Diplomarbeit zum Thema

Verifikation von Echtzeit Architekturen

zur Erlangung des akademischen Grades
Diplom-Informatiker (TU-Berlin)

vorgelegt dem
Fachbereich Informatik
Institut für Telekommunikationssysteme

Yi Shen
1. Januar 2008

Gutachter:	Prof. Dr. Hans-Ulrich Heiß
Gutachter:	Prof. Dr. Hartmut Ehrig
Betreuer:	Prof. Dr. Matthias Werner
Betreuerin:	Dipl.-Inform. Ulrike Prange

Danksagung

Ich möchte mich bei allen Mitarbeitern des Lehrstuhls KBS und TFS bedanken, die mir bei dieser Arbeit geholfen haben und mich durch inspirierende Gespräche und Rat unterstützt haben.

Mein besonderer Dank gilt meiner Betreuerin Ulrike Prange und meinem Betreuer Matthias Werner die mir stets mit wertvollem Rat und tatkräftiger Unterstützung zur Seite standen.

Für sprachliche Hilfe und Korrekturlesen bedanke ich mich bei meinem Kollegen Andreas Jobst und meinem Gemeindebruder Christian Gries.

Schließlich und doch nicht zuletzt bedanke ich mich bei meiner Frau, die mir stets den Rücken frei gehalten hat und mich immer wieder ermutigte und anspornte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Neben meinem Studium arbeite ich seit Jahren in der Qualitätssicherung. Zu meinen Aufgaben gehört unter anderem die Verifizierung von Softwareprodukten gegenüber deren Spezifikation. Dazu gehört auch die Feststellung von Risiken. Ich liebe diese Arbeit und stelle doch fest, wie komplex diese Aufgaben bereits bei relativ kleinen Softwareprodukten sind. Um so mehr staune ich über die Softwarequalität von Betriebssystemen, wie beispielsweise Mac OS X, Windows Vista oder so komplexen Systemen wie Verteilte Systeme, Eingebettete Systeme etc. Die Sicherung der Qualität derartig umfangreicher Systeme durch Testen und andere geeignete Verfahren erlebe ich außerordentlich herausfordernd und faszinierend.
In der Praxis der Softwareentwicklung und Qualitätssicherung wird meistens an Teilkomponenten gearbeitet. Sind die einzelnen Komponenten hinreichend geprüft und getestet, werden sie zusammengebaut und als Ganzes mit allen Interaktionen und Relationen nochmals die Prüfungen unterzogen, die das Gesamtbild abrunden. Obwohl Softwareentwickler meist hochkompetent und kreativ arbeiten, wird das Gesamtbild oft übersehen und an speziellen Baustellen zu lange optimiert. Für das gesamte Projekt ist der Nutzen meist gering und Verzögerungen sind die Folge. Auch wird das einwandfreie Funktionieren des Gesamtproduktes oft dadurch vernachläßigt.
In vielen modernen Branchen werden Softwaresysteme oft als eingebettete Systeme eingesetzt. Beispiele hierfür sind: Automobil-Industrie, Multimedia, Telekommunikation und Vehrkehrswesen.
Wie können wir das Problem lösen?
Für den Menschen ist nichts unmöglich. Beispiele dafür sind: Testautomatisierung, Modellierung, Simulationsverfahren und -methoden. Mit Hilfe solcher Verfahren können wir Fehlern vorbeugen, sie vermeiden und beseitigen. So habe ich zu meinem Thema gefunden. Dabei kann ich lernen und mich

weiterentwickeln. So finde ich analytische und verifizierte Verfahren, die nachvollziehbar sind und den Weg zur Lösung vieler komplexer Problemstellungen weisen.

1.2 Problemstellung

Die Analyse des Themas zeigt, daß wir es hier im wesentlichen mit zwei Problemstellungen zu tun haben. Verifikation und Architektur in Informatik. Verifikation ist abhängig von Verfahren. Architektur in Informatik betrifft nicht nur ein System, sondern auch den Aufbau beliebiger Systeme. Charakteristisch für diese Architektur ist der Begriff Echtzeit, der eine nichtfunktionale Eigenschaft ist. Dieser Aspekt wird im weiteren besonders untersucht. Schließlich wird eine konkrete Beispielarchitektur mit dieser Eigenschaft verifiziert.

1.3 Lösungsidee

Vorrangiges Ziel der Arbeit ist, eine generische Theorie und Methode für diese Architektur zu entwickeln. Besonders in Bezug auf die Komposition solcher Architekturen. Das wird in Kapitel 2 durchgeführt.
Danach werden in Kapitel 3 Definitionen von Verifikation vorgestellt und die passende Verifikationsmethode ausgewählt. Außerdem werden einige Verfahren vorgestellt und verglichen.
Im dritten Teil wird eine Verifikation einer konkreten MSS (Kapitel 4) mit verwandten Theorien von Kapitel 2 und ausgewählten Verfahren durchgeführt.
Kapitel 5 schließt ab mit zusammenfassenden Bemerkungen und einem Ausblick, wie weit diese Arbeit fortgesetzt und erweitert werden kann.

Kapitel 2

Kategorientheorie

In diesem Kapitel wird die Kategorientheorie zur Spezifikation von Architekturen beschrieben. Der Grundbegriff Architektur wird in Abschnitt 2.1 eingeführt und definiert. Danach kommt die Beschreibung in Abschnitt 2.2 mit einer Einleitung, wofür dieser Theorie eingesetzt werden kann. Anschließend werden auf dieser Basis in den Abschitten 2.3 und 2.4 Definition und Strukturierung beschrieben.

2.1 Definition (Architektur)

Mit Hilfe von Kategorientheorie können wir die Architektur spezifizieren. Vor Definition der Architektur müssen wir das Bauelement der Architektur kennenlernen. Es wird „System“ genannt, das als ein Automat definiert wird:
Ein System ist ein (möglicherweise unendlicher und möglicherweise zeitbehafteter) Automat $S(V, F)$, wobei V eine Menge von Variablen $v_1, v_2, \ldots, v_n$ ist und F eine Menge von Funktionen $f_1, f_2, \ldots, f_n$, die den Variablen zugeordnet sind, um Folgezustände zu berechnen, d.h.: $v_1^{'} = f_1(v_1, \ldots, v_n)$, $v_2^{'} = f_1(v_1, \ldots, v_n)$, $v_n^{'} = f_1(v_1, \ldots, v_n)$. Im Fall eines zeitbehafteten Automaten hängt $f_0, \ldots, f_n$ auch von der Zeit t ab.
Der Anfangszustand $v_1^0, \ldots, v_n^0$ zur Zeit t_0 ist bekannt.
Die Menge aller Systeme wird als S bezeichnet. Über dem (mehrfachen) Kreuzprodukt dieser Menge S mit sich selbst gibt es Operatoren, die mögliche Kompositionen beschreiben:
Ein Kompositionsoperator $\circ$ ist ein Abbildung $S \times S \rightarrow S$.
Bislang haben wir das System und Kompositionsoperator definiert, also Architektur wird geschrieben:
Eine Achitektur A ist eine Menge von Tupeln $(\circ, (S_1, S_2))$, wobei $\circ$ ein Kompositionsoperator und $S_1, S_2 \in S$.
A beschreibt damit alle gültigen Kompositionen.

2.2 Einleitung

Die Kategorientheorie ist eine mathematische Theorie, deren Entwicklung vor etwa 50 Jahren mit dem Ziel begonnen hat, ähnliche Konstruktionen in verschiedenen Bereichen der Mathematik gemeinsam zu beschreiben und zu analysieren. Heute ist die Kategorientheorie neben Algebra und Logik die wichtigste mathematische Theorie zur formalen Beschreibung von Strukturen in der Informatik und eine der wichtigsten Grundlagen der strukturellen Mathematik.
Dementsprechend gestattet die Kategorientheorie eine gemeinsame Untersuchung vieler bekannter Strukturen der Mathematik und Informatik. Ihr hohes Abstraktionsniveau und ihre Universalität bei gleichzeitiger Anschaulichkeit ermöglichen eine einfache Beschreibung von komplex aufgebauten Strukturen. Kennzeichnend dabei ist, daß die 'Objekte' nicht durch ihre innere Struktur, sondern durch ihre Beziehung zu anderen 'Objekten' beschrieben werden. Wichtige kategorielle Methoden und Ergebnisse lassen sich elegant auf viele Bereiche der Mathematik (z.B. Mengen, Abbildungen, Relationen, Ordnungsstrukturen, Graphen, lineare Algebra, universelle Algebra, Analysis, Topologie, etc.) und Informatik (z.B. Automaten, Datenstrukturen, Abstrakte Datentypen, Algebren, Graphgrammatiken, Petri-netze, λ-Kalkül, Logik, Programmierkonzepte, Programmiersprachen, Formale Spezifikation, Software-Engineering, Parallele Prozesse, Verteilte Systeme, etc.) anwenden und werden zur Beschreibung und Analyse von Kalkülen und ihrer Modelle eingesetzt.
Die Relevanz kategorieller Methoden in Mathematik und Informatik lässt sich in den folgenden 5 Punkten zusammenfassen:

1. Klassifikation und einheitliche Theorie
   Die Kategorientheorie gestattet eine Klassifikation und einheitliche Theorie von vielen verschiedenen Strukturen innerhalb der Mathematik und Informatik. Dazu gehört in der Mathematik eine gemeinsame Behandlung von algebraischen, analytischen, topologischen und geordneten Strukturen und in der Informatik eine einheitliche Beschreibung von Syntax und Semantik verschiedener Formalismen zur Spezifikation, Programmierung und Beweistechnik.

2. Paradigma der Beschreibungs- und Beweistechnik
   Während das klassische Paradigma zur Beschreibungs- und Beweistechnik auf den Grundbegriffen „Menge“ und „Element“ basiert, beruht das Paradigma der Kategorientheorie auf den Begriffen „Objekt“ und „Morphismus“. Damit werden Strukturen nicht mehr extensional beschrieben sondern durch universelle Eigenschaften charakterisiert. Entsprechendes gilt für Beweise, wobei das Dualitätsprinzip der Kategorientheorie eine zentrale Rolle spielt.

3. Flexible horizontale Strukturierung und vertikale Verfeinerungstechnik
   Das Konzept von Diagrammen aus Objekten und Morphismen einer Kategorie gestattet eine einheitliche Technik für verschiedene Typen von komplexen Strukturen. Die einzelnen Komponenten einer komplexen Struktur und deren Beziehungen werden durch die Objekte und Morphismen des Diagramms beschrieben, die flache Version der komplexen Struktur durch den Limes oder Kolimes des Diagramms. Vertikale Verfeinerung von komplexen Strukturen kann durch Morphismen zwischen den einzelnen Komponenten dieser Strukturen beschrieben werden, die dann einen Morphismus zwischen den komplexen Strukturen induzieren.

4. Instrumentarium zur Übertragung von Konzepten und Ergebnissen
   Das wichtige Konzept von Funktoren in der Kategorientheorie, d.h. strukturverträgliche Abbildungen zwischen Morphismen und Objekten von zwei Kategorien, liefert ein Instrumentarium für die formale Beschreibung von Übergängen (Transformationen) zwischen verschiedenen Ebenen einer Beschreibungstechnik, etwa zwischen der syntaktischen und der semantischen Ebene.

5. Instrumentarium für gegenseitige Befruchtung von Gebieten
   Durch das Konzept von Funktoren $F : C_1 \to C_2$ ist es auch möglich, Konstruktionen und Problemstellungen in einem Bereich der Mathematik und Informatik, beschrieben durch die Kategorie C_1, auf einen anderen Bereich, definiert durch die Kategorie C_2, zu übertragen und damit die beiden Bereiche gegenseitig zu befruchten. Ein klassisches Beispiel ist dabei die Klassifikation von geometrischen Strukturen im Rahmen der Topologie durch die Untersuchung von algebraischen Strukturen, genannt Homologiegruppen. Die Übertragung erfolgt dabei durch Funktoren von der Kategorie der topologischen Räume in die Kategorie der kommutativen Gruppen.

Bedarf an geeigneter Abstraktion von Problembeschreibungen besteht immer dann, wenn Arbeit beim Definieren und Beweisen von Konzepten doppelt geleistet wird. Hier bietet die Kategorientheorie eine angemessene Abstraktionsebene und eine solide Grundlage an Konzepten, die es gestatten, die gemeinsame Essenz verschiedener Beschreibungen zu formulieren. So beschreibt zum Beispiel das kategorielle Koprodukt die Konzepte Disjunkte Vereinigung und Disjunktion von Propositionen. Eine angemessene kategorielle Beschreibung von Ergebnissen und Beweisen liefert eine klare Abgrenzung von allgemeinen und speziellen Problemeigenschaften. Hier bietet sich die Gelegenheit zur "horizontalen Verallgemeinerung", d.h. wir können ähnliche Probleme lösen, indem wir deren allgemeine Eigenschaften

beibehalten und beweisen, dass die speziellen Eigenschaften in den neuen Problemen ebenfalls auftreten.
Insbesondere für die Komposition von Objekten einer Kategorie (also zum Beispiel die Verbindung mehrerer kleiner P/T-Netze zu einem großen) sind Kolimiten von Interesse. Kolimiten repräsentieren eine kategorielle Sichtweise der Komposition: gemeinsame Teile treten im komponierten Objekt nur einmal auf, verschiedenartige Teile werden ohne Identifikation nebeneinandergesetzt. Allgemein sind Kolimiten über beliebigen Diagrammen formuliert.
In dieser Arbeit werden nicht alle Konzepte wie Disjunkte Vereinigung oder Disjuktion von Propositionen verwendet. So stelle ich hier nur die Strukturierung durch Union vor, die auch Pushout genannt wird.

2.3 Definition

Eine Kategorie $C = (Ob_c, Mor_c, \circ, id)$ ist gegeben durch

1. eine Klasse Ob_c von Objekten,
2. für je zwei Objekte $A, B \in Ob_c$ eine Menge $Mor_c(A, B)$ von Morphismen,
3. für je drei Objekte $A, B, C \in Ob_c$ eine Kompositionsoperation

$$\circ : Mor_c(A, B) \times Mor_c(B, C) \rightarrow Mor_c(A, C)$$

4. für jedes Objekt $A \in Ob_c$ eine Identität $id_A \in Mor_c(A, A)$,

so dass die folgenden Bedingungen erfüllt sind:

Assoziativität Für alle $f \in Mor_c(A, B)$, $g \in Mor_c(B, C)$ und $h \in Mor_c(C, D)$ gilt: $(h \circ g) \circ f = h \circ (g \circ f)$.

Neutralität Für alle $f \in Mor_c(A, B)$ gilt: $f \circ id_A = f$ und $id_B \circ f = f$.

Für einen Morphismus $f \in Mor_c(A, B)$ heisst das Objekt A Quelle (domain) und B Ziel (codomain) von f. Man schreibt dann auch $f : A \rightarrow B \in Mor_c$. Mit dieser Schreibweise lässt sich die Komposition von Morphismen in Diagrammen Abbildung 2.1 folgendermassen veranschaulichen.

2.4 Strukturierung durch Union

Die Union ist eine Konstruktion, die zwei Objekte bezüglich einer gemeinsamen Schnittstelle (dem Interface) vereinigt. Für Mengen bedeutet dies, im

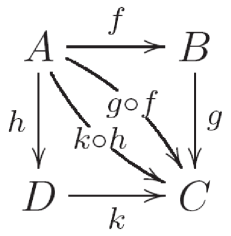

Abbildung 2.1: Morphismus

Gegensatz zur normalen Vereinigung, dass nur die als Schnittstelle ausgezeichnete Teilmenge in beiden Mengen identifiziert wird. Der Rest (die Komplemente) werden dagegen disjunkt vereinigt. Die kategorielle Konstruktion, die die Union beschreibt, ist das Pushout.

2.4.1 Definition von Pushout

Ein Pushout von zwei Morphismen $f_1 : A_0 \to A_1$ und $f_2 : A_0 \to A_2$ einer Kategorie C ist ein Objekt A_3, genannt Pushout-Objekt, zusammen mit zwei Morphismen $g_1 : A_1 \to A_3$ und $g_2 : A_2 \to A_3$ in C, so dass das Diagramm (PO) in folgender Abbildung 2.2 kommutiert und die folgende universelle Eigenschaft erfüllt ist:

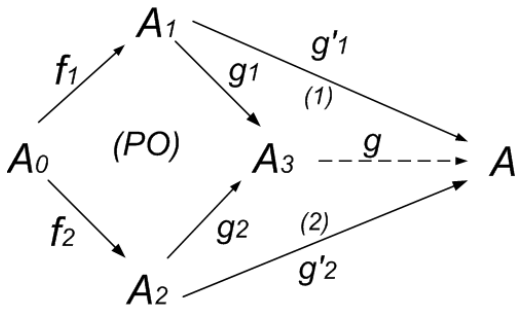

Abbildung 2.2: Pushout

Für alle Objekte A und Morphismen $g_1^{'} : A_1 \to A$ und $g_2^{'} : A_2 \to A$ in C mit $g_1^{'} \circ f_1 = g_2^{'} \circ f_2$ existiert genau ein Morphismus $g : A_3 \to A$ in C, so dass die folgenden Diagramme (1) und (2) kommutieren.
Das Diagramm (PO) heißt in diesem Fall Pushout-Diagramm. Wenn für alle $A_1 \overset{f_1}{\leftarrow} A_0 \overset{f_2}{\rightarrow} A_2$ das Pushout $A_1 \overset{g_1}{\rightarrow} A_3 \overset{g_2}{\leftarrow} A_2$ existiert, dann hat die Kategorie C Pushouts. Wir schreiben vereinfachend:

$$A_3 = A_1 +_{A_0} A_2,$$

wobei die Morphismen nicht explizit genannt, aber wesentlich sind. A_0 nennen wir Interface.

2.4.2 Komposition und Dekomposition von Pushout

Gegeben seien die folgenden Diagramme (1) und (2) in der Abbildung 2.3 in einer Kategorie C, dann gilt:

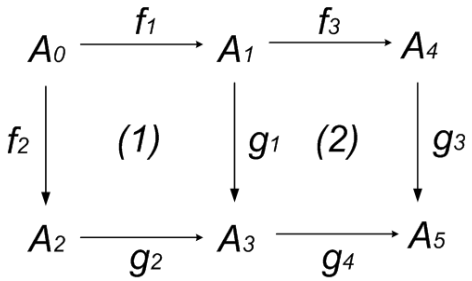

Abbildung 2.3: Komposition von Pushout

1. Komposition : Falls (1) und (2) Pushout-Diagramme sind, dann ist auch das zusammengesetzte Diagramm (1) + (2) Pushout-Diagramm.

2. Dekomposition : Falls (1) + (2) und (1) Pushout-Diagramme sind und (2) kommutiert, dann ist auch (2) Pushout-Diagramm.

Beweis

1. Es ist zu zeigen, dass A_5 mit Morphismen $g_4 \circ g_2$ und g_3 Pushout von $f_3 \circ f_1$ und f_2 ist. Die Kommutativität folgt unmittelbar:

$$g_3 \circ f_3 \circ f_1 \overset{(2)}{=} g_4 \circ g_1 \circ f_1 \overset{(1)}{=} g_4 \circ g_2 \circ f_2$$

   Die universelle Eigenschaft kann man folgendermaßen zeigen. Seien A, $k_1 : A_2 \to A$ und $f_4 : A_4 \to A$ mit $f_4 \circ f_3 \circ f_1 = k_1 \circ f_2$ gegeben. Dann existiert wegen Pushouteigenschaft von (1) genau ein $k_2 : A_3 \to A$ mit $k_2 \circ g_2 = k_1$ und $k_2 \circ g_1 = f_4 \circ f_3$. Damit folgt aus der Pushouteigenschaft von (2) genau ein $k_3 : A_5 \circ A$ mit $k_3 \circ g_3 = f_4$ und $k_3 \circ g_4 = k_2$. Damit gilt auch $k_3 \circ g_4 \circ g_2 = k_2 \circ g_2 = k_1$ und $k_3 \circ g_3 = f_4$. Die Eindeutigkeit von k_3 mit dieser Eigenschaft folgt aus der Eindeutigkeit von k_3 bzgl. Diagramm (2) und der Eindeutigkeit von k_2 bzgl. (1).

2. Es ist zu zeigen, dass A_5 mit den Morphismen g_3 und g_4 Pushout von f_3 und g_1 ist. Die Kommutativität des Diagramms (2) gilt laut Voraussetzung. Gegeben seien A, $k_1 : A_4 \to A$ und $k_2 : A_3 \to A$ mit

$k_2 \circ g_1 = f_3 \circ k_1$ als (3).
Es gilt $k_2 \circ g_2 \circ f_2 \overset{(1)}{=} k_2 \circ g_1 \circ f_1 \overset{(3)}{=} k_1 \circ f_3 \circ f_1$ als (4).
Dann existiert wegen der Pushouteigenschaft von (1)+(2) genau ein Morphismus $k_3 : A_5 \to A$ mit $k_3 \circ g_4 \circ g_2 = k_2 \circ g_2$ als (5) und $k_3 \circ g_3 = k_1$ als (6).
Ferner folgt aus (4) wegen der universellen Eigenschaft des Pushouts (1), dass $\exists! k_{2x} : A_3 \to A$ mit $k_{2x} \circ g_2 = k_2 \circ g_2$ und $k_{2x} \circ g_1 = k_1 \circ f_3$.
Aus der Eindeutigkeit von k_{2x} folgt dann $k_{2x} = k_2$.
Infolge von Gleichung (4) und da $k_3 \circ g_4 \circ g_1 \overset{(2)}{=} k_3 \circ g_3 \circ f_3 \overset{(3)}{=} k_1 \circ f_3$, erfüllt auch $k_3 \circ g_4$ die Eigenschaften von k_{2x}, und damit gilt: $k_3 \circ g_4 = k_2 (= k_{2x})$ und $k_1 = k_3 \circ g_3$ (laut(6)). Die Eindeutigkeit von k_3 mit dieser Eigenschaft folgt aus der Eindeutigkeit von k_3 bzgl. Diagramm (1) + (2) (da für alle $k_3^{'} : A_5 \to A$ mit $k_3^{'} \circ g_3 = k_1$ und $k_3^{'} \circ g_4 = k_2$ gilt $k_3^{'} \circ g_3 = k_1$ und $k_3^{'} \circ g_4 \circ g_2 = k_2 \circ g_2$).

Kapitel 3

Modellierungsverfahren

Ein Überblick über bekannte Verifikationsmethoden wird in Abschnitt 3.1 als Grundlage sowie Definition zuerst darstellt. Danach werde ich die untersuchten Verfahren und Methoden wie kontinuierlichen Modelle, Temporale Logik, Petri-Netz und Automaten in den Abschnitten 3.2, 3.3, 3.4 und 3.5 vorstellen. Ein Verfahren wird davon ausgewählt und die Vertiefung wird geschaffen.

3.1 Verifikationsmethoden

Verifikation wird zum Überprüfen der Korrektheit eines Entwurfs (z.B. von eingebetteten Echtzeitsystemen) und zum Nachweis von Fehlfunktionen eingesetzt. Dabei kann ein Modell des Systems durch Verifikation auf bestimmte Eigenschaften, z.B. die Existenz von Nebenläufigkeiten oder das Einhalten von angegebenen Zeitrestriktionen, untersucht werden. Es wird auch überprüft, ob ein gewünschtes Systemverhalten eingehalten wird bzw. ob unerwünschtes, eventuell auch gefährliches Systemverhalten auftreten kann. Der Begriff Verifikation hat verschiedene Definitionen:
„Verifikation ist die Überprüfung des Wahrheitsgehaltes eines Modells gegen seine Spezifikation.“
oder
„Proof by formal or analytical means that the specification of a software system is consistent and complete.“[40]
Es gibt verschiedene Arten und Möglichkeiten zur Verifikation von eingebetteten Echtzeitsystemen, die sich in Komplexität und Aussagekraft unterscheiden. Dazu gehören Simulation, Test, deduktive Methoden (Theorem Proving) und Model Checking oder formale Verifikation an Hand von formalen Modellierungsmitteln wie z.B. Petri-Netzen.
Unter Simulation versteht man „die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirk-

lichkeit übertragbar sind.". Simulation dient dem Zweck, eine Prognose über das Verhalten des modellierten Systems zu erhalten. Um Simulation durchführen zu können, braucht man ein ausführbares Modell des Systems. Bei der Simulation können meistens nicht alle, sondern nur bestimmte Fälle untersucht werden. Das führt dazu, dass im Allgemeinen nicht alle Fehler entdeckt werden können. Deshalb wird die Fehlerfreiheit des gesamten Systems dabei nicht garantiert.
Bei Testverfahren am realisierten System wird auch nur das Systemverhalten in ausgewählten Fällen untersucht. Es gilt damit auch die Einschränkung der Simulation der nicht garantierten Fehlerfreiheit.
Verifikationsverfahren wie Model Checking, darunter auch Temporal Model Checking und Symbolic Model Checking, unterstützen die Systemüberprüfung auf bestimmte Eigenschaften an Hand eines Modells. Da die Suche in einem Zustandsraum des Modells läuft, ergibt sich eine große Anzahl von Zuständen, wie sie in realen Systemen auftreten. Es gibt schon Tools, die Zustandsräume mit 10100 Elementen beherrschen. Diese Methode wird vorzugsweise bei der Verifikation von Hardware und Protokollen eingesetzt. Beim Temporal Model Checking wird die Erreichbarkeit einzelner Zustände im Gegensatz zum Symbolic Model Checking untersucht, wo das gesamte Systemverhalten berechnet wird. Als größere Anwendungsfälle für Model Checking sind Überprüfungen des IEEE Futurebus+, IEEE SCI und ISDN [5, 6, 10] und des Druckreglers eines elektronischen Bremssteuersystems für Nutzfahrzeuge [1] durchgeführt worden.
Als weitere Verifikationsmethode ist die formale Verifikation mit Petri-Netz-Modellen verbreitet. Diese stellt mit mathematischen Beweisen sicher, dass ein System die geforderten Eigenschaften erfüllt. Viele Methoden davon sind schon lange bekannt und sehr gut untersucht. Es werden aber ständig Weiterentwicklungen vorgenommen. Die Methoden wurden in größeren Projekten, wie von der Deutschen Telekom in einer Studie „Höhere Petrinetze im Bereich Intelligenter Netzwerke“ [2, 3, 4] , „WAP Class 2 Transaction Service“ [14] und TCP-Protokollvarianten vom Hewlett-Packard CPN Centre [13], angewendet.
Nachdem wir einen Überlick über verschiedene Verifikationsmethoden kennengelernt haben, werden wir nun einige konkrete Methoden untersuchen.

3.2 Kontinuierlichen Modelle

Die kontinuierlichen Modelle, in welchen sich die Werte der Zustandsvariablen innerhalb eines endlichen Zeitintervalls unendlich oft ändern und einen beliebigen Wert annehmen können, werden vorwiegend mit Differenzialgleichungen beschrieben. Diese beschreiben die Zusammen-hänge zwischen Funktionen einer oder mehrerer Eingangsvariablen und deren Ablei-

tungen. Dabei kommen verschiedene Gleichungen (lineare und nichtlineare Gleichungen, partielle Gleichungen, welche mehrere unabhängige Variablen haben, Differenzialgleichungen mit expliziter und impliziter Schreibweise) in Betracht.
Die häufig für die Modellierung von kontinuierlichen Systemen verwendeten Gleichungen sind explizite Differentialgleichungen, die aus gewöhnlichen linearen oder nichtlinearen Differentialgleichungen 1. Ordnung und aus algebraischen Gleichungen in expliziter Form bestehen:

$$x = f(x, u, t)$$

$$y = g(x, u, t)$$

Diese Gleichungen beschreiben ziemlich adäquat viele kontinuierliche Systeme. Die Lösung solcher Gleichungssysteme geschieht mit relativ einfachen numerischen Verfahren.
Manche Systeme lassen sich aber mit der impliziten Form leichter beschreiben, die eine weitere Möglichkeit zum Modellieren von solchen Systemen darstellt. Sie besteht genau wie die oben genannte aus linearen oder nichtlinearen Differentialgleichungen 1. Ordnung und aus algebraischen Gleichungen und ist in impliziter Form angegeben:

$$F = (x, x, u, t) = 0$$

Dabei benötigt die Lösung von solchen Gleichungssystemen aufwendige numerische Verfahren sowie eine umfangreichere Spezifikation der Anfangsbedingungen. Deshalb existieren nur wenige Simulationswerkzeuge, die solche Modelle unterstützen.
Wenn bei der Modellierung von kontinuierlichen Systemen neben zeitlichen auch räumliche Prozesse auftreten, können solche Systeme mit partiellen Differentialgleichungen beschrieben werden. Wegen der aufwendigeren Lösungsverfahren und speziellen Einsatzgebiete werden diese von den meisten kontinuierlichen Simulationswerkzeugen nicht unterstützt.

3.3 Temporale Logik

Temporale Logik ist eine spezielle Art der modalen Logik [8, 11]. Die *modale Logik* wurde ursprünglich von Philosophen entwickelt, um unterschiedliche Wahrheitsmodelle untersuchen zu können. Während eine bestimmte Behauptung in einer „Welt“ wahr sein kann, könnte dieselbe Behauptung in einer anderen „Welt“ falsch sein. Bei der temporalen Logik verändern sich die Wahrheitsattribute in Abhängigkeit von der zugeordneten Zeit. In der temporalen Logik werden unterschiedliche temporale Operatoren bereitgestellt, um zu beschreiben und zu begründen, in welcher Weise die Wahrheitswerte von Behauptungen und Aussagen unter wechselnden Zeitmodi

variieren können. Es war Pnueli, der in den späten Siebziger Jahren des vergangenen Jahrhunderts [31] argumentiert hat, dass temporale Logik einen nützlichen Formalismus bereitstellen könne, um Computerprogramme zu spezifizieren und zu verifizieren. Temporale Logik ist besonders geeignet für reaktive Systeme, also Systeme, die nicht nur feste Funktionen aus einem bestimmten Input erzeugen, sondern die kontinuierlich und ohne abzubrechen mit der Umwelt kommunizieren können. Ausgehend von dieser grundsätzlichen Idee wurden verschiedene Unterarten der temporalen Logik entwickelt: solche mit linearem Zeitverlauf, mit verzweigender Zeit, in Echtzeit, sowie Logiken mit und ohne zeitlich vorangestellten Operatoren, Logiken mit Bezug zu Zeitintervallen anstelle von Zeitpunkten, etc. Verifikationstechniken werden angeboten, mit denen zeitbasierte Formeln in ihrem Bezug zu den verwendeten Zeitmodellen überprüft werden können, und dies oft mit der Unterstützung eines Softwaretools.
Diese Methoden der vorstehenden beschriebenen temporalen Logik erscheinen mir jedoch der Verwendung von Petri-Netzen nicht ebenbürtig zu sein, welche einfacher zu verstehen und anzuwenden sind.

3.4 Petri-Netz-basierte Modelle

Die in den 60er Jahren von C.A. Petri in seiner Dissertation „Kommunikation mit Automaten“ [30] entwickelte Theorie der Petri-Netze stellt ein formales Mittel zur Beschreibung von verteilten nebenläufigen Systemen dar. Da es eine Vielzahl verschiedener allgemeiner Veröffentlichungen zu Petri-Netzen gibt, basiert die folgende Darstellung auf einer Zusammenfassung aus [12, 19, 25, 26, 33, 35, 36, 37]. Mittlerweile sind Petri-Netze ein annerkannter Beschreibungsformalismus, der sich erfolgreich in verschiedenen Gebieten der Informatik, Steuerungs- und Automatisierungstechnik, der Produktionsmodellierung und beim Workflow Entwurf zur Modellierung anwenden lässt. Da eine deutliche Tendenz zur Ersetzung bzw. Ergänzung algebraischer Beschreibungsmittel durch grafische existiert, gewinnen solche Beschreibungstechniken wie Petri- Netze bei der Modellierung von parallel ablaufenden Prozessen und Systemen immer mehr an Bedeutung. Mit Hilfe von Petri-Netzen werden kausale Zusammenhänge von diskreten Systemen beschrieben. In Petri-Netzen werden zwei Arten von Netzknoten verwendet: Transitionen und Plätze, dabei wird die Systemstruktur über gerichtete Kanten, die Plätze und Transitionen miteinander verbinden, festgelegt. Die Veränderung der Markierung durch Schalten von Transitionen bildet die Dynamik des modellierten Systems ab. Die Beschreibung nebenläufiger Prozesse wird mit Petri-Netzen besonders günstig möglich.
Verschiedene Erweiterungen der einfachen Petri-Netze unterstützen die Beschreibung von komplizierterem Verhalten der modellierten Systeme. Diese

Erweiterungen beziehen sich auf die Einführung von höheren Petri-Netzen (z.B. Colored Petri Nets, CPN) [19, 20, 21], welche im Unterschied zu einfachen Petri-Netzen (z.B. Platz-Transitions-Netzen) nicht anonyme Marken sondern attributierte Marken verwenden. Diese Klasse von Netzen stellt ein adäquates Modellierungsmittel von komplexen Systemen dar.
Als eine wichtige Erweiterung ist die Einführung von Zeitkonzepten [26] zu nennen. Die Zeitbewertung kann zu Transitionen, Plätzen, Kanten oder Marken zugeordnet werden und beschreiben damit die Zeiteigenschaften der zu modellierenden Prozesse. Das ermöglicht auch die Untersuchung auf gewünschtes oder unerwünschtes Zeitverhalten.
Die Einführung von Hierarchiekonzepten [17] erlaubt hierarchische Anordnungen bei der Modellierung von komplexen Systemen.
Die Gefärbten Petri-Netze lassen sich in Platz-Transitions-Netze transformieren, was die Anwendung von für diese Netzklasse entwickelten Analysemethoden ermöglicht.
Die Petri-Netze sind dank ihrer einfachen grafischen Darstellung und dem gut entwickelten mathematischen Analyseapparat in wissenschaftlichen Arbeiten ziemlich weit verbreitet. Der umfassende Einsatz als Beschreibungsmittel in der Praxis ist weniger zu beobachten. Das liegt vermutlich gerade daran, dass sie sehr formal und mathematisch orientiert verwendet werden können und sollten. Demzufolge sind oft Ansätze erfolgreich, wo Petri-Netze Konzepte in verbreitete Beschreibungsmittel integriert werden und wo mit Petri-Netzen andere Beschreibungsmittel formalisiert und der Verifikation zugänglich gemacht werden.

3.5 Zeitabhängige Automaten

Zur Beschreibung zeitabhängiger Systeme sind in den vergangenen Jahren eine Reihe unterschiedlicher, in weitesten Sinn als Automaten zu bezeichnede Formalismen entwickelt worden. Das erste, wohl auch einfachste und am besten studierte Modell geht auf Alur und Dill zurück [9] und ist unter der Bezeichnung „timed automata“ bekannt. Dabei handelt es sich um einen klassischen endlichen Automaten, der durch die Hinzunahme einer endlichen Zahl von Uhren an die Belange, die bei der Beschreibung zeitlichen Verhaltens auftreten, anpaßt wurde.
Zeitgleich wurden in [15] von Harry R. Lewis „timed state diagramms“ eingeführt. Darauf folgten weitere Modelle: „timed transition systems“ [38], „interval automata“ [32], „parallel timer processes“ [23], „action timed graphs“ [44] und „guarded-command real-time programs“ [39]. Alle diese Systeme sind insofern von gleicher Konzeption, als sie mit einem wie auch immer gearteten Mechanismus versehen sind, der es erlaubt, gleichzeitig eine begrenzte Anzahl zeitlicher Abstände verfolgen und messen zu können.

Es ist deshalb auch nicht verwunderlich, daß auf den erste Blick höchst unterschiedliche Systeme dieser Art. wie z.B. „timed automata“ und „time state diagrams“, gleiche Ausdruckstärke besitzen. Wir wollen diese Automaten unter dem Begriff Abstandsautomaten zusammenfassen.
Hier wird das Automatenmodell vorgestellt, das als „timed automaten“ in [9] eingeführt wurde und das Paradebeispiel für einen Abstandsautomaten ist. Wir wollen diesem Modell den anschaulichen Namen „Uhrautomat“ geben.

3.5.1 Berechnungsstruktur für Uhrautomaten

Die Berechnungsstruktur Σ_{UA} ist definiert durch:

$$\Sigma_{UA} = (P_0, \{0\}, \{add\}, I)$$

Ein Σ_{UA} Automat $A = (Q, R, (s, \sigma), \Delta, F)$ ist ein spezieller Automat, dessen Verhalten durch Σ_{UA} eingeschränkt wird. Da Σ_{UA} nur ein Anfangselement, nämlich Null, besitzt, ist σ identisch Null. Außerdem gibt es in Σ_{UA} nur eine Schrittfunktion, so daß bei festem Registersatz nur ein Fortschreibung möglich ist. Schließlich kommt bei jeder Rücksetzung nur ein Wert in Frage, auf den zurückgesetzt werden kann. Das führt uns zu der folgenden vereinfachenden Definition.

3.5.2 Definition (Uhrautomat)

Ein Uhrautomat A ist ein Quintupel

$$A = (Q, R, s, \Delta, F),$$

bei dem Q, R und F wie bei einem Σ_{UA}-Automat definiert sind, $s \in Q$ Anfangzustand genannt wird und Δ aus Transitionen der Form $(q, a, \phi, \varsigma, q\,')$ besteht. Er wird mit Σ_{UA}-Automat $(Q, R, (s, \sigma), \Delta\,', F)$ identifiziert, für dessen Anfangsbelegung $\sigma(r) = 0$ für alle $r \in R$ gilt und dessen Transitionsrelation $\Delta\,'$ für jede Transition $(q, a, \phi, \varsigma, q\,')$ aus Δ die Transition $(q, a, \vartheta, \phi, \varsigma, q\,')$ enthält, wobei $\vartheta(r) = add$ für jedes $r \in R$ gewählt wird.

Vergleich mit Petri-Netze

Zur Beschreibung komplizierter zeitlicher Zusammenhänge wie MSS, in denen auch abgesehen von zeitlichen Abständen sich mit der Zeit verändernde Größen eine wichtige Rollen spielen, sind Abstandsautomaten sowie Uhrautomat zu schwach. Automaten unterstützen ein zustandsorientierte Modellbildung. Wenn man die einzelnen Zustände des zu beschreibenden Systems

definiert und die Zuständsübergänge identifiziert hat, kann man den Automaten direkt angeben. Demgegenüber stellen Petrinetze das dynamische Verhalten des Systems als ein Folge von Teilprozessen dar. Wenn man die Teilprozesse zusammen mit den Bedingungen für den Beginn der Prozesse und die Ergebnisse zum Ende der Prozesse zusammengestellt hat, kann man das Petri-Netz aufstellen, wobei die Stellen die Bedingungen für den Start bzw. die Ergebnisse der Teilprozessen beschreiben und Transitionen die aktivierten Teilprozesse.
Diese Unterschiede führen dazu, dass man bei Petri-Netzen besser erkennen kann, welche Ereignisse (bzw. Teilprozesse) sequenziell und welche parallel auftreten.
Automaten und Petri-Netze haben ähnliche Beziehungen, die in der folgenden Tabelle 3.1 darstellt werden. Hier werden die Analogien kurz in beiden Beschreibungsformen zusammengefasst.

Stardandautomat	**Petri-Netz**
Berechnung des Nachfolgezustandes	Berechnung der Nachfolgemarkierung
aktive Ereignisse	aktive Transitionen
Menge der möglichen Nachfolgezustände	Menge der möglichen Nachfolgemarkierungen
Der Automat beschreibt zusammengehörige Zustands- und Ereignisfolgen	Das Petri-Netz beschreibt zusammmengehörige Markierungs- und Transitionsfolgen
Die Sprache des Automaten ist die Menge aller möglichen Ereignisfolgen	Die Sprache des Petri-Netzes ist die Menge aller möglichen Transitionsfolgen

Tabelle 3.1: Vergleich Automaten Petri-Netz

3.6 Gefärbte Petri-Netze

Höhere oder Gefärbte Petri-Netze (Coloured Petri Net oder abgekürzt als CPN) stellen eine Erweiterung von klassischen Petri-Netzen dar, welche durch Einführung von unterscheidbaren, z.B. gefärbten, Marken entsteht. Im Vordergrund steht dabei die kompaktere Darstellung von ähnlichen, sich oft wiederholenden Teilen. Diese treten typischerweise in parallel laufenden Prozessen auf.

3.6.1 Definition von ac-CPN

Da in der Literatur keine einheitliche Definition von gefärbten Petri-Netzen existiert, wird hier auf eine formale Definition für ein möglichst allgemeines gefärbtes Petri-Netz eingegangen [7, 19].
Ein gefärbtes P/T-Netz ist ein Tupel ac-CPN=(P, T, F, C, cd, V, M_0). Für die Definition der Färbung der Stellen und der Flussrelation wird der Begriff der Multimenge gebracht. Als Multimenge über einer nicht leeren Menge A bezeichnet man eine Abbildung bg von A in die Menge der natürlichen Zahlen, $bg : A \to \mathbb{N}$. Sei $C_T(A)$ die Menge aller Multimengen über der Menge A. Die Länge der Multimenge ist $|bg| := \sum_{a \in A} bg(a)$, falls $|bg| = 0$, dann wird $\emptyset$ für die leere Multimenge angezeigt.
Ein gefärbtes ac-P/T-Netz besteht dann aus den folgenden Elementen:

1. P und T sind Mengen, deren Elemente Stellen (places) bzw. Transitionen genannt werden.
2. $F \subseteq (P \times T) \cup (T \times P)$ ist eine zweistellige Relation, die Flussrelation von ac-CPN.
3. $C = \{c_1, c_2, c_3, \ldots\}$ ist eine Menge von Farben.
4. $cd : P \to \mathcal{P}(C)$ definiert die Menge der erlaubten Farben für jede Stelle ($\mathcal{P}$ ist die Potenzmenge).
5. V ist die Vielfachheit der gefärbten Kanten; $\forall(p,t)$, $(t,p) \in F$ gilt $V(p,t) \in C_T(cd(p))$ und $V(t,p) \in C_T(cd(p))$.
6. M_0-Anfangsmarkierung: $M_0(p) \in C_T(cd(p))$.

Dabei ist $C_T(cd(p))$ für jede Stelle p die Menge aller Multimengen über der Menge $cd(p)$ der Farben dieser Stelle. Ist z.B die Menge der Farben $C = \{c_1, c_2, c_3, c_4, c_5\}$, $cd(p) = \{c_1, c_2, c_3\}$, dann sind die Mengen $\{c_1, c_1, c_2\}$, $\{c_1, c_2, c_3, c_3, c_3\}$. Multimengen über $cd(p)$, bzw. $\{2c_1 + c_2, c_1 + c_2 + 3c_3\} \in C_T(cd(p))$. Die Standardoperationen von Multimengen werden im Folgen definiert. Für alle bg,bg_1,$bg_2 \in C_T(A)$ und alle $n \in \mathbb{N}$, gilt:

- $bg_1 + bg_2 := \sum_{a \in A}(bg_1(a) + bg_2(a))'a$ (Addition).
- $n \cdot bg := \sum_{a \in A}(n \cdot bg(a))'a$ (Skalare Multiplikation).
- $bg_1 \geq bg_2 := \forall a \in A, bg_1(a) \geq bg_2(a)$ ($\leq$ und $=$ sind analog definiert).
- Falls $bg_1 \leq bg_2$, gilt $bg_2 - bg_1 := \sum_{a \in A}(bg_2(a) - bg_1(a))'a$ (Subtraktion).

Für die Definition des Schaltverhaltens erweitern wir V auf alle Elemente $(p,t) \in P \times T$ und $(t,p) \in T \times P$ mit $V(p,t) = V(t,p) = 0$ wenn $(p,t), (t,p) \notin F$.

Für die beschriebenen ac-CPN gilt als Schaltregel:

1. Eine Transition t ist unter einer Markierung M schaltfähig, wenn in allen Vorplätzen von t ausreichend Token vorhanden sind.

$$\forall p \in \bullet t : M(p) \geq V(p,t), mit \bullet t = \{p|(p,t) \in F\}$$

2. Beim Schalten wird in den Vorplätzen von t entsprechend der notwendigen Einsetzung in den Kantengewichten subtrahiert und in den Nachplätzen von t addiert, so entsteht die Folgemarkierung $M^{'}$

$$\forall p \in P : M^{'}(p) = M(p) - V(p,t) + V(t,p)$$

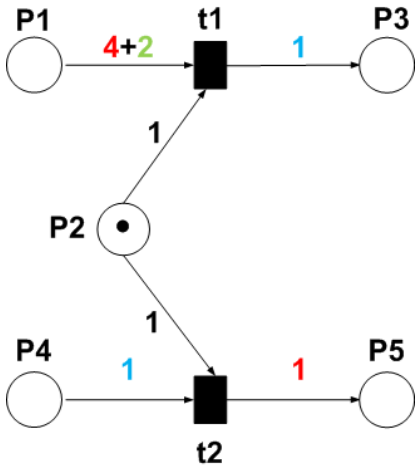

Abbildung 3.1: ac-CPNetz

Beispiel

In der Abbildung 3.1 ist ein einfaches gefärbtes ac-P/T Netz zu sehen, mit

$$P = \{p_1, p_2, p_3, p_4, p_5\}, T = \{t_1, t_2\}, C = \{rot, grün, blau, schwarz\}$$

$$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_4, t_2), (p_2, t_2), (t_2, p_5)\}.$$

Zur einfacheren Beschreibung werden die Farben durch ihren Anfangsbuchstaben identifiziert, d.h $C = \{r, g, b, s\}$. Die erlaubten Farben auf den Stellen sind :

$$cd(p_1) = cd(p_4) = \{r, g, b, s\}, cd(p_2) = \{s\}, cd(p_3) = \{b\}, cd(p_2) = \{r\}$$

Die Kantegewicht sind in Abbildung 3.1 gegeben durch:

$$V(p_1, t_1) = 4r + 2g, V(p_4, t_1) = b, V(p_2, t_1) = V(p_2, t_2) = s,$$

$$V(t_1, p_3) = b, V(t_2, p_3) = r$$

$$M_{initial} = (\emptyset, s, \emptyset, \emptyset, \emptyset)$$

Falls mindestens 4 rote und 2 grüne Token in p_1 oder ein blaues Token in p_4 vorhanden sind (Regel 1), kann t_1 bzw. t_2 von M_0 nach M_1 schalten (Regel 2). Sind z.B zusätzlich zu $M_{initial}$ in p_1 5 rote und in p_4 ein blaues Token sind vorhanden, berechnet sich die Folgemarkierung M_1 nach dem Schalten von t_2 durch:

$$M_1 = (5r, s, \emptyset, b, \emptyset) - (\emptyset, s, \emptyset, b, \emptyset) + (\emptyset, \emptyset, \emptyset, \emptyset, r)$$

$$M_1 = (5r, \emptyset, \emptyset, \emptyset, r)$$

3.6.2 priorisiertes zeitbewertetes ac-CPN

Ein Tupel ac-$CTPN = (P, T, F, C, cd, V, \Upsilon, Z)$ ist ein gefärbtes priorisiertes zeitbewertetes P/N-Netz, wenn gilt:

1. ac-$CPN = (P, T, F, C, cd, V)$ ist ein gefärbtes P/T-Netz
2. Υ ist Priorität und wird den Transitionen zugeordnet Υ: $T \rightarrow \mathbb{N}$
3. Z ist Zeitdauer und wird den Transitionen zugeordnet Z: $T \rightarrow \mathbb{R}$

Priorität

In Abbildung 3.1 haben wir nur in p_4 ausreichend viele Tokens für das Schalten von t_2 gehabt. Falls im Netz jetzt in p_1 5 rote, 2 grüne und in p_4 ein blaues Token vorhanden sind, welche der Transitionen t_1, t_2 wird geschaltet? Es tritt eine typische Problemsituation (Konflikt) auf.
Ein Petri-Netz ist konfliktbehaftet, falls beim Schalten zweier verschiedener Transitionen eine gleiche Stelle die Token für beide Transitionen liefert (Vorplatzkonflikt). Konkret bedeutet diese Bedingung für den Vorplatzkonflikt, dass zwei verschiedene Paare (x, y) und (x, z) mit der selben Stelle x in der Flussrelation enthalten sind. Um solche Konfliktsituation zu verhindern verwendet man Prioritäten Υ, die als natürliche Zahlen, beginnend mit 0, neben einer Transition notiert werden. Ein höherer Prioritätswert bedeutet, dass die Transition gegenüber einer Transition mit einem niedrigeren Prioritätswert bevorzugt wird.

Definition (Konflikt in ac-CPNet)

Sei $N = (P, T, F, C, cd, V)$ ein ac-CPNetz, dann gilt:
$t_1 \neq t_2 \in T$ befinden sich in einem $Vorplatzkonflikt$, falls $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, das heißt: $\exists p \in P$ mit Nichtdeterminismus an der Stelle p.

In Abbildung 3.1 stehen t_1 und t_2 im Vorplatzkonflikt. Weil beide auf die Stelle p_2 zugreifen. So verwendet man hier die Funktion Υ zur Priorisierung. Jeder Transition wird genau ein Prioritätswert zugeordnet, als $\Upsilon(t)$. Für t_1, t_2 , wenn $\Upsilon(t_2) < \Upsilon(t_1)$, dann hat t_1 höhe Priorität als t_2. Falls $\forall p \in {\bullet}t_1 : M(p) \geq V(p, t_1)$ und $\forall p \in {\bullet}t_2 : M(p) \geq V(p, t_2)$, wird die Transition mit höherer Priorität, also Transition t_1 ausgewählt, und von M_0 nach M_1 geschaltet über t_1, also $M_0 \xrightarrow{t_1} M_1$.

Zeit

Es gibt drei häufig benutzte Arten von Funktionen für die Zeiten:

- Zeitverzögerung t_v
- Zeitdauer t_d
- Zeitintervall t_i

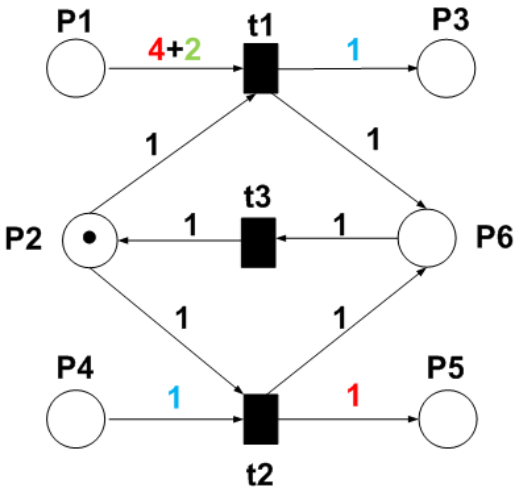

Abbildung 3.2: ac-CTPNet

Da für die Verifikationsverfahren von MSS im Verlauf dieser Arbeit die gefärbten Zeitdauer P/T Netze von Interesse sind, wird hier etwas näher auf diesen Typ t_d von zeitbewerteten Petri Netzen eingegangen. Die Zeitdauer t_d wird in [34, 43] so beschrieben: Wenn die Transition zum Zeitpunkt t_i zu schalten anfängt, wird die Markierung der Vorplätze sofort geändert, aber in den Nachplätzen erscheint die Änderung der Markierung erst nach dem Ablauf der angegebenen Zeitdauer t_d, also zum Zeitpunkt $t_i + t_d$. Falls $t_d = 0$, wird die Transition wieder sofort ohne Zeitdauer geschaltet
Ein priorisiertes ac-CTPN ist in Abbildung 3.2 darstellt. Es ist eine Erweiterung von der Abbildung 3.1 mit $p_6 \in P, t_3 \in T$ und es gibt die neuen

Flussrelationen $(t_1, p_6), (t_2, p_6), (p_6, t_3), (t_3, p_2) \in F$ und neue Kantengewichte $V(t_1, p_6) = V(t_2, p_4) = V(p_6, t_3) = V(t_3, p_2) = s$. Die Zeitdauerfunktion in ac-CTPNet wird als reelle Zahlen, beginnend mit 0, neben einer Transition notiert. In der Abbildung 3.2 wird jetzt jeder Transition $t \in T$ ein Zeitdauerwert $Z(t) \in \mathbb{R}$ zugeordnet. Falls $\forall p \in \bullet t : M(p) \geq V(p, t)$, kann t von M nach $M^{'}$ nach $Z(t)$ schalten.
Sei z.B die Zeitdauer von $Z(t_1) = 3$, $Z(t_2) = 2$ und $Z(t_3) = 4$ für jede Transition in dem priorisierten ac-CTPN in Abbildung 3.2. Als Zeiteinheit wird hier Sekunde verwendet. Falls 5 rote, 2 grüne und ein blaues Token in p_1 bzw. p_4 vorhanden sind, hat (t_1) höhe Priorität als (t_2). t_1 wird zuerst geschaltet und folgende Schaltsschritte sind möglich:

$$M_0 \xrightarrow{(t_1)Z} M_1 \xrightarrow{(t_3)Z} M_2 \xrightarrow{(t_2)Z} M_3 \xrightarrow{(t_3)Z} M_4$$

Berechnung:
Nach 3 Sekunden,

$$\begin{aligned} M_1(p) &= (5r + 2g, s, \emptyset, b, \emptyset, \emptyset) - (4r + 2g, s, \emptyset, \emptyset, \emptyset, \emptyset) + (\emptyset, \emptyset, b, \emptyset, \emptyset, s) \\ &\overset{3sec}{=} (r, \emptyset, b, b, \emptyset, s) \end{aligned}$$

Nach weiteren 4 Sekunden,

$$\begin{aligned} M_2(p) &= (r, \emptyset, b, b, \emptyset, s) - (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, s) + (\emptyset, s, \emptyset, \emptyset, \emptyset, \emptyset) \\ &\overset{4sec}{=} (r, s, b, b, \emptyset, \emptyset) \end{aligned}$$

Nach weiteren 2 Sekunden,

$$\begin{aligned} M_3(p) &= (r, s, b, b, \emptyset, \emptyset) - (\emptyset, s, \emptyset, b, \emptyset, \emptyset) + (\emptyset, \emptyset, \emptyset, \emptyset, r, s) \\ &\overset{2sec}{=} (r, \emptyset, b, \emptyset, r, s) \end{aligned}$$

Nach weiteren 4 Sekunden,

$$\begin{aligned} M_4(p) &= (r, \emptyset, b, \emptyset, r, s) - ((\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, s) + (\emptyset, s, \emptyset, \emptyset, \emptyset, \emptyset) \\ &\overset{4sec}{=} (r, s, b, \emptyset, r, \emptyset) \end{aligned}$$

Die Folgemarkierung hat insgesamt von M_0 bis M_4 13 Sekunden verbraucht. Falls die Startzeit t_{start} ist, ist die Endzeit $t_{ende} = t_{start} + Z(t_1) + Z(t_3) + Z(t_2) + Z(t_3) = t_{start} + 13$.

3.6.3 Darstellung von Prio ac-CTPN

Matrix-Darstellung

Hier beschäftige ich mich mit der Darstellung von P/T-Netzen durch Matrizen und Vektoren. Das hat folgende Gründe:

- Die Berechnung von Schaltsequenzen lässt sich durch eine Matrix-Vektor-Multiplikation ausdrücken und ist somit die Basis für eine automatische Berechnung.
- Paralleles Schalten lässt sich durch eine Vektor-Addition ausdrücken.
- Invarianten können berechnet werden.
  Invarianten sind entweder Zustände, die sich im Laufe des Ablaufs nicht ändern, also invariant bleiben (Stellen-Invarianten), oder Schaltsequenzen, die immer wieder durchlaufen werden können (Transitions-Invarianten). Beide stellen eine Methode zur Überprüfung der Korrektheit eines Netzes da.

Definition (Prio ac-CTPN in Matrix-Darstellung)

Aus einer endlichen Menge von Transitionen T besteht ein priorisiertes ac-$\underline{\text{CTPN}}$ $=(P, T, \underline{pre}, \underline{post}, C, cd, \Upsilon, Z)$ in Matrix-Darstellung, einer endlichen Menge von Stellen P, zwei positiven Matrizen $\underline{pre}, \underline{post} : P \times T \rightarrow C_T(C)$, C, cd, Υ, Z wie in ac-CTPN so dass für alle $p \in P$, $t \in T$ gilt $pre(p,t)$, $post(p,t) \in C_T(cd(p))$.
Sei ac-CTPN$=(P, T, F, V, C, cd, \Upsilon, Z)$ ein priorisiertes gefärbtes Netz. Dann ist durch ac-$\underline{\text{CTPN}}$ $= (P, T, \underline{pre}, \underline{post}, C, cd, \Upsilon, Z)$ die Darstellung von ac-CTPN gegeben, mit

$$\underline{pre}(p,t) = \begin{cases} V(p,t) & ; \quad falls \quad p \in \bullet t \\ \emptyset & ; \quad sonst \end{cases} \tag{3.1}$$

und

$$\underline{post}(p,t) = \begin{cases} V(t,p) & ; \quad falls \quad p \in t\bullet \\ \emptyset & ; \quad sonst \end{cases} \tag{3.2}$$

Ein markiertes Netz in Matrix-Darstellung ac-$\underline{CTPN}_{M_0}$ ist ein ac-CTPN in Matrix-Darstellung zusammen mit einem P-Vektor M_0 mit $\forall p \in P : M_0(p) \in C_T(cd(p))$ als Anfangsmarkierung. Die Verifikation durch Matrix-Darstellung ohne Zeitbewertung oder Unabhängigkeit von Zeitbewertung liefert Aussagen zur Erreichbarkeit von Zuständen des modellierten Systems, zur Lebendigkeit, Sicherheit usw.
Dabei gibt $\underline{pre}$ an, wieviele Token die Transition t von p entnehmen muss, und $\underline{post}$ beschreibt, wieviele Token t auf p ablegt. Die Matrix-Darstellung eines Netzes ist also nicht viel mehr als eine tabellarische Darstellung der Flussrelation zusammen mit den Kantengewichten. Markierungen können wir als Vektoren über der Stellenmenge des betreffenden Netzes mit Einträgen aus $C_T(C)$ schreiben. Das prio ac-CTPN aus Abbildung 3.2 wird in der folgenden Tabelle 3.2 in Matrixdarstellung mit $\underline{pre}$, $\underline{post}$ Matrizen und Anfangsmarkierung M_0 dargestellt.

$\underline{pre}/\underline{post}$	t_1	t_2	t_3	M_0
p_1	$4r+2g/\emptyset$	$\emptyset/\emptyset$	$\emptyset/\emptyset$	$5r+2g$
p_2	$s/\emptyset$	$s/\emptyset$	$\emptyset/s$	s
p_3	$\emptyset/b$	$\emptyset/\emptyset$	$\emptyset/\emptyset$	$\emptyset$
p_4	$\emptyset/\emptyset$	$b/\emptyset$	$\emptyset/\emptyset$	b
p_5	$\emptyset/\emptyset$	$\emptyset/r$	$\emptyset/\emptyset$	$\emptyset$
p_6	$\emptyset/s$	$\emptyset/s$	$s/\emptyset$	$\emptyset$

Tabelle 3.2: Matrix-Darstellung

Schaltverhalten der prio ac-CTPN in Matrix-Darstellung

Sei $\underline{N}$ ein prio ac-CTPN in Matrix-Darstellung. Ein Schaltvektor wird definiert als positiver Vektor $v : T \to \mathbb{N}$, der jeder Transition $t \in T$ die Häufigkeit ihres Schaltens zuordnet. Nach Schaltregel 1 wird v aktiviert wenn genügend Token in den Vorplätzen liegen, also:

$$M \geq \underline{pre} \cdot v$$

Ein M-aktivierter Schaltvektor v bestimmt eine Folgemarkierung $M^{'}$ von M durch die Schaltregel 2 :

$$M^{'} = M - \underline{pre} \cdot v + \underline{post} \cdot v = M + (\underline{post} - \underline{pre}) \cdot v$$

Für jede Transition $t \in T$ führen wir noch den charakteristischen Vektor $cv_t : T \to N$ ein mit

$$cv_t(x) = \begin{cases} 1 & ; \quad falls \quad x = t \\ 0 & ; \quad sonst \end{cases} \tag{3.3}$$

Für das prio ac-CPN aus Abbildung 3.2 hat die Folge der Transition t_1, t_3, t_2, t_3 den Schaltvektor $v{=}cv_{t_1} + cv_{t_3} + cv_{t_2} + cv_{t_3}{=}(1,0,0) + (0,0,1) + (0,1,0) + (0,0,1){=}(1,1,2)$. Die Zeitbewertung wird in diesem Beispiel nicht gebraucht. Wenn Zeitdauern für Transitionen gegeben sind, kann man die Vektoren z.B durch $cv_{t_1} = (1(3), 0, 0)$ darstellen.
Die Matrix $I = post - pre$ heißt Inzidenzmatrix des Netzes. Diese Inzidenzmatrix wird für die Berechnung der Invarianten verwendet.
Unsere prio ac-$\underline{CPN}$ aus Abbildung 3.2 hat folgende Inzidenzmatrix:

$$\underline{I} = \begin{pmatrix} \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & s \\ b & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset \\ \emptyset & r & \emptyset \\ s & s & \emptyset \end{pmatrix} - \begin{pmatrix} 4r+2g & \emptyset & \emptyset \\ s & s & \emptyset \\ \emptyset & \emptyset & \emptyset \\ \emptyset & b & \emptyset \\ \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & s \end{pmatrix} = \begin{pmatrix} -4r-2g & \emptyset & \emptyset \\ -s & -s & s \\ b & \emptyset & \emptyset \\ \emptyset & -b & \emptyset \\ \emptyset & r & \emptyset \\ s & s & -s \end{pmatrix} \tag{3.4}$$

Stellen-Invarianten

Sei N ein ac-CPTNetz und $I_p : C \otimes P \rightarrow Z$ ein Stellenvektor, dann ist I_p eine Stellen-Invariante von Netz N, wenn für alle $M \xrightarrow{t} M'$ gilt: $\sum_{p \in P} I(p) \cdot M(p) = \sum_{p \in P} I(p) \cdot M'(p)$.
I_P ist ganzzahliger Lösungs-Vektor des linearen Gleichungssystems (mit transponierter Inzidenzmatrix $\underline{I}_P^T$):

$$\underline{I}_P^T \cdot x = \emptyset$$

Beispielsberechnungen werden in Kapitel 4 durchgeführt.

Monoid-Darstellung

Hier führe ich die universell-algebraische Sichtweise von Netzen ein, die zur Monoid-Darstellung von priorisierten ac-$CTPN$ führt. Dabei wird die Netzstruktur durch zwei Abbildungen ähnlich wie bei Graphen beschrieben. Diese Abbildungen geben für jede Transition ihren Vor- und Nachbereich an, wobei der Zielbereich keine Menge, sondern das freie kommutative Monoid $P^{\oplus}$ über der Menge P aller Stellen ist. Diese Darstellung bildet die Grundlage für die Netz-Morphismen und Strukturierung von Netzen. Die Verwendung der Monoid-Darstellung für P/T-Netze hat folgende Vorteile:

- Das nebenläufige Schalten mehrerer Transitionen und das Schalten einer einzelnen Transition wird einheitlich notiert und behandelt.
- Netz-Morphismen lassen sich analog zu Homomorphismen von Algebren konstruieren und bewahren das Schaltverhalten.
- Durch Einbettung der Netze in Monoid-Darstellung in den mathematischen Kontext der Kategorientheorie werden deren Ergebnisse und Methoden für die Strukturierung und Analyse von Netzen anwendbar.

Definition (Freies kommutatives Monoid über eine Menge)

Sei M eine Menge, dann lässt sich das freie kommutative Monoid $M^{\oplus} = (M, \lambda, \oplus)$ über M mit neutralem Element λ und einer zweistelligen Verknüpfung $\oplus$ folgendermaßen rekursiv beschreiben:

1. $M \cup \lambda \subseteq M^{\oplus}$
2. $v, w \in M^{\oplus} \Longrightarrow v \oplus w \in M^{\oplus}$
3. $v \oplus \lambda = v = \lambda \oplus v$ (neutrales Element)
4. $u \oplus (v \oplus w) = (u \oplus v) \oplus w$ (Assoziativität)

5. $v \oplus w = w \oplus v$ (Kommutativität)

Ein Element w lässt sich in Normalform schreiben als $w = \sum_{i=1}^{n} k_i a_i$, mit $a_i \in M, k_i \in \mathbb{N}$.
Allgemein gilt für die Lineare-Summen-Schreibweise von Elementen:

1. $w \oplus 0a_i = w$ für alle $a_i \in M$
2. $\lambda = \sum_{i=1}^{0} k_i a_i$
3. Die Addition von $w = \sum_{i=1}^{n} k_i a_i$ und $w' = \sum_{j=1}^{n} k'_j a_j$ mit $a_i = a_j$ für $i, j = 1, \ldots, n$ ist definiert durch $w \oplus w' = \sum_{i=1}^{n} (k_i + k'_i) a_i$.

Freie kommutative Monoide sind äquivalent zu Multimengen, d.h. $M^{\oplus} \sim C_T(M)$.

Definition (Prio ac-CTPN in Monoid-Darstellung)

Bei der Monoid-Darstellung eines Netzes ohne Kapazitätsbeschränkungen stellen wir die Flussrelation durch Abbildungen *pre* und *post* zwischen der Menge T der Transitionen und dem freien kommutativen Monoid $(C \otimes P)^{\oplus}$ über der Menge P der Stellen und der Menge C der Farben dar. Dabei gilt:

$$C \otimes P = \{(c, p) | p \in P, c \in cd(p)\}$$

Für $(c, p) \oplus (d, p)$ schreiben wir kurz $(c + d)p$ oder auch, wenn $c = d, 2cp$.
Zu prio $\underline{\text{ac}}$-$\underline{CPTN} = (P, T, \underline{pre}, \underline{post}, C, cd, \Upsilon, Z)$ in Matrix-Darstellung ist dann ein priorisiertes ac-$\overline{CTPN} = (P, T, pre, post, C, cd, \Upsilon, Z)$ in Monoid-Darstellung gegeben, mit der Menge P der Stellen, der Menge T der Transitionen und den Abbildungen $pre, post : T \to (C \otimes P)^{\oplus}$, die den Vorbereich *pre* und den Nachbereich *post* der Transitionen beschreiben, sowie C, cd, Υ, Z in $\underline{\text{ac}}$-$\underline{\text{CPTN}}$. Dann gilt $pre(t) = \sum_{p \in P} \underline{pre}(p, t)$ und $post(t) = \sum_{p \in P} \underline{post}(p, t)$. Eine Markierung ist gegeben durch ein Element $: M \in (C \otimes P)^{\oplus}$.
Für das ac-CPNetz in Abbildung 3.2 sind die Funktionen *pre* und *post* folgendermaßen gegeben:

$pre(t_1) = (4r + 2g)p_1 \oplus (s)p_2$, $\quad post(t_1) = (b)p_3 \oplus (s)p_6$
$pre(t_2) = (b)p_4 \oplus (s)p_2$, $\quad post(t_2) = (r)p_5 \oplus (s)p_6$
$pre(t_3) = (s)p_6$, $\quad post(t_2) = (s)p_2$

3.6.4 Die Kategorie Prio ac-CTPN

Nun soll die Kategorie der priorisierten ac-CTPNetze entwickelt werden. Die Objekte dieser Kategorie sind prio ac-CTPN in Monoid-Darstellung, die Morphismen prio ac-CPTN-Morphismen.

Definition der priorisierten ac-CTPN Morphismus

Seien $N_i := (P_i, T_i, pre_i, post_i, C, cd_i, \Upsilon_i, Z_i)$, für i = 1,2 priorisierte ac-CTPNetze. Dann ist

$$f : N_1 \to N_2 := (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$$

ein prio ac-CTPN-Morphismus wenn Abbildung 3.3 kommutiert. d.h.

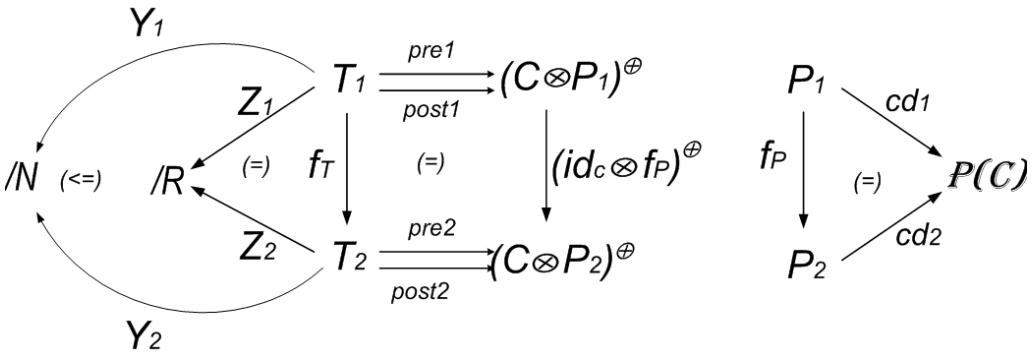

Abbildung 3.3: Prio ac-CTPN Morphismus

$$Z_1 = Z_2 \circ f_T, \Upsilon_1 \leq \Upsilon_2 \circ f_T, cd_1 = cd_2 \circ f_P,$$

$$(id_C \otimes f_p)^{\oplus} \circ pre_1 = pre_2 \circ f_T, (id_C \otimes f_p)^{\oplus} \circ post_1 = post_2 \circ f_T.$$

Beispiel

Betrachten wir die beiden priorisierten ac-CPTN N1 und N2 in Abbildung 3.4. Dann bilden

$$f_P := \{p_6 \to p_4, p_7 \to p_5, p_8 \to p_2\}$$

$$f_T := \{t_3 \to t_2\}$$

einen Netz-Morphismus nach Definition der priorisierten ac-CTPN Morphismen, denn die Vor- und Nachbereiche der Transitionen stimmen überein, und es gilt:

$$
\begin{aligned}
Z_1(t_3) &= 2 = Z_2(t_2) = Z_2(f_T(t_3)) \\
\Upsilon_1(t_3) &= 5 = \Upsilon_2(t_2) = \Upsilon_2(f_T(t_3)) \\
cd_1(p_6) &= b = cd_2(p_4) = cd_2(f_P(p_6))
\end{aligned}
$$

usw für alle anderen Stellen p_8, p_7 auch noch.

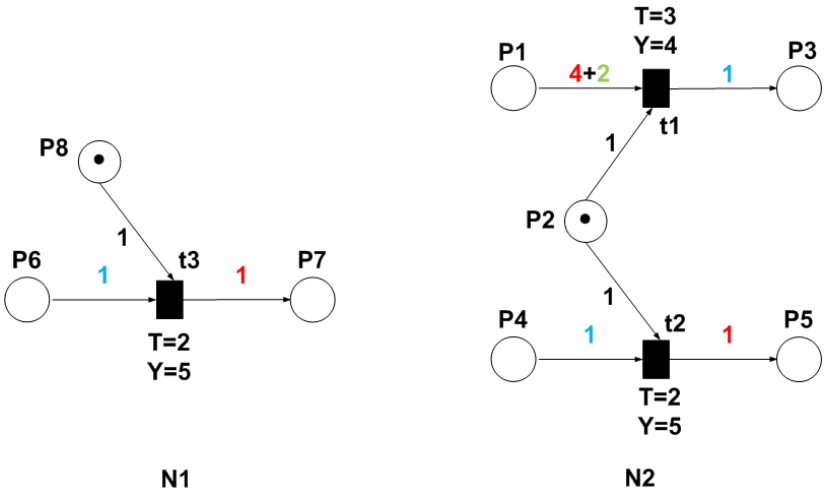

Abbildung 3.4: Netz-Morphismus

Satz der Kategorie Prio ac-CTPN

Die Klasse der priorisierten ac-$CTPN := (P, T, pre, post, C, cd, \Upsilon, Z)$ bildet zusammen mit den priorisierten ac-CTPN-Morphismen die Kategorie **Prio ac-CTPN**.

- $\circ$: Die Komposition ist komponentenweise definiert.

$$\forall f : N_1 \to N_2, g : N_2 \to N_3 : g \circ f = (g_P \circ f_P, g_T \circ f_T)$$

- $id_N = (id_P, id_T)$: Die Identität ist komponentenweise definiert für P und T.

Beweis:

Seien N_i mit $i = 1,2,3$ Prio ac-CTPNetze und $f : N_1 \to N_2$ und $g : N_2 \to N_3$ Netz-Morphismen. Hier müssen wir zeigen, dass $g \circ f : N_1 \to N_3$ ebenfalls Netz-Morphismus ist. Wir zeigen, dass die folgenden Abbildungen 3.5 und 3.6 komponentenweise kommutieren.
Da f und g prio ac-CTPN-Morphismen sind, folgt dass die beiden Komponentendiagramme kommutieren. Also kommutiert auch das gesamte Diagramm, da $(id_c \otimes g_P)^{\oplus} \circ (id_c \otimes f_P)^{\oplus} = (id_c \otimes (g_P \circ f_P))^{\oplus}$, $Z_1 = Z_3 \circ f_T \circ g_T$,

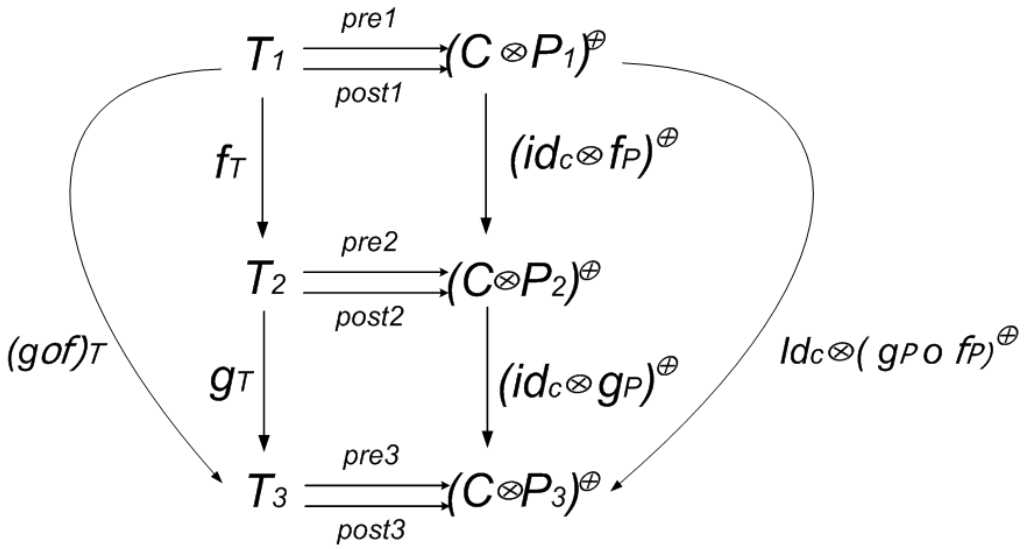

Abbildung 3.5: Komposition

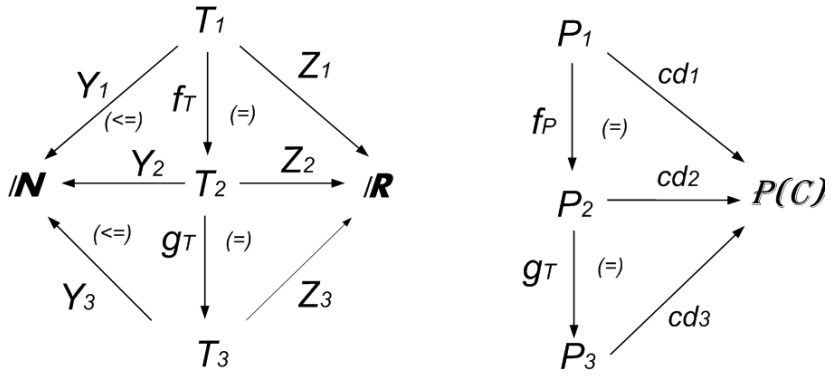

Abbildung 3.6: Komposition2

$\Upsilon_1 \leq \Upsilon_3 \circ f_T \circ g_T$, $cd_1 = cd_3 \circ f_T \circ g_T$.
Die Eigenschaften der Assoziativität und der Identität folgen aus der komponentenweisen Komposition wegen den zugrundeliegenden Abbildungen in Mengen.

3.6.5 Pushout in Prio ac-CTPN

Die Kategorie Prio ac-CTPN hat Pushouts, die in Abschnitt 2.4.1 schon definiert wurden.

Konstruktion

Seien die Netze N_0, N_1, N_2 mit $N_i := (P_i, T_i, pre_i, post_i, cd_i, C, \Upsilon_i, Z_i)$ für i = 0,1,2 und Prio ac-CTPNetz Morphismen $f_1 : N_0 \to N_1$ und $f_2 : N_0 \to N_2$ gegeben. Dann lässt sich Pushout $N_2 \overset{g_2}{\to} N_3 \overset{g_1}{\leftarrow} N_1$ über $N_2 \overset{f_2}{\leftarrow} N_0 \overset{f_1}{\to} N_1$ wie folgt konstruieren.
Wir konstruieren jeweils komponentenweise Pushout über den Stellen und Transitionen und erhalten die beiden Pushouts (P) und (T) in Abbildung 3.7. Dann sind $g_i = (g_{i,P}, g_{i,T})$ und $f_i = (f_{i,P}, f_{i,T})$ für $i = 1, 2$ die geforderten priorisierten ac-CTPN Morphismen und das Pushout-Objekt $N_3 = (P_3, T_3, pre_3, post_3, cd_3, C, \Upsilon_3, Z_3)$ mit:

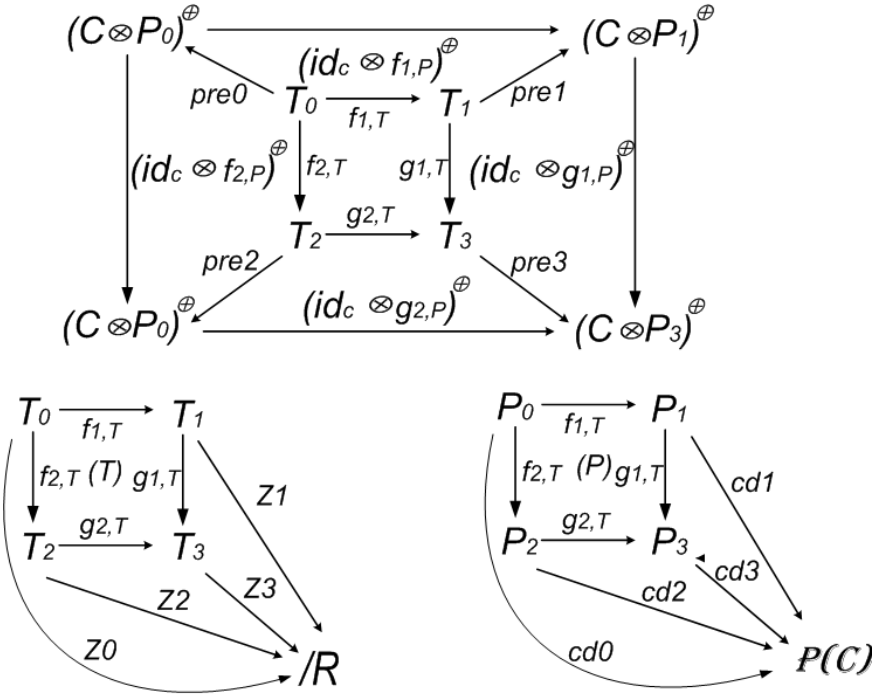

Abbildung 3.7: induzierte Morphismen

- pre_3 ist der von Pushout (T) und den Morphismen $(id_c \otimes g_{2,P})^{\oplus} \circ pre_2$ und $(id_c \otimes g_{1,P})^{\oplus} \circ pre_1$ induzierte Morphismus (siehe Abbildung 3.7).
- $post_3$ ist der von Pushout (T) und den Morphismen $(id_c \otimes g_{2,P})^{\oplus} \circ post_2$ und $(id_c \otimes g_{1,P})^{\oplus} \circ post_1$ induzierte Morphismus.
- Die Zeitbewertung Z_3 ist der von Pushout (T) und den Morphismen Z_1 und Z_2 induzierte Morphismus (siehe Abbildung 3.7).
- Die Färbung cd_3 ist der von Pushout (P) und den Morphismus cd_1 und cd_2 induzierte Morphismus (siehe Abbildung 3.7).
- Die Priorität Υ_3 für alle $t_3 \in T_3$ ist $\Upsilon_3(t_3) = \min \Upsilon(t)$ mit $t \in g_1^{-1}(t_3) \cup g_2^{-1}(t_3)$

Der Beweis, dass die angegebene Konstruktion wohldefiniert und ein Pushout tatsächlich in prio ac-CTPNet ist, läuft analog zu dem für Satz 9.13 (Pushout in PTNet) in [22] mit Kommutativität $g_1 \circ f_1 = g_2 \circ f_2$ und Universeller Eigenschaft. Wir zeigen dies noch einmal explizit für die neu eingeführten Elemente der Netze, also für die Zeitbewertung, die Färbefunktion und die Prioritäten.
Aus der Pushout-Eigenschaft von (T) folgt direkt $Z_3 \circ g_{2,T} = Z_2$ und $Z_3 \circ g_{1,T} = g_1$.
Analog folgt aus der Pushout-Eigenschaft von (P) $cd_3 \circ g_{2,T} = cd_2$ und $cd_3 \circ g_{1,T} = cd_1$.
Für die Prioritäten gilt nach Konstuktion : $\Upsilon_1(t) \leq \Upsilon_3(g_{1,T}(t))$, $\Upsilon_2(t) \leq \Upsilon_3(g_{2,T}(t))$.

3.6.6 Definition CPNetz

Bislang haben wir uns immer auf das ac-CPN konzentriert. Tatsächlich kann eine Stelle oder eine Transition in einem normalen P/T-Netz oder ac-CPNetz eine Menge von Stellen oder Transitionen darstellen. Diese Vereinigung von Stellen und/oder Transitionen heißt **Faltung** und die umgekehrte Operation heißt **Entfaltung**.
In dem Beispiel in Abbildung 3.1 sehen wir, dass die Teilnetze die gleiche Struktur haben. Jetzt falten wir nur die Stellen des Netzes in der folgenden Abbildung 3.8 A).
Nach der Faltung gilt für das Netz immer noch die Definition von ac-CPNetz. Wenn wir noch die Transitionen falten, siehe Abbildung 3.8 B), dann gilt die Definition von ac-CPN nicht mehr, weil es keine konstante Flussrelation mehr gibt, sondern verschiedene Schaltmodi für einer Transition t. Wir müssen die Schaltenmodi der Transition in CPN definieren. Um später leichter zu verifizieren, wird hier die Matrix-Darstellung für die Definition verwandt.

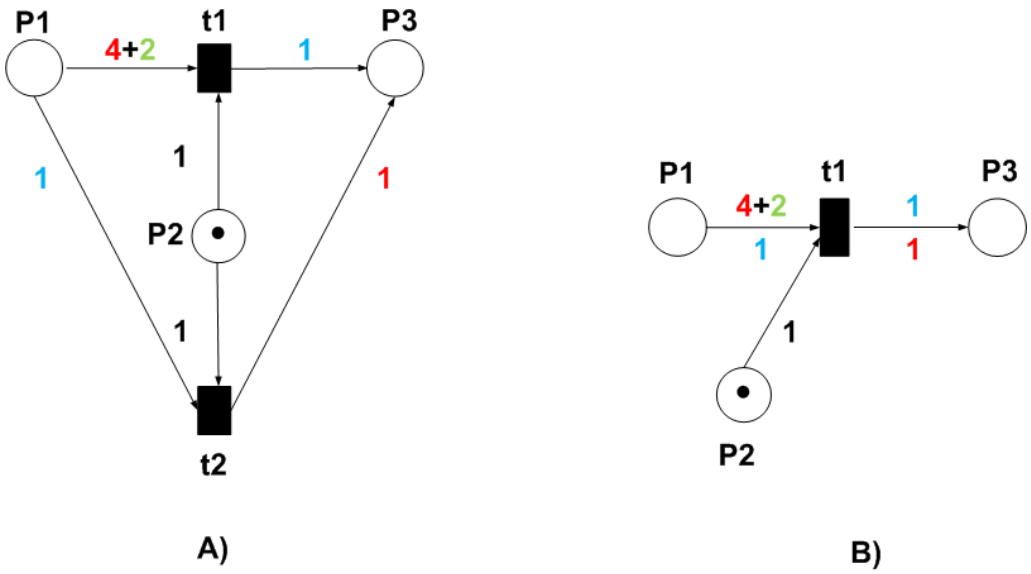

Abbildung 3.8: CPT Netz

Ein Tupel $\underline{\text{CPN}}=(P, T, C, cd, \underline{pre}, \underline{post}, S, guard)$ ist ein gefärbtes P/T-Netz, für das gilt:

1. P und T sind Mengen, deren Elemente Stellen (places) bzw. Transitionen genannt werden.
2. C ist eine Menge von Farben.
3. die Stellen werden auf Menge von Farben abgebildet, als $cd : P \rightarrow \mathcal{P}(C)$ geschrieben.
4. S ist die Menge der Schaltmodi der Transitionen, wobei $guard : T \times S \rightarrow Bool$ die erlaubten Schaltmodi jeder Transition definiert.
5. $\forall p \in P$, $\forall t \in T$ gilt $\underline{pre}$, $\underline{post} : P \times T \times S \rightarrow C_T(C)$,
   für $\forall (p,t), (t,p) \in P \times T$ gilt $\underline{pre}(p,t,s) \in C_T(cd(p))$ und
   $\underline{post}(t,p,s) \in C_T(cd(p))$, wenn $guard(t,s) = true$,
   sonst $pre(p,t,s) = post(p,t,s) = \emptyset$.

Vor einem Beispiel wollen wir noch ein endliche Variablenmenge Var auf den Schaltmodi der Transitionen t definieren, um die Kantenbeschriftung zu vereinfachen. Die Variablenbelegung werden in den Transitionen notiert um die gültigen Schaltmodi darzustellen. Mit Hilfe von Var kann man das CTPNetz besser in der Matrix-Darstellung analysieren und berechnen.

Für die beschriebenen CPN gilt als neue Schaltregel:

1. Auswahl einer Transition t und eines Schaltmodus s mit:

$$guard(t, s) = true$$

2. Eine Transition t ist schaltfähig, wenn in allen Vorplätzen des Schaltmodus s von t ausreichend Token vorhanden sind.

$$\forall p \in \bullet t : M(p) \geq \underline{pre}(p, t, s)$$

3. Beim Schalten wird in den Vorplätzen des Schaltmodus s der Transition t entsprechend der notwendigen Einsetzung in den Kantengewichten subtrahiert und in den Nachplätzen des Schaltmodus s der Transition t addiert.

$$\forall p \in P : M^{'}(p) = M(p) - \underline{pre}(p, t, s) + \underline{post}(p, t, s)$$

und $M \xrightarrow{(t,s)} M^{'}$ als Folgemarkierung geschrieben.

Bei CPNetz werden Prioriotät Υ und Zeitdauer Z nicht mehr nur den Transitionen, sondern den Schaltmodi der Transitionen zugeordnet, so dass gilt:

- $\Upsilon : T \times S \to \mathbb{N}$
- $Z : T \times S \to \mathbb{R}$

So haben wir das priorisierte zeitbewertete gefärbte Petri Netz als ein Tupel $\underline{\text{CTPN}}=(P, T, C, cd, \underline{pre}, \underline{post}, S, guard, \Upsilon, Z)$ bekommen. Auch der Begriff des Konflikts in CPNet wird erweitert. Falls $\exists t \in T$ und es gibt Schaltmodi $s_1, s_2 \in S$ mit $guard(t, s_1) = guard(t, s_2) = true$, das heißt, dass es für eine Transition mehr als einen Schaltmodus gibt, dann tritt ein $Vorplatzskonflikt$ auf.
Das Beispiel Netz in Abbildung 3.2 wurde gefaltet und in Abbildung 3.9 erweitert.
In der Abbildung 3.9 haben wir $v_1, v_2, v_3 \in Var(t_1)$ von t_1, so dass wir die Schaltmodi $S(t_1)$ von t_1 zusammenfassend schreiben können.

$$\begin{aligned} S(t_1) = \quad & (v_1, v_2, v_3) | (v_1 = 4r + 2g \wedge v_2 = b \wedge v_3 = s) \\ & \vee (v_1 = b \wedge v_2 = r \wedge v_3 = s) \\ S(t_3) = \quad & (s) \end{aligned}$$

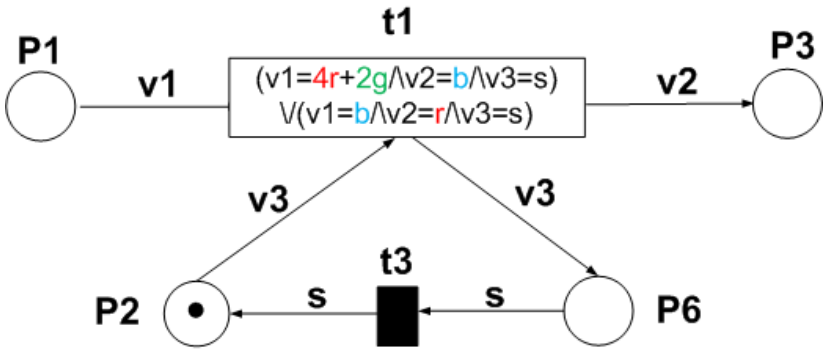

Abbildung 3.9: variable CPNet

Wir zeigen das CPTNetz in den folgenden Tabellen 3.3 und 3.4 mit pre- und post-Matrix-Darstellung.

$\underline{pre}$	t_1 : $(v_1 = 4r + 2g \wedge v_2 = b \wedge v_3 = s) \vee (v_1 = b \wedge v_2 = r \wedge v_3 = s)$	t_3 : $S(t_3)$
p_1	v_1	$\emptyset$
p_2	v_3	$\emptyset$
p_3	$\emptyset$	$\emptyset$
p_6	$\emptyset$	s

Tabelle 3.3: pre-CTPN-Matrix

$\underline{post}$	t_1 : $(v_1 = 4r + 2g \wedge v_2 = b \wedge v_3 = s) \vee (v_1 = b \wedge v_2 = r \wedge v_3 = s)$	t_3 : $S(t_3)$
p_1	$\emptyset$	$\emptyset$
p_2	$\emptyset$	s
p_3	v_2	$\emptyset$
p_6	v_3	$\emptyset$

Tabelle 3.4: post-CTPN-Matrix

Das Schalten von $M_0(p) \overset{t_1,s_1}{\rightarrow} M_1(p)$ geht analog. Ich gebe hier ein explizierte Beispiel (Abbildung 3.10).
Berechnung in folgendem Formel
Nach 3 Sekunden:

$$M_1(p) = M_0(p) - \underline{pre}(p, t_1, s_1) + \underline{post}(p, t_1, s_1)$$

$$M_1(p) = \begin{pmatrix} 5r+2g+b \\ s \\ \emptyset \\ \emptyset \end{pmatrix} - \begin{pmatrix} v_1\,[v_1 = 4r+2g, v_2 = b, v_3 = s] \\ v_3\,[v_1 = 4r+2g, v_2 = b, v_3 = s] \\ \emptyset \\ \emptyset \end{pmatrix} + \begin{pmatrix} \emptyset \\ \emptyset \\ v_2\,[v_1 = 4r+2g, v_2 = b, v_3 = s] \\ v_3\,[v_1 = 4r+2g, v_2 = b, v_3 = s] \end{pmatrix}$$

$$M_1(p) = \begin{pmatrix} 5r+2g+b \\ s \\ \emptyset \\ \emptyset \end{pmatrix} - \begin{pmatrix} 4r+2g \\ s \\ \emptyset \\ \emptyset \end{pmatrix} + \begin{pmatrix} \emptyset \\ \emptyset \\ b \\ s \end{pmatrix} = \begin{pmatrix} r+b \\ \emptyset \\ b \\ s \end{pmatrix}$$

Abbildung 3.10 zeigt mal auch die Schaltregel.

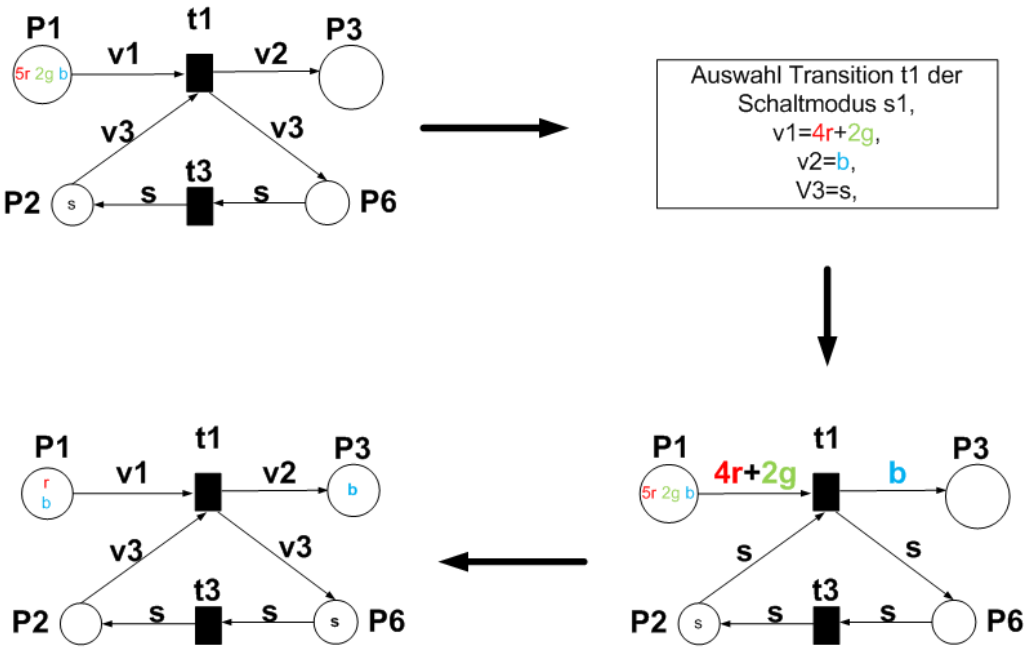

Abbildung 3.10: CPN-Schaltregel

3.6.7 Hierarchische P/T Netze

Durch die Verfeinerung einer Stelle oder einer Transition kann man einen durch ein Netzelement auf einer höheren Abstraktionsebene dargestellen Vorgang genauer darstellen. Abbildung 3.11 zeigt ein Beispiel.
Wie in der Abbildung 3.11 zu sehen ist, werden Transitionen durch *transitionsberandete Netze* und Stellen durch *stellenberandete Netze* ersetzt. Damit wird erreicht, dass sich der Markenfluss im übergeordneten Netz in den detaillierten Netzen fortsetzt. Fließt im übergeordneten Netz von Abbildung 3.11 eine Marke von der Stelle p_1 über die Transition t_1 zu dem Stellen p_2 und p_3, so kann man auf der detaillierteren Beschreibungsebene genauer erkennen, was beim Schalten der Transition t_1 passiert. Die Marke fließt zunächst über die Transition t_{11} und markiert die darauffolgende Stelle. Dann schalten nacheinander die Transitionen t_{12} und t_{13}. Die rechts aus dem Teilnetz herausfließende Marke ist im übergeordneten Netze die Marke, die die Stellen p_2 und p_3 repräsentiert.

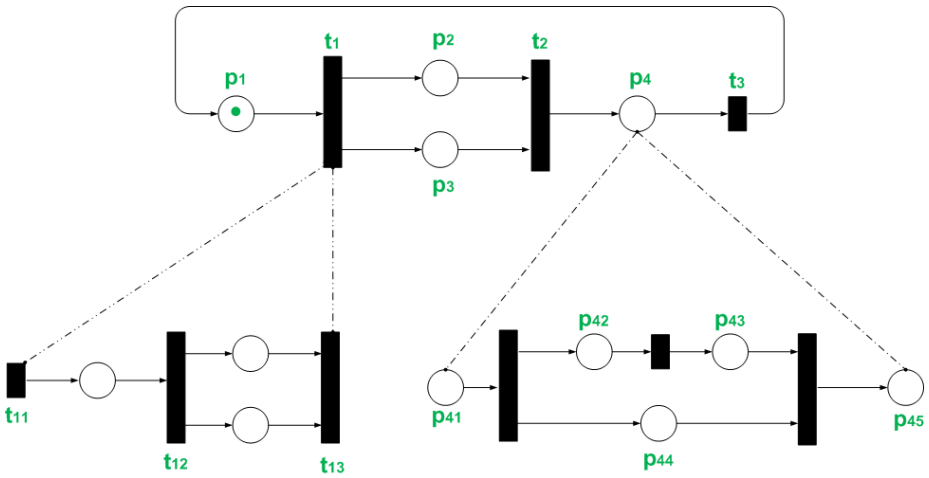

Abbildung 3.11: Hierarchisches Petri-Netz

Im ähnlicher Form ist das stellenberandete Netz zu interpretieren, das die Stelle p_4 des übergeordneten Netzes ersetzt. Sobald die Stelle p_4 markiert ist, beginnt im detaillierten Netz der Markenfluss von der Stelle p_{41} zur Stelle p_{45}.
Man könnte natürlich die detaillierteren Beschreibungen direkt in das übergeordnete Netz einsetzen. Dies führt zu einem detaillierten Modell, das alle Informationen enthält, dessen Verhalten jedoch schwer zu überblicken ist. Die hierarchische Modellierung verfolgt deshalb das Ziel, Übersicht durch eine grobe Darstellung mit Detaildarstellungen für eine genauere Analyse von Teilsystemen oder Teilprozessen zu verbinden. Aus diesem Grund ist die in der Abbildung 3.11 gezeigte Systemdarstellung auf mehreren Abstrakti-

onsebenen zweckmäßig.
Bei der Verwendung hierarchischer Modelle wird man typischerweise mit dem groben Modell beginnen und das Systemverhalten zunächst durch ein Grundmuster von Teilprozessen darstellen. Diejenigen Teile des Netzes, die man bei einer Analyse oder einem Entwurf genauer darstellen muss, wird man dann durch Verfeinerung von Stellen oder Transition erfassen, wobei die Verfeinerungsschritte auf die gezeigten stellen- bzw. transitionberandeten Teilnetze führen. Natürlich kann man bei einer Analyse auch den umgekehrten Weg wählen und eine detaillierte Darstellung durch schrittweises Zusammenfassen von Teilnetzen zu Stellen oder Transitionen so vereinfachen, so dass man einen Überblick über die wichtigsten dynamischen Eigenschaften des beschriebenen Prozesses erhält (Vergröberung).

3.6.8 Transformationen von P/T-Netzen

Transformationen findet man in den unterschiedlichsten Bereichen des Lebens. Dabei handelt es sich um eine Umwandlung oder Verfeinerung von Objekten. Diese Verfeinerung stellt in der Softwaretechnik eine etablierte Technik für die schrittweise Entwicklung von Petri-Netzen dar. Verschiedenartige Konzepte der Verfeinerung werden in der Literatur [22, 28, 29] beschrieben. Sie beschäftigen sich hauptsächlich mit der Verfeinerung von Stellen und Transitionen, sowie der Verfeinerung von Teilnetzen durch andere Teilnetze. Ein wichtiges Resultat ist die Verträglichkeit zwischen Union (Pushout) und Transformation. Das ist wichtig, da im Entwicklungsprozess die unterschiedlichen Teilnetze unabhängig von einander modifiziert werden, aber dennoch weiterhin zueinander passen sollen.
Da für die Verifikationsverfahren von MSS im Verlauf dieser Arbeit Transformationen nicht verwendet werden, werden sie nicht weiter vorgestellt.

Kapitel 4

Verifikation von MSS

Zuerst betrachten wir als Beispiel eine Echtzeit Architektur, bezeichnet als *Message Scheduled System*. Seine Spezifikation, Grundfunktionalitäten und angewandtes Scheduling werden vorgestellt. Dann werden wir das vorherige Modellverfahren und die Kategorietheorie dieser MSS Architektur anwenden.
Schließlich wird noch die Verifikation in zwei Teilen durchgeführt. Zuerst überprüfen wir die Eigenschaften des Petri-Netzes, auf seine Erreichbarkeit und Korrektheit der Struktur. Danach wird ein *Timed Demand Analyse* Verfahren verwendet, um die nicht funktionale Eigenschaft (Zeit) zu berechnen. Ist das angewandte Scheduling des MSS noch korrekt?

4.1 Spezifikation der MSS Architektur

4.1.1 Beschreibung

Ein System der Architektur Message Scheduled System besteht physikalisch aus mehreren Rechnerknoten (hier Abbildung 4.1 mit 2 Knoten), die an einen Broadcast Kommunikationssystem angeschlossen sind. An diese grundlegende Struktur stellt die Systemarchitektur noch eine Reihe von Anforderungen. Begriffe wie Knoten, Task, Nachricht, externes Ereignis, System, Eigenschaft usw... werden diesem Abschnitt erklärt.

Nachricht

Nachrichten sind im MSS keine Elemente im eigentlichen Sinne, sondern die Grundlage des gesamten Systems und seiner Funktion, da Nachrichten die einzige Möglichkeit der expliziten Interaktion zwischen Komponenten sind. Nachrichten werden im MSS als Ereignis betrachtet.

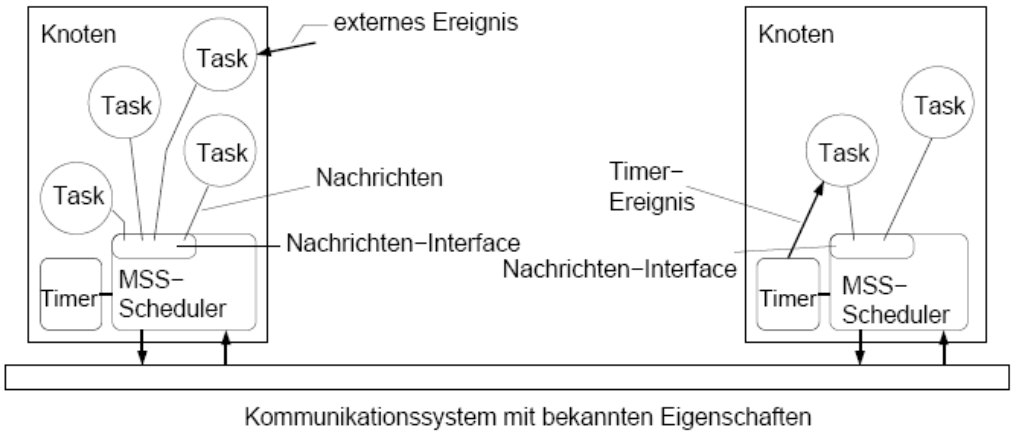

Abbildung 4.1: Message Scheduled System

externes Ereignis

Die in MSS existierenden Typen von Ereignissen sind „externe Ereignisse“, die es erlauben, auf Stimuli der Umgebungen, reagieren zu können, die nicht in Form einer MSS-Nachricht vorliegen. Ein externes Ereignis ist im Gegensatz zu einer Nachricht lokal, kann also nur auf dem entsprechenden Knoten direkt behandelt werden.

Task

Eine Task ist eine reine Softwarekomponente, die ausgehend von einem Satz eingehender Ereignisse (also insbesondere auch Nachrichten) ausgehende Nachrichten berechnet, wobei diese Berechnung durch das Auftreten eines speziellen Ereignisses ausgelöst wird.
Eine Task $T \in \Gamma$, wobei Γ die Menge aller Tasks des MSS-Systems ist, läßt sich über folgende Parameter beschreiben:

- $I(T)$ - Menge von Eingangsereignissen mit Menge von logischen Nachrichten und externen Ereignissen
- $O(T)$ - Menge von physikalischen Nachrichtentypen
- $w(T)$ - Worst Case Execution Time
- $p(T)$ - Deadline Time mit $p(T) \geq w(T)$
- $W(T)$ - Menge von *wake-up* Ereignissen mit $W(T) \subseteq I(T)$
- $a(T)$ - ein Bezeichner (Architektur)

Knoten

Aus Sicht des Kommunikationssystems sind die daran angeschlossenen Knoten Endgeräte, die Nachrichten empfangen und senden können. Knoten sind Rechnerelemente mit einem oder mehreren Prozessoren (CPU), eigenem Speicher, einer Schnittstelle zum Kommunkationssystem, sowie optionale Schnittstellen zur Umgebung(I/O) und optionale Betriebssoftware.
Für die Systemsicht von MSS muss diese abstrakte Darstellung erweitert werden. Die Knoten kommen in einem bereits komponierten System sowohl für sich (also ohne Tasks), als auch in der Komposition mit Tasks (also Knoten mit einem Satz darauf laufender Tasks) vor.
Ein solcher Knoten $N \in M$ (M ist die Menge aller Knoten im System) besitzt damit folgende Parameter:

- $a(N)$ - ein Bezeichner (Architektur)
- Γ^N - Menge von Tasks
  Bezeichnung: Menge der auf N laufenden Tasks, wobei gilt: $\forall T \in \Gamma^N : a(T) = a(N)$. Für einen Knoten ohne Tasks gilt: $\Gamma^N = \emptyset$

System

Ein MSS-System kann aus einem einzelnen Knoten mit den eventuell darauf laufenden Tasks mitsamt Kommunikationssystem oder aber aus mehreren Knoten (eventuell mit darauf laufenden Tasks) mitsamt Kommunikationssystem bestehen. Da die Beschreibung der Knoten bereits Informationen über die darauf laufenden Tasks enthält und das Kommunikationssystem über die daran angeschlossenen Knoten parametrisiert werden kann.
Ein System S in MSS wird damit über die Menge N der in S enthaltenen Knoten bestimmt. Diese Menge von Knoten umfaßt deren Beschreibungen, so daß die vollständige Beschreibungen des Systems sich letztendlich hierarchisch aus den Beschreibung der Knoten und Tasks zusammensetzt.

Eigenschaft

1. Jede Task-Deadline $p(T)$ ist bekannt und wird eingehalten.
2. Jede Nachrichten-Deadline $p(M)$ ist bekannt und wird eingehalten.
3. Jede durch Komposition entstehende End-zu-End-Deadline ist vorab berechenbar.
4. Jede Deadline bleibt durch Komposition unverändert

In dieser Diplomarbeit werde ich nicht für komplette MSS Architekturen deren Korrektheit beweisen, sondern eine Ebene (*Abarbeitung von Tasks*)

aus diesem System. Daher wird diese Ebene vorgestellt. Deshalb wird auch die Bedingung und Garantie für diese Ebene eines gültigen Systems untersucht.
Eine Task kennt in MSS vier Zustände. Das Auftreten eines wake-up events führt dazu, dass die Task von Zustand SUSPENDED in WAIT Zustand wechselt. Im WAIT Zustand wird solange gewartet, bis die Deadline der vorherigen Instanz der Task erreicht ist. Dann wechseln die Task auf READY Zustand. Zwischen den Zuständen READY und RUNNING wechselt die Task in Abhängigkeit von der Entscheidung des Lokalen Taskschedulers des Knotens. Hier kann es sein, dass die Task mit höherer Priorität eine laufende Task mit niedriger Priorität unterbricht. Die Schwerigkeit des Beweises liegt im folgenden: Garantiert eine Task, die zum Zeitpunkt T_{start} vom Zustand WAIT in den Zustand READY wechselt, ob sie spätestens zum Zeitpunkt $T_{end} = T_{start} + p(T)$ vollständig abgearbeitet ist? Eine Deadline zum Zustandwechsel von WAIT nach READY wird also eingehalten?

Schedulingebene 1: Lokales Taskscheduling

Als Schedulingverfahren für die Abarbeitung der Taskebene wird Rate Monotonic Scheduling (RMS) benutzt. RMS arbeitet mit festen Prioritäten. Task erhält eine Priorität reziprok zu seiner Periode (Deadline). Eine laufbereite Task mit höherer Priorität unterbricht stets eine Task niederer Priorität. Ein Menge T von n Tasks ist mit RMS unter den Standardbedingungen immer dann ausführbar, wenn folgendes gilt

$$Gesamtlast : U = \sum_{i=1}^{n} \frac{w(T_i)}{p(T_i)} \leq n(2^{\frac{1}{n}} - 1).$$

Und Grenzwert für Lastschranke

$$U_\infty = \lim_{n \to \infty} n(2^{\frac{1}{n}} - 1) = \ln 2 \approx 0.693$$

Die Kompositionentscheidung des lokalen Taskschedulings

MSS als komponierbare Architektur (komponierbar in Bezug auf das zeitliche Verhalten) muss die Einhaltung der zeitlichen Garantie für die Ausführung von Tasks (Taskdeadlines) sicherstellen. Insbesondere bedeutet dies, daß Hinzufügen oder Entfernen von Komponenten nicht verletzt wird.
In einem MSS System müssen Ressourcen auf verschiedenen Ebenen verwaltet werden, so daß die Abbildung von Komponierbarkeitsentscheidungen auf Scheduling Tests auf mehreren Ebenen erfolgt. In dieser Arbeit wird nur die erste Ebene von Interesse sein. Lokales Schedulings aller Tasks auf einen Knoten, Betrachte Ressource CPU-Zyklen der Prozessors.

Berechnung des lokalen Task-Schedulings sowie die neu hinzukommenden Auslastungen nach *RMS*, wie ich sie vorher schon dargestellt habe, ein Beispiel dafür:
Seien zwei Tasks T_1 , T_2 auf einen Knoten N_1, mit $p(T_1) = 10$, $p(T_2) = 5$, $w(T_1) = 3$, $w(T_2) = 1$, muss das lokale Scheduling nur einmal betrachtet werden. Es gilt für die Auslastung :

$$U(N_1) = \sum_{i:T_i \in T^N} \frac{w(T_i)}{p(T_i)} = \sum_{i:T_i \in T_1,T_2} \frac{w(T_i)}{p(T_i)} = \frac{3}{10} + \frac{1}{5} = 0.5$$

Bei Benutzung von RMS als Schedulingverfahren für diese beiden Tasks ist wegen $0.5 < 0.69$ stets ein Schedule vorhanden, die Vergabe der Prioritäten erfolgt nach RMS so, daß T_1 aufgrund $p(T_1) < p(T_2)$ die höhere Priorität erhält.
Dieses Beispiel erweitert das System um einen weiteren Knoten N_2, auf dem eine weitere Task T_3 läuft, mit $p(T_3) = 20$, $w(T_3) = 10$. Da die Komposition auf den Knoten des vorhandenen System keinerlei Task hinzufügt, betreffen diese Berechnung lediglich den Knoten N_2, dessen Last einzig durch T_3 bestimmt wird:

$$U(N_1) = \sum_{i:T_i \in T^{N_2}} \frac{w(T_i)}{p(T_i)} = \sum_{i:T_i \in T_3} \frac{w(T_i)}{p(T_i)} = \frac{10}{20} = 0.5$$

Die Knotenauslastung muss für alle in S_c (bereits komponiertes System) enthaltenen Knoten durch Addition neu berechnet werden. Sie betrifft in diesem Fall lediglich N_2 und gestaltet sich sehr einfach, da es diesen Knoten im gültigen System S_1 nicht gibt, also $U^{S_1}(N_2) = 0$ und neu gültige System S_2 ergibt sich:

$$U^{S_2}(N_2) = U^{S_1}(N_2) + U^{S_c}(N_2) = 0 + 0.5 = 0.5 \leq 0.693$$

Damit ist die Schedulingentscheidung auf dieser Ebene positiv.
Die Berechnungen auf der ersten Ebene des Schedulings sind rein lokal möglich, der MSS-Scheduler des jeweiligen Knotens (bzw. der jeweiligen Knoten, falls die Komposition mehrere Tasks und Knoten umfasst) kann sie ohne die Notwendigkeit von Kommunikation selbst durchführen und hat alle notwendigen Daten lokal verfügbar. Falls die Berechnungen auf dieser Ebene zu einem positiven Ergebnis geführt haben, wird mit der Berechnung des Bus-Schedulings fortgefahren und es werden die entsprechenden Lastparameter des lokalen Schedulings angepaßt, anderenfalls wird die Komposition abgebrochen.

4.1.2 Modellierung einer Task

Ich verwende hier Modellverfahren Prio ac-CTPN. In Abbildung 4.2 wird einer MSS Task in prio ac-CTPNetz Diagramm dargestellt: Die Beweisführung

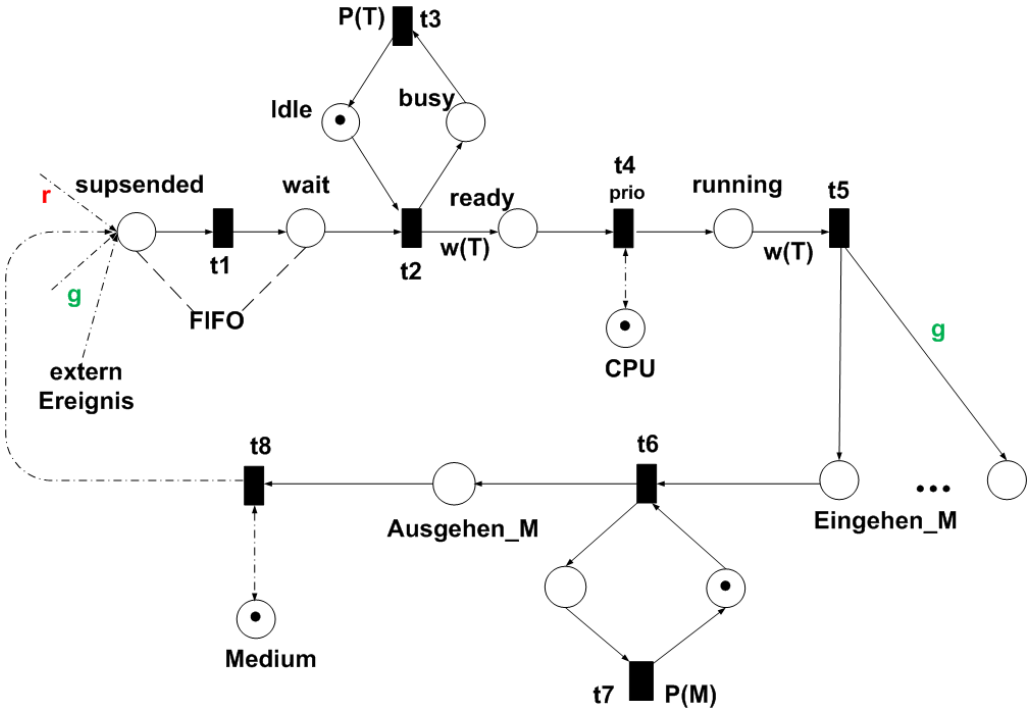

Abbildung 4.2: Spezifikation einer MSS-Task

dieser Diplomarbeit bezieht sich nur auf die Taskebene. Die Taskebene von MSS wird hier in Tabelle 4.1 darstellt.
In diesem Modell ist der Prozessor die Schnittstelle von den Tasks. An dieser Stelle ist also der lokale Prozessor zu spezifizieren. Zur Modellierung eines Knotens mit k Prozessoren werden k Marken auf dem Platz P_{CPU} benutzt. Für wie lange ein Task die Ressource „CPU-Zeit“ erhält, gibt das Scheduling Quantum an, bevor eine neue Scheduling-Entscheidung stattfindet. Es legt damit zugleich die Granularität möglicher Unterbrechungen fest, denn Unterbrechungen sind jeweils nur nach Ablauf des Quantums möglich. Das Quantum wird als atomare Einheit der Zeit betrachtet.
Womit alle Zeit wie $p(T)$ oder $w(T)$, Vielfache des Quantums sind. Die Transition t_3 einer Task T hat $p(T)$-fache Quantum Zeitdauer. Für jeden Task werden $w(T)$ Marken erzeugt und auf Platz P_{ready} gelegt. Tasks werden ausgeführt nach Quantum Zeitdauer. Unterbrechung oder Wechsel einer Task ist nur erst nach Ablauf eines Quantums Zeit möglich. Im MSS dürfen mehrere Tasks auf dem gemeinsamen Prozessor zugreifen. Es wird durch lokales Scheduling nach Priorität entschieden. So werden die Tasks priorisiert, daß der Transition t_4 die entsprechende Priorität zugewiesen wird. Um die Tasks zu identifizieren, hat jede Task genau eine Farbe zugeordnet. Aber die gemeinsame Schnittstelle P_{CPU} hat eine eigene Farbe, unterschiedlich von

$P_{supsended}$	Task wird durch das Auftreten eines Ereignisses aus dieser Menge aufgerufen.
P_{wait}	warten bis Deadline $p(T)$ der vorherigen Instanz der Task erreicht ist.
P_{Idle}	Deadline einer Task erreicht. Diese wartende Task darf jetzt zum Ready-Zustand wechseln.
P_{busy}	Task arbeitet gerade.
P_{ready}	Task ist bereit zu arbeiten.
$P_{running}$	Task wird ausgeführt.
P_{CPU}	Prozessor führt die laufende Task nach Scheduling aus.
t_1	aktiviert, falls die Task aufgerufen ist
t_2	aktiviert, falls Task angekommen ist und Deadline der vorherigen Instanz dieser Task erreicht ist.
t_3	Freigabe der laufenden Instanz der Task, nach Ablauf der angegebenen Zeitdauer $p(T)$ sowie Deadline.
t_4	Tasks werden in Abhängigkeit von der Entscheidung des lokalen Taskschedulers ausgeführt.

Tabelle 4.1: Interpretation der Taskebene von MSS

den Farben der Tasks. Bis jetzt haben wir eine Task modelliert. Tatsächlich werden ein Menge Tasks in MSS zusammengebaut und komponiert. Im folgenden Abschnitt wird die Taskebene der MSS Architektur modelliert und aufgebaut.

4.1.3 Modellierung MSS Architektur

Auf einen Knoten wird jede Task genau eine Farbe zugeordnet und modelliert. Der Prozessor hat eine eigene Farbe, wie vorher schon definiert.
Seien zwei Tasks(N_1, N_2), die schon Bedingungen für ein gültiges System erfüllen und den Algorithmus für die Kompositionsentscheidung [18] ausgeführt haben. Und Prozessor N_0 als prio ac-CTPN Modell in einem gültigen System. Verwenden werden wir es gemäß Abschnitt 3.6.4 und 3.6.5. Dann haben wir die Kategorie Prio ac-CTPN und durch Morphismus und Pushout das Netz N_3 im gültigen System erhalten. In folgender Abbildung 4.3 wird das System mit zwei Tasks aufgebaut und spezifiziert. Falls im vorhandenen

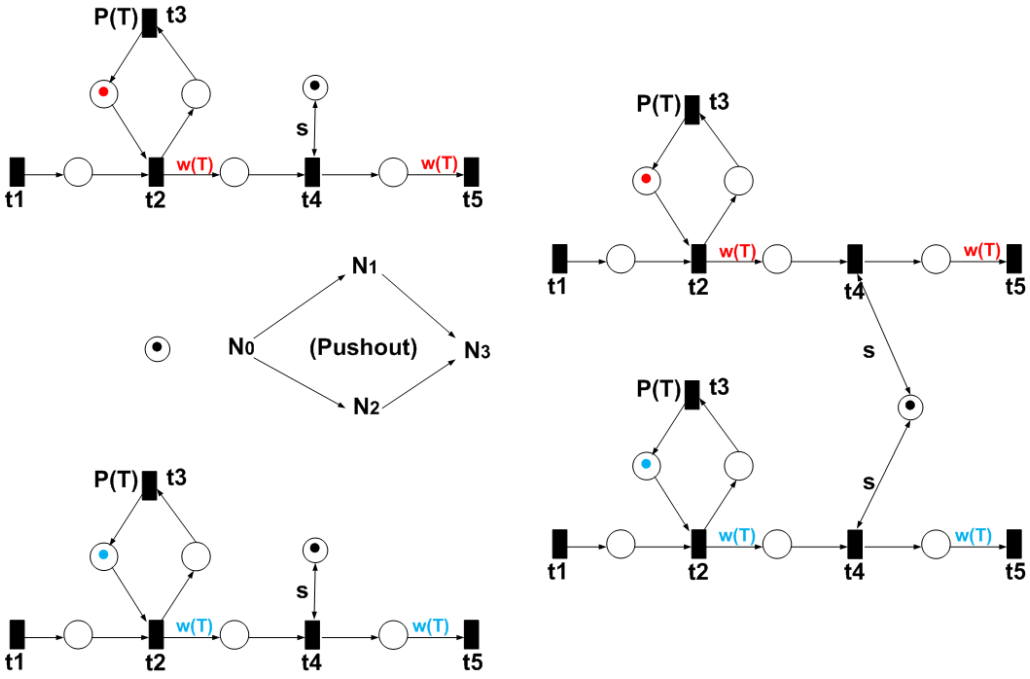

Abbildung 4.3: Task-Pushout

gültigen System ein weiterer Task dazu kommt und die komponierbare Bedingung erfüllt wird, so kann die Task auch in diesen vorhandenen Modell durch Komposition von Pushout hinzugefügt werden. Das heiß, N_2, N_3, N_4 wird im Pushout N_5 erzeugt. Abbildung 4.4 gibt ein Beispiel.
Analog ist: Falls in einem vorhandenen gültigen System weitere beliebige

Tasks laufen und die komponierbare Bedingung erfüllt wird, so kann wiederum trivialerweise durch Komposition von Pushout hinzugefügt werden. In der Formel ist das gültige MSS System auf einen Knoten zusammen-

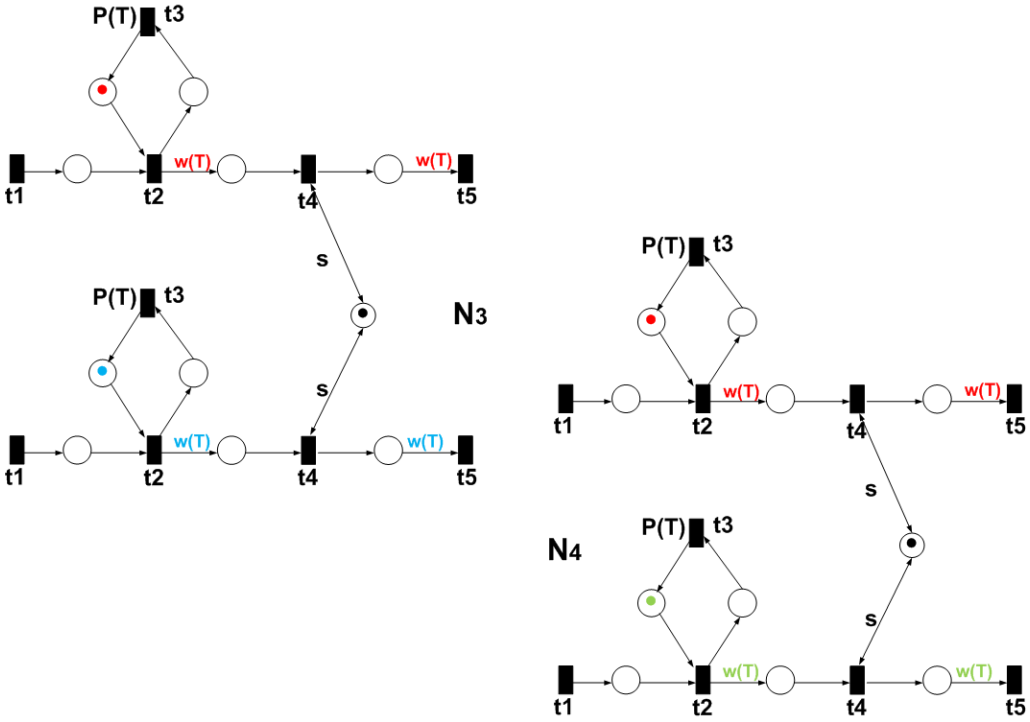

Abbildung 4.4: more-Tasks-Pushout

gefasst, für alle $Task_j$, mit j=1,2,3..., gibt es Prio ac-CTPNet N_i, mit i= 0,1,2,3... Nach der Verwendung von Netzmorphismus und Pushout hat der j Tasks das Diagramm N_i, wobei $i = 2j - 1$.
Bislang haben wir beliebige Tasks dieses MSS Systems durch Prio ac-CTPN aufgebaut. Ein MSS System ist eine Komponierbarkeit eingebetteter Echtzeitsysteme. Eine der Haupteigenschaften von eingebetteten Echtzeitsystemen ist die Nebenläufigkeit von Tasks. Diese Diagramme N_{2j-1} stellen die gemeinsame Schnittstelle „Prozessor“ dar, viele gleiche Zustände. Das ist visuell sehr unübersichtlich und aufwendig in der grafischen Darstellung. Um noch dazu die Abhängigkeiten und Verbindungen zwischen diesen Diagrammen zu zeigen, muss man den komplizierten Mechanismus des Ereignissaustauschens durch Zwischenzustände benutzen. Deshalb ist auf diesem Gebiet ein Mechanismus für die leistungsfähige Modellierung sinnvoll. Der Ausweg aus dieser Problematik ist die Erweiterung der vorhandenen Standards auf gefärbte Diagramme. Diese entstehen durch Faltung von mehreren änlichen Diagrammen. Für j Tasks werden diese Diagramme N_{2j-1} j-mal gefaltet. Dann haben wir endlich ein einfaches Modell von diesem gültigen

MSS System auf einen Knoten. Die Beschreibung der Stellen und Transitionen des Diagrammes zeigen eine Variante des Modells. Jetzt entspricht es nicht mehr der Definition von priorisiertem ac-CTPN, sondern Definition von priorisiertem CTPN.

4.2 Beweis der Korrektheit

4.2.1 Korrektanforderung

Jetzt haben wir ein einfaches priorisiertes CTPNetz Diagramm mit beliebigen Tasks zu analysieren. Das komponierte Diagramm steht in Abbildung 4.5. Nach der Anwendung von Petri-Netz und Kategorie spezifizieren wir die MSS Architektur von der komplexen Struktur des Diagramms zu diesem einfachen Diagramm. Jetzt verifiziere ich die Anforderung nach Spezifikati-

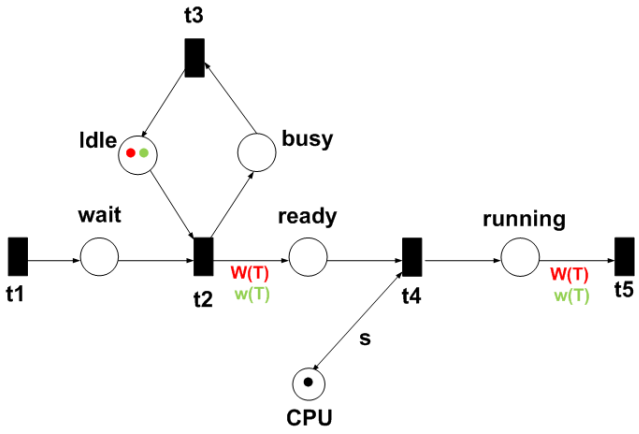

Abbildung 4.5: Komponierte Tasks

on und Garantie der Taskebene. Normalweise liefert die Verifikation ohne Zeitbewertung Aussagen zur Erreichbarkeit von Zuständen des modellierten Systems, zur Lebendigkeit, Konfliktfreiheit, Beschränktheit und Sicherheit. Konfliktfreiheit haben wir durch Prioriät auf Transition gelöst. Die Kapazität jeder Stelle in unserem Modell ist unbeschränkt. So haben wir kein Problem mit Beschränktheit. Sicherheitsanforderungen sind Anforderungen an die Stellen bzw. Markierungen, z.B obere und untere Schranken, die die Markierung einer Stelle nicht über- oder unterschreiten darf. Sicherheitsanforderungen können durch P-Invarianten verifiziert werden. Lebendigkeitsanforderungen sind Anforderungen an die Transitionen bzw. an das Schaltverhalten und fordern z.B. , dass das Hintereinanderschalten bestimmter Transitionen die Markierung bewahrt. Lebendigkeitsanforderungen können durch T-Invarianten verifiziert werden.

Die Verifikation des Zeitverhaltens eines Systems zeigt, ob die zeitlichen Restriktionen erfüllt werden. Deshalb verwende ich *Timed Demand Analysis* in dieser Architektur und berechne es für Tasks. Vorher müssen alle Tasks die gültige Bedingungen des System erfüllen.
Dann haben wir ein gültiges MSS System, siehe Diagramm „Komponierte Tasks“. Nach Abschnitt 3.6.7 Hierachische P/T Net Definition wird das Diagramm in folgender Abbildung 4.6 erweitert.

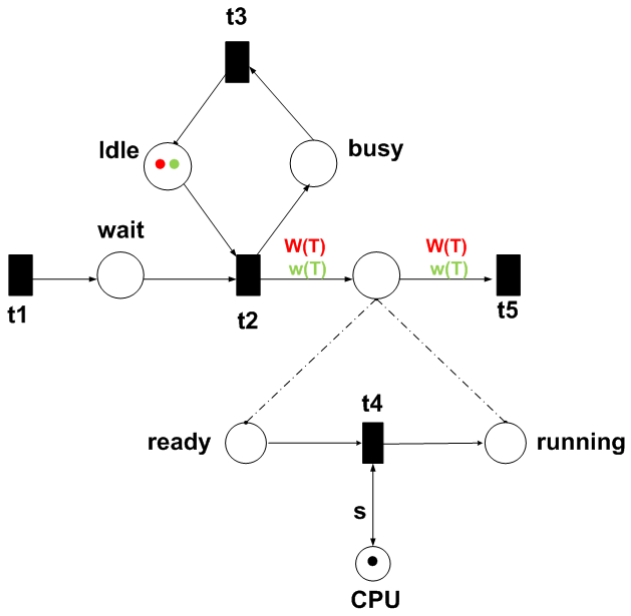

Abbildung 4.6: MSS Taskebene Hierarchie

4.2.2 Beweis

Es sei ein gültiges MSS-System S. Es existiert ein Knoten N in diesen System. Das dazugehörige Zeitdauernetz nach Kategorietheorie und Hierachie ist vereinfacht in Abbildung 4.6 darstellt. In diesem Netz $\forall T_i \in T^N$ gilt:

1. $M(P_{busy}(T_i)) \leq x \; ; \; M(P_{busy}(T_i)) > \emptyset \Rightarrow M(P_{idle}(T_i)) = \emptyset$
2. $M(P_{idle}(T_i)) \leq x \; ; \; M(P_{idle}(T_i)) > \emptyset \Rightarrow M(P_{busy}(T_i)) = \emptyset$
3. $M(P_{ready}) \leq y \; ; \; M(P_{ready}) = y \Rightarrow M(P_{running}) = \emptyset$
4. $M(P_{running}) \leq y \; ; \; M(P_{running}) = y \Rightarrow M(P_{ready}) = \emptyset$
5. Deadline einhalten, sowie die zeitlichen Restriktionen erfüllen:

$$\begin{aligned} & M(P_{ready}(T_i)) \neq \emptyset \quad \Rightarrow M(P_{idle}(T_i)) = \emptyset \\ \wedge & M(P_{idle}(T_i)) \neq \emptyset \quad \Rightarrow M(P_{ready}(T_i)) = \emptyset \end{aligned}$$

Berechnen der P-Invariants, um die Sicherheitsanforderung zu überprüfen. Aus Abbildung 4.6 hat das Netz der Task seine $\underline{pre}$ und $\underline{post}$ in Matrix-Darstellung (Tabelle 4.2 und 4.3).

$\underline{pre}$	t_1	t_2	t_3
$P_{supsended}$	s	$\emptyset$	$\emptyset$
P_{wait}	$\emptyset$	s	$\emptyset$
P_{idle}	$\emptyset$	s	$\emptyset$
P_{busy}	$\emptyset$	$\emptyset$	s
P_{ready}	$\emptyset$	$\emptyset$	$\emptyset$

Tabelle 4.2: pre-Matrix

$\underline{post}$	t_1	t_2	t_3
$P_{supsended}$	$\emptyset$	$\emptyset$	$\emptyset$
P_{wait}	s	$\emptyset$	$\emptyset$
P_{idle}	$\emptyset$	$\emptyset$	s
P_{busy}	$\emptyset$	s	$\emptyset$
P_{ready}	$\emptyset$	$w(t) \cdot s$	$\emptyset$

Tabelle 4.3: post-Matrix

Indexmatrix $\underline{I} = \underline{post} - \underline{pre}$:

$$\underline{I} = \begin{pmatrix} -s & \emptyset & \emptyset \\ s & -s & \emptyset \\ \emptyset & -s & s \\ \emptyset & -s & -s \\ \emptyset & w(t) \cdot s & \emptyset \end{pmatrix}$$

Lösung des Gleichungssystems $\underline{I}^T \cdot x = \emptyset$:

$$\underline{I}^T = \begin{pmatrix} -s & s & \emptyset & \emptyset & \emptyset \\ \emptyset & -s & -s & s & w(t) \cdot s \\ \emptyset & \emptyset & s & -s & \emptyset \end{pmatrix}$$

Durch Zeilenaddition $Z_2' = Z_2 + Z_3$ erhalten wir :

$$\begin{matrix} -s & s & \emptyset & \emptyset & \emptyset \\ \emptyset & -s & \emptyset & \emptyset & w(t) \cdot s \\ \emptyset & \emptyset & s & -s & \emptyset \end{matrix}$$

Also

$$x_1 = x_2 = w(t) \cdot x_5$$

$$x_3 = x_4$$

Für $x_3 = s$, $x_1 = \emptyset$ erhalten wir $x_4 = s$, $x_2 = \emptyset$ und $x_5 = \emptyset$, also als P-Invariante den Stellenvektor $I_{S_1} = (\emptyset, \emptyset, s, s, \emptyset)^T$. d.h $\Rightarrow$

$$M(P_{idle}) + M(P_{busy}) = s \tag{4.1}$$

Für $x_4 = \emptyset$, $x_1 = s$ erhalten wir $x_3 = \emptyset$, $x_2 = s$ und $x_5 = \frac{s}{w(t)}$, $I_{S_2} = (s, s, \emptyset, \emptyset, \frac{s}{w(t)})^T$. d.h $\Rightarrow$

$$M(P_{supsended}) + M(P_{wait}) + \frac{s}{w(t)} \cdot M(P_{ready}) = s \tag{4.2}$$

Verifikation:

- $M(P_{idle}(T_i)) \leq s$ folgt aus Formel 4.1
- $(M(P_{idle}(T_i)) \neq \emptyset \Rightarrow M(P_{busy}(T_i)) = \emptyset)$ folgt aus Formel 4.1
- $M(P_{busy}(T_i)) \leq s$ folgt aus Formel 4.1
- $(M(P_{busy}(T_i)) \neq \emptyset \Rightarrow M(P_{idle}(T_i)) = \emptyset)$ folgt aus Formel 4.1

Damit ist die Sicherheitsanforderung erfüllt.
Die Korrektanforderung von Punkt 1, 2 haben wir schon durch die Berechnung für die einzelne Task in Formel 4.1 gezeigt. Wir entfalten das Teilnetz inklusive $t_1, t_2, t_3, P_{wait}, P_{idle}, P_{busy}$ nach jeder Task. So hat jeder Task mit seiner eigenen Farbe nun das Teilnetz wieder. Es gilt: $M(P_{idle}(T_i)) + M(P_{busy}(T_i)) = x$ mit x, das immer Farbe von T_i zugeordnet wird, weil die Tasks in dem Teilnetz parallel laufen. Die Formel gilt analog für alle Tasks auf den Knoten N.
Im Architektur Diagramm besitzt dieser prio CTPN Hauptnetz Synchronisationgraph, die für diese Petri-Netzklasse spezifischen Eigenschaften: Jede Stelle besitzt genau eine Vorgängertransition. Nach Anwendung von Hierarchie haben wir das Teilnetz (Scheduling auf Prozessor) bekommen, trotzdem das Teilnetz immer noch zeitbehaftet ist, wie in vorheriger Spezifikation. Transition ist nach Zeit Quantum Einheit geschaltet. So ist es zeitlich unabhängig von der Abarbeitung der Task. Die laufende Task kann nur unterbrochen werden durch Tasks mit höherer Priorität. Hier zeige ich, dass die Schaltmodi der Transition nach Priorität Sequenz

$$\Upsilon_0 > \Upsilon_1 \ldots > \Upsilon_n, (n \geq 1)$$

geschaltet ist. Es gilt folgende Sequenz:

$$s_0 < s_1 < s_2 \ldots < s_n$$

ebenfalls

$$M_0 \overset{(t_0,s_0)}{\rightarrow} M_1 \overset{(t_1,s_1)}{\rightarrow} M_2 \ldots \overset{(t_{n-1},s_{n-1})}{\longrightarrow} < M_n$$

Es ist genauso wie ein normales ungefärbes Petri-Netz Schaltverhältnis. Durch Matrix-Darstellung von prio CTPN wird P-Invariant des Teilnetzes berechnet Es wird die Sicherheitsanforderung geprüft, dass alle Tasks eines Knotens im gültigen MSS System durch dieses Modell nach dem Bedingungen korrekt ausgeführt worden sind.

$\underline{post}$	t_4	$\underline{pre}$	t_4
P_{ready}	$\emptyset$		y
P_{CPU}	s		s
$P_{running}$	y		$\emptyset$

Tabelle 4.4: pre-post-Matrix

Indexmatrix $\underline{I} = \underline{post} - \underline{pre}$:

$$\underline{I} = \begin{pmatrix} -y \\ \emptyset \\ y \end{pmatrix}$$

Lösung des Gleichungssystems $\underline{I}^T \cdot x = \emptyset$:

$$\underline{I}^T = \begin{pmatrix} -y & \emptyset & y \end{pmatrix}$$

Also

$$x_1 = x_3$$

Für $x_1 = y$, $x_2 = \emptyset$ erhalten wir $x_3 = y$, also als P-Invariante den Stellenvektor $I = (y, \emptyset, y)^T$. d.h $\Rightarrow$

$$M(P_{ready}) + M(P_{running}) = y \tag{4.3}$$

y hat alle gefärbte Token Menge auf Stelle P_{ready}.
Verifikation der Punkt 3, 4 :

- $M(P_{ready}) \leq y$ folgt aus Formel 4.3
- $M(P_{ready}) = y \Rightarrow M(P_{running}) = \emptyset$ folgt aus Formel 4.3
- $M(P_{running}) \leq y$ folgt aus Formel 4.3
- $(M(P_{running}) = y \Rightarrow M(P_{ready}) = \emptyset)$ folgt aus Formel 4.3

Damit ist die Sicherheitsanforderung erfüllt.
Bislang haben wir dieses Modell teilweise nach den Eigenschaften von priorisiertem CTPNetzen untersucht und gezeigt, daß die Korrektanforderung erfüllt ist. Um zu prüfen, ob für die Zeit besonders die Deadline eingehalten wird, untersuchen wir die *response time* R von Tasks, mit $\forall T_i \in T^N$, $w(T_i) \leq R(T_i) \leq P(T_i)$.
Wie vorher der Formel der Invariant von P_{ready} und P_{idle}:

$$\begin{array}{ll} M(P_{ready}(T_i)) \neq \emptyset & \Rightarrow M(P_{idle}(T_i)) = \emptyset \\ \wedge M(P_{idle}(T_i)) \neq \emptyset & \Rightarrow M(P_{ready}(T_i)) = \emptyset \end{array}$$

Es gibt die Möglichkeit, daß eine höher priorisierte Task die niedriger priorisierte Task unterbricht, wenn diese nicht rechtzeitig innerhalb der Deadline ausgeführt wurde. In diesem Fall gilt die Formel nicht und das MSS geht zu einem Fehlerzustand. Wie im vorherigen Beweis, haben wir noch nicht die Beziehung zwischen Periode und Ausführungszeit bewiesen. Man kann nicht mit diesem Modell die Zeitanforderung von Scheduling berechnen. Deshalb verwenden wir als zusätzliche Verifikationmethode **TDA** dienen.

4.2.3 Time Demand Analysis

Mit Hilfe von *Time Demand Analysis* kann man die Zeitanforderung testen und berechnen. Die Task T_i, mit $j < i$, hat höhere Priorität als T_i. Dann können wir ihre worst-case response time R_i mit folgender Formel definieren:

$$R_i = w(T_i) + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P(T_j)} \right\rceil \times w(T_j) \tag{4.4}$$

Die Gleichung der Formel 4.4 ist schwer zu lösen, da R_i auf beiden Seiten erscheint.
Hier stelle ich das Verfahren vor, wie man die response times berechnen kann. Ebenso ihren Bezug zur Anzahl der Taskaufrufe innerhalb eines Intervalzeitraums und Ausführungszeit für jeden Aufruf der Task.
Wir verwenden zuerst eine Form, die: *halb-open* Interval $\left[t, t^{'}\right), t < t^{'}$, inklusive aller Werte von t bis $t^{'}$, aber nicht gleich $t^{'}$. Ein Funktion $Inputs(\left[t, t^{'}\right), j)$, deren Wert die Ankunftsanzahl von Tasks mit Prioriotät j innerhalb des Zeitraumes von *halb-open* Interval $\left[t, t^{'}\right)$ ist, wir definieren in Abbildung 4.7

$$Inputs(\left[t, t^{'}\right), j) = \left\lceil t^{'}/T_j \right\rceil \Leftrightarrow \lceil t/T_j \rceil$$

Die Ausführungszeit wird für die gesamte Anzahl von Taskaufrufen verbraucht.

$$Inputs(\left[t, t^{'}\right), j) \times w(T_j)$$

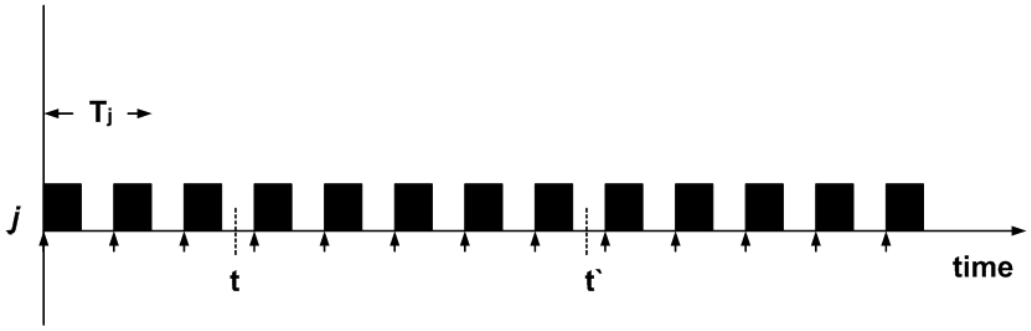

Abbildung 4.7: Input([t,t'),j)=5

Deshalb können wir die Ausführungszeit für die Anzahl der Tasksaufrufe wie folgt definieren: $Comp(\left[t,t^{'}\right),i)$. Alle Prioritäten sind höherer als i:

$$Comp(\left[t,t^{'}\right),i) = \sum_{j=1}^{i-1} Inputs(\left[t,t^{'}\right),j) \times w(T_j)$$

Die response time der Task mit Priorität i innerhalb des Intervals $\left[t,t^{'}\right)$ lassen wir durch die Funktion $R(t,t^{'},i)$ berechnen. Die Ausführungszeit auf Priorität i innerhalb des Intervals $\left[t,t^{'}\right)$ ist $t^{'} \Leftrightarrow t$. Dann ist die komplette Ausführungszeit innerhalb dieses Intervals für höhere Prioritäten $0 \ldots i \Leftrightarrow 1$ ist $Comp(\left[t,t^{'}\right),i)$. Falls das Ergebnis Null ist, wird der Prozess innerhalb dieses Intervals nicht preemptiv und erhält für diese gesamte Zeit der Ausführung der Task die Priorität i. Ist allerdings die Ausführungszeit nicht Null, d.h $Comp(\left[t,t^{'}\right),i) > 0$, so kann die response time nicht mehr kleiner als $t^{'} + Comp(\left[t,t^{'}\right),i)$ werden. Es existiert die folgende rekursive Definition dieser Funktion $R(t,t^{'},i)$:

$$\begin{aligned} R(t,t^{'},i) = & \text{ if } Comp(\left[t,t^{'}\right),i) = 0 \text{ then } t^{'} \\ & \text{else } R(t^{'},t^{'} + Comp(\left[t,t^{'}\right),i),i) \end{aligned}$$

Die worst-case response time kann also wie folgt definiert werden:

$$R_i = R(0, w(T_i), i)$$

Wenn keine Ausführungszeit auf $0 \ldots i \Leftrightarrow 1$ nötig ist, dann ist die response time auf Priorität i genau der Ausführungszeit $w(T_i)$. Die Auswertung der response time wird garantiert fertig werden, falls die folgende Formel gilt:

$$\sum_{j=1}^{i} (w(T_j)/P(T_j)) \leq 1$$

Die Korrektheit der response time war auch zu beweisen. Wir werden zeigen, daß die Lösung der Gleichung in Ausdrücken der Funktion R korrekt ist.

$$R_i = w(T_i) + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P(T_j)} \right\rceil \times w(T_j)$$

Zuerst erkennen wir, daß die Funktion $Comp$ über einem Interval definiert ist, daß es einige t_2 gibt, für die gilt:

$$Comp([t_1, t_3), i) = Comp([t_1, t_2), i) + Comp([t_2, t_3), i), t_1 \leq t_2 \leq t_3$$

Beweis

Sei $t^{'}$ die Summe der Ausführungszeit innerhalb des Intervals $[0, t)$ auf Priorität $0 \ldots i \Leftrightarrow 1$ plus die benötigte Zeit auf Priorität i. Also eine Invarianz INV für die rekursive Gleichung R ist

$$INV : Comp([0, t), i) + w(T_i) = t^{'}$$

Schritt 1: Die Initialbedingung $R(0, w(T_i), i)$ erfüllt die Invarianz.
Schritt 2: Bei der Induktionsannahme, gilt $R(t^{'}, t^{'} + Comp([0, t), i), i)$ der Invarianz auch.

$$w(T_i) + Comp(\left[0, t^{'}\right), i) = t^{'} + Comp(\left[t, t^{'}\right), i)$$

Danach wird für $0 \leq t \leq t^{'}$, Interval Arithmetik angewendet,

$$Comp(\left[0, t^{'}\right), i) = Comp([0, t), i) + Comp(\left[t, t^{'}\right), i)$$

Subtraktion beider Formeln. So erhalten wir,

$$Comp([0, t), i) + w(T_i) = t^{'}$$

Schritt 3: $R_i = t^{'}$ und

$$INV \wedge Comp(\left[t, t^{'}\right), i) = 0$$

Substitution für INV, also

$$Comp(\left[0, t^{'}\right), i) + w(T_i) = t^{'} \wedge R_i = t^{'}$$

Substitution für $Comp$, also

$$(\sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P(T_j)} \right\rceil \times w(T_j)) + w(T_i) = t^{'} \wedge R_i = t^{'}$$

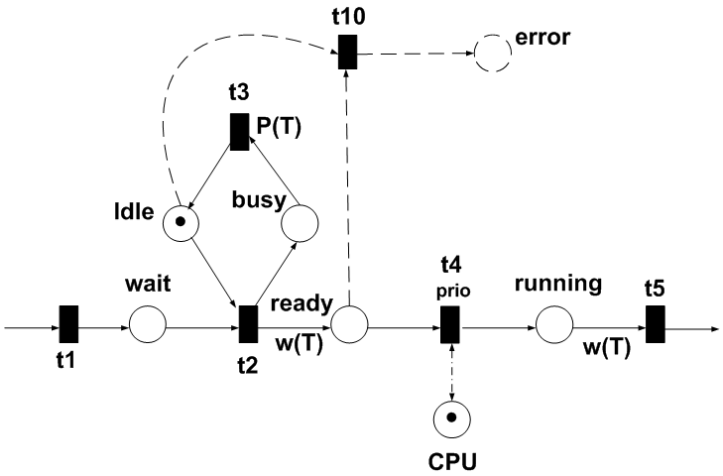

Abbildung 4.8: Modell mit Fehlerzustand

Dadurch können wir eine hinreichende und notwendige Bedingung für ein ausführbares System mit n Priorität jetzt definieren : $\forall i, 1 \leq i \leq n, R_i \leq P(T_i)$
Als nächsten betrachten wir eine Task, die wir in Abbildung 4.8 modelliert haben. Wie wir vorher schon bewiesen haben. Zusätzlich haben wir eine *Error* Stelle dem Modell hinzugefügt. Wenn die Deadline der Task nicht eingehaltet wird, tritt der Fehlerzustand auf. Hier setzen wir die höhere Priorität t_{10} als t_4 ein. Die gesamte Modellsstruktur haben wir schon vorher bewiesen, nur um die Komponierbarkeit von Tasks zu betrachten. Es kommt nur zum *Error*, falls eine Task ihre Deadline nicht einhalten kann.

Eine Task

Es sei eine Task T_i im MSS mit ac-CTPNetz 4.8 modelliert, Anfangszustand M_0 hat ein Token beliebiger Farbe, weil es im MSS nur eine Task gibt, falls $w(T_i) = R(T_i) \leq P(T_i)$, hier ist z_1 Quantums Zeitduaer der CPU und $P(T_i) = z_2$:

$$M_0 \xrightarrow{t_2} M_1 \overset{w(T_i)(t_4,z_1)}{\longrightarrow} M_2 \overset{t_3,z_2-w(T_i)z_1}{\longrightarrow} M_0 \tag{4.5}$$

und weitere Zustände werden durch $M_2 \xrightarrow{t_5} M_3$ aktiviert. Das führt uns zur zweiten Ebene, die wir nicht weiter betrachten. Wie in Formel 4.5 ist trivialweise die Garantie für die Anzahl der Tokens auf P_{wait} Stelle immer erfüllt. Die Task hat also immer ihre Deadline eingehalten. Nur wenn $w(T_i) = R(T_i) > P(T_i)$, dann kommt es zu einem *Error* Zustand. MSS kann jetzt nicht mehr aktiv sein und auch nicht weiter ausgeführt werden, da sich keine Tokens mehr auf *Idle* Stelle befinden. Offensichtlich gilt auch ohne Analyseberechung von TDA.

Komposition von zwei Tasks

Seien N_0, N_1, N_2, N_3 ac-CTPNetz in MSS. N_0 ist CPU der Knoten. N_1, N_2 sind zwei verschieden farbige Tasks auf die Knoten, mit $\forall i \in 1, 2, w(T_i) \leq P(T_i)$ erfüllt. Die Task der kurzen Deadline hat immer eine höhere Priorität. Durch Pushout in Prio ac-CTPNetz bekommen wir das Netz N_3. Dann verwenden wir die Faltung für das ac-CPTNetz, so erhalten wir das prio CT-PNetz in folgender Abbildung 4.9.

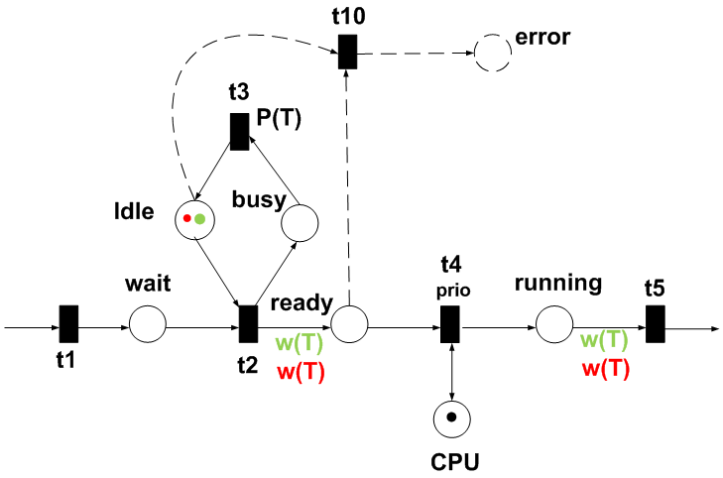

Abbildung 4.9: 2 Taskfalten

Die Ausführungszeit von jeder Task ist kleiner als ihre Periode. Ihre gesamte Last U ist kleiner gleich $2(2^{1/2} - 1) = 0.828$. Das entspricht RMS. Man kann jetzt TDA dafür verwenden und verifizieren, ob unter RMS Schedulingsverfahren, die zwei komponierten Tasks immer noch ausführbar sind, d.h $R(T_i) \leq P(T_i)$.

Beispiel:

$$P(T_1) = 3, w(T_1) = 1$$
$$P(T_2) = 9, w(T_2) = 3$$

Offenbar ist die Last der Tasks

$$U = 1/3 + 3/9 = 0.666 < 0.828$$

Also berechnen worst-case $R(T_2) = R(0, 3, 9)$ mit dem Startzeitpunkt 0. So untersuchen wir die Teilnetze durch Hierarchische P/T Netze in der Abbildung 4.10 gekennzeichnet, ob alle komponierten Tasks alle rechtzeitig vor ihrer Deadline ausgeführt sind.

$$\begin{aligned} R(0,3,9) \quad & = \text{if } Comp([0,3),9) = 0 \text{ then } 3 \\ & \text{else } R(3, 3 + Comp([0,3),9), 9) \end{aligned}$$

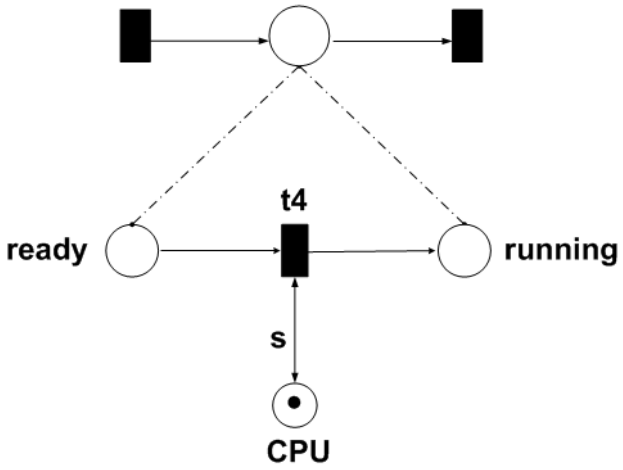

Abbildung 4.10: Teilnetze

$$\begin{aligned} Comp([0,3),9) &= Input([0,3),3) \times 1 \\ &= 1 \end{aligned}$$

Weiter berechnen für $R(3,4,9)$:

$$\begin{aligned} Comp([3,4),9) &= Input([3,4),3) \times 1 \\ &= 1 \end{aligned}$$

weiter für $R(4,5,9)$:
$Comp([4,5),9) = 0$
Die response time $R(0,3,9) = R(0,5,9)$ für Task T_2 ist 5 Einheitszeit. Eindeitig haben wir gesehen, daß beide Tasks erfolgreich komponiert werden und ihre Deadline eingehalten wird. Ist es ein Sequenzschaltungsverhältnis, gilt folgende Markierungsänderung:
$M_0 \overset{t_4(r,1)}{\rightarrow} M_1 \overset{t_4(g,1)}{\rightarrow} M_2 \overset{t_4(g,1)}{\rightarrow} M_3 \overset{t_4(r,1)}{\rightarrow} M_4 \overset{t_4(g,1)}{\rightarrow} M_5$
Man kann auch beliebig Ausführungszeit und Periode für 2 Tasks nach TDA unter RMS Bedingung verifizieren, ob sie immer korrekt eingehalten werden.

Komposition von drei Tasks

Analog zu ihrer Struktur haben wir die *pushout* der Task verwendet und sie gefaltet. Diese werden auch durch TDA berechnet. Die zwei Tasks bleiben unverändert. Es kommt Task T_3 dazu, mit $P(T_3) = 6, w(T_2) = 2$. Last von der Tasks $U = 1/3 + 2/6 + 3/9 = 0.999 < 1$ und $0.999 > 3(2^{1/3} - 1)$. Berechnen für $R(0,3,9)$:

$$Comp([0,3),9) = Input([0,3),3) \times 1$$

$$
\begin{aligned}
&+Input([0,3),6) \times 2 \\
&= 3
\end{aligned}
$$

Weiterhin für R(3,6,9):

$$
\begin{aligned}
Comp([3,6),9) &= Input([3,6),3) \times 1 \\
&= 1
\end{aligned}
$$

Weiter für R(6,7,9):

$$
\begin{aligned}
Comp([6,7),9) &= Input([6,7),3) \times 1 \\
&+Input([6,7),6) \times 2 \\
&= 3
\end{aligned}
$$

Weiter für R(7,10,9):

$$
\begin{aligned}
Comp([7,10),9) &= Input([7,10),3) \times 1 \\
&= 1
\end{aligned}
$$

$Comp([10,11),9) = 0$
Die response time $R(0,3,9) = R(0,11,9)$ für Task T_2 ist 11 Einheitszeit. Deadline von T_2 ist nun nicht mehr eingehalten, so sind diese Systeme nicht mehr sicher und es kommt zum $Error$.
Wenn die Periode von T_3 von 6 nach 60 geändert wird, dann ist $U = 1/3 + 2/60 + 3/9 = 0.699 < 3(2^{1/3} - 1)$ unter RMS erfüllt. Es wird nochmal berechnet: alle Deadlines sind korrekt eingehalten.

Komposition von n Tasks

Für beliebige Tasks können wir, das Verfahren, dass wir für zwei oder drei Tasks verwendet haben, entsprechend erweitern. Anschließend berechnen wir die gesamt Last, die unter RMS Bedingung erfüllt ist. Danach verifizieren wir nun durch TDA, um die Korrektheit nachzuprüfen und zu beweisen, ob das MSS die gewünschte Erreichbarkeit erhält.

4.2.4 Ergebnis von Verifikation

Durch obige Analyse, Verifikation und Berechnung des zweiten Teilbeweises erhalten wir das Ergebnis, daß die Garantie der Korrektheit der Taskebene von MSS, sowie seine Struktur und Scheduling eingehalten werden. Außerdem haben wir weiter entdeckt, daß wir durch Verwenden des TDA sogar noch die Tasklast optimieren können. z.B die drei Tasks, die wir vorher verwendet haben: Falls die Periode von T_3 von 6 nach 8 geändert wird, dann ist $U = 1/3 + 2/8 + 3/9 = 0.916 > 3(2^{1/3} - 1)$ unter RMS Bedingung nicht erfüllt. Trotzdem können die drei Tasks ausführbar sein und immer ihre Deadline einhalten. So können wir also dieses Verfahren einsetzen, um mehr Leistung zu erzielen.

Kapitel 5

Zusammenfassung und Ausblick

Die ständig steigende Komplexität und die Anforderungen für immer mehr Sicherheit von komponierbaren, eingebetteten Echtzeitsystemen fordern die Einbeziehung von graphischen und objektorientierten Darstellungsmitteln, sowie Werkzeugen für den Entwurf solcher Systeme. Diese ermöglichen einen effektiven Entwurf und dessen parallele Überprüfung.

Zentrale Ergebnisse dieser Arbeit: Spezifikation von Echtzeit Architekturen und Verifikation der Architekturen mit passendem Verfahren.
Es besteht aus drei Teilen:

- Durch Architektur Definition können wir seine konkreten Eigenschaften kennenlernen. Dann beschreibe ich, wie Kategorietheorie auf eine Menge verschiedener Modellierungsverfahren angewendet werden kann. Damit können Spezifikation von Architekturen erfüllt werden.

- Danach stelle ich einige populäre Verifikationsmethoden vor. Ich vergleiche ihre Gemeinsamkeiten, Vorteile und Nachteile. Dadurch können wir genauer erfahren und deutlich erkennen, welche konkreten Verfahren auf welche Situation angewendet werden können. So erhalten wir Gefärbte Petri-Netze als Verifikationsmethode. Ich habe also die Standard Definition mit Zeit und Priorität erweitert. Ich werde darauf Kategorie erfolgreich anwenden. Außerdem werden ein paar nützliche Netzstrukturen vorgestellt, wie z.B Netz-Hierarchie, Transformation und die Netzeigenschaften Faltung und Entfaltung. Damit kann ich das Modell optimal erfüllen und verifizieren.

- Ich gebe ein konkretes Beispiel-MSS mit seinen Zeiteigenschaften, Parametern und Garantie der Deadline. Nach Modellierung mit CTPN verifiziere ich die Korrektheit mittels P/Netz Eigenschaften. Dann be-

rechne ich seine response time durch TDA Algorithmus. Durch Überprüfung von response time und Periodezeit ist auch die Einhaltung der Deadline garantiert.

Folgende Betrachtung beschäftigt sich mit der Unterstützung der Transformationen durch praktisch einsetzbare Tools. Dabei ist es wünschenswert, ein gemeinsames Tool für die drei dynamischen Diagrammtypen zu schaffen. Das soll nicht nur Algorithmen zur direkten Transformation in Gefärbte Petri-Netze, sondern auch die Rücktransformation und die Transformation von Diagrammen ineinander, unter Einbeziehung der ausgearbeiteten Verifikationsalgorithmen unterstützen. Ein geeignetes Petri-Netz-Tool mit graphischer Darstellungsmöglichkeit sollte auch das Eingabeformat des Transformationstools exportieren können. Gleiches gilt für die Verifikation auf Basis Gefärbter Petri-Netze, deren Ergebnisse auf das Niveau der Ursprungsdiagramme zurück transformiert werden sollen. Realisierung von Simulatoren wäre auch vorteilhaft für die bessere Untersuchung der Modelle.

Literaturverzeichnis

[1] Bitsch, F., Gunzert M.: *Formale Verifikation von Softwarespezifikaionen in ASCET-SD und MATLAB.* (2000). - Fachtagung Verteilte Automatisierung - Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung, http://www.ias.unistuttgart.de/de/forschung/pub/paper/va2000 paperbt.pdf

[2] Capellmann C., Dibold H.; *Petri Net Based Specifications of Services in an Intelligent Network. Experiences Gained from a Test Case Application.* In: Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in Computer Science Vol. 691 (1993), S. 542-551

[3] Capellmann C., Christensen S., Herzog U.; *Visualising the Behaviour of Intelligent Networks.* In: Services and Visualization, Towards User-Friendly Design, Lecture Notes in Computer Science Vol. 1385 (1998), S.174-189

[4] Capellmann C., Dibold H., Herzog U.; *Using High-Level Petri Nets in the Field of Intelligent Networks.* In: Application of Petri Nets to Communication Networks, Lecture Notes in Computer Science Vol. 1605 (1999), S.1-36

[5] Chaves J.: *Formal Methods at AT&T*: An industrial usage report. In: Proceedings of Formal Description Techniques IV (1992), S. 83-90

[6] Clarke E. M., Grumberg O., Hiraishi H., Jha S., Long D. E. ; McMillian, D. L. ; Ness, L. A.: *Verification of the Futurebus+ cache coherence protocol.* In: Proceedings of CHDL (1993)

[7] Claude G., Rüdiger V.; *Petri Nets for Systems Engineering , -A Guide to Modeling, Verification, and Applications*; Publishing House of Electronics Industry; ISBN 7-121-00781-9 -Beijing: 2005.6.

[8] Colin S.; *Modal and Temporal Logics*; In Background: Computational structures, volume 2 of Handbook of Logic in Computer Science; Clarendon Press, OXFORD, 1992 S.477-563

[9] David D., Rajeev A.; *Automata, Languages and Programming: 17th International Colloquium*; Band 443 aus Lecture Notes in Computer Science", Warwick, England, European Assoc. Theoret. Comput. Sci., Springer-Verlag: 1990.

[10] Dill D. L., Drexler A. J., Hu A. J., Yang C. H.: *Protocol verification as a hardware design aid.* In: IEEE International Conference on Computer Designs: VLSI in Computers and Processors (1992), S. 522¨C525

[11] E.A. Emerson; *Temporal and Modal Logic* In Formal models and semantics, volume B of Handbook of Theoretical Computer Science; Elsevier, 1990. S.997-1072

[12] Fengler W.; *Technische Applikation von Petrinetzten, Petri-Netze in Technik und Wirtschaft*; Technische Universität Ilmenau, Institut für Technische und Theoretische Informatik, Fachgebiet Rechnerarchitekur, 88

[13] Figueiredo J.C.A., Kristensen L.M.; *Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols.* In: Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN (1999), S. 21-40

[14] Gordon S., Billington J.; *Modelling and Initial Analysis of the Resource Reservation Protocol Using Coloured Petri Nets.* In: Proceedings of the Workshop on Practical Use of High-level Petri Nets (2000), S. 91-110

[15] Harry R. Lewis; *A logic of concrete time intervals.* In: Fifth Annual IEEE Symposium on Logic in Computer Science; Philadelphia, Pennsylvania. IEEE Computer Society Press (1990), S. 380-389

[16] Hooman J. ; *Specification and compositional verification of real time systems*; ISBN 3-540-54947-1; -Berlin: Springer, 1991.

[17] Hubert P., Jensen K., Shapiro R.M.; *Hierarchies in Coloured Petri Nets.* In: Advances in Petri Nets 1990. Lecture Notes in Computer Science 483, Springer Verlag, 1991.

[18] Jan R.; *Komponierbarkeit eingebetteter Echtzeitsysteme*; Dissertation.

[19] Jensen K.; *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use Volume 1*; EATCS Monographs on Theoretical Computer Science. Springer-Verlag, -Berlin, 1992.

[20] Jensen K.; *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use Volume 2*; EATCS Monographs on Theoretical Computer Science. Springer-Verlag, -Berlin, 1995.

[21] Jensen D., Rozenberg G.; *High-level Petri Nets - Theory and Application*; Springer-Verlag, Berlin, 1991

[22] J. Padberg, U. Prange, H. Ehrig, C. Ermel, K. Hoffmann; *Petri-Netze-Skript, Modellierung, Strukturierung und Kompositionalität*; 2005.

[23] Karlis .C; *Decidability of bisimulation equivalences for parallel timer processes*; In G.V. Bochmann and D.K.Probst(Herausgeber), Computer Aided Verification, Fourth International Workshop, CAV92, Band 663 aus Lecture Notes in Computer Science. Springer-Verlag, Montreal, -Kanada, 1992

[24] Knoke M. ; *Verteilte optimistische Simulation von stochastischen farbigen Petrinetzen* -Berlin: dissertation.de, 2007, ISBN 978-3-86624-207-4.

[25] König R., Quaeck L.: *Petri-Netze in der Steuerungstechnik*; -Berlin, Verl.Technik, c 1988.

[26] König R. und Quäck L.; *Petri-Netze in der Steuerungs- und Digitaltechnik*; R.Oldenbourg Verlag, München 1989.

[27] Lunze J.; *Ereignisdiskrete Systeme: Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen*; ISBN 978-3-486-58071-6, ISBN 3-486-58071-X -München: Oldenbourg 2006.

[28] Julia P.; *Safety preserving transformation of coloured Petri Nets* -Berlin ; Fachbibliothek Informatik, 2002.-21 Bl. (Forschungsberichte der Fakultät IV, Elektrotechnik und Informatik ; 2000,13)

[29] Julia P. ; *Specification architectures : definition, semantics and transformation* -Berlin ; Techn. Univ. Berlin, Fakultät IV, Elektrotechnik und Informatik, 2002.- (Forschungsberichte der Fakultät IV, Elektrotechnik und Informatik ; 2002,11)

[30] Petri C. A.; *Kommunikation mit Automaten*; Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn 1962.

[31] Pnueli A.; *The Temporal Logic of programs* In Proc. 18th Ann. IEEE Symp. on Foundation of Computer Science; 1977 S.46-57

[32] Rajeev A.; *Techniques for Automatic Verification of Real-time Systems*; Dessertation, Stanford University, Kaliformnien; 1992.

[33] Reisig W.; *Systementwurf mit Netzen*; Berlin, Heidelberg, New York, Tokyo: Springer, 85

[34] René David, Hassane Alla; *Discrete,Continuous,and Hybrid Petri Nets*; ISBN 3-540-22480-7 -Berlin: Springer, 2005.

[35] Rosenstengel B., Winand U.; *Petri-Netze: Eine anwendungsorientierte Einführung. 4.*, verbesserte und erweiterte Auflage, Friedr. Vieweg and Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1991.

[36] Schnieder E.; *Prozessinformatik: Automatisierung mit Rechensystemen*; Einführung mit Petrinetzen. 2., erweiterte Auflage, Friedr. Vieweg and Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1993.

[37] Starke M.; *Petri-Netze*; Herausgegeben vom Zentralinstitut für Kibernetik und Informationsprozesse der Akademie der Wissenschaften der DDR, Berlin, Juni 1978.

[38] Thomas.A Henzinger, Zohar Manna and Amir Pnuell; *Temporal proof methodologies for real-time systems* In: Conference Record of the 18th Annual ACM Symposium on Principles of Programming Languages; Orlando, Florida; 1991, S 353-366

[39] Thomas.A Henzinger, Xavier Nicollin, Joseph Sifakis and Sergio Yovinne; *Symbolic model checking for real-time systems* In: Seventh Annual IEEE Symposium on Locic in Computer Science; Santa Cruz, Kalifornien; 1992. IEEE Computer Society Press, S. 394-406

[40] Verification & Testing Research Group. 2000. - http://www.dcs.shef.-ac.uk/research/groups/vt/mission.html

[41] Wilke T. ; *Automaten und Logiken zur Beschreibung zeitabbängiger Systeme*; Inst. für Informatik und Praktische Mathematik der Christian-Albrechts-Uni. zu Kiel, 1994.

[42] Wintz S.; *Hierarchische Analyse technischer Systeme mit Petri-Netzen*; -Düsseldorf; VDI-Verlage; 1995. (Berichte aus dem Institut für Regelungstechnik, RWTH Aachen)

[43] Wu ZheHui; *Grundlage der Petri nets*; ISBN 7-111-18278-2, -Beijing: China Machine Press, 2006.4.

[44] Xavier N., Joseph S. und Sergio Y.; *From ATP to timed graphs and hybrid systems*; Acta Information 30(2), 1993, S. 181-202