Modellvalidierung von zeitbegrenzten logistischen Prozessketten mit Interval Timed Coloured Petri Nets

Abstract: Durch Zeitrestriktionen in logistischen Prozesskettenmodellen wird die Spezifikation der Anforderungen an Dienstleistungen oder in Transportketten integrierte Informationssysteme vereinfacht. Der neue zeitliche Aspekt birgt bei Modellerstellung auch die Gefahr, unbemerkt strukturelle Fehler in das Modell zu bringen. Eine simulative Analyse garantiert nicht die Erkennung dieser Fehler. Es wird ein Ansatz basierend auf Interval Timed Coloured Petri Nets vorgestellt, der Modellierungsfehler in zeitbegrenzten logistischen Prozesskettenmodellen durch Modellvalidierung erkennen lässt.

1 Einleitung

Moderne logistische Systeme greifen auf externe Dienstleistungen und Informationsquellen zurück, insbesondere bei lose gekoppelten und selbstorganisierenden Systemen. Für diese werden zeitliche Obergrenzen in Service Level Agreements (SLAs) mit den Dienstleistern vereinbart.
Die Beschreibung und Modellierung logistischer Netzwerke kann mit Hilfe des Prozesskettenparadigmas [Kuh95] geschehen. Hier kann es schon bei der Modellierung erstrebenswert sein, die maximale Prozessdauer entsprechend den Vereinbarungen zeitlich zu begrenzen. Die SLAs wären so überprüfbar und die Alternativen bei deren Überschreitung können in das Modell mit einfließen. Timeouts in logistischen Prozesskettenmodellen sind daher ein Baustein für die Modellierung von flexiblen und reaktiven Logistiknetzen, die auf kürzfristige Veränderungen reagieren können. Eine Reaktion unterliegt zeitlichen Beschränkungen, ansonsten ist sie wirkungslos oder kontraproduktiv.
Die im Sonderforschungsbereich „Modellierung großer Netze in der Logistik“(SFB 559) [BC09] entwickelte Modellierungssprache *ProC/B* [BBF$^+$02, BBT04] formalisiert das Prozesskettenparadigma und macht macht es für Analyseverfahren wie die Simulation zugänglich. Zuletzt wurden auch Timeouts in die Sprache integriert [BBKV08].
Durch die unumgängliche Abstraktion der Realität für die Modellkonstruktion können Fehler entstehen, die das Verhalten des Modells verfälschen. Diese teilweise sporadisch auftretenden Fehler sind bei einer rein simulativen Auswertung nicht sicher erkennbar und können die Ergebnisse verfälschen. Für *ProC/B* wurden daher Validierungsverfahren [BK07, BC09, KT06] entwickelt, die strukturelle Fehler im Modell durch funktionale Analyse erkennen.
Für die Validierung von Modellen mit Timeouts wird in dieser Arbeit ein Konzept entworfen, um diese in die funktionale Modellanalyse im *ProC/B* Toolset zu integrieren. Die Problemstellung, der Ansatz und die dazu erarbeiteten Methoden werden vorgestellt.

Zu Beginn wird die Modellierungssprache *ProC/B* vorgestellt. Als neues Sprachelement werden Timeouts von Prozessdiensten und in Abschnitt 3 ausgewählte Fehlerquellen im Modell besprochen, die erst durch eine funktionale Analyse erkennbar sind. Zielvorgaben für die Validierung werden in Abschnitt 3 beleuchtet. Es folgt in Abschnitt 4 die Beschreibung der Interval Timed Coloured Petri Nets (ITCPN), welche als Grundlage für die Modelltransformation dienen. Das Vorgehen bei der Modelltransformation und Validierung von ProC/B Modellen mit Hilfe der ITCPN bildet Abschnitt 5. Die Ergebnisse werden abschließend zusammengefasst.

2 *ProC/B*

ProC/B wird in einer grafischer Notation angegeben [BBK00]. Es gibt zwei grundlegende Sprachelemente in *ProC/B*: Prozesskettenelemente (Process Chain Elements, PCEs) und Funktionseinheiten (Function Units, FUs). Die PCEs beschreiben das Verhalten und die FUs die hierarchische Struktur eines Systems. Zur Beschreibung komplexer Aktivitäten werden PCEs zu Prozessketten zusammengefasst. Funktionseinheiten stellen Entitäten im System dar, die PCEs Dienste anbieten. Atomare Funktionseinheiten sind Server zur Abbildung von begrenzten Ressourcen und Warteschlangen, Counter und Storage realisieren die Zwischenlagerung von Objekten. Darüber hinaus gibt es die Option, eigene Funktionseinheiten durch Komposition zu erstellen, Dienste der zusammengesetzten Funktionseinheiten werden durch eigene Prozessketten beschrieben, deren PCEs wieder auf lokale FUs zurückgreifen können. Diese selbstähnliche Beschreibung erlaubt den Aufbau hierarchischer Modelle.

Abbildung 2 zeigt ein Beispiel für ein Modell eines automatischen Beladesystems für Container. Die oberste Ebene des Modells wird durch die Funktionseinheit `Order_system` dargestellt. Sie beinhaltet eine Prozesskette `Container` und mit `Storage_system` eine Funktionseinheit. `Storage_system` ist eine zusammengesetzte FU mit zwei offerierten Diensten namens `send_order` und `load`.
Die Prozesskettenelemente `send_order` und `load` rufen die entsprechenden Dienste der Funktionseinheit `Storage_system` auf. Dies wird in *ProC/B* ähnlich der Syntax objektorientierter Programmiersprachen durch Angabe der Funktionseinheit, eines Punktes und des jeweiligen Dienstes realsiert, hier z.B. `Storage_system.send_order`. Das Prozesskettenelement `count_fullfilled_orders` hat keine direkte Funktion für das Modell, es zählt hier die fertig beladenen Container durch Erhöhen einer Statistik.

Die Semantik des Modelles in Abbildung 2 ist wie folgt: Die mit `Source` bezeichnete Quelle generiert alle 1000 Zeiteinheiten fünf Objekte, die die Prozesskette von links nach rechts durchlaufen. Menge und Zeit der Quelle sind durch beliebige Zufallsverteilungen parametrisierbar. Durchläuft ein Objekt ein Prozesskettenelement, so wird beim Dienstaufruf einer Funktionseinheit das Objekt an die jeweilige Prozesskette in tieferer Ebene übergeben. Ist der Dienst abgeschlossen, kehrt das Objekt zurück und führt seinen Weg durch die Prozesskette fort. Dieser endet in der Senke `Sink`, das Objekt wird hier zerstört. Gehörte die Senke einer Funktionseinheit in tieferer Hierarchieebene so würde das Objekt eine Ebene höher zum aufrufenden Prozesskettenelement gesendet.

Der Fortschritt der Modellzeit ist in *ProC/B* wie folgt definiert: Ein Dienstaufruf und die Bewegung entlang der Prozesskette sind zeitlos. Ein Zeitverbrauch findet erst an den atomaren Funktionseinheiten statt, deren Kapazität begrenzt ist und Objekte auf einen Dienst warten lassen. Die atomaren FUs vom Typ Server bieten hierzu verschiedene Bedienstrategien [BBF+02], die Bearbeitungszeit eines Objektes lässt sich über einen Parameter in der Aufrufenden PCE steuern. Zusammengesetzte Funktionseinheiten übernehmen entsprechend den kummulierten Zeitverbrauch der inneren atomaren Funktionseinheiten. Eine weitere Möglichkeit, Zeitverbrauch zu definieren, sind dedizierte DELAY Prozessketten-elemente für einfache Verzögerungen des Objektes.
Zur präziseren Modellierung bietet *ProC/B* weitere Operatoren. Die in den Prozessketten bearbeiteten Objekte können Variablen tragen, welche in der Prozesskette veränderbar sind. Prozesse können mit OR-Konnektoren und über den Variablen definierten Bedingungen in verschiedene Teilbäume verzweigen. Parallele Ausführung und Synchronisation von Prozessen kann mit Hilfe von AND-Konnektoren stattfinden.

2.1 Timeouts

Bei Portierung der Modellierungssprache auf die Simulationsbibliothek *OMNeT++* wurde *ProC/B* um die Semantik der Timeouts erweitert [BBKV08]. Sie werden als Attribut an PCEs angegeben und begrenzen die Zeit, die ein Dienstaufruf an einer Funktionseinheit maximal dauern darf. Abbildung 1 zeigt die Syntax eines Prozesskettenelementes

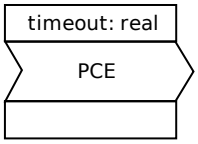

Abbildung 1: Prozesskettenelement mit Timeout

in *ProC/B* mit Timeout-Angabe. Eine maximale Laufzeit t_{max} wird in einen Rechteck überhalb des Prozesskettenelements spezifiziert. Ist die Dauer des Dienstaufrufes zum Zeitpunkt α kürzer als t_{max} so verhält sich das Prozesskettenelement normal. Zusätzlich wird ein Flag `in_time` an das Objekt angehängt. Ist Zeit t_{max} seit dem Aufruf verstrichen und dieser noch nicht beendet, so tritt der Timeout ein und es gilt folgende Semantik: Eine Kopie des verspäteten Objektes wird vom Prozesskettenelement bei $\alpha + t_{max}$ weitergesendet. Dieses entspricht in seinen Variablen dem Objekt zum Zeitpunkt α. `in_time` wird auf `false` gesetzt. So ist eine Behandlung des Timeouts durch bedingte Verzweigung der Prozesskette möglich.
Das Originalobjekt wird erst bei der Rückkehr zur wartenden Funktionseinheit verworfen. Auf die Option, das Objekt sofort beim Eintritt des Timeouts aus dem Modell in beliebiger Hierarchieebene zu entfernen, wurde verzichtet. Es bestünde die Gefahr, dass das Objekt in einen Synchronisationsvorgang eingebunden wäre, ein Entfernen des Objektes würde den betreffenden Modellteil unweigerlich blockieren. Eine detaillierte Beschreibung die

Implementierung von Timeouts für *ProC/B* auf *OMNeT++* ist in [BBKV08] zu finden.

Das Beispiel wurde um Timeouts erweitert (Abbildung 3). Die Prozesskettenelemente `send_order` und `load` haben jetzt eine Zeitvorgabe von 60 Zeiteinheiten (ZE) für die Auftragsübergabe bzw. 2000 ZE zum Beladen des Containers. Hinter jedem Prozesskettenelement ist eine bedingte Verzweigung angebracht, die die Variable `in_time` für eine Fehlerbehandlung testet[1]. Diese besteht in diesen Modell aus Mitzählen der abgebrochenen Dienstaufrufe durch Statistik `failed_orders`, weitere Reaktionen auf den Fehlerfall sind denkbar.
In Abbildung 4 ist nun auch der innere Aufbau von Funktionseinheit `Storage_system` gezeigt. Die Prozesskette `send_order` empfängt die Bestellliste, was durch ein Delay-Element mit 50 ZE realisiert ist. Weiterhin legt sie diese in das Auftragsverwaltungssystem `Database` ab, was 20 ZE beansprucht.
Die zweite Prozesskette `load` besitzt ein auf 400 ZE begrenztes Prozesskettenelement `pick_order_items`. Es startet das Kommissioniersystem `Commission_system`. Passiert dabei kein Timeout, so wird dies in `Database` vermerkt (100 ZE), ebenso der Fehler durch `generate_error`. Für das Kommisioniersystem nehmen wir an, dass nur stochastische Annahmen zur Bearbeitungszeit bekannt sind. Auf der rechten Seite in Abbildung 4 ist die Funktionseinheit `Commission_system` dargestellt. Offensichtlich ist ihre einzige Funktion eine negativ exponentiell verteilte Verzögerung des Auftrages mit Parameter $\lambda = \frac{1}{200}$. Dies ergibt zwar den Erwartungswert 200 ZE, die mögliche Dauer ist aber unbegrenzt. Gäbe es keine Timeouts im Modell, so würde `Order_system.load` eine potentiell unbegrenzte Ausführungsdauer haben.

3 Validierungsziele von zeitbegrenzten Prozessketten

Die Modellvalidierung für Prozesskettenmodelle hat das Ziel, kritische Fehler im Modell noch vor der Simulation zu erkennen. Dabei steht nicht eine Überprüfung der Güte von Eingabedaten im Vordergrund, sondern ob das vom Benutzer erstellte Modell des Systems ein plausibles und maschinell auswertbares Verhalten zeigt. Dies ist hilfreich, da sich Fehler, obwohl schon strukturell im Modell vorhanden, erst nach zeitaufwendigen Simulationsläufen auftreten. Entweder ist der Fehler so gravierend, dass die Simulation niemals zu einem Ergebnis kommt, oder der Fehler betrifft nur einen Teil des Modells und verfälscht so unbemerkt die Messergebnisse.
Für das ProC/B Toolkit wurden Verfahren entwickelt [BK07, BC09, KT06], um im Modell nach bestimmten Fehlerarten zu suchen und diese zu erkennen. Es basiert auf einer Transformation der Modelle in Petri-Netze, diese werden dann einer funktionalen Analyse unterzogen. Die Validierung von *ProC/B* Modellen basiert auf Beschränktheit und Lebendigkeit als Eigenschaft der resultierenden Petri-Netze [KT06], welche durch Invarianten oder den Erreichbarkeitsgraph des Netzes ermittelbar sind. Beispiele für die durch Modellvalidierung erkannten Fehler in logistischen Prozessketten sind nichtplausible Angaben zur Anlieferungs- und Auslieferungsmenge in Lagern und Verklemmungen durch wechselseitiges Warten auf Ressourcen. In [BK07] wird beschrieben wie nicht-ergodische

[1] Prefix `data.` dient zur Unterscheidung von globalen Variablen in *ProC/B*.

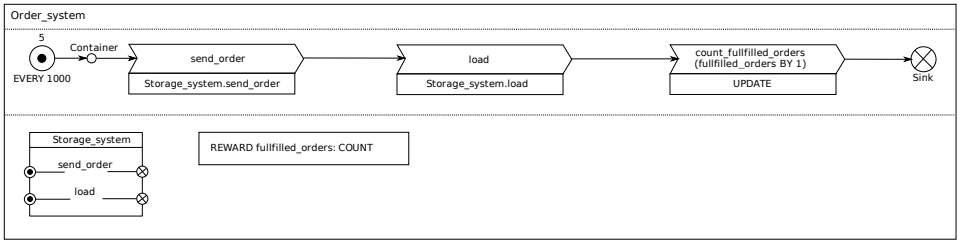

Abbildung 2: Ein einfaches Bestellsystem

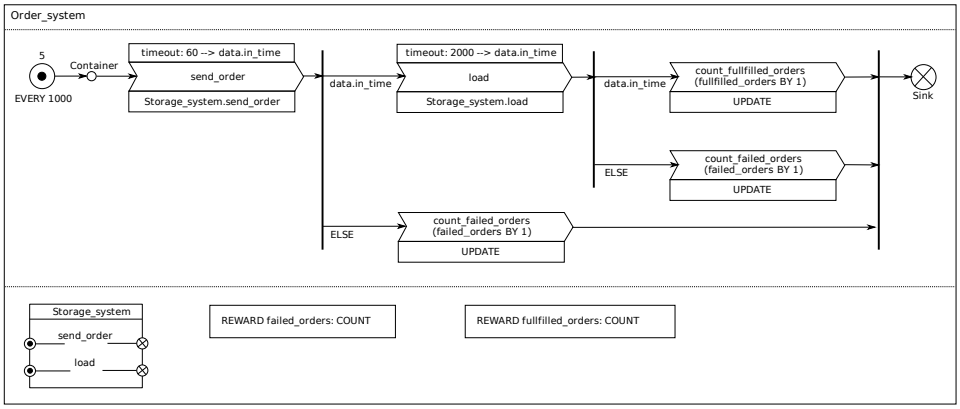

Abbildung 3: Bestellsystem mit Timeouts

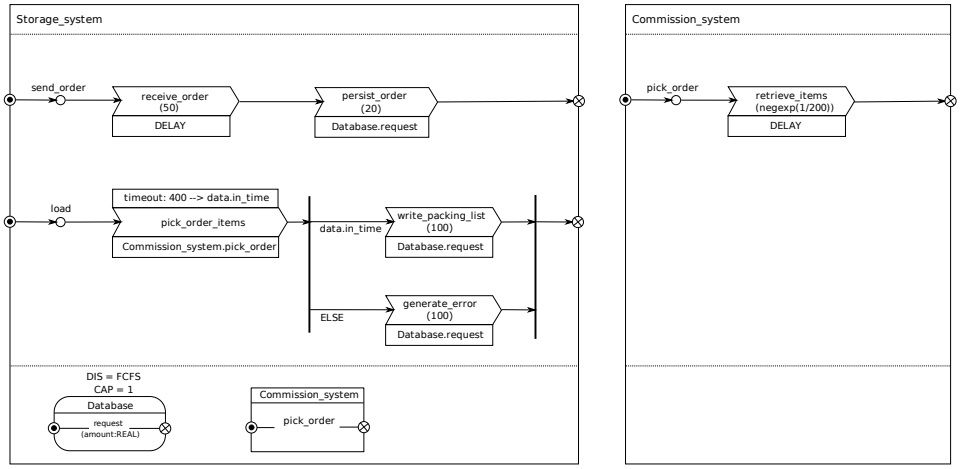

Abbildung 4: Funktionseinheiten `Storage_system` und `Commission_system`

Modellteile erkannt werden, bei denen sich kein stationärer Zustand einstellt.

Für das neue Sprachelement der Timeouts in *ProC/B* wird nun ein Ansatz vorgestellt, welcher Modellierungsfehler in Bezug auf Timeouts erkennt. Zwei Validierungsziele werden dazu im Folgenden beschrieben.

3.1 Restriktive Zeitgrenzen

Ein Fehler beim Setzen von Zeitgrenzen kann sein, die Vorgaben zu restriktiv zu halten. Dieser Fall tritt auf, wenn der Timeout t_{max} des aufrufenden Prozesskettenelementes so knapp bemessen ist, dass die Summe aller minimal möglichen Bearbeitungszeiten in der aufgerufenen Funktionseinheit den Wert t_{max} übersteigt. Ein solches Prozesskettenelement würde bei jeder Aktivierung nur einen Timeout erhalten, eine Beendigung des Arbeitsschrittes wäre nicht möglich. Ein Ziel der Validierung ist das Erkennen dieser unerfüllbaren Anfordungen.
Dem aufmerksamen Leser mag aufgefallen sein, dass die Zeitgrenzen des Prozesskettenelementes `send_order` in Abbildung 3 nicht erfüllbar sind. Die Grenze liegt bei 60 ZE, die in Dienst `send_order` angegebenen PCEs `receive_order` und `persist_order` brauchen zusammen minimal 70 ZE. Hier liegt ein Modellierungsfehler vor, der jedoch einen Simulationslauf nicht verhindern würde. Erst nach der Simulation würde sich anhand Rewards `failed_orders` zeigen, dass jede Auftragsbearbeitung abgebrochen ist. Dieses Beispiel zeigt nur eine Hierarchiestufe, das generelle Ziel der Validierung ist eine Überprüfung in beliebiger Tiefe.
Zur Korrektur des Fehlers wird die Timeout-Zeit von `send_order` auf 200 ZE heraufgesetzt, was aber einen neuen Fehler in das System bringt. Dieser kann mit dem zweiten hier vorgestellten Validierungsziel gefunden werden.

3.2 Absicherung von Zeitvorgaben

Das zweite Ziel der Validierung soll sein, die Absicherung von Zeitvorgaben unter einer definierten Systemlast zu überprüfen. Dies kann zur Eingrenzung von Antwortzeiten in Realzeitsystemen oder zur Überprüfung von Service Level Agreements (SLAs) wünschenswert sein. In einen hierarchischen *ProC/B* Modell könnte eine Absicherung der Zeitvorgabe erreicht werden, wenn auf jeder Hierarchiestufe die Arbeitsschritte zeitbegrenzt wären. Oft ist dies aber nicht möglich oder bei der Modellierung nicht sinnvoll.
Die Quelle des Beladungsprozess aus Abbildung 3 generiert alle 1000 ZE fünf Container gleichzeitig. Innerhalb dieses Zeitintervalls müssen die Container das System passieren da dies dem Fahrplan eines Transportmittels entspricht, welches die Container mitnimmt. Es lässt sich zeigen, dass diese Zeitgrenze verletzt werden kann.
Ein kompletter Durchlauf eines einzigen Containers durch das Modell in Abbildung 3 und Abbildung 4 dauert 570 ZE und liegt so innerhalb der Vorgabe. Dies setzt sich zusammen aus 70 ZE für `send_order` und in `load` höchstens 400 ZE für `pick_order_items`.

Unabhängig vom Eintreten des Timeouts von `pick_order_items` dauert ein Verbuchen in die Datenbank 100 ZE. Die Kommunikation mit dieser hypothetischen Datenbank ist jedoch der Schwachpunkt bei höherer Systemlast. Sie arbeitet nach dem First Come First Serve Prinzip und kann nur eine Anfrage gleichzeitig bedienen, andere müssen warten. Laufen nun wie im Modell spezifiziert fünf Container durch das System so verlängert sich die Antwortzeit lastabhängig für den zweiten bis fünften Container. Das Prozesskettenelement `send_order` könnte der fünfte Container erst nach 150 ZE verlassen. In `load` verweilt er sogar bis zu 900 ZE, da sich das Schreiben auf die Datenbank auf 500 ZE verlängert und `pick_order_items` auf 400 ZE begrenzt ist.
Der fünfte Container würde mit maximal 1050 ZE nicht die Zeitvorgabe von 1000 ZE einhalten können. Die Auswertung einer Simulation des Modells führt nicht zwangsläufig zu dieser Erkenntnis. Da in `Commission_system` der Erwartungswert der Exponentialverteilung nur 200 ZE ist sind Simulationsläufe denkbar, in denen `pick_order_items` nicht nur unter der Timeout-Grenze, sondern auch unter 350 ZE bleiben würde. Es besteht das Risiko, dass die Verletzung der Zeitvorgabe von 100 ZE nicht auffällt. Mit dem hier vorgestellten Validierungsansatz lässt sich dieser Modellierungsfehler sicher und ohne Simulation erkennen.

3.3 Vorgehen bei Validierung von Timeouts

Die Modellvalidierung von *ProC/B*-Modellen mit Timeouts basiert auf einer Transformation der Prozessketten in eine spezielle Variante der Petri-Netze, der Interval Timed Coloured Petri Nets (ITCPN) [vdA93]. Die ITCPN sind gefärbte Petri Netze mit Zeitmarkierungen an den Tokens und Zeitintervallen an Transitionen [vdA93]. Ihre Besonderheit liegt darin, Zeit ohne Zufallsverteilungen angeben zu können. Die oberen Grenzen der Intervalle bestimmen, wann die nächste Transition spätestens aktiviert und sofort feuern muss. Sie eignen sich daher zur Überprüfung von Zeitgrenzen in Modellen.
Die Validierung der Modelle findet mit Hilfe des reduzierten Erreichbarkeitsgraphen der Zustandsklassen des ITCPN statt. Aus ihm lassen sich Eigenschaften des Netzes abgelesen [vdA94] bzw. ausschließen. Die erkannten Merkmale können auf das *ProC/B*-Modell zurück übertragen werden.

Ebenso sind die Time Petri Nets nach Merlin [MF76] eine Möglichkeit, zeitabhängige System zu validieren [BD91]. In ihnen kann eine Transition nach Aktivierung innerhalb eines Intervalls feuern, zuvor kann sie jedoch von anderen Transitionen deaktiviert werden. Die Time Petri Nets würden sich auch zur Validierung von *ProC/B*-Modellen eignen. Es wurde aber dagegen entschieden, da Time Petri Nets keine Färbung der Tokens besitzen. Aufgrund der Komplexität vieler logistischer Modelle wird die kompaktere Darstellung [Jen03] von gefärbten Netzen bevorzugt.

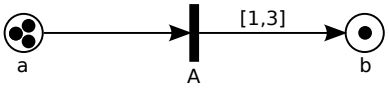

Abbildung 5: Beispiel eines ITCPN

4 Interval Timed Coloured Petri Nets

Im folgenden wird die Definition der Interval Timed Coloured Petri Nets nach van der Aalst aus [vdA93] vorgestellt. Die Beschreibung der ITCPN ist nur soweit ausgeführt wie sie zur Konstruktion eines Erreichbarkeitsgraphen notwendig ist. Für eine vollständige Definition verweisen wir auf [vdA93, vdA94, Bou08]
Zunächst werden für die Beschreibung der ITCPN Zeitintervalle benötigt [vdA93]: Die reele Zeitmenge ist $TS = \{x \in \mathbb{R} \mid x \geqslant 0\}$. Ein Intervall ist gegeben durch zwei Zeitangaben: $INT = \{[y, z] \in TS \times TS \mid y \leqslant z\}$ Grundlegend ist auch das Konzept der Multisets (MS) [Jen03]. Ein Element a kommt in Multiset b mit der Häufigkeit $b(a)$ vor. A_{MS} ist die Menge aller Multisets über Menge A [vdA93].
Die ITCPN werden in [vdA93, Bou08] wie folgt definiert: Ein Interval Timed Coloured Petri Nets ist ein Tuple $N = (\Sigma, P, T, C, CT, F, tm_0)$ mit den folgenden Eigenschaften:

- Σ ist eine endliche Menge von Färbungen
- P ist eine endliche Menge an Stellen
- T ist eine endliche Menge an Transitionen, $P \cap T = \emptyset$
- C ist eine Farbfunktion. Sie ist definiert von P nach Σ, $C \in P \rightarrow \Sigma$ und ordnet jeder Stelle eine Menge von Farben zu.
- $CT = \{\langle p, v\rangle | p \in P \wedge v \in C(p)\}$ is die Menge aller möglichen farbigen Tokens (ohne Zeit).
- F ist die Transitionsfunktion. Sie ist definiert von Transitionen T in beliebige Funktionen. Wenn $t \in T$ dann $\mathrm{dom}(F(t)) \rightarrow (CT \times INT)_{MS}$, wobei $\mathrm{dom}(F(t)) \subseteq CT_{MS}$ den partiellen Definitionsbereich angibt [Bou08].
- $tm_0 \in (CT \times TS)_{MS}$ ist der initiale Zustand [Bou08].

Ein Token ist in ITCPNs durch drei Attribute definiert: [vdA94] seine Position $p \in P$, seine Färbung $v \in C$ und einen Zeitstempel $x \in TS$. Es lässt sich als Tripel $\langle\langle p, v\rangle, x\rangle$ notieren. Der Zeitstempel des Tokens ist die zeitliche Verzögerung des ITCPN, er bestimmt, ab wann das Token verfügbar ist.
Stellen und Transitionen sind durch gerichtete Kanten verbunden. Diese sind durch die Transitionsfunktion F implizit über den Definitionsbereich und die Abbildung gegeben. Sie nimmt hier die Stelle der Inzidenzfunktionen [Jen03] in anderen Netzarten ein [vdA93].
Abbildung 5 zeigt ein kleines ITCPN mit Stelle *a* im Vorbereich und *b* im Nachbereich

einer Transition *A*. Die Stellen sind mit Tokens der Farbe r versehen. Die Markierung ist somit $3\langle a, r\rangle + \langle b, r\rangle$.
Die formale Semantik der ITCPN wird über Zustände und Ereignisse definiert. Eine Markierung $m \in CT_{MS}$ ist ein Multiset von Tokens. Ein Zustand $tm \in (CT \times TS)_{MS}$ ist eine Markierung mit Zeitstempeln an den Tokens [Bou08]. Übergänge zwischen Zuständen finden durch das Feuern von Transitionen statt. Der Zustandsübergang durch Feuern von $t \in T$ wird als Ereignis $e = \langle t, b_{in}, b_{out}\rangle$ zusammengefasst, $b_{in}, b_{out} \in (CT \times TS)_{MS}$. E ist die Menge aller Ereignisse. Es werden dabei b_{in} Tokens aus dem Vorbereich von t entfernt und b_{out} Tokens im Nachbereich erzeugt. Wenn $\langle\langle p, v\rangle, x\rangle \in b_{out}$ dann erzeugt e ein Token auf Stelle p mit Färbung v und Verzögerung x.
Die Besonderheit der ITCPN liegt in der Auswahl der Verzögerung x. Der Zeitstempel wird aus einem Intervall $[a, b] \in INT$ gewählt, welches jeder ausgehenden Kante einer Transition durch F zugewiesen ist. In Abbildung 5 ist dieses Intervall $[1, 3]$.
Ein Ereignis $\langle t, b_{in}, b_{out}\rangle$ muss aktiviert sein bevor es eintreten kann. Dies entspricht der Aktivierung einer Transition in anderen Varianten von Petri-Netzen, geschieht aber hier in den den Stufen aktiviert („enabled") und zeitaktiviert („time enabled") [vdA93].
Ein Ereignis $\langle t, b_{in}, b_{out}\rangle \in E$ ist aktiviert in Zustand tm, wenn ausreichend Tokens im Vorbereich vorhanden sind, $b_{in} \leqslant tm$, und diese so auf den Stellen positioniert sind, dass die zeitlose Markierung m von b_{in} zum Definitionsbereich der Transitionsfunktion gehört: $m(b_{in}) \in dom(F(t))$.
Da hier die Zeitstempel der Tokens noch keine Rolle spielen, können mehrere Ereignisse gleichzeitig aktiv sein. Erst ein zeitaktiviertes Ereignis darf eintreten. Die Aktivierungszeit

$$ET(\langle t, b_{in}, b_{out}\rangle) = max_{\langle\langle p,v\rangle,x\rangle \in b_{in}} x$$

eines Ereignisses $\langle t, b_{in}, b_{out}\rangle$ ist das Maximum aller Zeitstempel der beim Feuern einer Transition t verbrauchten Tokens. Ein Ereignis ist zeitlich aktiviert , wenn es kein weiteres Ereignis mit einer niedrigeren Aktivierungszeit gibt. Die Maximumsbildung sorgt dafür, dass auf Stellen platzierte Tokens erst nach Ablauf der Verzögerungszeit x sichtbar und für ein Ereignis nutzbar werden.
Im Beispiel in Abbildung 5 ist zuvor das Ereignis $\langle A, \langle\langle a, r\rangle, 2\rangle, \langle\langle b, r\rangle, 1.7\rangle\rangle$ zum Zeitpunkt 2 eingetreten. Das Token auf *b* wird, unter Berücksichtigung des Intervalls $[1, 3]$, erst 1.7 Zeiteinheiten nach dem Ereignis verfügbar sein. Der neue Zustand ist $3\langle\langle a, r\rangle, 3\rangle + \langle\langle b, r\rangle, 3.7\rangle$. Als nächstes werden zum Zeitpunkt 3 die drei anderen Tokens auf *a* verfügbar und das nächste Ereignis wird zeitaktiviert.

Ein Zustand tm_2 wird direkt erreichbar von tm_1 genannt durch das Auftreten des Ereignisses $e = \langle t, b_{in}, b_{out}\rangle$. Die Schreibweise ist $tm_1 \xrightarrow{e} tm_2$.

4.1 Erreichbarkeitsgraphen

Erreichbarkeitsgraph $RG = (N, V)$ eines ITCPN enthält für jeden erreichbaren Zustand $tm_i \in (CT \times TS)_{MS}$ einen Knoten $n_i \in N$. Eine gerichtete Kante $v_j \in V$ zwischen n_1 und n_2 gibt an, dass durch Eintreten von Ereignis $e = \langle t, b_{in}, b_{out}\rangle$ der Zustandsübergang $tm_1 \xrightarrow{e} tm_2$ möglich ist.

Ein Erreichbarkeitsgraph kann aus dem Erreichbarkeitsbaum generiert werden [BK02]. Statt mit erreichbaren Markierungen wird bei den ITCPN mit den zeitbehafteten Markierungen, also Zuständen gearbeitet. Der Erreichbarkeitsbaum enthält zu Beginn nur den initialen Zustand tm_0 als Wurzel. Für jedes Ereignis $e_j \in E$ mit $tm_0 \xrightarrow{e_j} tm_i$ wird eine Kante v_j eingefügt und Blatt n_i angehangen. Dies wiederholt sich an den Blättern als neue Ausgangsknoten. Der Graph wird erstellt, indem schon vorher erreichte Zustandsknoten verschmolzen werden.

4.2 Modified Transition System Reduction Technique

Die Modified Transition System Reduction Technique (MTSRT) erstellt einen reduzierten Erreichbarkeitsgraphen für ein ITCPN [vdA94]. Ein nach obiger semantischer Definition der ITCPN konstruierter Erreichbarkeitsgraph wäre unendlich groß. Beim Eintreten eines Ereignisses $\langle t, b_{in}, b_{out}\rangle$ wird für jedes Token $\langle\langle p, v\rangle, x\rangle \in b_{out}$ eine Verzögerung x aus Zeitintervall $[a, b] \in INT$ gesetzt. Für das Intervall $[a, b]$ mit Grenzen aus $\mathbb{R}$ sind unendlich viele Werte für x möglich. In [vdA93] wird daher die Semantik des ITCPN so verändert, dass Zustandsklassen durch Aggregierung von Zuständen gebildet werden können. Ein mit Hilfe der MTSRTerstellter Erreichbarkeitsgraph eines ITCPN verwendet Zustandsklassen als Knoten.
Der Grundgedanke der MTSRT ist, dass Tokens nicht mehr nur einen Zeitstempel mit sich führen, sondern ein Zeitintervall [vdA93]. Die früheste Verfügbarkeit eines Tokens zur Aktivierung eines Ereignisses wird nun als Zeitspanne definiert. Der Zustand tm wird somit zu einer Zustandsklasse: $\overline{tm} = (CT \times INT)_{MS}$.

Somit muss auch die Aktivierungszeit eines Ereignisses sich jetzt an der unteren und oberen Intervallgrenze aller Tokens orientieren. Die minimale und die maximale Aktivierungszeit eines Ereignisses $\langle t, b_{in}, b_{out}\rangle \in \overline{E}$ sind:

$$ET_{min}(\langle t, b_{in}, b_{out}\rangle) = max_{\langle\langle p,v\rangle,[y,z]\rangle \in b_{in}} y$$

$$ET_{max}(\langle t, b_{in}, b_{out}\rangle) = max_{\langle\langle p,v\rangle,[y,z]\rangle \in b_{in}} z$$

Ein Ereignis e in Zustandsklasse tm_C kann zwischen $ET_{min}(e)$ und $min\{ET_{max}(e) \mid e \in \overline{E}$ und e ist aktiviert in $tm_C\}$ zeitaktivert sein und eintreten. Spätere Zeitpunkte gehören zu anderen Zustandsklassen.
Mit dieser veränderten Semantik der ITCPNs lässt sich der reduzierte Erreichbarkeitsgraph eines Netzes basierend auf Zustandsklassen erzeugen. Ist das Netz beschränkt, so wird auch der Erreichbarkeitsgraph endlich sein. Das Verfahren wird ausführlich in [vdA93, vdA94] sowie darauf aufbauend in [Bou08] vorgestellt.

5 Modelltransformation

Im Folgenden werden nun die Transformation von zwei *ProC/B*-Elementen in ITCPN beschrieben, die von der Einführung von Timeouts in die Sprache betroffen sind. *ProC/B*-

Elemente ohne Zeitverbrauch werden hier nicht dargestellt, ihr ITCPN entspricht der bisherigen Transformation [BK07, KT06] in hierarchische gefärbte Petri-Netze [Jen03].
Zuvor sind die Intervallgrenzen für die ITCPN aus Zeitangaben im Modell zu bestimmen.

5.1 Bestimmung der Zeitintervalle

Die ITCPN können keine Zufallsverteilungen für die Zeit beinhalten, sie haben nur Intervalle. Viele *ProC/B*-Modelle beinhalten jedoch Zufallsverteilungen, die nun auf Intervallgrenzen sinnvoll abgebildet werden müssen. Insgesamt sieht die Sprache *ProC/B* keinen direkten Weg vor, Zeit durch Intervalle zu spezifizieren. Zufallsverteilungen bei der Modelltransformation einfach zu entfernen, würde das Modell verfälschen. Daher werden für die Transformation von *ProC/B* zu ITCPNs die Intervalle wie folgt bestimmt:

- Konstante Werte a werden als Intervall $|a, a|$ angenommen. Dies ist insbesondere bei Spezifikationen für Timeouts der Fall, da diese als Konstanten angegeben werden.
- Ist eine Gleichverteilung zwischen a und b gegeben, so lautet das Intervall $|a, b|$
- Für andere Verteilungen, deren Dichtefunktion bestimmbar ist, werden auf Intervallgrenzen unter Vorgabe eines Parameters ϕ mit $0 < \phi < 1$ abgebildet. Sei $f(x)$ die Dichtefunktion und E der Erwartungswert der Verteilung, dann sind die Intervalgrenzen $|a, b|$ derart bestimmt, dass gilt:

  $$\int_a^E f(t)\,dt = \frac{\phi}{2} \text{ sowie } \int_E^b f(t)\,dt = \frac{\phi}{2}$$

  Somit werden um den Erwartungswert fixierte Grenzen bestimmt, in die Anteil ϕ aller Werte der erreichbaren Zufallszahlen fällt. Ein möglicher Wert wäre z.B. $\phi = 0,9$. Wird die obere Grenze b nicht sinnvoll bestimmbar, so wird sie mit $b = \infty$ angenommen, z.B. Exponentialverteilung mit $\lambda = 1$ ergibt $\phi = 0.9$ das Intervall $|\frac{1}{5}, \infty|$.

Eine Aussage über die Fehlerfreiheit des Modells kann nur unter Berücksichtigung des Parameters ϕ getroffen werden, da das Modellverhalten außerhalb der Intervallgrenzen nicht betrachtet wird.

5.2 Verzögernde Prozesskettenelemente

Delay-PCEs verzögern Prozesse in *ProC/B* um eine Zeitspanne, die Konvertierung in ein ITCPN ist somit einfach. Ein ITCPN des Delay-Elementes zeigt Abbildung 6. Im ITCPN wird ein Token auf Stelle *a* um eine Zeitspanne verzögert, die aus dem Intervall [`lower, upper`] stammt. Die Auswahl der Intervallgrenzen geschieht nach dem Verfahren aus Abschnitt 5.1. Erst nach Ablauf der Zeitspanne steht es auf Stelle *b* im Nachbereich zur Verfügung und bewirkt so die Verzögerung.

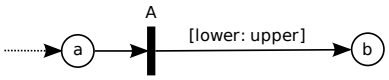

Abbildung 6: ITCPN eines Delay Elements

5.3 Aufrufende Prozesskettenlemente

Abbildung 7 zeigt das ITCPN eines aufrufenden Prozesskettenelementes mit Timeout. Im *ProC/B*-Modellen haben diese PCEs die Fähigkeit, das Eintreten des Timeouts durch die boolesche Variable `in_time` zu markieren und damit eine Fehlerbehandlung zu ermöglichen (vgl. Abbildung 3). Diese Variable kann nicht in die ITCPN übernommen werden, stattdessen wird hier über die Struktur des Netzes mit zwei Ausgängen unterschieden. Die Alternative wäre gewesen, den Timeout durch die Färbung der Tokens zu markieren. Dies hätte jedoch zu einer unnötigen Vergrößerung des Zustandsraumes geführt.
Stelle *a* nimmt das Token an, welches ein eintreffendes Objekt symbolisiert. Stelle *b* symbolisiert das Subnetz der aufzurufenden Funktionseinheit [HJS91, BK07], Stelle *d* speichert die unveränderte Kopie des Objektes, welche im Falle des Timeouts das Original ersetzt (vgl. Abschnitt 2.1). In Stelle *c* wird eine Markierung abgelegt, die später einen wechselseitigen Ausschluss zwischen pünktlichen und unpünktlichen Objekt erzwingt. Kehrt das Objekt aus dem Subnetz bei Stelle *b* zeitig zurück so feuert Transition *B* und verbraucht dabei das Token von *c* und erzeugt je eines auf *e* und *g*. Das Token auf *g* dient dazu, *E* zu aktivieren und das Token für den Fall des Timeouts auf Stelle *d* zu entfernen. Dies kann erst nach *t_max* Zeit geschehen, da erst hier das Token auf *d* verfügbar wird.
Der Timeout ist spiegelbildlich zum Normalfall aufgebaut, mit der Ausnahme dass Transition *C* feuern kann nachdem in Stelle *d* das Token verfügbar wurde. Stelle *f* erhält ein Token, sodass Transition *D* später das unpünktliche Objekt aus *b* entfernen kann.

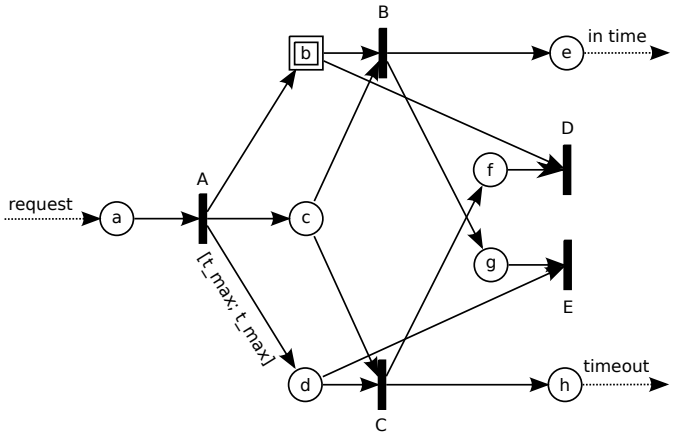

Abbildung 7: ITCPN eines Prozesskettenelementes mit Timeout

5.4 Erkennen der Validierungsziele im Erreichbarkeitsgraphen

Zu eng gesetzte Zeitvorgaben bewirken das ausschließliche Auftreten eines Timeouts am Prozesskettenelement (siehe Abschnitt 3.1). Dies geschieht schon bei minimaler Last wenn nur einen Objekt die Prozesskette durchläuft. Im abgeleiteten ITCPN N entspricht dies der Situation, dass in den Subnetzen für PCEs (Abbildung 7) nur Transition C für den Fehlerfall aktiviert werden kann, Transition B niemals. Transition B ist somit tot in N.
Für die Validierung wählen wir die initiale Markierung TM_0 so, dass nur für ein Modellobjekt ein Token enthalten ist. Der Graph der erreichbaren Zustandsklassen $RG(N, TM_0)$ wird damit erstellt.
Eine tote Transition B ist nicht im Erreichbarkeitssgraphen $RG(N, TM_0) = (V, E)$ als Kante $e_B \in E$ enthalten, ansonsten gäbe es einen Zustand $v \in V$ in dem B aktiviert werden könnte.
Sei $T_B(N)$ die Menge aller in ITCPN N enthaltenen Transitionen, die wie Transition B in Abbildung 7 die rechtzeitige Beendigung eines durch eine PCE aufgerufenen Dienstes repräsentieren. In einen *ProC/B*-Modell sind die Zeitgrenzen nicht zu restriktiv gesetzt, wenn gilt: $T_B(N) \cap E \neq \emptyset$ für E aus $RG(N)$. Die Prozesskettenelemente, deren Transitionen in der Menge $T_{fail} = T_B(N) \cap RG(N, TM_0)$ enthalten sind, besitzen den Modellierungsfehler einer zu strikten Zeitbegrenzung, sie produzieren nur Timeouts.

Das zweite Validierungsziel ist die Absicherung von Zeitvorgaben (siehe Abschnitt 3.2). Es soll geprüft werden, ob eine festgelegte Anzahl an Objekten mit dem modellierten System innerhalb einer Zeitschranke bearbeitet werden kann. Hierzu kann nicht mehr über das Feuern der Transitionsmenge T_C argumentiert werden, da Timeouts Bestandteil der Modellierung sind. Die Validierung geschieht daher über die Intervalle der Zustandsklassen, in denen die zu untersuchende Menge an Objekten auf einer Stelle im Ausgangsbereich des Systems liegt. Dieses Vorgehen ist abgeleitet von der in [vdA94] gezeigten Methode zur Berechnung der Performanz von Systemen mit Hilfe von ITCPNs. Wir setzen voraus, dass das Modell an sich keine gravierenden Fehler enthält (z.B. totaler Deadlock), der die Bearbeitung der Objekte verhindert. Abbildung 8 zeigt das abstrakte ITCPN eines Modelles mit Eingangsstelle *a* und Ausgangstelle c. Die Platzhalterstelle *b* steht für den Rest des gesamten Systems.
Es soll die Zeitvorgabe t_{max} für n Objekte getestet werden. Die Startmarkierung TM_0 des ITCPN wird so gewählt, dass an der Eingangsstelle *a* in Abbildung 8 n Tokens mit gleicher Zeitmarkierung liegen. Der Erreichbarkeitsgraph $RG(N, TM_0)$ wird gebildet. Im Graphen gibt es mindestens eine Zustandsklasse, in dem die n Tokens auf der Ausgangsstelle c liegen. Es kann hier mehrere Klassen geben, da noch im Inneren des Systems auch nach Ablage der n Tokens Transitionen feuern können.
Es werden nun die Menge $S_C \in RG(N, TM_0)$ der Zustandsklassen im Graphen gesucht, in denen die n Tokens auf c liegen und die durch Feuern der Transition *B* erreichbar sind, d.h. eine mit e_B markierte eingehende Kante aufweisen. Wir betrachten nun die Zeitintervalle der Klassen in S_C. Die kleinste aller Intervallgrenzen sei y, die größte z. Es lassen sich folgende Fälle unterscheiden:

- $t_{max} < y$. Die Zeitvorgabe ist auf keinen Fall erfüllbar.

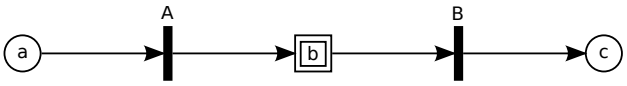

Abbildung 8: ITCPN für Eingangs- und Ausgangsbereich

- $y \leqslant t_{max} < z$. Die Zeitvorgabe kann vom System eingehalten werden, ist aber nicht sicher.
- $z \leqslant t_{max}$. Die Zeitvorgabe wird eingehalten.

6 Fazit und Ausblick

In dieser Arbeit wurde die Einführung von Timeouts in die prozesskettenbasierte Modellierungssprache *ProC/B* beschrieben, welche in ihren Grundzügen erläutert wurde. Mögliche Modellierungsfehler bei der Verwendung von Timeouts wurden aufgezeigt. Es wurde dargestellt, dass diese nicht sicher durch Simulation zu erkennbar sind und nur eine Validierung des Modells diese erkennt.
Der hierzu entwickelte Ansatz sieht vor, die Validierung durch funktionale Analyse basierend auf einer Transformation der *ProC/B*-Modelle in Interval Timed Coloured Petri Nets durchzuführen. Diese wurden formal vorgestellt sowie die Erstellung des reduzierten Erreichbarkeitsgraphen durch MTSRT erläutert. Die Transformation der *ProC/B*-Bausteine zu ITCPN unter Beachtung der Zeitgrenzen wurde anhand zweier grundlegender Prozesskettenelemente gezeigt. Da ITCPN Zeitinformationen in Form von Intervallen tragen, müssen etwaige Zufallsverteilungen des *ProC/B*-Modells auf Zeitintervalle unter Annahme eines Parameters geschätzt werden. Weiterhin wurde vorgestellt, wie sich zu restriktive Zeitgrenzen und die mangelnde zeitliche Absicherung als Modellierungsfehler durch den reduzierten Erreichbarkeitsgraphen erkennen lassen.
Die weiteren Forschungsziele sind die Erfassung weiterer temporaler Modellierungsfehler und eine bessere Nutzung der zeitlichen Informationen innerhalb des reduzierten Zustandsgraphen. Diese Informationen lassen sich zur Berechnung weiterer quantitativer Leistungsmaße nutzen [vdA94].

Literatur

[BBF+02] Falko Bause, Hans Beilner, M. Fischer, P. Kemper und M. Völker. The ProC/B Toolset for Modelling and Analysis of Process Chains. In T. Field, P.G. Harrison, J. Bradley und U. Harder, Hrsg., *TOOLS 2002*, number 2324 in Lecture Notes in Computer Science, Seiten 51–70. Springer Verlag, 2002.

[BBK00] Falko Bause, H. Beilner und Peter Kemper. Modellierung und Analyse von Logistiknetzwerken mit Prozeßketten. In *ASIM-Symposium Simulationstechnik*, Seiten 63–67,

September 2000.

[BBKV08] Falko Bause, Peter Buchholz, Jan Kriege und Sebastian Vastag. A Framework for Simulation Models of Service-Oriented Architectures. In S. Kounev, I. Gorton und K. Sachs, Hrsg., *LLNCS*, Jgg. 5119, Darmstadt, Germany, June 27-28 2008. SPEC International Performance Evaluation Workshop 2008, SIPEW 2008, Springer.

[BBT04] Falko Bause, Peter Buchholz und Carsten Tepper. The ProC/B-Approach: From Informal Descriptions to Formal Models. In *ISoLA — 1st International Symposium on Leveraging Applications of Formal Method*, 2004.

[BC09] Peter Buchholz und Uwe Clausen, Hrsg. *Große Netze der Logistik. Die Ergebnisse des Sonderforschungsbereichs 559*. Springer, 2009.

[BD91] Bernard Berthomieu und Michel Diaz. Modeling and Verification of Time Dependent Systems Using Time Petri Nets. *IEEE Trans. Softw. Eng.*, 17(3):259–273, 1991.

[BK02] Falko Bause und Pieter S. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, 2nd edition. Auflage, 2002.

[BK07] Falko Bause und Jan Kriege. Detecting Non-Ergodic Simulation Models of Logistics Networks. In *Proc. of 2nd International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2007)*, Nantes, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Bou08] Hanifa Boucheneb. Interval timed coloured Petri net: efficient construction of its state class space preserving linear properties. *Form. Asp. Comput.*, 20(2):225–238, 2008.

[HJS91] P. Huber, K. Jensen und R.M. Shapiro. Hierarchies in coloured Petri nets. In *Proceedings on Advances in Petri nets 1990 table of contents*, Seiten 313–341. Springer-Verlag New York, Inc. New York, NY, USA, 1991.

[Jen03] Kurt Jensen. *Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use. Volume 1 (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, April 2003.

[KT06] Peter Kemper und Carsten Tepper. A Petri net approach to debug simulation models of logistic networks. In *Proc. 5th Mathmod Vienna*, Jgg. 30, 2006.

[Kuh95] Axel Kuhn. *Prozessketten in der Logistik - Entwicklungstrends und Umsetzungsstrategien*. Verlag Praxiswissen, Dortmund, 1995.

[MF76] P. Merlin und D.J. Farber. Recoverability of Communication Protocols–Implications of a Theoretical Study. *Communications, IEEE Transactions on*, 24(9):1036–1043, Sep 1976.

[vdA93] W. M. P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, Seiten 453–472, London, UK, 1993. Springer-Verlag.

[vdA94] W. M. P. van der Aalst. Using interval timed coloured Petri nets to calculate performance bounds. In *Proceedings of the 7th international conference on Computer performance evaluation : modelling techniques and tools*, Seiten 425–444, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.