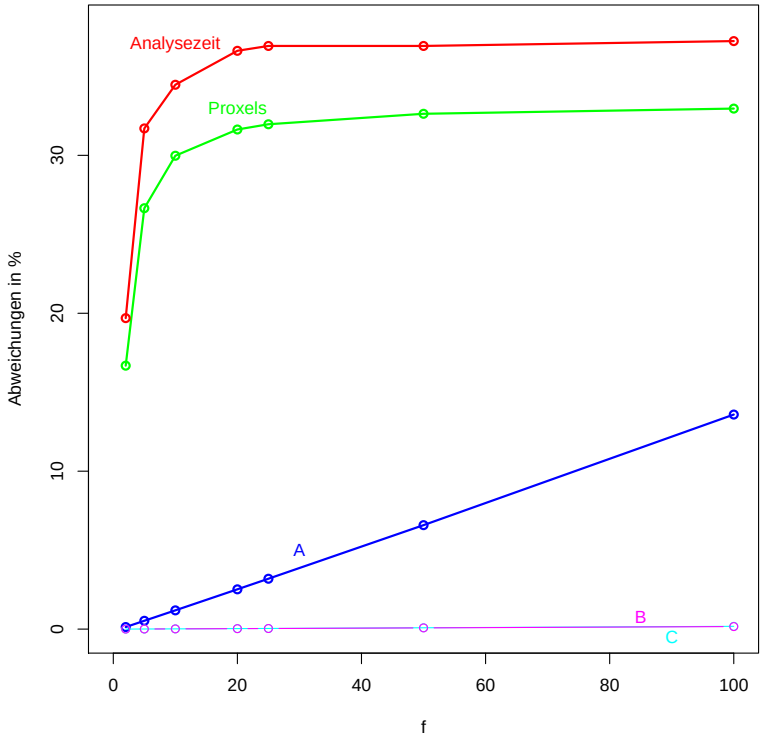

Abbildung 4.7.: Relative Abweichungen für Modell 1

beschränktem Träger lassen sich besonders große Beschleunigungen erreichen. Bei der Analyse läßt sich ein Kompromiß für die Skalierung der vergrößerten Zeitschrittweite finden, ab dem keine nennenswerten Geschwindigkeitsgewinne mehr verzeichnet werden können, sondern nur noch der Approximationsfehler ansteigt. Das Verhältnis der Schrittweiten ist bei diesem Kompromiß weniger extrem als das Verhältnis der Raten.

Der veränderte Algorithmus setzt voraus, daß die zu verwendenden Zeitschrittweiten zur Laufzeit schon bekannt sind oder durch eine Berechnung bekannt werden. Solange kein Verfahren für eine automatisierte Berechnung der Schrittweiten zur Laufzeit existiert, müssen sie durch den Anwender vorgegeben werden. Da dies Kenntnisse in der Stochastik und der Proxel-basierten Simulation erfordert und somit einem unversierten Anwender nicht zumutbar ist, ist das Verfahren noch nicht reif genug für die allgemeine Verwendung und bedarf weiterer Erforschung.

Die soeben getroffenen Aussagen waren das zweite Hauptziel der Diplomarbeit, welches somit als erreicht angesehen wird. Nachstehend werden als eine Vorarbeit für die weitere Erforschung des Verfahrens zusätzliche Theorien und Gedankenexperimente erörtert.

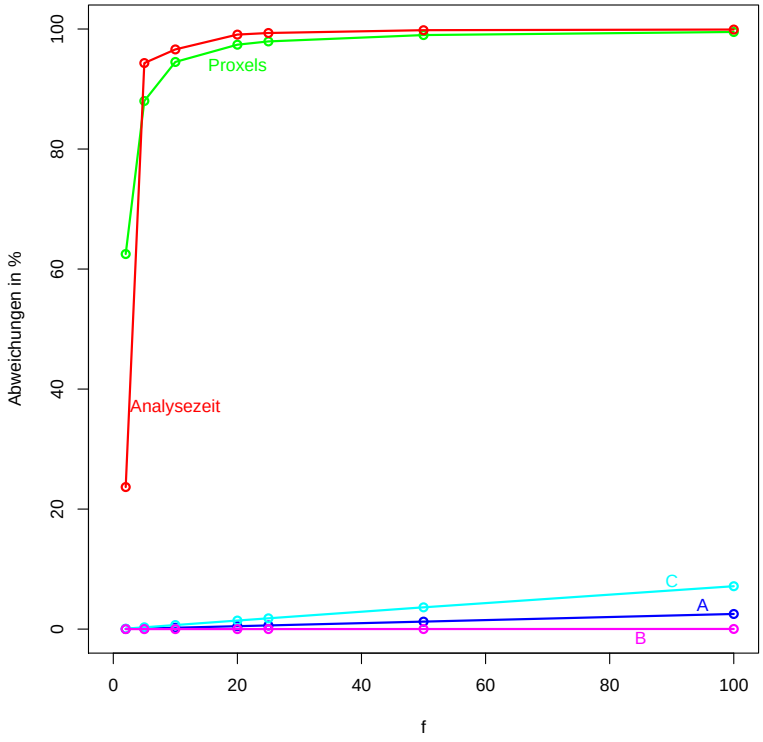

Abbildung 4.8.: Relative Abweichungen für Modell 2

4.5. Gedanken zur Berechnung adaptiver Schrittweiten

In diesem Abschnitt wird die automatisierte Wahl der Zeitschrittweiten theoretisch erörtert. Eine derartige Automatisierung ist notwendig, da das Verfahren ansonsten lediglich durch den Benutzer vorgegebene Werte verwendet. Dies erfordert Benutzerkenntnisse in der Stochastik und der Proxel-basierten Simulation, die im Regelfall nicht vorausgesetzt werden können. Zudem ist die manuelle Parametrierung sehr zeitaufwendig und macht somit die erzielten Geschwindigkeitsvorteile schnell zunichte.

Der folgenden Vorschlag basiert darauf, daß ein sehr kleiner Wert τ_0 als Basisschrittweite zur Verfügung steht. Dieser Wert entspricht der Schrittweite in einem Integrationsverfahren. Alle verwendeten Schrittweiten im Verfahren sind ganzzahlige positive Vielfache von τ_0.

Eine Möglichkeit der Berechnung ist nun, die Schrittweiten für die verschiedenen Übergänge zueinander in das umgekehrte Verhältnis zu setzen, das die Flußraten der aktiven Übergänge im Moment bilden. Für den Übergang mit der größten Flußrate wird die kleinste Schrittweite (τ_0) eingesetzt. Die übrigen Flußraten werden entsprechend skaliert. Ist es nicht möglich, das Verhältnis der Schrittweiten durch eine Menge von ganzzahligen Vielfachen von τ_0 anzunähern läßt, ist τ_0 entsprechend zu verringern. In alle Folge-Proxels fließt dann der gleiche Anteil an Wahrscheinlichkeitsmasse, sie sind aber

von unterschiedlich langer Gültigkeit. Soll oder kann τ_0 nicht verringert werden, so ist für alle Schrittweiten ein entsprechend geringer skaliertes Produkt von τ_0 einzusetzen.

In den durchgeführten Experimenten wurde festgestellt, daß das Verhältnis der Schrittweiten weit weniger extrem sein kann als das Verhältnis der Flußraten, um das Beschleunigungspotential nahezu auszuschöpfen. In den Experimenten war das Verhältnis der Flußraten fünfmal so hoch wie das empfohlene Verhältnis der Schrittweiten. Sollte dies auch im allgemeinen Fall gelten, so ist es bei der Berechnung der Schrittweiten zu berücksichtigen.

Für Verteilungsfunktionen mit beschränktem Träger erscheint es sinnvoll, einen möglichst großen Zeitschritt zu machen. Die Weite dieses Trägers ist das Intervall vom jetzigen Alter des Zustandsübergangs bis zum Ende des Nullstellenintervalls, um dann mit kleinerer Schrittweite fortzufahren. Es werden beim originalen Verfahren zwar ebenfalls keine neuen Proxels erzeugt, wenn die Flußrate gleich 0 ist. Durch die Sperrung des Übergangs als Folge des gemachten Zeitsprungs werden jedoch Rechenschritte eingespart.

Bisher wurde davon ausgegangen, daß die Übergangswahrscheinlichkeit w_T der errechnete Wert ist und $h_T(x_T)$ als Funktion sowie τ vom Benutzer als Wert oder durch eine Heuristik vorgegeben sind (siehe Gleichung 4.1). Denkbar ist aber auch, daß stattdessen ein Basiswert τ_0 und ein Zielwert für w_T angegeben werden, anhand derer in einer einfachen Iteration ein natürlicher Skalierungsfaktor k bestimmt wird, für den die folgende Beziehung gilt.

$$\min |w_T - h_T(x_T) \times k \times \tau_0| \tag{4.3}$$

$$k = \texttt{round}\left(\left|\frac{w_T}{h_T \times \tau_0}\right|\right) \tag{4.4}$$

Damit wird ausgedrückt, daß jeder Zustandswechsel mit der Wahrscheinlichkeit w stattfindet und es wird berechnet, wie lange gewartet werden muß, bis diese Wahrscheinlichkeit erreicht ist. Zu beachten ist, daß der Betrag von $h_T(x_T)$ im Regelfall nicht konstant für $0 \leq x_T \leq T_{max}$ ist und somit ein zusätzlicher Fehler in der Approximation entstehen kann.

$$h_T(x_T) \times 2 \times \tau_0 \neq (h_T(x_T) + h_T(x_T + \tau_0) \times \tau_0) \tag{4.5}$$

Nach Gleichung 4.3 würden dann im Algorithmus so lange Proxels erzeugt, bis keine Proxels mehr für Zeitpunkte kleiner als T_{max} in der Datenstruktur (siehe Abschnitt 4.2) existieren, das heißt, bis der kleinste Zeitpunkt in der Datenstruktur größer gleich T_{max} ist. Alle verbleibenden Proxels sind für den Zeitpunkt T_{max} gültig.

Gelingt es, die Schrittweitenberechnung mit einem der obigen oder einem gleichgerichten Verfahren zu automatisieren, so läßt sich eine erste allgemeine Implementierung des Proxel-basierten Verfahrens mit variablen Zeitschritten schaffen. Weitere Untersuchungen müssen zeigen, welche Berechnungsmethode am besten geeignet ist.

5. Reflexionen über die Arbeit

Im abschließenden Kapitel der Arbeit wird eine kurze Zusammenfassung über die vorgenommenen Arbeiten und deren Ergebnisse gegeben. Darauf folgt eine rückblickende Bewertung aus persönlicher Sicht. Zum Ende werden in einem Ausblick die Möglichkeiten für Folge- und Ergänzungsarbeiten ausgelotet.

5.1. Zusammenfassung

In der vorliegenden Diplomarbeit wurde zuerst ein industriell genutztes Werkzeug für Petri-Netz-Analysen vorgestellt und anschließend die Grundlagen der Proxel-basierten Simulation erläutert.

Im darauffolgenden Kapitel wurde der bisherige Stand der Erkenntnisse über die Proxel-basierte Simulation eingesetzt, um in einem existierenden Werkzeug der Daimler-Chrysler AG eine allgemeingültige Implementierung zur Proxel-basierten Simulation von erweiterten Stochastischen Petri-Netzen zu schaffen. Dabei sind Methoden entwickelt worden, wie sich die erweiterten Modellierungsmöglichkeiten des Werkzeugs, wie zum Beispiel Server-Multiplizitäten und funktionale Parameter, auf das neue Verfahren abbilden lassen und wie Ergebnisstatistiken aus den durch die Proxels getragenen Informationen zu berechnen sind. Die durchgeführten Experimente zeigen, daß die Umsetzung einige der wichtigen Garantie-Modelle des Industriepartners deutlich schneller analysiert, als die ansonsten eingesetzte Monte-Carlo-Simulation.

Im vierten Kapitel wurde eine Idee zur Beschleunigung der Verfahrens vorgestellt, die auf dem Einsatz unterschiedlicher Zeitschrittweiten basiert. Es wurde beschrieben, inwiefern der Algorithmus und die Datenstrukturen verändert werden müssen, um die Idee bei der Analyse umzusetzen. Anhand von Experimenten wurde gezeigt, daß mit dem Verfahren eine deutliche höhere Effizienz erreicht wird, als mit dem ursprünglichen Verfahren. Darüberhinaus wurden Ansätze vorgeschlagen, um Methoden für die automatische Berechnung der Zeitschrittweiten zu entwickeln.

Bevor nun Vorschläge für mögliche Erweiterungen gemacht werden, soll eine persönliche Bewertung vorgenommen werden.

5.2. Bewertung und Rückblick

Die in Kapitel 1 gestellten Aufgaben wurden in der Arbeit vollständig bearbeitet. Die im selben Kapitel formulierten Ziele wurden ebenfalls beide erreicht. Der allgemeingültige Proxel-Simulator wurde der DaimlerChrysler AG zur Verfügung gestellt und kann dort

vorteilhaft zur Analyse von Petri-Netzen eingesetzt werden. Gleichzeitig wurde gezeigt, daß der Einsatz von variablen Zeitschritten sehr lukrativ ist. Diese Erkenntnis stellt einen notwendigen Schritt bei der Erforschung der Proxel-basierten Simulation dar.

Die größte Schwierigkeiten bei der allgemeinen Implementierung bereitete der Entwurf von Datenstrukturen für die Speicherung von Markierungen, Alterungen und Nachbarschaftsbeziehungen zwischen den Proxels. In der einschlägigen Literatur wird diese Herausforderung ebenfalls als ein weißer Fleck auf der wissenschaftlichen Landkarte betrachtet. Nachträglich ist festzustellen, daß insbesondere die Speicherung der Markierungen in einer Liste nachträglicher Verbesserungen bedarf. Es war jedoch nicht mehr möglich, diese noch im Bearbeitungszeitraum vorzunehmen.

Bisher können mit der geleisteten Arbeit lediglich Klassen von kleinen Modellen effizienter als mit der Monte-Carlo-Methode analysiert werden. Klein bedeutet in diesem Fall eine geringe Anzahl von Zuständen und konkurrierenden Transitionen. Innerhalb dieser Klassen befinden sich bereits Modelle, wie zum Beispiel das in der Arbeit beschriebene, für den Industriepartner wichtige Garantiemodell, für die sich die Proxel-basierte Simulation bereits jetzt schon als sehr viel effizienter erweist.

Bei der Untersuchung der variablen Zeitschritte übertrafen die Geschwindigkeitszunahmen (insbesondere bei Modellen, die Verteilungen mit beschränktem Träger umfassten) die persönlichen Erwartungen sogar. Im Gegensatz dazu wurde nicht erwartet, daß sich eine automatisierte Berechnung von Zeitschrittweiten als die größte Herausforderung präsentieren würde. Zum Bedauern des Autors war es bisher nicht möglich, weitere Untersuchungen zu diesem als hochgradig interessant angesehen Thema anzustellen.

5.3. Offene Fragen

Bei der Bearbeitung der Aufgabenstellung sind mit dem Gewinn neuer Erkenntnisse auch neue Fragen entstanden, deren Antworten noch in der Zukunft verborgen sind und die durch weitere Forschungsarbeit entdeckt werden müssen. Andere Fragen existierten bereits im Vorfeld und konnten durch die vorliegende Arbeit nicht beantwortet werden.

Seit Beginn der Erforschung der Proxel-basierten Simulation besteht Bedarf nach einer optimalen Datenstruktur, in der mit minimalem Aufwand geprüft werden kann, ob ein Proxel mit bekannten Zustandskoordinaten bereits in der Struktur existiert. Eine theoretische und triviale Lösung wäre die Verwendung eines mehrdimensionalen Arrays, in dem die Proxels entsprechend ihren Zustandskoordinaten adressiert werden. In der Praxis ist die räumliche Komplexität eines solchen Arrays viel zu hoch, um in einem realen Arbeitsspeicher gehalten zu werden. Baumartige Strukturen mit einer hohen Vermaschung durch Referenzen sind viel effizienter, aber auch komplizierter zu handhaben. Zudem haben sie der exponentiellen Natur des Verfahrens langfristig nichts entgegenzusetzen.

Die für die Erstellung eines allgemeingültigen Simulators dringendste Frage ist, wie sich bei der Verwendung von variablen Zeitschritten weitestgehend automatisiert bestimmen läßt, wann welche Schrittweite für die Erzeugung eines neuen Proxels für einen Zustandsübergang zu verwenden ist. Die Berechnung sollte in der Lage sein, beliebig

kleine (bzw. große) Schrittweiten vorzugeben, wenn dies für die Genauigkeit notwendig , bzw. für die Effizienz sinnvoll ist.

Direkt daraus erwächst die Frage, wie bereits zur Laufzeit der Analyse festgestellt werden kann, mit welche Genauigkeit der Algorithmus fortschreiten soll, um auch diesen Parameter automatisiert anzupassen oder gar in der Berechnung der Schrittweiten zu verwenden.

Weiterhin gilt es zu bestimmen, ob eine Heuristik existiert, um den beschriebenen Kompromißfaktor zu finden, mit dem bei der Skalierung der Zeitschrittweiten eine optimale Balance zwischen Approximationsfehler und Geschwindigkeitsgewinnen erreicht werden kann. Es besteht Grund zur Annahme, daß der Weg zu dieser Heuristik auch zu einem Verfahren führt, um die vorher beschriebene Frage nach der notwendigen Genauigkeit zu beantworten. Bei beiden Fragen ist die Information, welche Genauigkeit momentan bereits erreicht wird, von zentraler Bedeutung. Fest steht schon jetzt, daß das Verhältnis der Schrittweiten nicht extremer sein sollte als das Verhältnis der Wahrscheinlichkeitsflußraten.

5.4. Ein langfristiges Ziel

Wie kann nun die Weiterentwicklung des Proxel-basierten Verfahrens langfristig aussehen? Betrachtet man die Gemeinsamkeiten des Proxel-Verfahrens mit einem Simulator für zeitkontinuierliche Modelle auf der Basis einfacher Differentialgleichungen, so wird deutlich, daß einige Gemeinsamkeiten vorliegen. In beiden Fällen wird ein kontinuierlicher Prozeß diskretisiert, beide Verfahren arbeiten deterministisch und benötigen keine replizierten Simulationsläufe.

Ein Ziel der Forschung kann dann ein Verfahren sein, welches das Modell wie ein kontinuierliche Simulator in einem einzigen Durchlauf analysiert (im Gegensatz zur mehrfachen Simulation mit einer Extrapolation der Ergebnisse). Das Verfahren ist in der Lage, für jeden Zustandsübergang und jede Analysezeit immer die sinnvollste Schrittweite zu bestimmen, basierend auf einer konstanten minimalen Schrittweite τ_0 als Gegenstück zu einer Intergrationsschrittweite. Für einen Anwender ist es nicht mehr notwendig, Änderungen an τ_0 vorzunehmen, es sei denn, um außergewöhnliche numerische Herausforderungen zu meistern.

Ein allgemeiner Simulator, der nach dem eben beschriebenen Verfahren arbeitet, mag das Potential haben, die Monte-Carlo-Simulation als vorrangige Methode zur Analyse zeitdiskreter Simulationsmodelle abzulösen, selbst wenn er weniger effizient als diese ist. Durch den Determinismus der Proxel-basierten Simulation und dem damit verbundenen Verzicht auf Pseudozufallszahlen werden die Ergebnisse frei von simulationsbedingten Varianzen sein.

Darüberhinaus ist das Verfahren auch in der Lage, hybride Modelle mit einem zeitkontinuierlichen und einem zeitdiskreten Anteil zu verarbeiten. In diesem Fall dient τ_0 auch zeitgleich als Integrationsschrittweite für die angebenen Differentialgleichungen.

Es ist zum momentanen Zeitpunkt unklar, ob und bis wann ein derartiges Ziel erreicht werden kann. Genausowenig steht fest, ob neue Erkenntnisse nicht noch weitere Mög-

lichkeiten aufzeigen, an die bisher noch nicht gedacht wurde. Wohin auch immer sich die Proxel-basierte Simulation entwickeln mag, so ist es laut Meinung des Autors doch zu erwarten, daß das Verfahren in Zukunft auch für komplexere Modelle realisierbar und somit von großem Nutzen sein wird.

A. Anhang

Nachstehend befinden sich lediglich Erläuterungen der geleisteten Tätigkeit und der dabei geschaffenen Software beim Industriepartner. Dieses Kapitel soll hilfreich für das Verständnis sein und die Programmierung nachvollziehbar werden lassen. Darüber hinaus wird Auskunft gegeben, wie die Implementierung von Anwendern bedient werden kann. Abschließend werden einige Mängel aufgezeigt und Ideen für Weiterentwicklungen vorgestellt.

A.1. Struktur der Implementierung

Der Proxel-Simulator befindet sich im Package `com.dcx.expect.pn_proxel`. Die zentrale Klasse ist dabei `ProxelAnalyzer`, eine Implementierung der Schnittstelle `AnalyzerInterface`. Hierin befinden sich die iterativen Algorithmen, welche die Proxel-basierte Analyse durchführen. Gleichzeitig implementiert die Klasse die Schnittstelle `Runnable` und bietet deswegen die Möglichkeit, die Analyse in einem eigenen Thread auszuführen. Der Proxel-Analyzer verlangt als Eingabe im Konstruktor eine Instanz von `PnNet` sowie eine Konfiguration `PNConfig`.

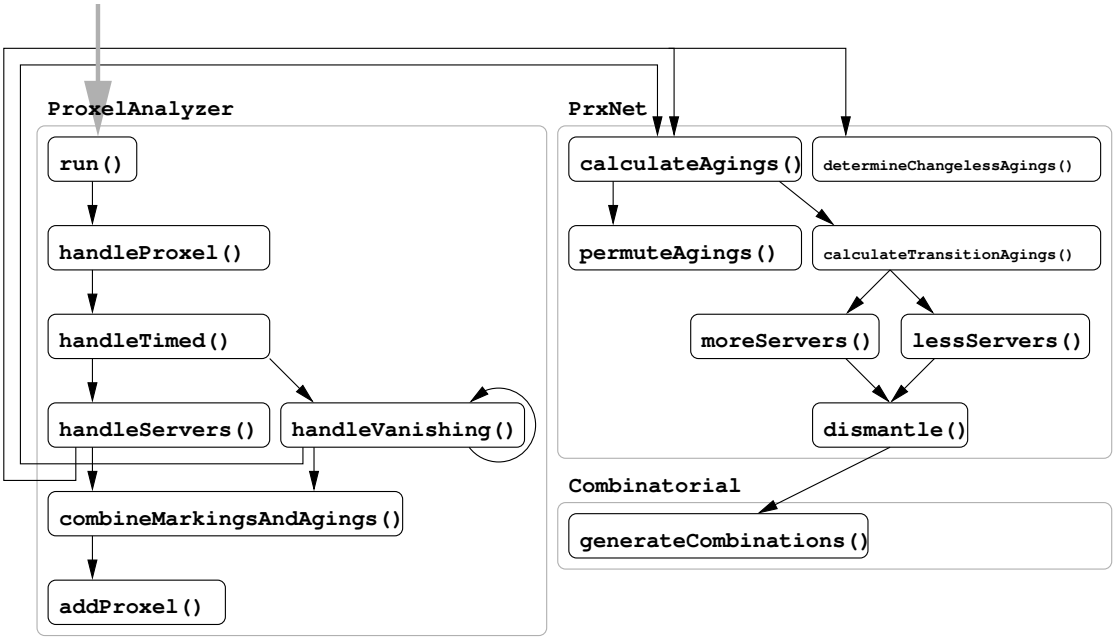

Abbildung A.1.: Aufrufefolge der Methoden

Von hoher Bedeutung ist daher auch die Methode `run()`, welche nach Einrichtung aller notwendigen Datenstrukturen die Iterationen startet. Pro Iteration werden dann alle vorhandenen Proxels für die $\frac{T_{max}}{\tau_i}$ Zeitpunkte betrachtet. Die Implementierung kennt nur zeitbehaftete Proxel; zeitlose Zustände werden sofort aufgelöst. Für jedes zeitbehaftete Proxel wird in `handleProxel(Proxel working, int time)` das Modell in die entsprechende diskrete Markierung versetzt und die Menge der aktivierten Transitionen (ENAS = *Ena*bling *S*et) bestimmt. Sind keine Transitionen aktiv, wird von einem absorbierenden Zustand ausgegangen und die Wahrscheinlichkeit wird vollständig an die folgenden Zeitschritte durchgereicht. Ansonsten werden in `handleTimed()` für jedes Proxel die Flußraten der aktivierten Transitionen (die Summe der Flußraten aller Server) bestimmt und entschieden, ob die darüber erreichten Zustände zeitlos oder zeitbehaftet sind. Aus den Flußraten und der Wahrscheinlichkeit des aktuellen Proxels wird die Wahrscheinlichkeit der Folgeproxels berechnet. Zu guter Letzt ergibt sich auch die Wahrscheinlichkeit, keinen Zustandswechsel durchzuführen (siehe Algorithmus 3.1).

Die Aufteilung der Wahrscheinlichkeit in einem zeitlosen Zustand erfolgt in `handleVanishing()`. Sollte durch das Feuern einer zeitlosen Transition wieder ein zeitloser Zustand erreicht werden, wird die Methode rekursiv aufgerufen, ansonsten entsteht ein weiteres Proxel. Die Berechnung der neuen Alterungs-Matrizen für ein neues Proxel erfolgt in Methoden der Klasse `PrxNet` in Abhängigkeit vom alten und vom neuen ENAS sowie der ursprünglichen Alterungs-Matrix. Diese werden dann (wieder zurück in `ProxelAnalyzer`) mit der erreichten Markierung verknüpft und als neue Proxel in die Datenstruktur eingefügt. Das Einfügen geschieht in `addProxel`. Dort werden auch die Nachbarbeziehungen zwischen Proxels erstellt und äquivalente Proxels akkumuliert.

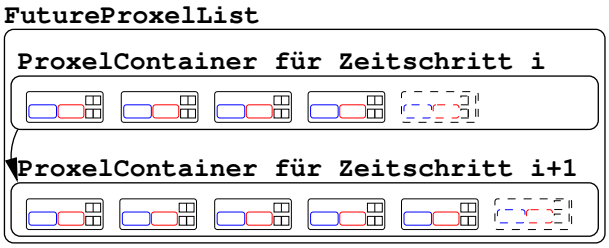

Abbildung A.2.: Datenhaltung

Die Datenhaltung der Proxels entspricht von der Klassenstruktur her der Abbildung 3.3. Die Proxels werden pro Zeitschritt in einer Instanz von `ProxelContainer` gehalten. Durch eine Instanz derselben Klasse halten auch die Markierungen Referenzen auf die ihnen zugehörigen Proxels. Die aktuelle Implementierung stellt einen Wrapper um eine `ArrayList` dar, in der die Proxels dann gespeichert werden. Für die Iteration über die Proxels eines Zeitschritts ist dies eine hinreichend effiziente Lösung, für das performante Suchen eines äquivalenten Proxels sind jedoch weitere Verbesserungen angebracht. Die Markierungen selbst werden in einer eigenen verketteten Liste gehalten, dies ist jedoch verbesserungswürdig (siehe Abschnitt A.3).

Die verschiedenen Container werden durch eine Instanz der Klasse `FutureProxel-`

`List` gespeichert. Bisher existieren maximal zwei Container in dieser Datenstruktur: der aktuell bearbeitete und der direkt darauf folgende. Diese Form der Datenhaltung ist darauf ausgelegt, leicht um variable Zeitschritte während der Proxel-Simulation erweitert werden zu können.

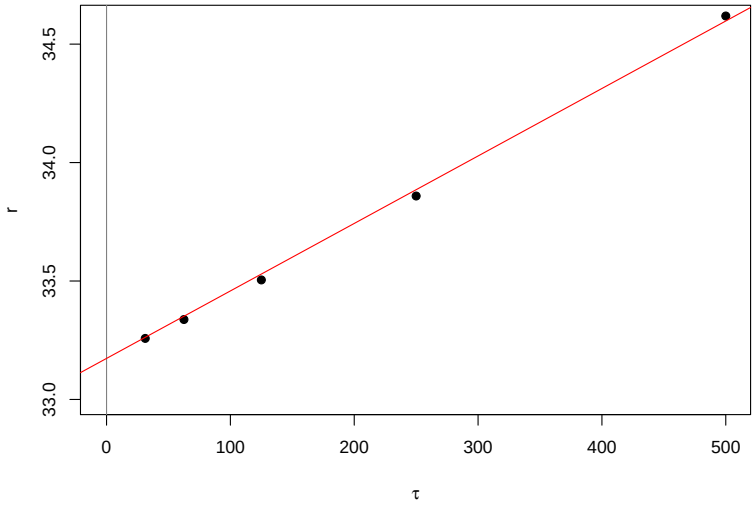

Abbildung A.3.: Regression über die Analyseergebnisse

Die Extrapolation der Ergebnisse erfolgt mittels linearer Regression über die vorhandenen Datenpunkte. Die Werte auf der x-Achse entsprechen dabei den verwendeten Zeitschrittweiten τ_i, auf der y-Achse verwendet man die approximierten Ergebniswerte r_i. Die Regressionsgerade $r = g(\tau) = a + b\tau$ beschreibt den gewonnenen Datensatz. Durch den Schnittpunkt dieser Gerade mit der Abszisse ergibt sich eine Extrapolation für ein theoretisches $\tau_0 = 0$. Die y-Koordinate dieses Schnittpunkts ist genau a, so daß sich für die Extrapolation des wahren Ergebnisses w die folgende Formel ergibt:

$$w = a = \frac{\sum_{i=1}^{n} \tau_i^2 \sum_{i=1}^{n} r_i - \sum_{i=1}^{n} \tau_i \sum_{i=1}^{n} \tau_i r_i}{n \sum_{i=1}^{n} \tau_i^2 - \left(\sum_{i=1}^{n} \tau_i \right)^2} \tag{A.1}$$

Die Speicherung aller Einzelergebnisse und die Berechnung der Extrapolation nach der soeben beschriebenen Methode ist in der Klasse `ProxelResult` implementiert. Die Extrapolation geschieht für den Nutzer transparent. Sollten nur Ergebnisse für eine einzelne Iteration vorliegen, wird keine Extrapolation vorgenommen und die Werte werden dem Anwender unverändert präsentiert.

A.2. Anleitung für die Proxel-basierte Simulation mit Expect

In Expect geschieht die Proxel-basierte Analyse in mehreren Iterationen, das heißt, es werden sequentiell mehrere Analysen mit veränderter Zeitschrittweite τ durchgeführt, aus denen dann „echte“ Ergebnisse extrapoliert werden.

Um nun mit Expect ein erstelltes Petri-Netz Proxel-basiert zu analysieren, ist in der Toolbar des Editorfensters für Petri-Netze das Proxel-Icon () aufzurufen. Es öffnet sich der Parametrierungsdialog für Proxel-basierte Simulation (Abbildung A.4). Die folgenden Optionen stehen darin zur Verfügung (in Klammern findet sich der Typ der Parameter):

- *Initialschrittweite:* Mit diesem Wert für die Zeitschrittweite τ_0 wird die erste Iteration der Proxel-Anaylse durchgeführt (`double`)
- *Skalierungsfaktor:* Der Skalierungsfaktor f verändert die Zeitschrittweite zwischen den Analysen: $\tau_{i+1} = f^{-1}\tau_i$ (`double`)
- *Extrapolationspunkte:* Der Parameter gibt an, wieviele Iterationen zur Extrapolation des echten Ergebnisses einbezogen werden. Stehen weniger Iterationsergebnisse zur Verfügung, wird der Wert intern automatisch reduziert. (`int`)
- *IRF-Werte cachen:* Gibt an, ob die errechneten numerischen Werte für die Instantaneous Rate Functions zwischengespeichert werden. Für Exponentialverteilungen und deterministischen Feuerzeiten werden keine Caches verwendet.
- *Maximale Cachegröße:* Gibt an, ob die Anzahl der zwischengespeicherten Werte begrenzt sein soll. Falls ja, steht ein Wertefeld für die Maximalanzahl zur Verfügung (`int`).
- *Schwellwert Wahrscheinlichkeit:* Gibt einen Minimalwert an, bis zu dem neue Proxels berücksichtigt werden. Proxels mit einer Wahrscheinlichkeit, die unter diesem Schwellwert liegt, werden verworfen. (`double`)
- *Simulationszeit:* Der Endzeitpunkt der Analyse. (`double`)
- *Zeitlose Sequenz:* Die maximale Anzahl von aufeinander folgenden zeitlosen Zuständen. Sollte eine Sequenz von mehr zeitlosen Zuständen auftreten, so wird dies als Endlosschleife im Modell betrachtet und die momentane Iteration wird abgebrochen. (`int`)

Alle Werte können auch im Optionsdialog eines Petri-Netz-Fensters im Expect vorab parametriert werden. Bis auf die Simulationszeit und den Wert für die zeitlose Sequenz befinden sich alle Werte im Reiter Proxel (ansonsten Simulation).

Wird der Start-Button betätigt, beendet sich der Parametierungsdialog und es öffnet sich ein (noch leerer) Ergebnisdialog. Im Hintergrund beginnt die erste Iteration. Sobald diese abgeschlossen ist, zeigt der Ergebnisdialog die statistischen Ergebnisse an.

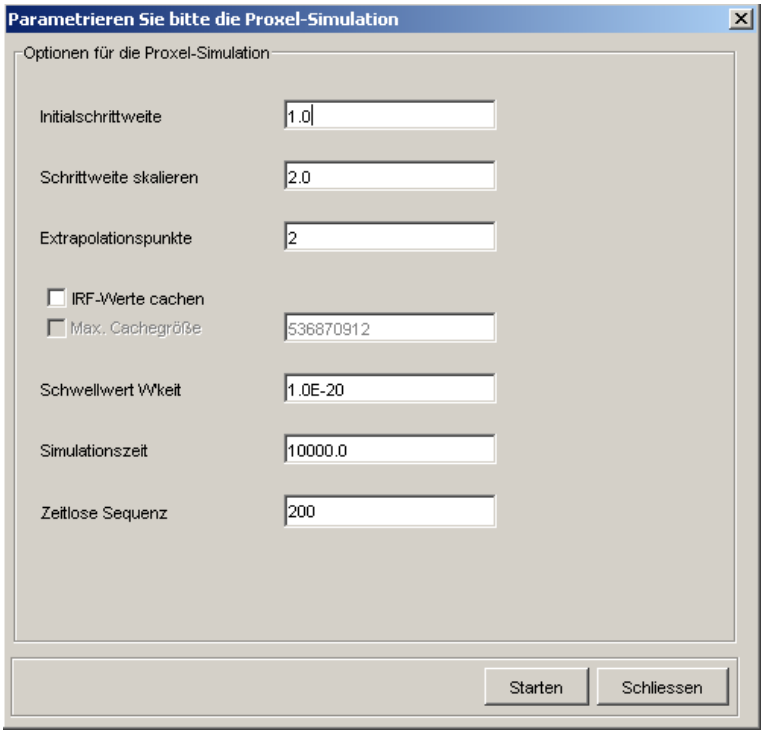

Abbildung A.4.: Der Parameterdialog

Die Darstellung entspricht der Monte-Carlo-Simulation, es werden jedoch keine Standardabweichungen präsentiert (vgl. Abbildung A.5).

Die Analyse terminiert nur dann, wenn vom Anwender der Stop-Button betätigt wird. Ansonsten wird nach jeder Iteration die Zeitschrittweite mit dem vom Anwender vorgegebenen Faktor skaliert und eine neue Iteration durchgeführt. Nach jeder abgeschlossenen Iteration werden die Ergebnisse aktualisiert. Ein Anwender erhält somit Rückmeldung über die Entwicklung der Genauigkeit bei den Ergebnissen. Er kann die Analyse jederzeit beenden, wenn die Genauigkeit seinen Ansprüchen genügt, oder aber auf weitere Iterationen warten.

A.3. Mängel der Implementierung

In der aktuellen Implementierung existieren aufgrund nicht ausreichend vorhandener mathematisch-statistischer Kenntnisse nicht für alle kontinuierlichen Verteilungsfunktionen, die durch Expect für zeitbehaftete Transitionen zur Verfügung gestellt werden, die Implementierungen für Instantaneous Rate Functions. Abgedeckt werden lediglich die Verteilungen Deterministisch, Uniform, Exponential, Weibull, Badewanne, Normal, Lognormal. Transitionen, die einer der übrigen Verteilungen (zum Zeitpunkt dieser Arbeit Cox, Erlang, GeneralErlang, HyperExponential, HypoExponential, Numeric, Polyline und Trace) unterliegen, können somit gegenwärtig nicht durch Proxel-basierte Analyse

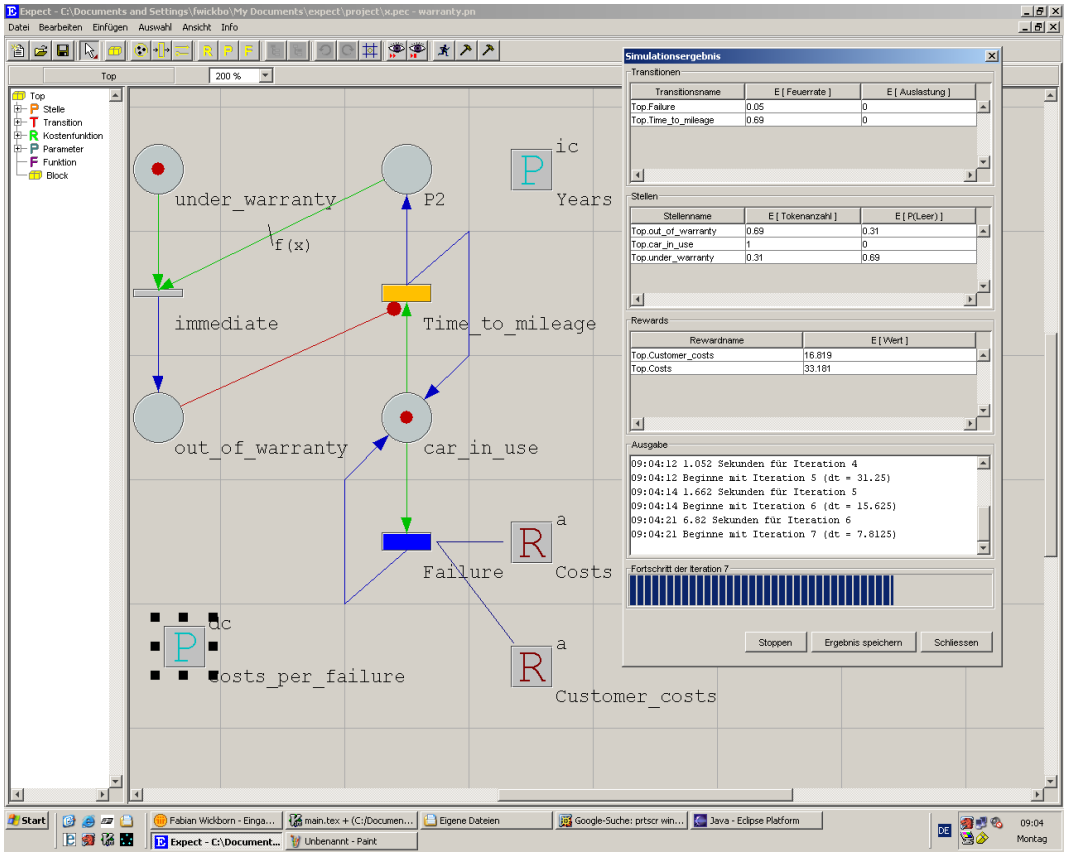

Abbildung A.5.: Expect in Aktion

verarbeitet werden. Zur Beseitigung des Mangels ist es lediglich notwendig, für die entsprechenden Verteilungsklassen die Methode `getIRF(double x, double dt)` korrekt zu implementieren.

Desweiteren ist die implementierte Speicherung der diskreten Markierung als einfache Liste suboptimal, da sich das Wiederauffinden eines Zustandes im Speicher mit linearem Aufwand als zu schwergewichtig gestaltet. Während der Bearbeitung der Aufgabenstellung lag das Hauptaugenmerk auf der korrekten Implementierung des Proxel-basierten Verfahrens und nicht auf einer optimalen Speicherung. Im Erzeuger für Erreichbarkeitsgraphen des Werkzeugs existiert ein Verfahren zur Speicherung der Zustände in einer Hash-Struktur, in dem auch auf die Massenspeicher des Computers zugegriffen wird. Dieses kann als sehr performant betrachtet werden. Es ist daher sinnvoll und von Vorteil, dieses Verfahren in Zukunft auch auf die Proxel-basierte Analyse zu übertragen.

A.4. Weitere Ideen

Neben der Beseitigung der oben angesprochenen Mängel sind zum Beispiel folgende kurzfristige Erweiterungen vorstellbar:

1. Ein automatischer Abbruch der Analyse, wenn sich die extrapolierten Ergebnisse durch eine Iteration nicht um mehr als einen vom Anwender vorgegebenen Schwellwert (zum Beispiel „bis auf drei Nachkommastellen genau“) ändern.

2. Neben der implementierten Extrapolation über die Regressionsgerade lassen sich weitere Verfahren zur Extrapolation wie zum Beispiel die Methode nach Lagrange einbauen.

Im Rahmen der Kooperation zwischen der Otto-von-Guericke-Universität und der DaimlerChrysler AG werden weitere Erkenntnisse über die Proxel-basierte Simulation ebenfalls in die Softwarewerkzeuge des Industriepartners einfließen.

Literaturverzeichnis

[Fin04] Finn, Stephan: *Quantitatve Analyse erweiterer Fehlerbäume mit nichttrivialen Basic Events.* Magdeburg, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, Mai 2004

[Ger00] German, Reinhard: *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets.* John Wiley & Sons, Inc., 2000. – ISBN 0471492582

[Gre04] Greiner, Stefan: Using Simulation to Predict Quality and Cost in the Automotive Business. In: *Proceedings of the 18th European Simulation Multiconference.* Magdeburg, Germany : SCS European Publishing House, 2004

[HGH02] Heller, Stefan ; Greiner, Stefan ; Horton, Graham: PeNeTo: A Petri Net Simulator for Fast Safety and Quality Analysis and Cost Prediction. In: *16^{th} European Simulation Multiconference: Modelling and Simulation 2002, June 3-5, 2002, Fachhochschule Darmstadt, Darmstadt, Germany.* Darmstadt, Germany, Juni 2002, S. 50–54

[Hor] Horton, Graham: *Introduction to Simulation.* Vorlesungsmaterialien. – http://www.sim-md.de/its

[Hor02] Horton, Graham: A New Paradigm for the Numerical Simulation of Stochastic Petri Nets with General Firing Times. In: *Proceedings of the European Simulation Symposium 2002.* Dresden : SCS European Publishing House, Oktober 2002

[Knu73] Knuth, Donald E.: *The Art of Computer Programming.* Bd. 1: Fundamental Algorithms. Second Edition. Addison-Wesley, Reading, Massachusetts, USA, 1973

[Knu04] Knuth, Donald E.: *The Art of Computer Programming, Pre-Fascicle 3A, A Draft of Section 7.2.1.3: Generating All Combinations.* Zeroth printing (revision 7). 2004. – http://www-cs-faculty.stanford.edu/~knuth/taocp.html

[LMH03a] Lazarova-Molnar, Sanja ; Horton, Graham: An Experimental Study of the Behaviour of the Proxel-Based Simulation Algorithm. In: *Simulation und Visualisierung 2003*, SCS European Publishing House, 2003

[LMH03b] Lazarova-Molnar, Sanja ; Horton, Graham: Proxel-Based Simulation of Stochastic Petri Nets Containing Immediate Transitions. In: *On-Site Proceedings of the Satellite Workshop of ICALP 2003 in Eindhoven, Netherlands.* Dortmund, Germany : Forschungsbericht Universität Dortmund, 2003

[LMH04] Lazarova-Molnar, Sanja ; Horton, Graham: Proxel-Based Simulation of a Warranty Model. In: *European Simulation multiconference 2004*, SCS European Publishing House, 2004

[Wic02] Wickborn, Fabian: *Erweiterung eines Petrinetzsimulators um die Fähigkeit der gleichzeitigen, parallelen Verarbeitung einer Simulationsaufgabe.* Magdeburg, Otto-von-Guericke-Universität Magdeburg, Studienarbeit, Juni 2002

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe. Es wurden keine anderen als die angegebenen Quellen verwendet. Zitate sind entsprechend kenntlich gemacht.

Magdeburg, den 16. November 2004

Fabian Wickborn