Modellierung und Verifikation von verteilten/parallelen Informationssystemen

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt

der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Msc. Inf. Khaoula Al Ali
geboren am 01.7.1970 in Damaskus, Syrien

Gutachter:
1. Univ.Prof. Dr.-Ing. habil. Wolfgang Fengler,TU Ilmenau
2. Prof. Dr.-Ing. habil. Herwig Unger, Fernuniversität Hagen
3. Dr.-Ing. Roger Knorr, IBM Stuttgart Deutschland GmbH

Datum der Einreichung: 10.11.2010
Datum der wissenschaftlichen Aussprache: 27.01.2011

urn:nbn:de:gbv:ilm1-2011000013

Danksagung

Die vorliegende Arbeit entstand im Fachgebiet Rechnerarchitektur der Fakultät Informatik und Automatisierung an der Technischen Universität Ilmenau.

An dieser Stelle möchte ich mich bei all denen bedanken, die zum Gelingen dieser Arbeit beigetragen haben. Herrn Prof. Dr.-Ing. habil. Wolfgang Fengler als Betreuer gilt mein besonderer Dank. Dieser umfasst die fortwährende Unterstützung in vielfältigster Form und die besonders wertvolle fachliche Beratung.

Besonders bedanke ich mich bei Herrn Dr. Ing. Bernd Däne für die Beratung zu allen meinen Fragen und als Konsultationspartner bei der Fertigstellung der schriftlichen Arbeit, sowie die sprachliche Korrektur.

Ich bedanke mich weiterhin bei Dr. Ing. Alexander Fleischer für die Beratungen zur Fertigstellung der schriftlichen Arbeit sowie sprachliche Korrektur.

Ich möchte mich hier bei allen Mitarbeiter des Fachgebietes Rechnerarchitektur bedanken, die mich während Promotionzeit durch Beratung und als Gesprächspartner unterstützt haben, vor allen Johanns Klöckner, Dr. Ing. Olga Fengler und Dr. Ing. Alexander Pacholik.

Außerdem bedanke ich mich bei Dr. Ing. Oswald Kowalski für ihre Unterstützung.

Besten Dank an meine Eltern, meine Töchter (Ghadir und Noura), meinen Mann, meine Geschwister, meine Schwiegereltern und meine Freunde, die mich immer motiviert und unterstützt haben.

Inhaltsverzeichnis

Zusammenfassung

Petrinetze werden in vielen Bereichen als Modellierungstechnik verwendet. Die verschiedenen Einsatzgebiete und Modellierungsziele erfordern mittel unterschiedliche Typen von Petrinetzen.
Die höheren Petrinetze eignen sich gut zur Formalisierung des Verhaltens verteilter Informationssysteme zum Zweck der Verifikation und Analyse.

Eine Klasse erweiterter gefärbte Petrinetze (HCPN-ST) mit strukturierten Marken wird in dieser Arbeit vorgestellt und am Beispiel erläutert. An diesem konkreten Modell wird die Analyse und Verifikation demonstriert.

Das Beispiel-Algorithmenmodell wird in ein CPN transformiert. Der Algorithmus der Transformation wird vorgestellt. Diese Transformation wird durchgeführt, um die erweiterte Methode mit Software -Tools zu analysieren und verifizieren. Mit Peneca Chromos wird das Modell editiert und simuliert und einige Eigenschaften werden analysiert. Die weitere Analyse und Verifikation erfolgt mit dem Tool INA. Es folgt die Erweiterung und Integration in den gesamten Analyseprozess und die Verifikationsmethodik. Abschließend wird die Notwendigkeit und Motivation zur Erweiterung von dynamischen Modellierungsansätzen für die Analyse von verteilten Informationssystemen behandelt und damit die Zielstellung der Arbeit erreicht.

Abstract

Petri nets are used in many fields as modelling technique. The different usage areas and modelling objectives require different classes of Petri nets.

Powerful high level Petri nets and especially coloured Petri nets are well suited for describing behavior of distributed information systems in order to verify and analyse them.

Extended coloured Petri nets with structured marks are presented in this work. An example is used in order to demonstrate the analysis and verification steps.

This example algorithm is modeled with extended coloured Petri nets (HCPN-ST). It is transformed into coloured Petri nets, in order to simulate, analyse and verify the method with existing software tools.

The model is simulated and analysed with PENECA Chromos tool, although it cannot verify all properties, but it allows to interoperate with INA tool. The remainder of the analysis and verification is done in the INA tool. The above mentioned steps are extended and integrated into the complete analysis process and the verification methodology. Finally, the need and motivation for the extension of dynamic approaches modelling for the analysis of distributed information system is elaborated to accomplish the goal of the work. We succeed to validate, that the extended formal method is an effective method to model and analyse distributed information systems.

Kapitel 1 Motivation und Einführung

1.1 Einführung

Die Entwicklung von schnellen lokalen Computernetzwerken (LAN, Local Area Networks), die im letzten halben Jahrhundert stattfand, erlaubte es, Hunderte von Maschinen innerhalb eines Gebäudes zu verbinden und innerhalb weniger Mikrosekunden Informationen zwischen den Maschinen auszutauschen. Dagegen können WAN (Wide Area Networks) Millionen von Maschinen auf der Welt mit unterschiedlichen Geschwindigkeiten verbinden.
Diese Technologien führten dazu, dass es heute nicht nur möglich, sondern unkompliziert ist, Computersysteme zusammenzustellen. Normalerweise spricht man von Computernetzwerken oder verteilten Systemen, im Gegensatz zu den früheren zentralen Systemen, die aus einem einzelnen Computer mit oder ohne seinen Peripheriegeräten bestanden. Verteilte Informationssysteme sind spezielle, meist komplexe verteilte Systeme, die sich durch folgende Merkmale auszeichnen[Ham05]:

- Meistens sind sie sehr groß.
- Sie sind sehr datenorientiert, d.h. die Datenhaltung steht im Zentrum der Anwendung. Üblicherweise werden sie in einer Datenbank verwaltet.
- Verteilte Informationssysteme sind meistens auch sehr nebenläufig, was sich durch eine große Anzahl an parallel arbeitenden Benutzern äußert.
- Sie sind extrem interaktiv durch graphische Benutzerschnittstellen

Als bekanntestes Beispiel wird das Internet in der Literatur behandelt, mit zahlreichen Anwendungen, die das Internet als Infrastruktur verwenden. Dieses Netz ist auf der ganzen Welt verteilt und transportiert dabei eine unglaubliche Menge von Informationen

1.2 Ziel der Arbeit

Verteilte Algorithmen sind Algorithmen in verteilten Systemen, und Peer to Peer (P2P) Systeme sind Beispiele für diese. Die P2P Algorithmen werden im Rahmen dieser Arbeit als konkretes Beispiel modelliert und verifiziert. Diese Algorithmen basieren auf Wanderern und wurden zur Unterstützung der Suche in P2P Netzen entwickelt.
Unter Methode wird hier im engeren Sinne eine Vorschrift für die Erstellung von Modellen verstanden.
Die Notwendigkeit der Anwendung formaler Beschreibungstechniken ist allgemein vor allen Beschreibungen wegen der Eindeutigkeitsproblematik und der Möglichkeit weiterer analytischer Behandlungen anerkannt [Kön87][Kru85].

Hauptanliegen dieser Arbeit ist es, einen Beitrag zur Verbesserung der formalen Spezifikation und Modellierung von verteilten Systemen auf Basis höherer Petrinetze zu leisten.
Es werden insbesondere zwei Richtungen verfolgt. Zum ersten sollen Ansätze untersucht werden. Dabei werden gezielt Erweiterungen und Einschränkungen bestehender Netzdefinitionen für unterschiedliche Ansätze entwickelt. Zum zweiten soll die Technik der Petrinetze durch Erweiterung und durch die Übernahme einer Vorgehensweise aus anderen Ansätzen weiter entwickelt werden, wobei die Möglichkeiten der Petrinetze benutzt werden.

Die Modellierungsmittel müssen einerseits immer leistungsfähiger werden, um diese zunehmende Komplexität zu beherrschen. Andererseits sollten sie ingenieurtechnisch gut handhabbar sein.
Petrinetze stellen eine Familie verschiedener Modellierungssprachen dar, die gemeinsamen

Prinzipien genügen, aber sehr unterschiedliche Ausprägungen haben und in verschiedenen Anwendungsbereichen eingesetzt werden.
Außerdem ist die Petrinetz-Theorie gut geeignet, das dynamische Verhalten komplexer, paralleler Systeme auf graphentheoretischer Grundlage zu beschreiben und deren Eigenschaften zu untersuchen. Deshalb hat die Wahl der Petrinetze als Modellierungsmittel eine besondere Bedeutung.
Die Wahl der höheren Petrinetze hat mehreren Gründe: Sie verfügen über eine intuitive grafische Darstellung, sie sind ausführbar; es können hierarchische Modelle konstruiert werden, es ist möglich, zeitbewertete Modelle für verschiedene Aktivitäten in einem System zu verwenden, und es gibt ausgereifte und erprobte Werkzeuge für die Erstellung, Simulation und Analyse von gefärbten Petrinetz-Modellen.
Trotzdem beschränken gefärbte Petrinetze die Möglichkeiten der Modellierung einer wichtigen Gruppe von Algorithmen, die als Lösung für die Suche und Selbstverwaltung auf P2P Netzen entwickelt werden. Diese Algorithmen werden in Kapitel 2 präsentiert und deren Analyse in den Kapiteln 5 und 6.

Ziel dieser Arbeit ist eine formale einheitliche Modellbildungsmethode für die verschiedenen Algorithmen. In dieser Arbeit wird diese Methode am Beispiel erläutert und gezeigt,dass diese Methode die Anforderungen erfüllt und sinnvoll ist. Außerdem ist die Modellierung dieses Algorithmus zur Simulation und formaler Verifikation vorsehen.

Die vorliegende Arbeit ist der Aufgabe gewidmet, durch neue Ansätze mit Petrinetzen den Bereich der Analyse von verteilten Systemen zu erweitern und damit einen Beitrag zur Modellierung der genannten Algorithmengruppen zu leisten.
Transformationen von dieser Methode in gefärbte Petrinetze sind für alle Elemente der Netze definiert. Die Transformation wird durchgeführt, um die Modelle mit Software-Tools simulieren, analysieren und verifizieren zu können. In dieser Arbeit werden die Software-Tools Peneca Chromos und INA benutzt. Damit werden die vorgestellten Modelle simuliert, analysiert und verifiziert. Diese beiden lassen sich zusammen günstig anwenden.
Bei den formalen Analysetechniken werden oft zwei unterschiedliche Ebenen der Verifikation angewendet [Kön87],[Fre90]. Mit der allgemeinen Analyse werden Eigenschaften nachgewiesen, die unabhängig sind von der speziellen Semantik des Modells. Hingegen wird die spezielle Analyse durch die Semantik des Algorithmenmodells bestimmt [Kön87]. In dieser Verifikationsebene wird versucht, die Korrektheit und Vollständigkeit des Modells gegenüber dem beschriebenen Verhalten zuweisen. Dazu gehören Datenspezifikationen, Fehlererkennung, Fehlerbehandlung und Zeitspezifikation. Aber die methodische Trennung zwischen allgemeiner und spezieller Analyse ist im Rahmen der Petrinetz-Analyse nicht immer eindeutig möglich.

Die funktionelle Analyse behandelt Eigenschaften, welche qualitativ (z.B Lebendigkeit, Beschränktheit) oder quantitativ (z.B. Platzkapazität) sein können.
In unserem Fall wird erstens der Algorithmus beispielsweise verbal beschrieben und mit erweiterten höheren Petrinetzen modelliert. Mit einem Werkzeug wird das Algorithmenmodell simuliert, damit wird die Korrektheit und der vollständig festgelegte Ablauf des Modells gegenüber dem in der spezifizierten Aufgabe, festgelegten Ablauf des Algorithmus, und seinen Mechanismen nachgewiesen. Dazu gehören Fehlererkennung und Zeitangaben.

1.3 Inhalt der Arbeit

Der Aufbau dieser Arbeit gliedert sich in mehrere Kapitel, wobei dieses Kapitel dazu dient, die Motivation des Themas darzulegen und weiterhin eine grundlegende Einordnung zu

liefern. Nach dieser Einführung und Erklärung des Ziels der Arbeit wird im zweiten Kapitel ein Überblick über verteilte Systeme gegeben, mit verschiedenen Definitionen für verteilte Systeme, die aus unterschiedlichen Quellen zitiert und aus unterschiedlichen Sichten betrachtet werden. Außerdem wird die Verteilung auf abstrakte Ebenen präsentiert. Schließlich wird gezeigt, welche Vor- und Nachteile diese Systeme mitbringen können. Als Übergang zu Peer to Peer Systemen wird Middelware identifiziert. Peer to Peer Systeme werden als verteilte Systeme definiert und klassifiziert. Es wird auf das Problem der Organisation und Suche in diesen Systemen eingegangen, um dieses als zentrales Problem zu identifizieren. Welche Mechanismen und Konzepte zur Suche in P2P Netzen verwendet werden, welche Vorteile und Nachteile sie haben, wird in Kapitel 2 beantwortet.Die Algorithmen, die zur Unterstützung der Suche in P2P Netzen entwickelt werden, werden auch in diesem Kapitel erläutert, weil einer dieser Algorithmen als konkretes Beispiel auf Basis der entwickelten formalen Methode modelliert wird, Diese Modellierungsmethode wird auf Basis höherer Petrinetze in dieser Arbeit erweitert.
In Kapitel 3 werden Modellierungsmethoden vorgestellt und entsprechend ihrer Anwendungen klassifiziert. Es folgt die Modellierung mit Wertbereichen, die als mathematische Grundlage für die formal definierte Modellierungsmethode dient und sich gleichzeitig als Modellierungsmittel für einen Teil eines Systems, wie z.B. eine Datenstruktur, eignet, Außerdem kann sie mit anderen Methoden integriert werden. Danach wird die Modellierung von Strukturen beschrieben.
Die bekannten Spezifikationssprachen werden so definiert, dass sie Forderungen nach Vollständigkeit, Eindeutigkeit und Verständlichkeit erfüllen.

In Kapitel 4 werden gefärbte Petrinetze und zeitbehafte Petrinetze vorgestellt. Am Anfang dieses Kapitels werden Petrinetze in Klassen klassifiziert. Dadurch werden gemeinsame Merkmale vorgestellt. Am Ende dieses Kapitels werden viele Software-Tools erläutert. Die Tools INA und Peneca Chromos werden kurz vorgestellt, um zu klären, welche Beitrag diese Software zum Ziel der Arbeit leisten kann.

Kapitel 5 stellt die entwickelte Methode vor, die im Zentrum dieser Arbeit steht. High Level Coloured Petrinetze mit strukturierten Token (HCPN-ST) sind eine formale Methode auf Basis gefärbter Petrinetze (CPN) [Jen92][Fen93] und Prädikat - Transitions- Netze (Pr/T) [Gen87][Gen,Lau81]. Dieses im Detail definierte Petrinetz und ein Algorithmus für P2P Systeme wird beispielhaft mit CPN und HCPN-ST modelliert. Im ersten Modell ist das CPN wegen des Aufwands und der Komplexität nicht geeignet, um die Ziele geschickt zu erreichen.
Kapitel 6 zeigt, wie das erweiterte gefärbte Petrinetze in gefärbte Petrinetze transformiert wird. Dabei wird der Transformationsalgorithmus detailliert vorgestellt..
Kapitel 7 zeigt die Verifikation der entwickelten Methode und Ergebnisse der Simulation und Analyse der vorgestellten Modelle. Dazu werden erweiterte Zeit-Petrinetze informal als Modell vorgestellt, weil die Zeit in den meisten Algorithmen und Protokollen eine große Rolle spielt. Im letztem Kapitel werden die Ergebnisse der Arbeit vorgestellt und eine Zusammenfassung sowie ein Ausblick gegeben.

Kapitel 2 Peer to Peer Systeme

2.1 Verteilte Systeme

In der Literatur gibt es verschiedene Definitionen für Verteilte Systeme und keine davon stimmt mit den anderen wirklich überein. Innerhalb dieser Arbeit beschäftigen wir uns mit verteilten Anwendungssystemen. Bevor ich mit meiner Problemvorstellung beginne, möchte ich einen Überblick über verteilte Systeme sowie über verwendete Begriffe und Konzepte präsentieren.

Die folgenden Definitionen sind laut Literatur generische Definitionen für verteilte Systeme:

George Coulouris definierte **verteilte Systeme** in seinen Buch „Verteilte Systeme-Konzepte und Design [Clo, Dol, Kin02]“ wie folgt: „Ein verteiltes System ist ein System, in dem sich Hardware- und Softwarekomponenten auf vernetzten Computern befinden und miteinander über den Austausch von Nachrichten kommunizieren“.Wie in Abbildung (2.1) [Ham05]dargestellt:

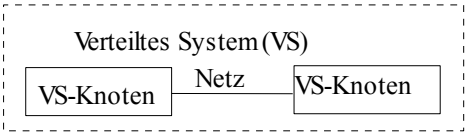


Abbildung 1: verteiltes System

Das Internet ist akzeptiert als das größte und bekannteste verteilte System. Millionen von Rechnern weltweit sind vernetzt.

Verteilte Systeme: Ein verteiltes System ist ein System mit mehreren Rechnern bzw. Prozessoren, auf denen mindestens eine verteilte Anwendung lauffähig installiert ist.

Ulrike Hammerschall [Ham05] definierte verteilte Anwendungen wie folgt: „Verteilte Anwendungen sind im Wesentlichen intelligente Anwendungsprotokolle, die auf Sender und Empfängerseite jeweils eine einfache Komponente zur Bearbeitung der Nachrichten zur Verfügung stellen. Zu den einfachsten verteilten Anwendungen zählen beispielweise Internetanwendungen wie das World Wide Web und Dateizugriffsdienste wie das File Transfer Protokoll.

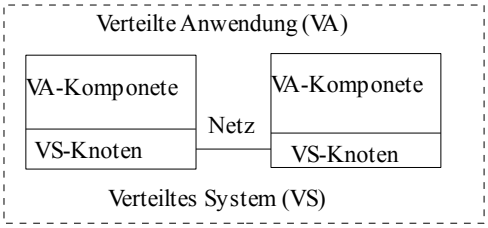


Abbildung 2 [Ham05]: Verteilte Anwendung

Eine verteilte Anwendung nutzt ein verteiltes System, um Anwendern eine in sich geschlossene fachliche Funktionalität zur Verfügung zu stellen.

Verteilte Anwendung (verteiltes Programm): Eine verteilte Anwendung ist eine Anwendung, die auf mehreren Rechnern bzw. Prozessoren abläuft und unter diesen Informationen austauscht.

„Es gibt verschiedene Varianten, verteilte Anwendung zu realisieren. Dadurch werden

verteilte Betriebssysteme, Multiprozessor-Timesharing-Systeme oder Programmsprache (wie Ada, Objektorientierte Sprache, Bibliotheken...) unterstützt.“

Kurmann [Kur04] beschreibt ein **verteiltes System** wie folgt: „Ein verteiltes System ist ein Softwaresystem, welches die Verteilung einzelner Bausteine eines Anwendungssystems auf verschiedene, räumlich getrennte Knoten eines Netzwerks (siehe Abbildung 2.2) unterstützt. Einzelne Teile des Systems können dezentral administriert werden und kommunizieren, um gemeinsam Aufgaben zu bewältigen. Ein verteiltes System unterstützt nebenläufige Verarbeitung von Geschäftsprozessen und ist sowohl für den Anwender als auch für den Entwickler transparent“.

Verteiltes System: Nach [Kie97] besteht ein verteiltes System aus mehreren Prozessen, die mittels Nachrichten miteinander kommunizieren.

So schreibt M. Stumm [Stu90] zusammenfassend: "Ein **verteiltes System** besteht aus einer Anzahl Rechner, die über ein (lokales) Netzwerk verbunden sind. Statt diese Rechner eigenständig arbeiten zu lassen, ist es von Vorteil, sie mit Software-Hilfsmitteln zu einem leistungsfähigen Gesamtsystem zusammenzufassen, das mehr als nur einfachen Austausch von Daten erlaubt. Damit wird es möglich, Ressourcen global zur Verfügung zu stellen und verteilte Applikationen zu unterstützen".

Zum Schluss verlängerte [Ens78]die Definition:

Verteiltes System = Verteilte Hardware + Verteilte Steuerung + Verteilte Daten

2.2 Anforderungen und Problemstellungen verteilter Systeme:

1. Gemeinsames Nutzen von Ressourcen (nicht nur Hardwarekompetenten, sondern Daten und Dienste): Um diesen Forderungen zu begegnen können, sollen verteilte Anwendungen nach dem Client/Server-Modell oder nach dem Peer – to -Peer- Modell gleichberechtigter Komponenten strukturiert (realisiert) sein.

2.Transparenz: Verteilte Systeme zeigen sich dem Benutzer transparent bezüglich verschiedener Gesichtspunkte, d.h. der Benutzer sieht in verschiedenen Ausprägungen nicht, dass er mit einem verteilten System arbeitet. Zugriffs- und Lokations-Transparenz sind bereits bei vielen Systemen weitgehend realisiert. Aber Replikations-, Migrations-, Nebenläufigkeits- Parallelitäts- und Fehler -Transparenzforderungen sind nur teilweise realisiert.

3.Skalierbarkeit: Ein System soll beliebig erweiterbar sein.

4.Offenheit: In einem verteilten System gibt es verschiedene Rechner mit unterschiedlichen Betriebssystemen und Software. Damit sie zusammenarbeiten, müssen die Schlüsselschnittstellen der Hardware- und Softwarekomponenten offen gelegt sein. Die Interprozesskommunikation auf Anwendungsebene muss einheitlich offen gelegt sein.

5. Flexibilität, Dynamik: Die Erweiterbarkeit muss vorhanden sein, um neue Entwicklungen, neue Dienste und neue Komponenten einfach ergänzen zu können.

6. Zuverlässigkeit: Wenn die vorherigen Punkte erfüllt sind, zeigt das verteilte System ein hohes Maß an Verfügbarkeit und Fehlertoleranz.

7. Effizienz: Das verteilte System arbeitet mit nebenläufigen Prozessen ohne gemeinsamen Adressraum.

Verteilte Systeme im Vergleich zu sequentiellen Systemen:

Wichtige Aspekte verteilter Systeme im Vergleich zu sequentiellen Systemen sind die Größe, die Komplexität, die Heterogenität, die Nebenläufigkeit, der Nichtdeterminismus und die Zustandsverteilung.

Diese Aspekte führen zu komplexerer Programmierung, aufwendigeren Tests und Verifikationen und schwierigerem Verständnis der Phänomene.

Von den Definitionen der verteilten Systeme ist bisher klar, dass die Kommunikation zwischen den Prozessen das Herz aller verteilten Systeme ist. Die Kommunikation in verteilten Systemen basiert immer auf der Nachrichtenübergabe auf unterster Ebene, die das darunterliegende Netzwerk bereitstellt.

Um heterogene Computer und Netzwerke zu unterstützen und gleichzeitig den Eindruck eines einzigen Systems zu präsentieren, werden verteilte Systeme häufig mithilfe einer Softwareschicht organisiert, die logisch zwischen einer höhere Ebene aus Benutzern und Applikationen und einer darunter liegenden Ebene aus Betriebssystemen platziert wird, wie in Abbildung (3) gezeigt. Entsprechend dieser Anordnung bezeichnet man ein solches verteiltes System manchmal auch als Middleware.[Tan, Ste03]

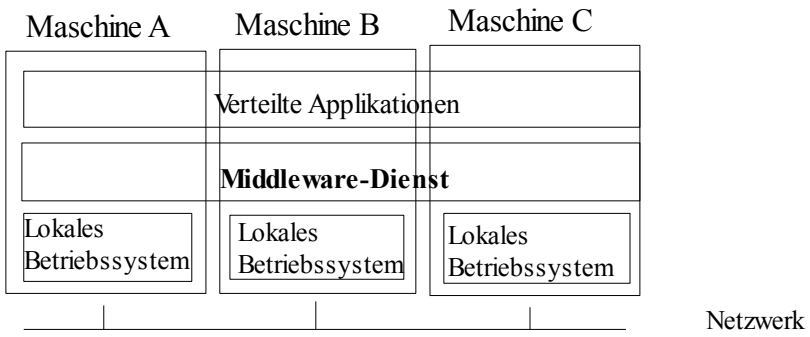


Abbildung 3: Ein verteiltes System, das als Middleware angeordnet ist. Es ist zu beachten sie, dass sich die Middleware-Schicht über mehrere Maschinen erstreckt.

2.3 Middleware und verteilte Systeme:

Wesentliche Basismechanismen verteilter Systeme sind von den verwendeten Netzwerktechnologien und weiterer Hardwarespezifika unabhängig. Diese Basismechanismen werden durch Middleware realisiert. Dabei handelt es sich um generische Softwareplattformen zur Überbrückung der Heterogenität unterschiedlicher Systeme und Netze, die gleichzeitig auch eine Reihe wichtiger Systemdienst wie etwa Sicherheitskonzepte, Transaktionsmechanismen und Verzeichnisdienste anbieten.

Definition der Middleware [Clo, Dol, Kin05]

„Middleware ist eine Softwareschicht, deren Ziel es ist, Heterogenität zu verbergen und dem Anwendungsprogrammierer ein passendes Programmiermodell zur Verfügung zu stellen. Middleware präsentiert sich durch Prozesse oder Objekte für eine Menge von Rechnern, die miteinander interagieren, um die Kommunikation und die Ressourcenverwaltung für verteilte Anwendungen zu implementieren.“

Peer to Peer Systeme stehen im Mittelpunkt unserer Betrachtungen als Sonderfall von verteilten Systemen. Die Abbildung 4 zeigt Klassifikation von Computersystemen. Die Computersysteme sind zweigeteilt. Auf einer Seite zählen zentralisierte Systeme als Erstere

(z.B.Mainframes), die aus einem leistungsfähigen Rechner bestehen, der von verschiedenen Terminals aus bedient werden kann. Aber alle Ressourcen sind nur auf dem zentralen Rechner vorhanden. Auf der andern Seite existieren Verteilte Systeme, die sich wiederum in Client-Server und Peer to Peer (P2P) Systeme unterteilen. So wie in Abbildung 4 ersichtlich, können P2P in drei Typen unterteilt werden.

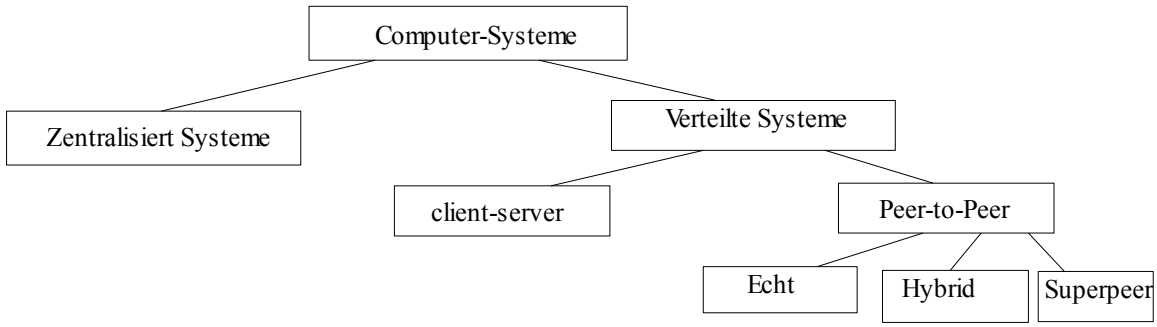


Abbildung 4: Übersicht über Computersysteme [Mil,Dej,Kal,Raj02],[Wul05]

P2P System sind den verteilten Computersystemen zuzuordnen. Der Peer to Peer Begriff wird in jedem Falle einem System zugeordnet, das aber aus einzelnen, mehr oder weniger unabhängigen Individuen (Peers) besteht, die miteinander kommunizieren können.

Die Peers werden durch Software dargestellt, die nicht notwendigerweise auf unterschiedlichen physikalischen Maschinen laufen. Jeder Peer verfügt über seine eigenen Daten, Metadaten und die Möglichkeit, mit anderen Peers kommunizieren.

Zum besseren Verständnis der Arbeit widmet sich das zweite Kapitel einer konkreten Vorstellung der P2P Systeme und wichtiger Themen auf diesem Gebiet, die mit dieser Arbeit zu tun haben. Die Algorithmen, die als Gegenstand dieser Arbeit modelliert sind, sind auf Anwendungsebenen angeordnet.

2.4 Peer to Peer Systeme

P2P Systeme sind heutzutage ein wichtiges aktuelles Forschungsgebiet und viele Projekte beschäftigen sich mit der Untersuchung dieser Technologie und der Entwicklung von Anwendungen auf P2P Basis, z.B. P2P Next[P2P], Peer-to-Peer-Internet-TV Projekt [Irt].

In dieser Arbeit wird eine Gruppe von Algorithmen analysiert und modelliert. Diese Algorithmen wurden zur Unterstützung der Suche in P2P Systemen entwickelt.

Deshalb wird das folgende Kapitel gezielt benötigte Grundlagen in Bereich der P2P Netze und Wanderer Algorithmen vermitteln. Hierbei werden die Eigenschaften und Vorteile von P2P Netzen beleuchtet. Daneben werden verschiedenen Ansätze und Strategien zur Suche in P2P Beispiel – Systemen vorgestellt.

Viele Autoren definierten P2P unterschiedlich. Die Definitionen entsprechen ihren Anwendungen und Architekturen.

Die Intel P2P working Group definierte P2P als „die gemeinsame Nutzung von Computer-Ressourcen und Diensten durch direkten Austausch zwischen den Systemen“.

P2P Netze werden definiert als „eine Art von Internet auf Anwendungsebene über dem Internet.“

„Eine wesentliche Abstraktion von P2P-Systemen ist die verwendete Overlay-Netzwerk-Topologie, die durch eingebettete Protokolle, Mechanismen und Algorithmen erzeugt wird.

Mehrere Ansätze sind als Kommunikations-Systeme vorgeschlagen worden, um eine effiziente und skalierbare Inter-Peer-Kommunikation zu ermöglichen. Diese Systeme sind auf der physikalischen Netzwerkinfrastruktur entwickelt als Overlay-Netze. Ihre Topologie und Funktionsmechanismen beeinflussen stark die Leistung von Routing- und Topologie-Wartungsalgorithmen und damit die Effizienz der beteiligten P2P-Systeme.“

In der Literatur gibt es viele Klassifikationen für Abstraktionsebenen. Diese Klassifikationen sind nicht einheitlich, da sich auf den jeweiligen Abstraktionsebenen P2P Systeme befinden.

Das P2P-Paradigma kann interessanterweise auf jeder der folgenden Ebenen unabhängig voneinander auftreten [Hau, Dus05]:

- **Netzwerkebene**: Grundlegende Routing–Dienste, die es ermöglichen, Requests in applikationsunabhängiger Weise über das physikalische Netzwerk zu routen.
- **Dienstebene**: Um höherwertige Dienst zur Verfügung zu stellen, hilft die Kombination und Erweiterung von Funktionalitäten der Datenzugriffebene. Die Bandbreite dieser Dienste kann von einfachem File-Sharing bis hin zu komplexen Geschäftsprozessen reichen.
- **Datenzugriffebene**: Suche und Änderung von Ressourcen mit Hilfe anwendungsspezifischer Zugriffsstrukturen.
- **Benutzerebene**: Gruppierung von Benutzern (Communities) und Unterstützung von Benutzer-Interaktionen unter Verwendung der Dienstebene für Community-Management und Informationsaustausch.

- Wie oben erwähnt wird, sind P2P den verteilten Systemen zuzuordnen.

Aber die Verteilung kann auf verschiedenen Ebenen existieren, bei P2P Systemen wie folgt [Wul05]:

1. Auf Hardwareebene: Das heißt, dass das P2P System aus verschiedenen eigenständigen Rechnern besteht und diese Rechner müssen miteinander verbunden sein, um Kommunikation zu ermöglichen.
2. Auf Benutzerebene: Hier ist eine andere Form von P2P Systemen vorhanden, bei dieser stellen die Anwender die eigentlichen Individuen dar.
3. Auf Anwendungsebene: Die dritte Variante von P2P System, hier werden Peers durch eine Software dargestellt. Und diese Software sollte nicht auf unterschiedlichen physikalischen Maschinen laufen. Jeder Peer bringt mit eigenen Daten und Metadaten und die Möglichkeit, mit anderen Peers zu kommunizieren. Diese Systeme benötigen ein vom Schema der Rechnernamen (DNS) unterscheidbares Adressierungsmodell. Meist verwenden P2P Systeme ein eigenes Adressierungsmodell im Gegensatz zu Client-Server Systemen, deren Adressierung wesentlich über DNS erfolgt.

Die beschriebenen Systeme und Algorithmen, die als Gegenstand in dieser Arbeit behandelt werden, sind auf Anwendungsebenen angeordnet.

Folgende charakteristische **Eigenschaften** besitzt P2P, wodurch es sich von anderen Architekturen für Verteilte Systeme unterscheidet:

- Es gibt keine zentrale Koordination, d.h., es gibt keinen zentralen Knoten, der die Interaktionen der Peers untereinander steuert oder koordiniert.
- Jeder Peer speichert einen Teil der im Gesamtsystem vorhandenen Daten und stellt diese zur Verfügung, aber kein Peer verwaltet den Gesamtdatenbestand.
- Kein Peer hat eine globale Sicht auf das System. Jeder Peer kennt also nur seine

Nachbarschaft, das sind die Peers, mit denen er interagiert.

- Das globale Verhalten des Systems entsteht aus der Kombination der lokalen Interaktionen zwischen den Peers.
- Peers sind autonom.
- Alle gespeicherte Daten im System müssen verfügbar sein, also trotz verteilter Speicherung müssen bei möglichen Verbindungsausfällen, unbekannten Glaubwürdigkeiten der Peers etc. die gesamten Daten jederzeit und von jedem Peer aus zugreifbar sein.
- Peers und Verbindungen sind nicht zwingend verlässlich. Da man nicht die Glaubwürdigkeit jedes einzelnen Peers kennt und Peers auch offline sein können, muss das System Mechanismen zur Verfügung stellen, um dies zu kompensieren.

2.4.1 Anwendungsgebiete von P2P Architekturen:

Wesentliche Anwendungsgebiet für P2P Architekturen sind parallelisierbare (verteilte) Anwendungen, eine verteilte Daten- und Inhaltsverwaltung sowie kollaborative Systeme.

- Parallelisierung, um berchnungsintensive Operationen in Teiloperationen zu zerlegen. Diese Operationen werden anschließend parallel auf einer Menge von Peers ausgeführt.
- Die Verwaltung von Daten und multimedialen Inhalten mit dem Ziel einer gemeinsamen Nutzung von Ressourcen in einem P2P-Netze.
- P2P Systeme ermöglichen Kollaboration ohne zentralen Server
- P2P können im Internet, in Intranets sowie in Ad-hoc-Netzwerken eingesetzt werden.

Ebene	Applikation-Domäne	Dienst	Beispielsystem
Netzwerkebene	Internet	Routing	TCP/DNC
Datenzugriffsebene	Overlay Netzwerke	Ressourcenlokation	Gnutella, Freenet
Dienstebene	P2P Applikationen	Dienst, verteilte Bearbeitung	Seti@home, Napster, Groove
Benutzerebene	Benutzercommunities	Kollaboration	eBay, Ciao

Tabelle 1: [Hau, Dus05] Anwendung des P2P-Paradigmas auf unterschiedlichen Systemebenen.

Alle Peer to Peer verwenden das Internet. Das Internet ist eher ein loser Zusammenschluss verschiedenartiger Rechnernetzwerke, im Gegensatz zu einer zentralen Kommunikationsstruktur mit homogener Hard- und Software [Mah, Sch07].
Aktuelle Anwendungsbereiche von P2P Systeme sind Instant Messaging, Internettelefonie, gemeinsamer Datenzugriff (Filesharing), kooperatives Arbeiten und verteiltes Rechnen. Ein wichtiger Aspekt dieser Systeme ist die Breitstellung dezentraler Ressourcen, bei denen es sich um Dienste, Datenspeicher, Kommunikationsverbindungen oder auch Nutzer handeln kann.

2.4.2 Typen von Peer to Peer Architekturen

P2P können, wie Abbildung 5, in drei grundlegende Typen unterteilt werden:
1. **Echte P2P**: In echten P2P Architekturen existiert kein zentralisierter Server. Sowohl die Suche nach dienstanbietenden Peers als auch die Dienstnutzung zwischen Peers erfolgt ohne Verwendung eines Servers. Die Peers kommunizieren direkt miteinander und nutzen dabei

gegenseitig Dienste bzw. stellen Dienste zur Verfügung. Siehe Abbildung (2-1) [Sch, Spr07]. Dieser Ansatz arbeitet jedoch sehr effizient. Ein Beispielsystem ist der Filesharing-Dienst Gnutella.

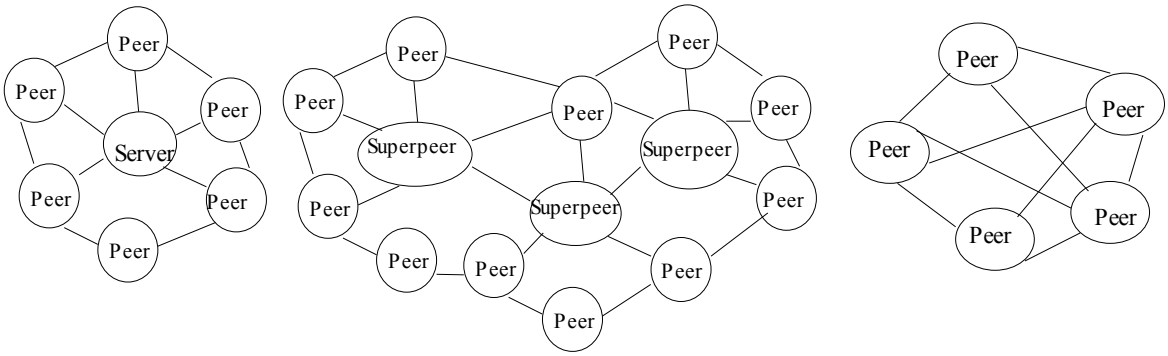


Abbildung 5: Typen von P2P-Architekturen: hybrid (links), Superpeer (Mitte), echt (rechts)

2. hybrid P2P: In der hybriden P2P Architektur gibt es ein zentrales Verzeichnis, bei dem sich dienstanbietende Peers registrieren können.
Der Ansatz erfordert eine zentrale Infrastruktur, die Information über alle teilnehmenden Peers enthält. Ein Beispielsystem ist der Filesharing-Dienst Napster.
3. P2P mit Superpeer: Eine Mischform den anderen beiden Typen besitzen P2P Architekturen mit Superpeers. Neben den Peers existieren im System so genannte Superpeers, die Informationen bzw. Dienste erhalten, die nicht von alle Peers angeboten werden.
Weil mehrere Superpeers im System existieren können, stellt ein Superpeer keinen zentralen Server dar. Als Beispielsystem ist der Filesharing-Dienst Ka Zaazu nennen.
Die Grenze zwischen P2P Systemen und Client-Server Systemenist also sehr unscharf. Deshalb sind die typischen Merkmale beider Systeme hilfreich besonders die oben genannten Eigenschaften von P2P Systeme.

2.4.3 P2P-Netze Struktur

In Abbildung (2-6) wird eine Übersicht über die Arten von P2P-Netzen hinsichtlich ihrer Struktur gegeben. Die P2P Netzen können in zwei Arten unterteilt werden.

Unstrukturierte P2P-Systeme

Alle derzeitigen unstrukturierten P2P-Systeme sind flach und lose gekoppelt. Sie haben den Vorteil, dass sie einen geringen Aufwand in die Organisation investieren und dadurch sehr gut mit den sehr dynamischen Netzen zurechtkommen.
Der typische Repräsentant dieser Klasse ist Gnutella [Cli05]. Das Protokoll von Gnutella besteht aus fünf Nachrichtentypen für Netzwerkverwaltung (Ping, Pong) und Datenmanagement (Query, QueryHit, Push). Das Routing von Nachrichten in Gnutella erfolgt durch einen einfachen „Constraint Broadcast“ Ansatz: Nach Erhalt einer Nachricht wird ihr Time-To-Live (TTL) Zähler verringert. Ist die TTL größer als 0 und wurde die Nachricht noch nicht empfangen (Vermeidung von Schleifen durch in die Pakete eingebettete Identifikatoren), wird die Nachricht an alle Peers in der Nachbarschaft, die mit dem Peer verbunden sind, weitergeleitet. Üblicherweise hat jeder Peer vier aktive Verbindungen zu anderen Peers, d.h., jedes empfangene Paket, wird an drei Peers weitergeleitet. Gleichzeitig überprüft jeder Empfänger, ob lokale Suchergebnisse vorhanden sind. Informationen über Suchergebnisse (QueryHit) werden entlang des Netzwerkpfades, auf dem eine Anfrage (Query) empfangen wurde, zurückgegeben.

Um an einem Gnutella-Netzwerk teilnehmen zu können, muss ein Peer einen Gnutella-Peer kennen. An diesen Peer wird eine Ping-Nachricht geschickt, die wie eine Suchanfrage weitergeleitet wird und von anderen Peers mit einer Pong-Nachricht beantwortet wird, welche ihre Adresse und einige Systemparameter enthalten. Aus den empfangenen Pong-Nachrichten wählt der Peer maximal 4 aus und stellt zu diesen Verbindungen her (unter der Voraussetzung, dass diese noch weniger als 4 aktive Verbindungen haben). Der Transfer von Dateien (basierend auf den via QueryHit gelieferten Informationen) erfolgt über ein vereinfachtes HTTP-Protokoll direkt zwischen zwei Peers. Push-Nachrichten werden verwendet, wenn ein Peer sich hinter einer Firewall befindet und der Transfer daher als „Callback“ initiiert werden muss.
Der Ansatz von Gnutella ist zwar einfach, aber effizient: Suchergebnisse werden schnell zurückgeliefert, und das Netzwerk ist sehr fehlertolerant. Sein Nachteil ist jedoch der hohe Bandbreitenverbrauch. Beispielsweise verursacht ein einziges Ping bei 4 Verbindungen (C) pro Peer und einer typischen TTL von 7, $2\times\sum_{(i=1)}^{(TTl)}(c\times(c-1)^{i})=26240$ Nachrichten.
Neuere Ansätze verringern diesen hohen Bandbreitenverbrauch [Lv, Cao, Coh, Li,She02]. Dazu zählen Expanding Ring Search oder Random Walker.

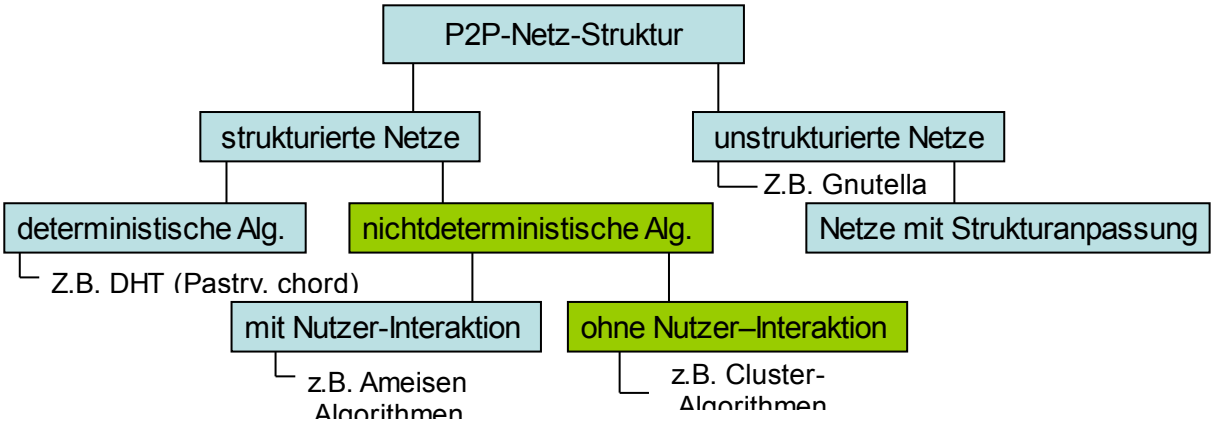


Abbildung 6: Kategorien für P2P Netze und ihr Algorithmen[Wul05]

Strukturierte P2P-Systeme

Die Merkmale der strukturierten P2P-Systeme sind:

- Auf alle Teilnehmer wird ein aufgeteilter verteilter Index konstruiert, der Engpässe und Asymmetrien verhindern kann.
- Komplexere Wartungsaufgaben, da nicht nur der Index, sondern auch die zur Verwaltung des Index aufgebaute verteilte Infrastruktur gewartet werden muss.
- Generell besitzen sie zwei Teile als globale Zugriffsstruktur, einen Teil des Gesamtindex und Routing-Tabellen.
- Suchoperationen werden durch Weiterleitung mittels der jeweiligen Suchstrategien aus der Routing-Tabelle an ausgewählte Peers durchgeführt.
- Die Routing-Tabellen werden dabei so aufgebaut, dass ihre Größe nur sehr langsam

anwächst und die Suche schnell durchgeführt werden kann.Damit skalieren diese Systeme sehr gut.

- Komplexität der Suche und Größe der Routing-Tabellen wachsen logarithmisch zur Anzahl der Peers im System.
- Der Aufbau und die Verwaltung von Routing-Tabellen erfolgt üblicherweise nach einer der folgenden Strategien: Ad-hoc Caching and Clustering entlang der Suchpfade, Distributed Hash Tables und Topologisches Routing.

[Hau, Dus05] Daneben bestehen die Hauptunterschiede zwischen den Systemen in Bezug auf ihre funktionale Flexibilität bei Replikation, Lastverteilung, unterstützten Suchprädikaten und Eigenschaften der Netzwerkverwaltungsprotokolle. Lose gekoppelte strukturierte P2P-Systeme wie Freenet und P-Grid haben mit unstrukturierten Systemen die flexible Netzwerk-Evolution gemeinsam. Eng gekoppelte Systeme wie CAN, Chord, Tapestry und Pastry hingegen benutzen starre Verwaltungsstrategien, um dadurch kontrollierbarer Durchsatz zu erhalten.

2.4.4 Vorteile von P2P Systemen

P2P Systeme werden in kurzer Zeit bekannt und sehr häufig angewendet. Das ist auf ihre vielen Vorteile zurückzuführen, die sie gegenüber den traditionellen Client- Server Systemen haben:

1. Geringen Kosten: In P2P Systemen werden die Kosten für den Aufbau und Betrieb des Gesamtsystems auf alle Peers aufgeteilt. Kommt ein Peer dazu, bringt er Rechenleistung, Netzbandbreite und Speicherkapazität mit in das System ein. Dies wird bei einem völlig dezentralen Ansatz, wie in Gnutella, SETI @home, realisiert.
2. Zuverlässigkeit und Stabilität des Systems steigen durch jede neue Ressourcen, die jeder Teilnehmer mitbringt. Dazu bringt jeder neuer Teilnehmer seine Leistung ein, im Gegenteil zu zentralen System, wo vorhandene Knoten mehr belastet werden.
3. Höhere Sicherheit gegen Angriffe auf das System durch die Verteilung, da eine Funktionalitätsstörung in einem Peer das Gesamtsystem nicht gefährdet.
4. Die Anwender in P2P System sind weitgehend autonom. Sie speichern benötige Daten lokal und können den anderen die Daten vermitteln. Bei Bedarf werden die Daten schnell lokal aktualisiert.

Nicht alle P2P Systeme besitzen alle diese Vorteile und in Client Server können diese nicht vorhanden sein.

Trotz aller Vorteile werden P2P Systeme vor allem durch das Suchproblem beeinträchtigt.

2.4.5 Wie werden die Strategien für die Suche in P2P-Systemen realisiert?

- Durch den Aufbau von Strukturen im Netz wird die Suche in P2P Netze unterstützt.
- Strukturierte P2P Netze besitzen eine Struktur, die durch die eingesetzten Protokolle erstellt wird. Sie dient dazu, das Netz zu organisieren, sodass bestimmte Operationen (wie z.B. die Suche) unterstützt werden.
- Zur Strukturbildung von P2P Netze gibt es zwei Arten von Ansätzen:
  - Hierarchische Struktur: Das Netz von leistungsfähigen Peers sind untereinander oder mit sogenannten Superpeer verbunden [Kaz05].

- Strukturbildungsalgorithmen: Die Verbindung der Peers ist dynamisch veränderbar (z.B. Peers mit ähnlichem Inhalt werdenstärker verbunden), also die Struktur an die von dem Peer angebotenen Inhalte anzupassen.

Allgemeine Strategie zur Suche in P2P Systeme

Die Suche nach Inhalten in dezentralen, verteilten Systemen stellt immer noch ein großes Problem dar. Zur Lösung dieses Problem werden verschieden Ansätze vorgestellt. Im folgenden wird ein Überblick über allgemeine Strategien zur Suche in P2P Systemen gegeben.

- Suche mit zentralem Verzeichnis: Dieses Modell wurde durch [NaP00] bekannt. Es gibt ein zentrales Verzeichnis, in dem alle in der P2P–Community verfügbaren Ressourcen aufgeführt sind. Die Suchanfragen eines Peers werden an das Zentrale Verzeichnis gerichtet. Dann wird die Adresse des Peers ermittelt, der die Anfrage am besten beantworten kann. Danach kontaktiert der anfragende Peer diese Adresse und der Datenaustausch erfolgt direkt von Peer zu Peer, nicht über den zentralen Server. Das Zentrale Verzeichnis macht diesen Ansatz zu einer Mischform aus Client-Server und P2P Anwendung.
- Suche mit Broadcast: Ein reiner P2P-Ansatz ist Broadcast. Hier wird das Netz mit den Anfragen geflutet. Ein Peer sendet also seine Frage an seine Nachbarn, die diese Anfrage nun wiederum an ihre Nachbarn weiterleiten usw. Aber die Anzahl an Weiterleitungen ist beschränkt. Dieses Modell wird beispielsweise vom Gnutella-Protokoll [Gnu05] verwendet. Der Vorteil ist, es existiert kein zentrales Verzeichnis und der Algorithmus ist relativ robust und leicht zu implementieren. Als Nachteil kann die hohe Netzlast angesehen werden und dazu ist nicht sicher, dass vorhandene Ressourcen gefunden werden.
- Suche mit inhaltbasiertem Routing: Bei diesem Verfahren hat jeder Peer eine ID. Wenn ein neues Dokument in der Community veröffentlicht werden soll, wird ihm ebenfalls eine ID zugewiesen. Diese wird als Hash aus dem Dokumentinhalt oder Dokumentnamen erzeugt. Dann wird das Dokument zu dem Peer geroutet, der eine ID besitzt, die der ID des Dokumentes am ähnlichsten ist und wird auf diesem Knoten gespeichert. Dieses Modell ist sehr gut für große Netze, aber gleichzeitig ist die Implementierung aufwendig und es muss die ID des Dokumentes bei der Suche bekannt sein.

Als **Qualitätsparameter für die Suche** können folgende Kriterien verwendet werden [Wul05]:

- Die Zeit, die benötigt wird, bis eine positive Antwort auf eine Anfrage empfangen wird, sollte möglichst kurz sein.
- Möglichst alle im System verfügbaren Informationen und Dienste, die dem Anfrageprofil entsprechen, sollten gefunden werden.
- Der zusätzliche Aufwand an Rechenzeit und Netzverkehr, der für die Suche notwendig ist, sollte möglichst gering sein.

Dann muss je nach Anwendungsfall entschieden werden, welcher der Punkte für die Arbeit mit dem entsprechenden System wichtiger ist.

2.4.6 Unterstützung der Suche in P2P Netze

Durch den Aufbau von Strukturen im Netz wird die Suche in P2P Netze unterstützt. Strukturierte P2P Netze besitzen eine Struktur, die durch die eingesetzten Protokolle gebaut wird. Sie dient dazu, das Netz zu organisieren, damit bestimmte Operationen wie z.B. die

Suche, unterstützt werden.

Anforderung an Strukturbildungsalgorithmen

Bei Entwicklung von Algorithmen zur Suche in dezentralen P2P Systemen müssen folgende wichtige Kriterien berücksichtigt werden[Wul05]:

- Nur lokal verfügbare Informationen über das System können verwendet werden.
- Die eingesetzten Algorithmen werden auf jedem Peer gleichermaßen ausgeführt.
- Die Algorithmen müssen unabhängig von dem bisherigen Aussehen der Struktur des Netzes jederzeit auf einem Peer gestartet werden können.
- Die Struktur muss sich nach dem Ausfall eines Peers selbstständig reparieren können.
- Neu hinzukommende Peers müssen schnell in eine bestehende Struktur aufgenommen werden können, damit eine gute Skalierbarkeit gewährleistet ist.
- Das System muss auch dann funktionieren,wenn nicht alle Peers verfügbar sind bzw. Sie nicht den gegebenen Algorithmus ausführen.

Die hier besprochenen P2P-Systeme werden auf der Anwendungsschicht aufgebaut (die darunterliegenden Netzprotokolle werden nicht betrachtet). Die Verbindungen zwischen den Peers sind logische Verbindungen und können auf der Anwendungsebene angepasst werden. Die physikalischen Verbindungen sind vorgegeben und nicht veränderbar. Darüber das Internet (OSI-Schichten 1-4 [Day, Zim83]) liegt das P2P-Netz, das auf Anwendungsebene gebaut wird. Die Verbindungen werden durch Adressen von Nachbarn erzeugt, die jeder Peer lokal speichert. Durch die Einführung einer weiteren Ebene kann eine logische Struktur auf P2P Netze bauen. P2P Systeme bestehen also aus Internet und logischen P2P Netzen.
Der P2P-Systemen zugrunde liegende Denkansatz beruht darauf, dass auch bei erhöhtem Bandbreitenverbrauch aufgrund der Server-Last gleichmäßig arbeitende Client-Server–Systeme möglich sind. Im Gegensatz dazu bezahlt man dafür allerdings mit höherer Komplexität der verwendeten Algorithmen und zusätzlichem Sicherheitsaufwand. Um Komplexität und fehlende Mechanismen zur Reparatur von Defekten, die durch ausfallende Knoten verursacht werden, in einigen Ansätzen zu beseitigen, wird immer über neue Algorithmen nachgedacht. Die Algorithmen, die im folgenden vorgestellt werden und bessere Strukturen bauen können, unterscheiden sich von den bisher existierenden Algorithmen. Diese Algorithmen sind nicht deterministisch (siehe Abbildung 2-2). Mit stochastischen Komponenten und mit Hilfe von Wanderern können sie eine Struktur bauen, die die Suche in P2P Netzen unterstützen kann. Vor der Vorstellung der Algorithmen sollen hier verwendete Begriffe definiert werden.

Definitionen:

1. Ein Algorithmus heißt verteilt, wenn er auf einer physikalisch oder logisch verteilten Architektur arbeitet. Solche Algorithmen werden praktisch zunehmend wichtiger.
2. Cluster: Die Gruppe der Knoten, die von ein und demselben Wanderer besucht werden.
3. Wanderer (Random Walker) [Wul05]: Hier sind Nachrichten, die sich nach vorgegebenen Regeln im Netz bewegen und Aufgaben der Strukturbildung übernehmen. Sie transportieren Daten, keinen Programmcode. Sie verhalten sich passiv und werden von den auf jedem Peer ablaufenden Programmen für verschiedene Aufgaben genutzt, abhängig vom jeweilsdem verwendeten Algorithmus.

Aufgaben der Algorithmen:

- Aufbauen von globaler Struktur nur unter Verwendung lokaler Information
- Der Algorithmus kann in kurzer Zeit Verletzung der Struktur (sich ändernde Netztopologie) reparieren
- Kommunikation zwischen Peers
- Suchanfragen
- Administrative Aufgaben

2.4.7 Strukturbildungs-Algorithmen ohne Berücksichtigung von Nutzeraktivitäten – Cluster - Algorithmen

- Diese Gruppe von Algorithmen zur Bildung von Clustern und gleichzeitig zur Kommunikation der Peers basierte auf der Nutzung von Wanderern [Wul05], [Ung, Wul02], die im folgenden erläutert werden.
- Die Zeit spielt die größte Rolle bei der Bestimmung der Größe der Cluster.

1. Nicht disjunkte Cluster:

- Einige Knoten sind in mehr als einem Cluster erhalten, das bedeutet, dass sie von mehr als einem Wanderer besucht werden. Das ist notwendig, um die Informationen in gesamten Netz auszutauschen und zu verteilen.
- Die **Wanderer** werden auf den betroffenen Knoten bearbeitet.
- Die **Wanderer** enthalten neben den Metadaten einen Plan, in dem die Adressen der zu besuchenden Knoten gespeichert sind.
- Der **Plan** enthält anfangs nur die Adresse seines Erzeugerknotens und wird zufällig an ausgewählte Nachbarn gesendet und die bleibenden Einträge sind leer. Dazu speichert ein Wanderer folgende Daten:
  - $\mathbf{T}_{\mathbf{cycle}}$: die durchschnittlich benötige Zeit für einen Zyklus, berechnet aus den letzten r Zyklen.
  - **P**: die aktuelle Planposition.
  - **q**: die Länge des Plans (konstante Werte).
- Jeder Knoten kann sich an einem oder mehreren freien Plätzen in Plan eintragen, wenn er das gewünschte Besuchsintervall eintragen darf. Dies wird in Kapitel 5 erläutert.

2. Disjunkte Cluster:

- Jeder Knoten kann nur von einem einzigen Wanderer besucht werden. Dies verhindert die Kommunikation zwischen den Clustern.
- Aber der Austausch von Nachrichten wird durch eine hierarchische Struktur realisiert. Das heißt, dass es Wanderer gibt, die die Aufgabe haben, nicht Knoten sondern Cluster miteinander zu verbinden. Dazu werden verschiedene Ebenen eingeführt.
- Die Wanderer, die für die Knoten verantwortlich sind, haben die Ebene 0. Wenn der Plan eines solchen Wanderers über eine bestimmte Zeit stabil ist, wird ein Knoten in diesem Cluster bestimmt, der einen Wanderer der Ebene 1 erzeugt.

- Wanderer der Ebene 1 können nur Knoten der stabilisierten Wanderer Ebene 0 aufnehmen. Die Kommunikation zwischen den Clustern der Ebene 1 wird ebenso durch Wanderer der Ebene 2 realisiert usw. Dadurch entsteht eine hierarchische Struktur, die es ermöglicht, Nachrichten schnell in dem gesamten P2P Netz zu verteilen.
- Die Knoten, die von verschiedenen Ebenen besucht werden, übernehmen den Informationsaustausch zwischen den Ebenen.

3. Clusterbildung durch Okkupation der Nachbarschaft

- Dieser Algorithmus zur Bildung logischer Cluster in einem P2P-Netz unterscheidet sich von den beiden vorhergehenden.
  - Die Grundidee ist der Versuch, einen Cluster zu erzeugen, der möglichst viele Knoten aus der Nachbarschaft in den eignen Cluster einbindet.
  - Jeder Cluster hat einen Hauptknoten, der die Veränderungen des Cluster kontrolliert und Informationen über die aktuelle Größe des Clusters besitzt.
  - Aber die Größe des Clusters wird durch die Kommunikationszeit t_c beschränkt. Jeder Knoten x hat eine vorgegebene Zeit t_{avg}, um mit den anderen Knoten des Clusters zu kommunizieren und soll diese Zeit nicht überschreiten.
  - Es gibt auch eine maximale Zahl d von Knoten, die zu einem Cluster gehören können.
- **Vier Klassen von Clustern** werden während des Aufbaus eines Clusters definiert.
  - **Class0**: Alle Knoten, die selbst mit dem Aufbau einem Cluster nicht beginnen.
  - **Class1** : Cluster dieser Klassen haben die Größe s erreicht,wobei $1 \le s \le n+1$, wobei n die Anzahl der Nachbarn des Hauptknotens ist.
  - **Class2:** Cluster dieser Klassen haben die Größe s erreicht,wobei $n+2 \le s \le d$ gilt, Ihre Größe liegt im Zielbereich.
  - **Class3:** Cluster dieser Klassen sind zu groß $d \le s$. Sie bilden sich, wenn zwei kleinere Cluster verschmelzen. Aber sie sollten so schnell wie möglich ihre Größe reduzieren.
- **Zusätzlich werden folgende Werte auf jedem Knoten gespeichert:**
  - ID(x): Eine eindeutige Identifikation des Clusters. Knoten beziehen sie von den Hauptknoten.
  - Size(x): Die aktuell Größe des Clusters
  - Class(x): Die Klasse, der dieser Cluster zur Zeit gehört. Diese wird vom Hauptknoten berechnet.

2.4.8 P2P Netzwerke in der Praxis

Anwendungen von Peer to Peer Netzen: File-Sharing, World-Wide-Web, Internet Telefonie, E-Mail: E-POST wird ein P2P E-Mail Dienst gestellt[Mis, Hae, Pos, Dru05], Grid-Computing für mehr Informationen [Mah, Sch07].

Folgende File-Sharing Protokolle sind momentan bekannt:

- Das eDonkey-Netzwerk besteht aus einer Client-Server Struktur und arbeitet ähnlich wie Napster. Die Client-Server Software erlaubt nun, mehrere Server in einer Server-Liste aufzunehmen und neben echten Servern gibt es sogenannte Fake-Server.
- Fasttrack: folgt einer hybriden P2P Netzwerk-Struktur. Die Peers werden in zwei Klassen

unterteilt: normaler Peer und Superpeer. Offizielle Clientsoftware von FastTrak ist Kazaa.

- Gnutella (1 und 2): Gnutella 1 ist reines P2P, aber Gnutella 2 hingegen ein hybride P2P aus Peer und Superpeer.
- Overnet und Kademlia: Echtes P2P, als Routing Algorithmus verwendet Overnet das Kademlia. In Kademlia erhalten Peers durch Anwenden einer Hash-Funktion auf die IP-Adresse eine ID.
- Skype: Skype bietet Grundfunktionalitäten: Die direkte Voice over IP (VoIP) Verbindung zwischen zwei Rechnern ohne Zwischenknoten. Wie die Netzwerkstruktur funktioniert ist nicht bekannt.
- Es gibt noch weitere Protokolle wie KaZaA, Morphius, Napster, Soulseek, WinMX, Apple.Juice und BitTorrent und diese Liste wird ständig erweitert.

	Unstrukturierte P2P Systeme			Strukturierte P2P System arbeiten nach dem Prinzip Verteilte Hash-Tabellen Komplexität der Suche hängt von verwendeten Routing - Mechanismen and Netzverbindungsstruktur ab			
	Web Operating System echtes P2P	Gnutella echtes P2P	Freenet echtes P2P	Pastry infarstruktur für große verteilte P2P	Tapstry	chord	CAN echtes P2P
Suche	Durch Nachrichten-Ketten	Query-Hit Nachrichten mit TTL	Jede Suche hat Schlüssel Routing Tabellen verwendet	Mit ID in Routing Tabelle gespeichert	Mit ID in Routing Tabelle gespeichert	Basiert auf verteilten Hash-Tabellen	Basiert auf verteilten Hash-Tabellen
Nachteile Suche	Netzlast keine Garantie für Erfolg einer Suche	Netzlast keine Garantie für Erfolg einer Suche	Netzlast	Latenzzeit noch nicht effizient	Nicht vollständig selbstorga-nisierend	Zeit der Suche noch nicht effizient	Fehlen von Mechanismen zur Reparatur von Defekten
Vorteile Suche	Anonymität	Sehr stabil und selbst organisiert robust	Anonymität	Sehr robust Latenzzeit verringert Fehlertoleranz, Skalierbarkeit	Schnell Suche und zuverlässig	Garantie des Erfolgs einer Suche nach vorhanden Datei	Garantie des Erfolgs einer Suche nach vorhanden Datei

Tabelle 2: P2P Netze und Suche Problem mit bekanntesten Beispiele

Kapitel 3 Modellierung

3.1 Grundlagen

Die Modellierung und Modellbildung spielt eine große Rolle in unterschiedlichen Forschungsbereichen und Phasen des Entwicklungsprozesses. Jeder Bereich hat dabei spezielle Techniken der Modellbildung, die sich für seine Aufgaben besonders gut eignen. Ein Beispiel sind Gebäudemodelle in dem Bereich Architektur, Bauzeichnungen und Berechnungen der Statik von Gebäuden oder Schaltpläne für elektrische Anlagen. Dieses Kapitel stellt einige Modellierungsmethoden vor, die mit dieser Arbeit im Zusammenhang stehen. Sie werden im Kontext ihrer Anwendungen klassifiziert und präsentiert. Zuerst wird die Modellierung mit Wertebereichen behandelt, die als mathematische Grundlage für formale Methoden benötigt wird. Die meisten dieser Definitionen sind meisten zum späteren Verständnis erforderlich. Die Modellierung mit Graphen in der bisher bekannten Form ist eine gute und einfache Methode, die allein oder zusammen mit anderen Methoden benutzt werden kann. Danach folgt die Modellierung von Strukturen, die einen Teil des Systems oder die ganze Struktur dieses System beschreiben. Schließlich wird die Modellierung von Abläufen behandelt. Endliche Automaten, UML-Verhaltensmodelle und Petrinetze werden kurz vorgestellt. Diese Beschreibungsmittel sind für die behandelte Thematik geeignet, sie besitzen aber unterschiedliche Mächtigkeiten. Deshalb werden Petrinetze als geeignete Modellierungsmethode ausgewählt. Die Gründe dafür wurden im ersten Kapitel erläutert. Außerdem werden im nächsten Kapitel die wichtigsten Petrinetze vorgestellt, die mit dieser Arbeit im Zusammenhang stehen.

In der Informatik werden überwiegend abstrakte Modelle basierend auf Kalkülen und formalen Sprachen verwendet. Informatiker müssen diese als ihr Handwerkszeug beherrschen, um für ihre Aufgaben nützliche Modelle mit geeigneten Kalkülen und formalen Sprachen anzufertigen.

3.1.1 Modellierung mit Wertebereichen

Die folgenden Definitionen basieren auf [Kas,Kle 08]:

Definition 3.1: Wertebereich

Ein Wertebereich W ist eine Menge von Werten, die im Sinne eines Modells als gleichartig angesehen werden, wo ein Wert eines Wertbereiches gefordert wird, kann prinzipiell jedes Element aus W diese Rolle übernehmen.

Definition 3.2: Menge

Eine Menge M ist eine Zusammenfassung von verschiedenen Objekten, den Elementen der Menge. Mengen können auf zwei prinzipielle Weisen angegeben werden.

- extensional: durch Aufzählung ihrer Elemente, wie z.B. {1,4, 9,16,25}.
- intensional: durch Angabe einer Bedingung, alle Werte, die diese erfüllen, sind Elemente der Menge, wie z.B. $\{a|a \in N, a \text{ ist Quadratzahl und } a < 30\}$.

Wenn wir Mengen als Wertebereiche definieren, geben wir diesen meist einen Namen. Im Kontext der Modellierung sollen möglichst ausdruckskräftige Namen verwendet werden. Alle in der Mathematik definierten Mengenoperationen können angewendet werden, um

Wertebereiche zu verknüpfen.

Definition 3.3: Potenzmenge

Die Potenzmenge einer Grundmenge U ist die Menge aller Teilmengen von U, geschrieben Pow(U) oder (U).

Als Formel: Pow (U) := { M|M $\subseteq$ U}, wie z.B. U={a,b}, Pow(U) := {Ø, {a}, {b}, {a,b}}.

Potenzmengen kommen auch bei folgender Modellierungstechnik zum Einsatz: Manche Aufgaben haben nicht immer genau eine Lösung, sondern mehrere oder keine Lösung, dann kann man die Potenzmenge benutzen, um die Lösung darzustellen.

Definition 3.4: Geordnetes Paar, Kartesische Produkte

Ein geordnetes Paar (x, y) besteht aus zwei Werten x und y, wobei x die erste und y die zweite Komponente ist. Das kartesische Produkt M×N zweier Mengen M und N ist die Menge aller der geordneten Paare mit erster Komponente aus M und zweiter Komponente aus N.

M×N:={ (x, y)| x $\in$ M und y$\in$ N}. Das wird verallgemeinert zum kartesischen Produkt von n >1 Mengen, als Menge von geordneten n-Tupeln, beispielsweise

$M_1 \times M_2 \times \quad \times M_n := \{ (a_1, a_2, \ldots., a_n) \mid a_i \in M_i \text{ und } i \in I \}$, I ={1,2,3,….n}.

Wenn $M_1 = M_2 = M_n$ ist, kann $M_1 \times M_2 \times \quad \times M_n := M^n$ geschrieben werden.

Definition 3.5: Vereinigung

Seien $W_1, \ldots W_n$ beliebige Mengen und n>1. Dann ist $V= W_1 \cup \ldots\ldots \cup W_n = \cup^n_{i=1} W_i$ der vereinigte Wertebereich.

Die Vereinigung von Wertebereichen drückt die Zusammenfassung von spezielleren Wertebereichen aus. Beispiel: Peer := {n1, n2, n3}, Superpeer := {S1, S2}, P2P Netz mit Superpeer = Peer $\cup$ Superpeer = {n1, n2, n3, S1, S2}

Definition 3.6: Wertebereich von Relationen

Eine n-stellige Relation R ist eine Menge von n-Tupeln, wobei jedes davon aus einem Wertebereich $M_1 \times M_2 \times \ldots \times M_n$ mit n >1 stammt, d.h. $R \subseteq M_1 \times M_2 \times \ldots \times M_n$. Solch eine Relation stammt aus dem Wertebereich Pow $(M_1 \times M_2 \times \ldots\ldots \times M_n)$, der i.a weitere Relationen gleicher Struktur enthält.

Die Kardinalität des Wertebereiches ist $|\text{Pow}(M_1 \times M_2 \times \ldots\ldots \times M_n)| = 2^k$ mit $k = \prod^n_{i=1} |M_i|$ falls alle M_i endlich sind. Die Eigenschaften, die von solchen Relationen definiert werden können, sind reflexiv, symmetrisch, anti-symmetrisch, asymmetrisch, transitiv und alternativ.

Definition 3.7: Funktionen

Eine Funktion f ist eine 2-stellige Relation f$\in$ Pow(D×B), für die gilt:

Aus (x, y)$\in$ f und (x, z)$\in$ f folgt y = z. Einem Wert aus D ist also höchstens ein Wert aus B zugeordnet. Die Menge D heißt Definitionsbereich und die Menge B Bildbereich

der Funktion f.

Eigenschaften von Funktionen:

Eine Funktion $f \in D \rightarrow B$ ist:

total, wenn es für jedes $x \in D$ ein Paar $(x, y) \in f$ gibt;

surjektiv, wenn es zu jedem $y \in B$ ein Paar $(x, y) \in f$;

injektiv, wenn es zu jedem $y \in B$ höchstens ein Paar $(x, y) \in f$ gibt;

bijektiv, wenn f zugleich surjektiv und injektiv ist.

Definition 3.8: Stelligkeit von Funktionen

Funktionen aus dem Wertebereich $D \rightarrow B$ sind **n-stellig,** wenn der Definitionsbereich D eine Menge von n-Tupeln ist. Wenn D nicht als kartesische Produkt strukturiert und D nicht die leere Menge ist, sind die Funktionen aus $D \rightarrow B$ 1-stellig und sie sind 0-stellige Funktionen oder Konstanten Funktionen, wenn D leer ist.

Definition 3.9: Terme

Eine Schreibweise für Formeln, mit denen man Eigenschaften und Zusammenhänge in Kalkülen ausdrücken kann. Sie bestehen aus Operatoren, die Operanden verknüpfen. Die Operanden sind wiederum Formeln oder elementare Konstante oder Variable.

Definition 3.9.1: Sorte

Eine Menge von Termen τ kann in disjunkte Teilmengen $S_1, \ldots\ldots, S_n$ eingeteilt werden, um die strukturelle Korrektheit der Terme zu definieren. Die S_i heißen dann Sorten der Terme in τ.

Z.B. $<$: ARITH$\times$ ARITH$\rightarrow$BOOL, dies bedeutet der ($<$) Operator bildet einen Term der Sorte BOOL und verknüpft zwei Terme der Sorte ARITH.

Definition 3.9.2: Signatur

Eine Signatur $\sum := (S, F)$ ist ein Paar aus einer Menge von Sorten S und einer Menge von Strukturbeschreibungen F. Eine Strukturbeschreibung aus F beschreibt ein n-stellige Operatorsymbol (OP) und hat die Form: Op: $S_1 \times S_2 \times \ldots\ldots \times S_n \rightarrow S_0$ mit $S_i \in S$ Sein i-ter S_i Operand muss ein Term der Sorte S_i sein. Der mit OP gebildet Term gehört der Sorte S_0 an.

Definition 3.9.3: Baumdarstellung

Ein n-stelliger Term mit der Operation f und den Untertermen $t_1, t_2, \ldots\ldots, t_n$ lässt sich als Baum darstellen. Beispiel: Die t_i in folgende Baumdarstellung sind Operatorsymbole.

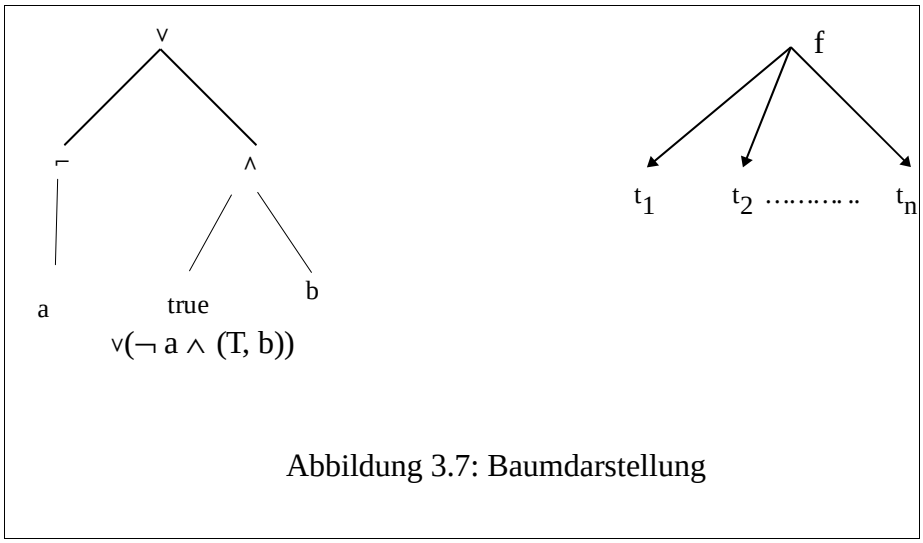


Abbildung 3.7: Baumdarstellung

Definition 3.10: Algebren

Eine Algebra ist eine formale Struktur. Sie besteht aus einer Menge grundlegender Elemente, Operationen über dieser Menge und Gesetze, die die Operationen zueinander in Bezug setzen.

In der Modellierung der Informatik spezifiziert man mit Algebren auch veränderliche Datenstrukturen wie Keller und Listen oder Abläufe in dynamischen Systemen.

Hierzu kann eine Unterscheidung von Algebra in zwei Ebenen erfolgen: abstrakte Algebra und konkrete Algebra.

Definition 3.11: Aussagenlogische Modellierung

Die Aussagelogik passt zur Repräsentation von statischem Wissen, welches in Elementaraussagen zerlegt und mit Hilfe der logischen Operatoren aufgebaut werden kann. Die Elementaraussagen dürfen dabei nur falsch oder wahr sein. Dynamische Systeme werden normalerweise nicht mit der Aussagenlogik formalisiert. Bei der Modellierung werden den einzelnen Elementaraussagen Atome zugeordnet, die von der Bezeichnung auf die Elementaraussagen schließen lassen. Für Übersetzung in eine entsprechende Formel mussdie passenden Verknüpfungen ausgesucht werden.

Beispiel: Der Aussage „Es regnet“ wird das Atom R zugeordnet und für „Die Straße ist nass.“ wird S gewählt.

- Es regnet nicht: $\neg R$,
- Entweder die Straße ist nass oder es regnet: $(R \vee S) \wedge (\neg R \vee \neg S)$.

Wir kommen schnell an die Grenzen der Aussagelogik, wenn es sich um Sachverhalte handelt. Wir gehen nicht weiter, weil sie für die definierte Problemstellung nicht geeignet ist.

Definition 3.12: Modellierung mit der Prädikatenlogik

Für die Problemstellung ist die Prädikatenlogik der ersten Stufe ausreichend. Im Gegensatz zu aussagenlogischen Formeln sind prädikatenlogische Formeln aus gewissen parametrisierten Elementaraussagen aufgebaut und sie können durch logische Operationen (Junktoren) verknüpft werden. Folgendes Beispiel zeigt, wie mit Hilfe prädikatenlogischer Formeln verschiedene Eigenschaften von Relationen beschreibbar sind:

$\alpha_1 = \forall x\ R(x, x)$, reflexiv,

$\alpha_2 = \forall x\ \neg R(x, x)$, irreflexiv,

$\alpha_3 = \forall x,y\ (R(x, y) \leftrightarrow R(y, x))$, symmetrisch,

$\alpha_4 = \forall x,y,z\ (R(x, y) \wedge R(y, z) \rightarrow R(x, z))$, transitiv,

$\alpha_5 = \forall x, \forall y\ (R(x, y) \vee x=y \vee R(y, x))$, Teilordnung.

3.1.2 Modellierung mit Graphen:

Graphen haben eine große Bedeutung für die Modellierung, weil sie gleichzeitig auf einfachen Relationen basieren, sehr vielfältig anwendbar und anschaulich darstellbar sind und als Datenstruktur in algorithmischen Lösungsverfahren verwendet werden. Der Kalkül der Graphen passt in vielen Aspekten gut zur Modellierung von Systemen und Aufgaben. Ein Graph besteht aus Knoten und Kanten als Abstraktion, die Knoten repräsentieren eine Menge gleichartiger Objekte jede der Kanten stellt eine Beziehung zwischen je zwei Objekten dar, die sie verbindet: Diese Graphen sind 2-stellige Relationen über der Knotenmenge, dazu sind sie verständliche Modelle und haben eine anschauliche Darstellung.

Grundlegende Definitionen:

Definition 3.13: Gerichteter Graph

Ein gerichteter Graph $G = (V, E)$ besteht aus einer endlichen Menge von Knoten V und einer Menge von gerichteten Kanten $E \subseteq V \times V$.

In der Graphentheorie und in der Modellierung haben ungerichtete und gerichtete Graphen jeweils eigenständige Bedeutungen.

Definition 3.14 : Ungerichteter Graph

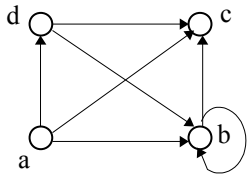
Ein ungerichteter Graph G ist ein Paar $G = (V, E)$ mit einer endlichen Menge von Knoten V und einer Menge von ungerichteten Kanten E: $E \subseteq \{\{x, y\} \mid x, y \in V\}$

Die Graphen kann man in der Modellierung einsetzen, um die Beziehung zwischen Objekten zu modellieren.[Kas,Kle08] Nachfolgend finden sich die vier bedeutendsten Darstellungsformen für Graphen:

a) Mengendarstellung:

Mengen V und E

V={ a ,b ,c ,d} , E = {(a ,b),(a ,c), (a ,d),(b ,b),(b ,c),(d ,b),(d ,c)}



b) Graphische Darstellung

	a	b	c	d
a	f	w	w	w
b	f	w	w	f
c	f	f	f	f
d	f	w	w	f

c) Darstellung als Adjazenzmatrix AM

a	(b,c,d)
b	(b,c)
c	( )
d	(b,c)

d) Darstellung als Adjazenzlisten

Abbildung 3.8: Die vier bedeutendsten Arten der Darstellung von Graphen

(a) Die abstrakte Angabe der Mengen V und E. (b) Die grafische Darstellung, die wegen ihrer Anschaulichkeit am häufigsten verwendet wird.(c) und (d) werden als Datenstrukturen für Algorithmen auf Graphen benutzt.

Definition 3.15: Grad in ungerichteter und gerichteter Graph

Sei G = (V, E) ein ungerichteter Graph. Dann ist der Grad eines Knotens v die Anzahl der Kanten (x, v)∈ E, die in v enden. Der Grad des Graphen G ist der maximale Grad seiner Knoten. In gerichteten Graphen: Grad v = Grad (v, x) + Grad (x, v) | (v, x) , (x, v) ∈ E oder Grad eines Knotens = Eingangsgrad dieses Knotens + Ausgangsgrad dieses Knotens. Für die in Abbildung 11 dargestellten Graphen ergibt sich der Grad: Grad (a) =1+2 =3 ,Grad (b) = 4, Grad (c) =3

Außerdem können die Knoten und Kanten mit Symbolen (Operatoren und Variablen) markiert werden. Aber die Markierungen, entweder der Kanten oder Knoten, muss durch eine Funktion definiert werden.

Beispiel: In diesem Beispiel gibt die Markierung an, für wie viele Verbindungen die Kante des Graphen steht. Die beiden Graphen haben die gleiche Form, allerdings erscheint der linke Graph anschaulicher. Diese Darstellungsform wird als Multigraph bezeichnet. Die Markierung m: E→N und m(v1,v2) = n | (v1,v2) ∈ E präsentiert die Kanten und v1,v2 ∈ V.

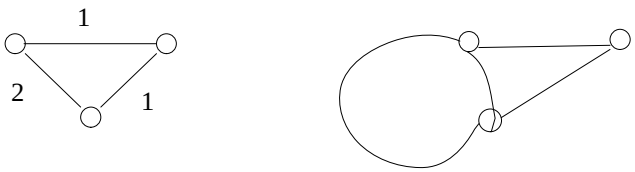


Abbildung3.9: Multigraph mit Mehrfachkanten

Definition 3.16: Weg

Sei G = (V, E) ein ungerichteter Graph. Eine Folge von Knoten (v_0, v_1, ..., v_n) mit $\{v_i, v_{i+1}\} \in E$ für $0 \leq i \leq n-1$ und $n \geq 0$ heißt ein Weg von v_0 nach v_n und hat die Länge n.

Wenn in einem Weg $v_0 = v_n$ gilt und dessen Kanten alle paarweise verschieden sind, wird der Weg im ungerichteten Graphen als Kreis und im gerichteten Graphen Zyklus genannt.

- **Euler-Weg bze Euler-Kreis:**

Sei G(V,E) ein ungerichteter, zusammenhängender Graph ohne Schleifen. Dann heißt ein Weg w Euler-Weg bzw. ein Kreis k heißt Euler-Kreis, wenn w bzw. k jede Kante aus E genau einmal enthält.

- **Hamilton-Kreis und -Weg:**

  Ein Weg w= (v_0, v_1, ..., v_n) in einem Graph G=(V, E) heißt Hamilton-Weg,wenn jeder Knoten aus V in v_0, v_1, ..., v_n genau einmal vorkommt und ein Hamilton-Kreis ist ein Kreis der jeden Knoten aus V in v_0, v_1, ..., v_{n-1} einmal enthält.

Diese Graphen werden zur Modellierung von verschiedenen Aufgaben benutzt. Zum Beispiel für Skizzen von Straßen, Autobahnverbindungen oder auch die Modellierung von Parallelrechnern.

Nachfolgendes Beispiel präsentiert eine typische Fragestellung im Bereich der Parallelrechner, die mit Hilfen von Graphen gelöst werden kann. Die Prozessoren des Parallelrechner sind in einem n*n Gitter verbunden. Eine Nachricht soll von einem Prozessor zu einem anderen weitergeben werden, jeden Prozessor erreichen und schließlich zum Initiator zurückkehren. Als Fragestellung ist gegeben, für welche Anzahl n ist das möglich? das bedeutet, gibt es einen Hamilton-Kreis? Mit speziellen Klassen von Graphen (Gitter) kann dieses Problem gelöst werden. Die Knoten des Gitters werden alternierend schwarz und weiß gefärbt, so dass verbundene Konten verschiedene Farben haben. Es gibt einen Hamilton-Kreis, wenn die Anzahl der Knoten n*n und die Kantenlänge n des Gitters gerade ist.

Verbindungsprobleme: Diese Probleme können mit ungerichteten Graphen modelliert werden.

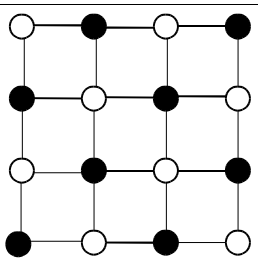


Abbildung 3.10: Hamilton-Kreis im Gitter mit n = 4

Definition 3.17: Ungerichteter Baum

Sei G = (V, E) ein ungerichteter, zusammenhängender Graph. Wenn G keine Kreise enthält, ist es ein ungerichteter Baum und alle Knoten, die den Grad 1 haben, werden Blätter genannt.

Definition 3.18: Spannbaum

Sei G (V, E) ein ungerichteter zusammenhängender Graph und G′= (V′, E′) ein ungerichteter Baum, der Teilgraph von G ist, und V=V′. Dann ist G′ ein Spannbaum von G.

Mit diesem Begriff wird ein bezüglich der Kanten kostengünstigerer Zusammenhang modelliert.

Beispiel: 8 Agenten werden mit A bis H benannt. Alle sollen direkt oder indirekt miteinander kommunizieren. Jede Kante wird mit einem Wert markiert, der das Risiko darstellt, dass die Kommunikation zwischen den beiden Knoten blockiert wird. Wir suchen für das Kommunikationssystem ein Netz mit minimalem Risiko. Dieses Problem lässt sich mit Hilfe des Spannbaums des Graphen lösen, so dass die Summe der Kantenwerte minimal wird.

3.1.3 Modellierung mit Bäumen

Definition 3-19: Gerichteter Baum

Ein gerichteter, zusammenhängender, azyklischer Graph G=(V, E) ist ein gerichteter Baum, wenn alle Knoten einen Eingangsgrad von 1 oder 0 haben. Der Knoten W ist die Wurzel von G, wenn dieser einen Eingangsgrad von 0 hat und der Baum hat die Höhe n, die gleich der Länge des längsten Weges von der Wurzel zu einem Blatt ist.

Der Binärbaum ist ein Baum, dessen Knoten einen Ausgangsgrad von höchsten 2 haben. Die Bäume werden zur Modellierung von Entscheidungen, Folge von Zuständen oder Strukturen benutzt.

Beispiele:

Modellierung von Klassen und Objekthierarchien von Rechnern durch gerichtete Bäume

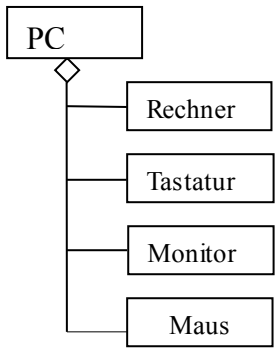


Abbildung 11: Klassen und Objektdiagramm

Definition 3.20: Bipartiter Graph

Ein Graph G = (V, E) heißt bipartit, wenn V in zwei disjunkte Teilmengen $V = V_1 \cup V_2$ zerlegt werden kann, so dass jede Kante zwei Knoten in verschiedenen Teilmengen verbindet.

3.1.4 Modellierung von Abhängigkeiten

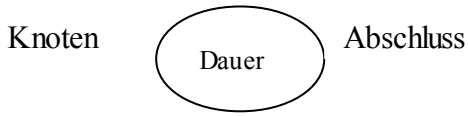
Mit gerichteten Graphen können Abhängigkeiten modelliert werden. Jeder Knoten modelliert eine Operation. Eine Kante (a,b) gibt an, dass die Ausführung von a Vorbedingung ist für Ausführung von b. Dafür kann es unterschiedliche Gründe geben.

Definition 3.21: Kritischer Pfad

In einem gerichteten, azyklischen Graphen G heißen Wege mit maximaler Länge kritische Pfade von G. Diese führen von einem Anfangsknoten, mit Eingangsgrad 0, zu einem Ausgangsknoten mit Ausgangsgrad 0.

- Manchmal stellt ein Graph sequentielle Operationen dar, aber die Operationen unterscheiden sich hinsichtlich ihres Zeitbedarfs, deshalb wird zur Modellierung eine Erweiterung des einfachen Graphen benötigt.

Solche Graphen haben Knoten mit zwei Markierungen, eine für die Repräsentation der Dauer und die andere zur Darstellung des Abschluss.



- Bei der Modellierung von Abhängigkeit kann die Rolle von Kanten und Knoten getauscht werden.
- Zum Schluss werden ergänzend weitere Anwendungen zur Modellierung mit Graphen vorgestellt, mit denen Abläufe beschrieben werden können. Ein Knoten modelliert einen Zustand, von dem aus der Ablauf zu unterschiedlichen Folgezuständen fortgesetzt werden kann. Eine Kante (a,b) modelliert den Übergang von Zustand (a) zu Folgezustand (b). Wenn es mehrere Folgezustände gibt, bleibt die Entscheidung innerhalb des Modells offen oder diese wird durch zusätzliche Kantenmarkierungen bestimmt.

3.2 Modellierung von Strukturen

Es werden zwei grundlegende Kalküle eingeführt, die zur Modellierung struktureller Eigenschaften von Systemen benutzt werden können: kontextfreie Grammatiken und das Entity-Relationship Modell.

3.2.1 Kontextfreie Grammatik:

Definition 3.22: Kontextfreie Grammatik (KFG)

Eine kontextfreie Grammatik G = (T, N, P,S) ist ein Tupel bestehend aus den endlichen Mengen: T Terminalsymbole (Terminal), N Nicht terminalsymbole (Nichtterminal), P Produktionen und dem Startsymbol $S \in N$. Die Mengen T und N sind disjunkt.

$V = T \cup N$ heißt auch Vokabular, die Elemente von V nennt man auch Symbole. Die Produktionen haben die Form $P \subseteq N \times V$.

Für eine Produktion (A, x) schreibt man A ::= x.

Definition 3.23 Ableitung

Sei G = (T, N, P, S) eine KFG, dann kann man mit der Produktion A::= x ∈ P das Nichtterminal A ∈ N in der Symbolfolge uAv ∈ V^+ durch die rechte Seite x der Produktion zu uxv ersetzen. Das ist ein Ableitungsschritt, werden mehrere Schritte nacheinander angewandt heißen diese Ableitung und werden notiert als $w_0 \Rightarrow w_1 \Rightarrow \ldots\ldots\ldots \Rightarrow w_n$ oder, falls die Einzelschritte nicht angegeben werden, $w_0 \Rightarrow^* w_n$.

Beispiel: HTML-Tabellen

In diesem Beispiel definieren die folgenden Produktionen eine vereinfachte Form von geschachtelten Tabellen in der Notation von HTML:

Table ::= '< table>' Rows '< /table>'

Rows ::= Row*

Row ::= '< tr>' Cells '< /tr>'

Cells ::= Cell*

Cell ::= '< td>' Text '< /td>'

Cell ::= '< td>' Table '< /td>'

3.2.2 Entity-Relationship-Modell

Das Entity-Relational-Modell (ER-Modell) ist ein formaler Kalkül zur Modellierung von Themenbereichen mit ihren Objekten, deren Eigenschaften und Beziehungen zwischen ihnen.

Für Spezifikationen im ER-Modell kann man eine grafische oder textuelle Notationen verwenden, wobei die grafische Notation anschaulicher ist.

Das ER-Modell basiert auf drei Grundbegriffen:

- Entity: Ein Objekt des Themenbereiches
- Relation: Eine Beziehung zwischen Objekten.
- Attribut: Ein Eigenschaft eines Objektes, beschrieben durch einen Wert.

Entity-Mengen: ist ein zentrale Begriff des ER-Modells.

Definition 3- 24: Entity–Menge

Eine Entity repräsentiert eine Zusammenfassung von Objekten, die im Modell als gleichartig gesehen werden, d.h. sie werden durch gleiche Attribute charakterisiert und können an den gleichen Relationen beteiligt sein. Eine konkrete Ausprägung zu einer Entity–Menge ist eine endliche Teilmenge der Objekte dieser Art. In den grafischen Notationen wird eine Entity–Menge durch ein Rechteck mit einem Namen daran angegeben. In einer konkreten Ausprägung wird die Objekt-Menge als Ellipse und die Objekte als Punkte mit identifizierenden Namen angegeben.

Objekte im ER-Modell stellen Gegenstände des modellierten Themenbereiches dar und jeder hat eine unterschiedliche Identität. Die Modellierung mit dem ER-Modell unterscheidet sich von der mit Wertemengen, die Elemente von Wertemengen werden nur durch ihre Werte unterschieden, die Elemente von Entity–Mengen durch ihre Identität.

3.2.3 Klassendiagramme in der UML

Die Klassendiagramme sind eine Teilsprache der Unified Modeling Language (UML), die am weitesten verbreitete und häufigsten eingesetzte Sprache zur Modellierung von Systemen. UML versucht die Modellierung unterschiedlicher Aspekte von Systemen möglichst umfassend abzudecken[Kas,Kle 08].

Klassendiagramme basieren auf den gleichen Grundkonzepten wie das Entity-Relationship-Modell. Sie werden zur Modellierung der Systemstruktur verwendet. Die Grundkonzepte der Kassendiagramme sind Klassen mit Attributen und Relationen zwischen den Klassen. Beispiel: P2P Netze

Abbildung 3-8: P2P Netze (Zwei Klassen mit Attributen und Assoziation).

3.3 Modellierung von Abläufen

Mit Verhaltensmodellen lässt sich das dynamische Verhalten eines Systems modellieren, daher eignen sich diese Modelle insbesondere für die als Schwerpunkt identifizierten parallelen und verteilten Systeme.

Nachfolgend werden die Techniken zur Modellierung von Abläufen eingeführt: endliche Automaten, dynamische Diagramme der UML und Petri-Netze. Sie werden eingesetzt, um das dynamische Verhalten von Systemen zu beschreiben.

3.3.1 Automaten:

Typische Beispiele dafür sind die Wirkung von Bedienoperationen auf reale Automaten oder auf Benutzungsoberflächen von Softwaresystemen; Schaltfolgen von Ampelanlagen; Abläufe von Geschäftsprozessen in Firmen und Steuerung von Produktionsanlagen. Solche Abläufe werden modelliert, in dem man die Zustände angibt, die das System einnehmen kann, und beschreibt, unter welchen Bedingungen es aus einem Zustand in einen anderen übergeht, sie werden einfach definiert und sind sehr anschauliche grafische Darstellungen. Endliche Automaten eignen sich zu Modellierung sequentieller Abläufe. Demgegenüber kann man mit Petri–Netzen nebenläufige Vorgänge beschreiben, bei denen Ereignisse im Prinzip gleichzeitig an mehren Stellen des Systems Zustandsänderungen bewirken können.

Endliche Automaten (Moore und Mealy):

Ein Automat ist ein besonderer Fall eines sequentiellen Verhaltensmodells. Ein endlicher Automat wird zur Modellierung einer realen oder abstrakten Maschine benutzt. Dabei wird insbesondere beschrieben wie die Maschine:

- auf äußere Ereignisse reagiert,
- ihren inneren Zustand ändert und

- gegebenenfalls Ausgaben produziert.

Definition 3-25[wikipedia.org][Pri, Wim08]: endlicher Automat

Formal ist der Automat durch ein Fünftupel (Σ, S, s0, δ, F) definiert,wobei

- Σ ist das Eingabealphabet (eine endliche nicht leere Menge von Symbolen),
- S ist eine endliche nicht leere Menge von Zuständen,
- s0 ist der Anfangszustand und ein Element aus S,
- δ ist die Zustandübergangfunktion: δ: S x Σ → S,
- F ist die Menge von Endzuständen und eine (möglicherweise leere) Teilmenge von S.

Endliche Automaten werden typischerweise eingesetzt, um

- Das Verhalten realer Maschinen zu spezifizieren;
- Das Verhalten von Software-Komponenten zu spezifizieren, z.B wie Benutzungsoberflächen auf Bedienereignisse reagieren;
- Sprachen zu spezifizieren als Menge der Ereignis- oder Symbolfolgen, die der endliche Automat akzeptiert, z. B. die Schreibweise von Bezeichnern und Zahlwerten in Programmen.

Ein Transduktor ist ein 6-Tupel (Σ, Γ, S, s0, δ, ω), wobei:

- Σ ist das Eingabealphabet (eine endliche nicht leere Menge von Symbolen),
- Γ ist das Ausgabealphabet (eine endliche nicht leere Menge von Symbolen),
- S ist eine endliche nicht leere Menge von Zuständen,
- s0 ist der Anfangszustand und ein Element aus S,
- δ ist die Zustandsübergangsfunktion: δ: S x Σ → S,
- ω ist die Ausgabefunktion.

Falls die Ausgabefunktion eine Funktion von Zustand und Eingabealphabet ist

(ω: S x Σ → Γ), dann handelt es sich um ein Mealy-Modell. Falls die Ausgabefunktion nur vom Zustand abhängt (ω: S → Γ), dann ist es ein Moore-Automat.

Zeichenfolgen über Alphabeten

Ein endlicher Automat definiert auch eine Sprache, sie umfasst die Folgen von Symbolen oder Ereignisse, die vom Automaten als legale Eingabe akzeptiert werden.

Definition 3-26: Alphabet

Ein Alphabet ist Menge von Zeichen. Sie dienen zur Bildung von Zeichenfolgen. Alphabete werden mit ∑ gezeichnet. Beispiele: { a, … ,z, A, … Z}; Dezimalziffern {0,1,2, … ,8,9}; Binärziffern {0,1}.

3.3.2 UML (2)-Diagramme zur Modellierung dynamischen Verhaltens

Zustandsübergangsdiagramme (Statecharts)

Ein Zustandsdiagramm zeigt eine Folge von Zuständen, die ein Objekt im Laufe seines Lebens einnehmen kann [Fen05].
Zustandsübergangsdiagramme beschreiben :

- das Verhalten von Objekten einer Klasse

• die Ereignisse, die einen Übergang (Transition) von einem Zustand zu einem anderen veranlassen
• die Aktionen, die sich aus diesem Zustandswechsel ergeben.
• die Aktivitäten und Aktionen, die in einem Zustand durchgeführt werden, sind:
entry/**Aktion,** do/**Aktivität,** exit/**Aktion** und event/**Aktion**
Jeder Zustandsautomat enthält Anfangszustandsmarke, Endzustandsmarke und Zustandsnamen.
Eine Aktion benötigt (konzeptuell) keine Zeit, hingegen benötigt eine Aktivität eine gewisse Zeit.
Eine **Entry-Aktion** wird immer beim Betreten des Zustandes ausgeführt,
während eine **Exit-Aktion** immer beim Verlassen des Zustandes ausgeführt wird.
Eine **Event-Aktion** wird durch ein Ereignis ausgelöst, aber der aktuelle Zustand wird beibehalten.
Die **do-Aktivität** wird ausgeführt, während der Zustand andauert und beendet entweder durch ein vorgesehenes (reguläres) Ende oder durch Auftreten eines Ereignisses, das einen Zustandsübergang verursacht.

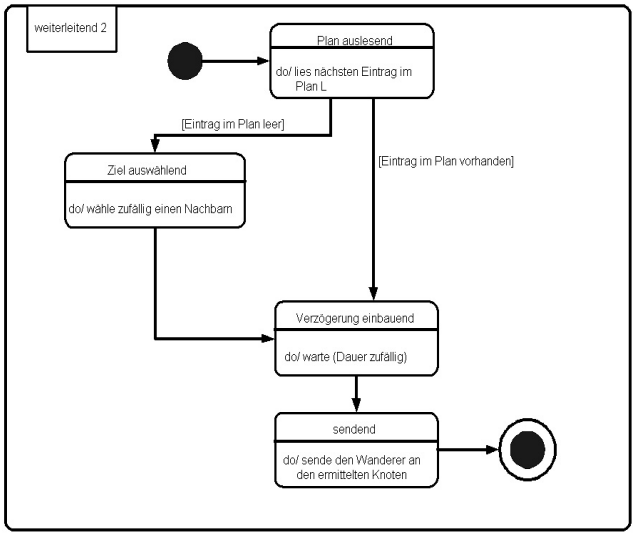


Abbildung 12 [Ger06]: Protokollzustandsautomat modelliert einen Teil des Algorithmus, wie ein Wanderer weiter geschickt wird entsprechend seinem Plan.

Beispiel: Sequenzdiagramme
Sequenzdiagramme beschreiben den Ablauf von Transaktionen und Interaktionen (Kommunikation). Sie zeigen den Informationsaustausch zwischen beliebigen Kommunikationspartnern innerhalb eines Systems oder zwischen Systemen generell. Sie ermöglichen die Modellierung von festen Reihenfolgen, zeitlich und logischen Ablaufbedingungen, Schleifen und Nebenhäufigkeiten.

Transaktion: Eine Menge von Nachrichten, die zwischen einer Gruppe von Objekten ausgetauscht wird, um eine bestimmte Operation oder ein bestimmtes Ergebnis zu veranlassen.
In vielen Anwendungsfällen müssen Transaktionen unteilbar sein und weitere Anforderungen erfüllen (z.B. im Anwendungsfall Datenbanken).

Interaktion: Austausch einer einzelnen Nachricht. Beispiel[Rupr,Hah,Que, ..05]:

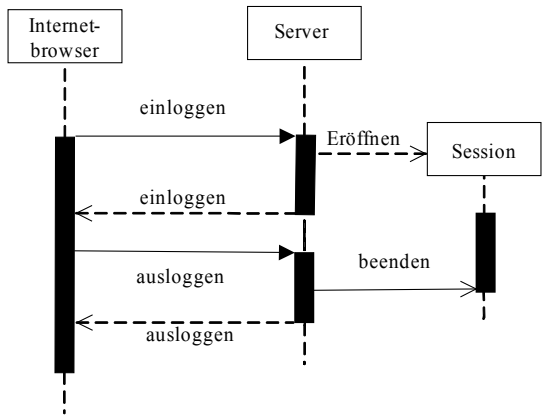


Abbildung 13: Sequenzdiagramm Internetsuche

Aktivitätsdiagramm:

Mit Hilfe dieser Diagrammart kann ein komplexer Verlauf unter Berücksichtigung von Nebenläufigkeiten, alternativen Entscheidungswegen und ähnlichem konkret modelliert und nachvollzogen werden. Die Semantik ist ähnlich dem Zustandsdiagramm und häufig werden die Elemente des Zustandsdiagramms auch in Aktivitätsdiagramm verwendet[Fen 05].
Elemente von Aktivitätsdiagrammen:
Aktionsknoten: kleinste ausführbare Funktionseinheit
Objektknoten: Datenbehälter (zum Weitergeben)
Kontrollknoten: für Entscheidungen und Zusammenführung im Kontrollfluss
Synchronisationsbalken: für nichtdeterministische Verzweigung undSynchronisation.
Aktivitätsbereiche: Zusammenfassung von Teildiagrammen zu höheren Einheiten.
Startzustand: Einsitzpunkt der Aktivität (muss nicht eindeutig sein)
Ablaufende: Ende eines Kontrollflusses (andere können noch aktiv sein)
Endzustand: Ende der gesamten Aktivität
Beispiel [Evg 07]:

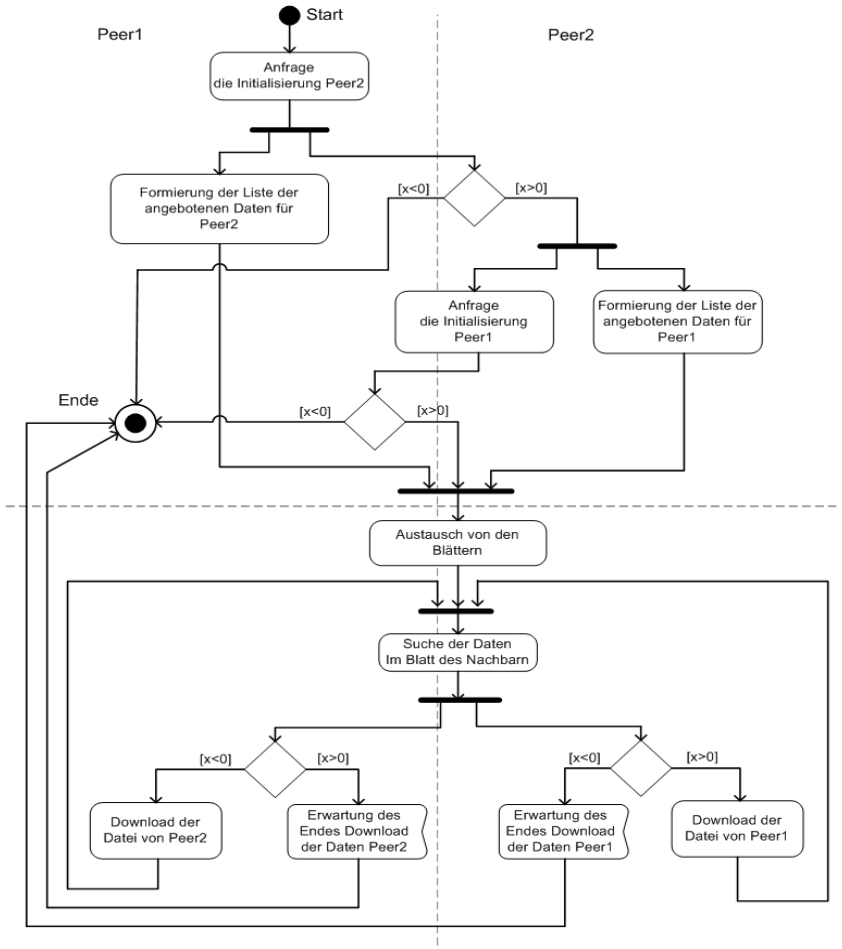


Abbildung 14: Die Suche in reinem P2P und Austausch von den Daten.

3.3.3 Petrinetze:

Petrinetzmodelle beschreiben insbesondere die Interaktionen zwischen Prozessen und die Effekte, dass Operationen prinzipiell gleichzeitig ausgeführt werden können.

Typische Beispiele für die Anwendung von Petrinetzen sind Modellierungen von realen oder abstrakten Automaten und Maschinen;

- kommunizierenden Prozessen in der Realität oder in Rechnern;
- Verhalten von Software- oder Hardware-Komponenten;
- Geschäftsabläufen;
- Spielen nach bestimmten Regeln.

Definition 3-27: Petrinetz

Ein Petrinetz ist ein Tripel PN = (P, T, F) mit einer Menge von Plätze P, einer endlichen Menge von Transitionen T und einer Relation F mit $F \subseteq P \times T \cup T \times P$. P bildet einen bipartiten Graphen mit den Knoten $P \cup T$ und den gerichteten Kanten F. Wobei:

- Plätze können mit mehreren Marken (Token) belegt sein.

- Die Schaltregel beschreibt Übergang der Marken von Vor- auf Nachplätze.
- Eine Transition darf schalten (feuern), wenn alle Vorplätze belegt sind.
- Beim Schalten einer Transition wird aus jeder Eingabestelle eine Marke entfernt und zu jeder Ausgabestelle eine Marke hinzugefügt.
- Die Plätze P repräsentieren Bedingungen oder Zustände des modellierten Systems. Sie werden in der Grafik durch Kreise dargestellt. Die Transitionen repräsentieren Zustandsübergänge oder Aktivitäten und werden durch Rechtecke dargestellt.

Der Zustand eines Petrinetzes P wird durch eine Markierung angegeben. Das ist eine Funktion M_p: $P \rightarrow N_0$, die jedem Platz eine Anzahl von Marken zuordnet.Die Marken werden in der Grafik als Punkte in den Plätzen dargestellt. Die Markierung kann als eine Folge von nicht-negativen Zahlen angeben werden.

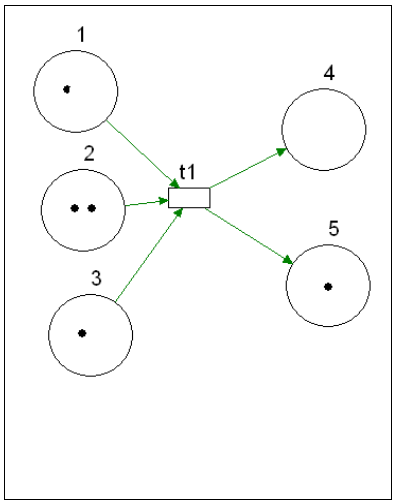
Definition 3-28 Petrinetz (Vor- und Nachbereich)

Zu einer Transition t in einem Petri-Netz PN sind zwei Mengen von Plätze definiert:

Vorbereich (t):={ P | (P, t) ∈ F}

Nachbereich (t):={ P | (t, P) ∈ F}.

Beispiel: t eine Transition mit ihrem Vorbereich {1,2,3} und ihrem Nachbereich {4,5}.



Definition 3-29 Schaltregel

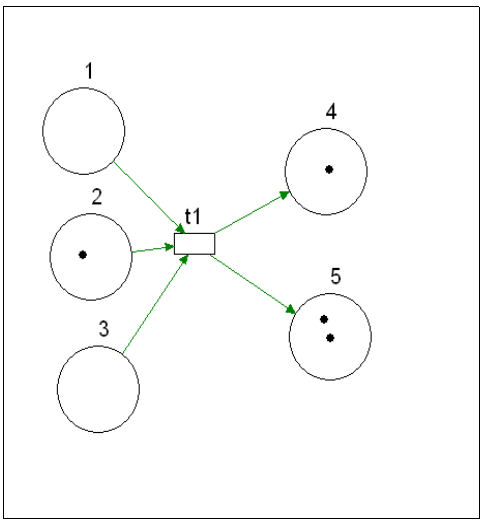
Eine Transition t eines Petrinetzes PN kann schalten, wenn für alle Plätze P ∈ Vorbereich(t) gilt M(P)≥ 1. Wenn zu einem Zeitpunkt mehrere Transitionen schalten können, wird eine davon nicht-deterministisch ausgewählt; sie schaltet als nächste. Wenn t schaltet, gilt für Nachfolgemarkierung M′:

M′(P)= M(P) -1 für alle v ∈ Vorbereich(t) - Nachbereich(t)

M′(n)= M(n) +1 für alle n ∈ Nachbereich(t) - Vorbereich(t)

M′(P)= M(P) für alle übrigen Plätze.

In vorigem Beispiel, wenn t schaltet, ergibt sich das folgende Petrinetz:



Durch die nicht-deterministisch Auswahl aus der Menge der Transitionen, die schalten können, sind aber ebenso viele unterschiedliche Abläufe möglich wie in dem modellierten parallelen System.

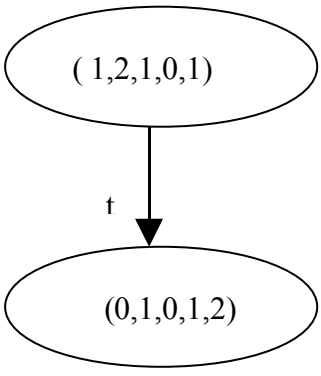
Definition 3-30 Konflikt

Zwei Transitionen t1 und t2 stehen in Konflikt, wenn Vorbereich (t1) ∩ Vorbereich (t2) ≠ ∅ und im Vorbereich nicht genügend Marken vorhanden sind, um beide Transitionen zu schalten. Das Schaltverhalten und das Durchlaufen der Markierungen eines Petrinetzes kann man durch einen Graphen beschreiben.

Definition 3-31 Erreichbarkeitsgraph

Man kann zu einem Petrinetz P und einer Anfangsmarkierung M_0 einen gerichteten Erreichbarkeitsgraph angeben: seine Knoten repräsentieren jeweils eine Markierung. Der Graph hat eine Kante x→ y, und der Pfeil wird mit t beschriftet, d.h. wenn x von der Anfangsmarkierung erreicht werden, t in x schalten kann und damit x in y überführt wird.

Erreichbarkeitsgraph für das vorige Beispiel:



Definition 3-32 Lebendige Transition

In einem Petrinetz P mit einer Anfangsmarkierung M_0 heißt eine Transition t lebendig, wenn es zu jeder von M_0 erreichbaren Markierungen, eine Markierung M gibt, die von M_0 erreichbar ist und in der t schalten kann.

Definition 3-33 Lebendiges Petrinetz

Ein Petrinetz mit einer Anfangsmarkierung M_0 heißt lebendig, wenn alle seine Transition lebendig sind.

Definition 3-34 Vor-, Nachbereich von Plätze

Sei $\sigma \subseteq S$ eine Teilmenge der Plätze eines Petrinetzes. Dann sind

Vorbereich $(\sigma) := \{t \mid \exists P \in \sigma : (t, P) \in F\}$

Nahbereich $(\sigma) := \{t \mid \exists p \in \sigma : (P, t) \in F\}$

Mengen von Transitionen, die auf Plätze in σ wirken bzw. die Plätze in σ als Vorbedingung haben.

Definition 3-35 Verklemmung

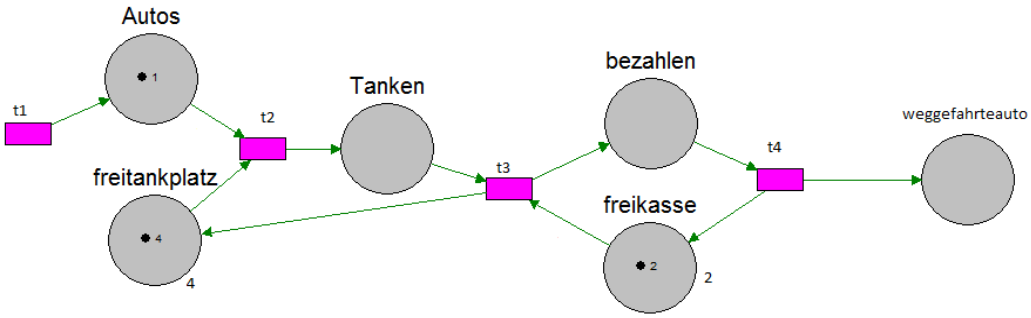
Eine nicht-leere Menge σ von Plätze eines Petri-Netzes heißt Verklemmung, wenn gilt Vorbereich $(\sigma) \subseteq$ Nachbereich (σ)

Definition 3-36 Kapazität

In einem Petrinetz P kann man Plätzen begrenzte Kapazitäten $k \in N$ zuordnen. Wenn ein Platz die Kapazität k hat und im Nachbereich einer Transition t liegt, dann kann t nur schalten, falls dadurch die Anzahl der Marken auf P nicht größer als k sein werde.

Definition 3-37 Vielfachheit

In einem Petrinetz P kann man Kanten eine Vielfachheit $v \in N$ zuordnen. Wenn eine Kante mit der Vielfachheit v am Schalten einer Transition beteiligt ist, müssen v Marken über diese Kante bewegt werden.



Typen von Petrinetzen

1. Bedingungs-/Ereignis-Netze (B/E-Netze)
• höchstens eine Marke pro Platz

2.Platz/Transitions-Netze (P/T-Netze)
• Plätze mit Kapazität (maximale Anzahl von Plätzen)
• Kanten mit Vielfachheit

• Eine Transition kann schalten, wenn jede ihrer Vorplätze mindestens so viele Marken enthält wie das Gewicht der dahinter liegenden Vorkante und jede ihrer Nachplätze höchstens so viele Marken enthält wie ihre Kapazität minus die Vielfachheit der davor liegenden Nachkante.

3.Prädikat/Transitions-Netze (Pr /T-Netze)
• Marken haben beliebigen Datentyp
• Transitionen mit Schaltbedingungen und Schaltwirkungen hierarchische PN.

4. Zeitbehaftete Petrinetze werden in den nächsten Kapiteln erläutert.

5. Gefärbte Petrinetze
Es gibt noch einige weitere Typen von Petrinetzen, diese werden in den nächsten Kapiteln ausführlicher betrachtet.

Kapitel 4 Höhere gefärbte und zeitbehaftete Petrinetze

Petrinetze bilden ein formales Hilfsmittel zur Modellierung paralleler Systeme, das durch die graphische Darstellung der Netze besonders attraktiv ist. Sie sind eine anerkannte Technik. Sie werden als formal fundierte Darstellungsmittel in den verschiedensten Ausprägungen eingesetzt, insbesondere für die Beschreibung und Analyse von Synchronisation, Kommunikation und Ressourcenvergabe zwischen verteilten und nebenläufigen Prozessen eignen sie sich gut.

Die Theorie der Petrinetze ist ihrer Herkunft und dem mit ihr verbundenen Anspruch nach eine Grundlagen-Theorie. Ihr Ziel ist es, die fundamentalen Phänomene der Nebenläufigkeit und der Veränderung inhaltlich und formal zu durchdringen[Gru88]. Deshalb sind Petrinetze die richtige Auswahl als Modellierungsmethode für Problemstellung dieser Arbeit im Bereich der parallelen verteilten Systeme.Dieses Kapitel behandelt verschiedene wichtige Klassen von Petrinetzen, die in vielen Bereichen als Modellierungstechnik verwendet werden und die im Zusammenhang mit dieser Arbeit stehen.

4.1 Klassifikation von Petrinetzen

„Der Begriff des Petrinetzes und die Netztheorie wurden auf die Suche nach natürlichen, einfachen und zugkräftigen Methoden zur Beschreibung und zu Analyse des Information-Kontrolldatenflusses in Informationsverarbeitenden Systemen entwickelt“[Sta90].
Petrinetze werden in zwei Kategorien eingeteilt, in schwarze Petrinetze und gefärbte Petrinetze, low level Petrinetze und high level Petrinetze, oder normale Petrinetze und erweiterte Petrinetze.
Diese drei Einteilungen in zwei Kategorien sind Synonym.

Schwarze Petrinetze

Als gemeinsames Merkmal für diese Kategorie haben sie genau eine Art von Marken, die als schwarze Marken (ununterscheidbare Marken) bezeichnet werden. Low level und klassisch (normal) nennt man auch alle Petrinetz Typen, die nur eine Art von ununterscheidbaren Marken für ihre Petrinetze zulassen.

- Kanal/Instanzennetz K/I: [Rei85] Diese Netze enthalten keine Marke und ihre Platzkapazitäten sind auf den Wert Null festgelegt. Sie sind rein statisch und werden verwendet, um strukturellen Zusammenhänge innerhalb eines Systems grafisch darzustellen. Die Transitionen sind die aktiven Instanzen und Kanäle sind passive Systemteile. Aber die hierarchische Darstellung dieser Netze wird verwendet, um die höchste Netzebene darzustellen.
- Bedingung / Ereignisnetze: In B/E Netze enthalten alle Plätze maximal eine Marke, wodurch sie als boolesche Variable, die die Zustände “belegt„ oder „nicht belegt“ annimmt, interpretiert wird. Die Bedingungen demzufolge gelten als erfüllt, sofern alle Eingangsplätze mit einer Marke belegt sind [Rei85].
- Platze / Transitionsnetze: Platzkapazitäten dieser Netze dürfen variable Werte größer Null annehmen, dazu werden Kantengewichten zugelassen, die bestimmen, wie viele Marken von den Plätzen des Vorbereichs entnommen und auf den Plätzen des Nachbereiches abgelegt werden[Fen05]. P/T-Netze gehören auch wie Elementare Netzsysteme [Tui97], [Roz87] zu den so genannten low Level Netzen.
- Da die Ausdrucksmächtigkeit dieser schwarzen Petrinetze für viele Anwendungen nicht

ausreichend ist, wurden in vergangen Jahren zahlreiche Erweiterungen vorgenommen.

Gefärbte Petrinetze [1]

Höhere Petrinetz-Typen besitzen Marken mit Identitäten, Plätze mit Multimengen von Marken und eingeschränktes Schaltverhalten von Transitionen durch Schaltmodi. Die Informationen der Netzstruktur werden durch Beschriftung der Netzelemente ausgenutzt.
CPNs sind anerkannte Modellierungsmittel, die eine Erweiterung von klassischen Petrinetzen und die bekanntesten höheren Petrinetze darstellen. Im graphischen Petrinetz-Modell wird der Zustand im Platz eines Petrinetzes durch markiert oder nicht markiert mit Farbmarken dargestellt, die Funktion durch Aktivierung von Schaltregeln realisiert und die Verweilzeitfunktion wird durch zeitbewertete Plätze oder Transitionen präsentiert. Diese wird in dem nächsten Abschnitt behandelt.

Es gibt zwei Art von Farbset [Jen95]:
1- atomic Farbset: atomic Farbset ist ein definiertes Farbset ohne Verweise auf andere bestehende Farbsets .
2- Im Gegenteil wird ein strukturiertes Farbset auf Basis eines Base Farbset mit verschiedenen Methoden und Mechanismen definiert. Z.B. Rekord, Union, List, oder Subset.

In der Literatur gibt es keine einheitliche formale Definition. Im folgenden wird auf eine formale Definition für ein möglichst allgemeines gefärbtes Petrinetz eingegangen [Jen92], [Rok92], [Kno95].

Definition 4.1

Das Tupel CN = (P, T, F, V, K, C, m_o, ζ) ist ein gefärbtes Petrinetz, mit:

1. P, T endliche nicht leere Mengen von Plätzen und Transitionen
2. F Flussrelation: Menge aller Kanten, wobei F die Vereinigung aus Vor- und Nachkanten ist, so dass gilt: $F \subseteq (P \times T) \cup (T \times P)$
3. V Vielfachheit der gefärbten Kanten; Abbildung der Kanten auf eine Menge, bestehend aus Kantengewichten: $P \times T \rightarrow BOOL(C) \cup T \times P \rightarrow C_t(C)$
4. K Kapazitätsfunktion K für Plätze: $P \rightarrow C_t(C)$
5. C Menge der Farben
6. $m0_c$ Anfangsmarkierung: $P \rightarrow C_t(C)$
7. ζ Farbkonvertierungsfunktion: $\zeta = \cup S(t)$ für alle $t \in T$ mit $S(t) = \cup S_i(t)$ oder leere Menge und $S_i(t) = (p,t) \rightarrow BOOL(C)$ und $(t,p) \rightarrow C_T(C)$

Wobei $C_t(C)$ eine Menge ist, in der jedes Element aus C höchstens nur einmal vorkommen kann und zwar in Verbindung mit einer natürlichen Zahl als Faktor. Die Menge BOOL(C) stellt eine Menge von Booleschen Ausdrücken mit logischem „und“ und „oder“ über den mit Faktoren versehenen Elementen aus C dar, wobei diese auch geklammert sein können.
Zu den Plätzen und Transitionen kommen als Erweiterung noch Mengen von Marken verschiedener Typen, welche durch Farben (C) unterschiedenen werden. Die Menge C besteht aus allen auftretenden Farben im Netz.
Die Kapazitätsfunktion für die Plätze wird wegen dem Vorkommen verschiedener Markentypen erweitert. Wobei jede Farbe im Platz einzeln begrenzt wird, sowie die maximale Summe von allen Marken angegeben wird. Wenn $m(p,c)$ die Anzahl von Marken der Farbe

1Coloured Petri nets (CPN) in Englisch

„c“ im Platz p ist, dann gilt $\forall c \in C$, $m(c, p) \leq K(p, c)$ und $\Sigma c \in C$, $m(p, c) \leq K(p, \{gesamt\})$ mit $p \in P$. Die Kantengewichte der Vorkanten von Transitionen stellen einen Booleschen Ausdruck (ohne Negation) aus einer Teilmenge der mit einem Faktor versehenen Farben dar. Andererseits sind die Kantengewichte der Nachkanten von Transitionen eine Teilmenge der mit einem Booleschen Ausdruck (ohne oder) mit einem Faktor versehenen Farben.
Anstelle der eben beschriebenen Kantengewichte können so genannte Farbkonvertierungsfunktionen genutzt werden. Diese gelten lokal im Vor- und Nachbereich einer Transition und enthalten verschiedene Schaltmodi (Menge S(t)), die untereinander mit logischem „oder“ verknüpft sind. Jeder Schaltmodus ($S_i(t)$) definiert für alle Vor- und Nachkanten der Transition eine Kombination von Kantengewichten in der oben beschriebenen Form. Die übliche Weise für die Angabe von Farbkonvertierungsfunktionen sind Tabellen, welche genau soviele Spalten haben, wie die Transition Vor- und Nachkanten hat, und für jeden Schaltmodus eine Zeile besitzen. In den Zeilen stehen entsprechend die Kantengewichte für die Vor- und Nachkanten.
Für die beschriebenen Gefärbten Petri-Netze gilt als Schaltregel:
• Eine Transition t ist schaltfähig, wenn in allen Vorkanten von t die Booleschen Ausdrücke den Wahrheitswert „1“ haben. Das heißt, dass farbrichtig ausreichend Marken vorhanden sind um beim Einsetzen in den Booleschen Ausdrücken den Wahrheitswert „1“ zu bekommen. Es ist notwendig, dass es in allen Nachplätzen von t ausreichende Kapazität gibt, um erzeugte ankommende Marken aufzunehmen. Die Berücksichtigung der oben genannten Gesamtkapazität eines Platzes muss gewährleistet sein.
• Beim Schalten wird in den Vorplätzen von t farbrichtig entsprechend der notwendigen Einsetzung in den Kantengewichten subtrahiert und in den Nachplätzen von t addiert.
• „Farbrichtig“ heißt in dem Zusammenhang, dass sowohl in der Markierung, der Kapazität und den Kantengewichten Terme mit der gleichen Farbe betrachtet werden.
Eine formale Definition zur Schaltregel enthält[Kno95].

Prädikat Transition Petrinetze Pr/T

Nach CPN wird das zweitbekannteste der High Level (höhere) Petrinetze als Pr/T Netze bezeichnet. Folgend wird Pr/T Petrinetze [Gen,Lau81]definiert.

Definition 4.2

Pr/T Netze sind ein Tripel $N = (S, T; F)$ ist direkte Netz wenn

1) $S \cap T = \emptyset$,
2) $S \cup T \neq \emptyset$,
3) $F \subseteq (S \times T) \cup (T \times S)$,
4) $dom(F) \cup cod(F) = S \cup T$

Für genannte Netz $N = (S, T; F)$ nennen wir

5) $X := S \cup T$ das Set der Elemente des Netzes, und
6) F Flussrelation Menge aller Kanten des N für $x \in X$,
7) $\bullet x := \{y \mid (y,x) \in F\}$ und $x \bullet := \{y \mid (y, x) \in F\}$ sind in dieser Reihenfolge Preset und Postset von x.

- Prädikate können an Plätzen und Transitionen sein.
- Eine Markierung M(s) stellt bei Prädikat-Transitions-Netzen eine Multimenge von Worten

aus einem Wertebereich D dar. Jedem Platz kann in diesem Zusammenhang ein bestimmter Datentyp zugeordnet sein, z.B. die Menge der reellen oder natürlichen Zahlen.

- Das Schalten einer Transition kann bei Netzen mit individuellen Marken von mehr als nur dem Vorliegen gewisser Anzahlen von Marken auf den Vorplätze abhängig gemacht werden. Zusätzliche Bedingungen (Prädikate) können qualitative Forderungen bzgl. der durch eine Transition verbrauchten bzw. erzeugten Marken ausdrücken.
- Die Schaltregel lässt sich bei Prädikat-Transitions-Netzen nun wie folgt definieren:

-Es kommt durch das Schalten einer Transition zu einem Abzug jeweils eines Wertes $a_i \in D$ (si) aus jeder Eingangsplatz si.

- Der Abzug der Marken aus einer Eingangsplatz und die Ablage der Marke in eine Ausgangsplatz ist an Bedingungen (Prädikate) für die Werte a_1, a_2, ... geknüpft. Damit eine Transition schalten kann, müssen alle ihr zugeordneten Prädikate wahr sein.
- Es kommt durch das Schalten einer Transition zu einer Ablage spezifizierter Ausgangswerte in den Nachbereichsplätzen.

Beispiel [Gen87]**:**

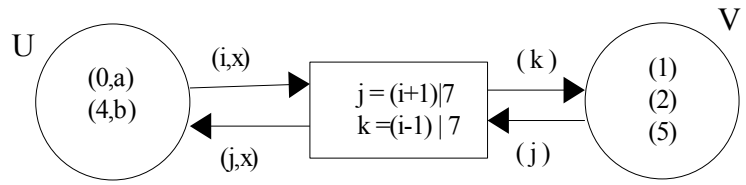


Abbildung 4.1 Bahnhof System mit Variable Relationen als Plätze

In Abbildung 4.1 stellen die Plätze zwei variable Relationen des Bahnhof Systems in eins zu eins Mode vor. Das wird durch Relationssymbole, d.h. Prädikate, dargestellt. Die Transition schaltet in einer Schaltmodi. Dann werden die Variablen x,i,j,k mit Konstanten ersetzt.

4.2 Zeitbehaftete Petrinetze (Zeit-Perinetze) [2]

Petrinetze werden als Zeit-Petrinetze bezeichnet, die Uhren und diesbezügliche Zeitanschriften verwenden. Zeit-Petrinetze werden verwendet, um zeitliche Aspekte in Systemen zu modellieren und vor allem zu analysieren.

Zeitkonzepte für Petrinetze sind zunächst für Zwecke der Simulation eingeführt worden. Inzwischen wurden alle Petrinetz-Elemente in verschiedenen Arbeiten mit einer Zeitbewertung versehen. Dabei wird das Schalten einer Transition davon abhängig gemacht, ob außer der Markierung auch eine bestimmte zeitliche Bedingung erfüllt ist. Zum einen die Quantifizierung der Verweildauern von Marken auf Plätzen und zum anderen die Quantifizierung von Verzögerungszeit an Transitionen. Zeitbewertung in Petrinetzen bedeutet eine Zeitbewertung der Netzelemente [Sta90]. Entsprechend besteht prinzipiell in einem Petrinetz die Möglichkeit, jedem der drei Elemente Plätze, Transitionen und Kante eine Zeitbewertung zuordnen. Um zeitliche Eigenschaften in einem Petrinetz abbilden zu können, muss der verwendete Petrinetz-Typ die Zeit mit einbeziehen. Dadurch werden diese Netze um eine Dimension erweitert, aber ohne ihre Kausalität zu verändern. Einen Überblick über verschiedene Ansätze zur Zeitbewertung geben u.a., [Bau92][Kön, Quä88], [Sta90], [Han92].

Zeitkonzepte wurden für low level Netze eingeführt, aber lassen sich nächst relativ einfach für high level Netze anpassen [Jen92], da sie weitgehend unabhängig von Petrinetz-konzepten sind.

Eines der ersten zeitbehafteten Netze dieser Art wird in [Ram74] vorgestellt. Diese Zeit-

2 Timed Petri nets

Petrinets bestehen aus P/T Netzen mit Zeitanschriften (als Label) an Transitionen. Jede Transition hat eine Uhr, die bei (0) gestartet wird, wenn die Transition konzessioniert ist (im Sinne eines P/T Netzes). Wenn eine zeitverzögerte Transition schaltfähig ist, beginnt sie zu schalten und beendet das Schalten nach der definierten Schaltdauer.
Als Stochastische Petrinetze werden im Allgemeinen Zeit-Petrinetze bezeichnet, die eine exponentiell verteilte Funktion als Zeitanschrift zulassen. Damit können Prozesse simuliert und analysiert werden, in denen eine als Zufallsgröße angenommene Bearbeitungszeit eine Rolle spielt [AMC87]. Eine Transition schaltet dann, wenn sie konzessioniert ist und die Präsenzzeit der entsprechenden Marken auf den Vorplätzen für einen aktuellen Wert der Verteilungsfunktion erreicht ist.
In folgenden wird eine systematische Gliederung der verschiedenen Netztypen nach dem zeitbewerteten Netzelement erfolgen.

4.2.1 Zeitbewertete Marken

Das Konzept der zeitbewerteten Marken wird u.a. in [Sta90]vorgestellt. Diese Marken haben zwei Zustände: verfügbar und nicht verfügbar. Jeder Marke kann eine individuelle, lokale Uhr zugeordnet werden. Die erzeugten Marken bekommen den Zustand nicht verfügbar beim Schalten einer Transition. Beim Eintreffen im Nachplatz der Transition startet die interne Uhr der Marke. Nach Ablauf einer festgelegten Zeit ändert die Marke ihren Zustand in verfügbar. Diese Zeit kann fest (deterministisch) sein oder einem statistischen Gesetz folgen. Bei dieser Art der Festlegung der Zeiten spricht man von einem stochastischen Petrinetz. Diese eignen sich besonders zur Modellierung zufällig eintretender Ereignisse. Einen Überblick über determinierte und stochastische Petrinetze gibt [Bau,Kri96].
Nachdem eine Marke ihr vorgegebenes, individuelles Mindestalter erreicht hat, steht sie für weitere Transitionen zur Verfügung. Vor Ablauf dieser Verweildauer ist die Marke zwar sichtbar, aber nicht verfügbar. Allen Marken, die von einer Transition produziert werden, soll das gleiche Mindestalter zugeordnet werden, dadurch kann man die Zeitbewertung durch eine Schaltzeit der Transition vereinfachen.

4.2.2 Zeitbehaftete Transitionen

Zeitbehafteten Transitionen kann eine Schaltdauer zugeordnet werden. Die Art des Schaltverhaltens dieser Transitionen kann unterschieden werden in **Zeitbewertung** und **Zeitverzögerung**.
Eine Transition kann schalten, wenn sie weiterhin konzessioniert ist und ihre Zeitanschrift kleiner oder gleich der Stellung ihrer Uhr ist und außerdem genügend Marken auf ihren Vorplätzen vorfindet. Das heißt, eine Transition schaltet nach einer gewissen Präsenzzeit der zum Schalten nötigen Marken. Eine Uhr wird ausgeschaltet, wenn die Transition ihre Konzession verliert.
Während **zeitbewertete** Marken auf den Plätze haften bleiben und weiterhin sichtbar sind, sind alle konsumierten Marken einer **zeitverzögerten Transition** für die Zeit der Schaltdauer nicht sichtbar. Wenn eine zeitverzögerte Transition schaltfähig, beginnt sie zu schalten und beendet das Schalten nach der definierten Schaltdauer.
Bei einer **zeitbewerteten** Transition muss die Transition über die gesamte Schaltdauer schaltfähig sein und entnimmt die Marken der Vorplätze nach Ablauf der Schaltdauer. Dabei kann die Transition ihre Konzession während der Schaltdauer durch konkurrierende Transitionen verlieren.
Mit dem **Reservierungskonzept** werden reservierende Transitionen realisiert, durch die zeitbewertete Transitionen die Konzession vorbehalten werden kann.
Eine reservierende Transition verhält sich bei diesem Konzept, welches auch in [Dra99] bei

den Hybriden Dynamischen Netzen verwendet wird, wieder wie eine zeitverzögerte Transition, d.h. bei Aktivierung der reservierten Transition werden die betreffenden Marken sofort in den Vorplätzen reserviert und stehen dadurch für konkurrierende Transitionen nicht mehr zur Verfügung. Das Reservieren kann dabei als vorläufiger Abzug interpretiert werden.

4.2.3 Zeitbewertete Plätze

Zeitbewerteten Plätzen beschreibt die Zeit, die eine ankommende Marke mindestens in dem entsprechenden Platz verharren muss, bis sie wieder zur Verfügung steht. Dieses Verhalten äquivalent Verhalten der zeitverzögerter Transitionen[Lic04].

4.2.4 Zeitbewertete Kanten

Zeitbewertete Vorkanten
Prinzipiell können zeitlich Prekanten begrenzt oder zeitlich verzögert (retardiert) bewertet werden. Deshalb ordnet man begrenzendes und verzögerndes Verhalten einer Kante zu. Das erfolgt durch eine Reihe von Möglichkeiten der Markendurchlässigkeit. Dabei verbleiben die Marken stets auf den Plätzen und sind für andere Transitionen bzw. konkurrierende Prekanten jederzeit verfügbar [Kön, Quä88].
Zeitstempelnetze
Durch die Überlagerung von Zeitbewertungen lassen sich Intervalle der Durchlässigkeit konstruieren, deren Länge und Anordnung auf der Zeitachse beliebig wählbar sind. In Zeitstempelnetzen sind durch diese Zeitfenster Marken im Vorplatz einer Kante nur für ein bestimmtes Zeitintervall verfügbar. Jede Marke in einem Zeitstempelnetz trägt einen Zeitstempel, der den Zeitpunkt ihres Entstehens auf dem jeweiligen Platz repräsentiert [Han,Lau98] In einem Zeitstempelnetz ist es deshalb möglich, dass eine Transition mit zwei oder mehreren Prekanten in ihren Vorplätzen ausreichend mit Marken versorgt ist, aber trotzdem nicht schalten kann, weil sich eine der Prekanten aufgrund der Zeit schon unwiderruflich geschlossen hat.
Zeitbewertete Postkanten
Die zeitliche Bewertung einer Postkante eignet sich dazu, den Markenfluss nach dem Schalten einer Transition zu verzögern. Wenn eine Transition schaltet, werden die Marken sofort aus denVorplätzen entfernt und entsprechend der Zeitbewertung werden die Nachplätze verzögert markiert[Lic04].

4.2.5 Intervall-Petri-Netze

Zeitbehaftete Petrinetze wurden eingeführt, um die Zeit, ohne die keine vollständige Anforderungsanalyse und Verifikation eines Systems möglich ist, in die klassischen Petrinetze zu integrieren. Es gibt zwei Grundarten von zeitbehafteten Petrinetzen. Einerseits kann einer Transition eine Zeit, die sie zum Feuern benötigt, zugeordnet werden. Andererseits kann jeder Transition ein Zeitintervall zugeordnet werden, innerhalb dessen die Transition feuern kann. Dies ist bei den hier verwendeten Intervall-Petrinetzen der Fall. Eine schaltfähige Transition darf nur innerhalb dieses Intervalls schalten und muss spätestens nach Intervallende geschaltet werden, es sei denn, sie verliert in der Zwischenzeit ihre Schaltfähigkeit. Wenn die Transition später ihre Schaltfähigkeit zurück erlangt, beginnt der Ablauf der im Intervall festgelegten Zeit von vorn. Das Intervall wird hierzu durch ein Tupel (a,b) von rationalen Zahlen, mit $0 \leq a \leq b \leq \infty$, $a < \infty$ dargestellt.

Eine weitere Möglichkeit, Zeitbewertungen vorzunehmen, bieten Schaltintervalle. Dieses Petrinetz-Modell wurde erstmals in [Mer74] vorgeschlagen. Bei den Intervall-Petrinetzen

(IPN) kann jeder Transition ein Schaltintervall [eft, lft] zugeordnet werden. Dabei beschreibt die Zahl eft (earliest firing time) die früheste Schaltzeit und die Zahl lft (latest firing time) die späteste, d.h. das Schalten dieser Transition erfolgt frühestens eft Zeiteinheiten und spätestens lft Zeiteinheiten nach Erhalten der Konzession. Wenn der Transition im Intervall [eft, lft] die Konzession entzogen wird, erfolgt kein Schalten. Die in [Hei, PoP97] vorgestellten Duration-Intervall-Petri-Nets (DIPN) verwenden zeitverzögerte Transitionen, denen ein Schaltdauerintervall zugeordnet werden kann.
Zusammenfassend ist festzustellen, dass zur Modellierung von Zeitaspekten mit Petri-Netzen eine Reihe interessanter Ansätze zur Verfügung steht. Besonders die Verwendung von Intervallen ist sehr sinnvoll, Einen Überblick über Konzept Zeit und Petrinetze und Zeit-Petrinetze gibt [Dia09].

4.3 Petrinetz Tools

Petri-Netze können in den unterschiedlichsten Bereichen eingesetzt werden. Deshalb sind, durch vorhandene Spezialisierungen und Anpassungen, die benötigten Tools an den Anwendungsbereich gebunden.
Petrinetze sind ein Werkzeug zur Modellierung von neben läufigen und verteilten Systemen. Sie ermöglichen sowohl eine anschaulich-grafische Darstellung als auch eine formale mathematische Definition und Analyse des Systems. Außerdem gibt es einige ausreichende Computer Tools, die verschiedene Arten und Klassen von Petrinetzen unterstützten. Im Internet [uni10] steht eine Reihe mit den Namen für existierend Petrinetze Tools begleitet mit Links zu den Seiten dieser Programme, mit denen eine Petrinetz-Modellierung bzw. Auswertung möglich ist.
Aufgrund von fortlaufenden Änderungen im Funktionsumfang und Komplexität der genannten Tools ist eine genaue Beschreibung schwer möglich und nicht Inhalt dieser Arbeit.
Beispiele davon sind **Netedit** (W. Nauber, TU Dresden), **INA** (P. Starke, Humboldt-Universität zu Berlin) **Peneca Chromos** (W. Fengler, TU-Ilmenau) und **DESIGN/CPN** Tool beinhaltet drei Teile: einen CPN-Editor (Coloured Petri Net) zum erstellen von CPN's, einen CPN-Simulator und eine grafische Auswertung. Hiermit könne hierarchische, farbige Netze erstellt werden.
Diesbezüglich Zeit gibt es ein **Analyseprogramm**, das an der Universität Koblenz entwickelt wird [POS03], und in [Han,Lau98] wird gezeigt, wie Zeitstempelnetze analysiert werden können. Dafür steht mit dem Integrierten Netz Analysator **INA** [Roc,Str01] ein leistungsfähiges und frei verfügbares Analyseprogramm zur Verfügung.Timed INA **(TINA)** existiert aber nur für schwarze Petrinetze.

Wichtige Überlegungen für Auswahl ein Petrinetz Tools

1. Netztyp (schwarz oder gefärbt)
2. Welche Leistungen soll das Tool haben und welche Aufgaben kann Tool leisten.
3. Auf welcher Plattform soll das Tool betrieben werden (Unix-, Windows,..).
4. Anwendungsweise und Freundlichkeit der Benutzerschnittestelle (User Interface).
5. Zuverlässigkeit und Verfügbarkeit
6. Möglichkeit der Anbindung an weiteren Programm bzw. Zusatzprogramme

In dieser Arbeit werden zwei Tools ausgewählt, Peneca Chromos (**P**etri **N**et **C**omputer **A**ided Software Engineering) und INA (**I**ntegrierter **N**etz **A**nalysator Version 2.1). Die beiden Tools werden zum Simulieren und Analysieren sowie zur Verifikation des erweiterten CPN-ST genutzt. Das wird im Kapitel 6 detailliert beschrieben. Deshalb wird in diesem Kapitel nur ein

kurzer Überblick über die beiden Tools gegeben. **Peneca Chromos (2.2beta):**
PENECA Chromos“ ist ein Entwurfssystem mit dem eine grafische Modellierung eines **gefärbten Petrinetzes** möglich ist, sowie dessen Simulation, Verifikation und statistische Auswertung. Peneca Chromos 2.2beta für Windows wird durch Doppelklicken des entsprechenden Icons aus ihrem Startmenü gestartet, hat eine freundliche Benutzerschnittestelle und ist akademisches Werkzeug. Es gibt ein Handbuch für die Anwendungsweise [Tui97].
PENECA Chromos besitzt die folgenden wesentlichen Systemkomponenten:

Die Benutzeroberfläche dient der graphischen Editieren von Netzen, zur Simulation und Abwicklung entworfener Steuerstrukturen, einschließlich Steuerung und Protokollierung der Simulation, und ermöglicht den Aufruf von benötigten Dienstfunktionen. Sie enthält eine umfangreiche Hilfe-Unterstützung.

Als Analysekomponente kann der Analysator INA (**I**ntegrierter **N**etz **A**nalysator) der Humboldt-Universität Berlin genutzt werden. Mit dessen Hilfe können Petrinetze verschiedenster Typen unter verschiedenen Schaltregeln, insbesondere unter Zeitbewertung, hinsichtlich allgemeiner Eigenschaften untersucht werden. Typische Eigenschaften, die durch die Analyse nachgewiesen werden, sind Beschränktheit von Plätzen, Lebendigkeit von Transitionen, Erreichbarkeit bzw. Nichterreichbarkeit von Markierungen bzw. Zuständen.

Folgende Ergänzungen und Erweiterungen des Systems konnten zum Teil bereits realisiert werden, zum Teil wurden sie konzeptionell erarbeitet:
1. eine Implementationskomponente ermöglicht die Compilation von PN-Strukturen in C++.
2. eine Kopplung des Tools mit der Fuzzy-Erweiterung OFCTcl (**O**bject **F**uzzy **C**ontrol) in der Scriptsprache Tcl wurde notwendig, um die mit Fuzzy-Logik bewerteten Modelle zu simulieren und zu testen. Dazu war die Integration von OPNTcl *(**O**bject **P**etri **N**et)* in PENECA Chromos notwendig, um die Kommunikation zwischen dem Petri-Netz-Simulator und denkbaren Erweiterungen in Tcl zu ermöglichen. Die Kommunikation erfolgt über das cne (**c**oloured **n**et**e**xchange)-Dateiformat für gefärbte Petrinetze.
3. mit einer Statistikkomponente können Petri-Netz-Simulationsläufe ausgewertet und statistische Kenngrößen berechnet werden.

INA :
Das Programm INA (Integrated Net Analyser) von der Humboldt-Universität Berlin ist ein sehr leistungsfähiges Werkzeug zur Durchführung umfangreicher formaler Analysen an PN.
Es werden ungefärbte und gefärbte Netze unterstützt, jedoch **keine Sonderkanten** und nur ein Teil der behandelten Platz- und Transitionseigenschaften.
Die Bedienung erfolgt ausschließlich über eine Textkonsole und ist gewöhnungsbedürftig.
In INA können für zeitbewertete Kanten nur Intervalle angegeben werden, was die Mächtigkeit der Kombination von limitierenden und retardierenden Kantenbewertungen einschränkt.
Leistung des INA :

1. Editor: Mit INA kann ein Netz editiert werden, was wird mit einem Menü und Fragen nach dem Netztyp durchgeführt wird. Aber gibt es die Möglichkeit, ein Netz aus einer Datei zu lesen, wenn nur das Analysieren des Netzes angefordert wird .
2. Schalten mit Zeit und ohne Zeit: Netzverhalten und -Elemente werden durch Erreichbarkeitsgraphen dargestellt .
3. Analysieren: Statischen und Dynamisch, Eigenschaften des Netzes und Invariantenanalyse.

Beispiel eines Menüs von INA:

Statische Analyse
Die statische Analyse liefert Eigenschaften, die bereits aus der Struktur des Petri-Netzes bestimmt werden können. Zuvor werden wieder einige Optionen angezeigt:

```
Computing information on elementary structural properties
...
Current name options are:
transition names to be written
transition colours to be written
place names to be written
place colours to be written
.....................................Reset options? Y/N
```

Die Antwort „Y“ führt zu einem Menü, welches die Änderung der Optionen erlaubt. Mit „N“ gelangen Sie zur nächsten Frage:

```
.........................Print the static conflicts? Y/N
```

die Sie mit „Y“ beantworten. Anschließend laufen bereits die Ergebnisse der statischen Analyse durch. Die Informationen werden automatisch im „Session-File“ gesammelt (siehe unten). Nach ein oder mehreren Tastendrücken erscheint das nächste Menü.

Nach dieser Zusammenfassung über Petrinetze und ihre Tools wird später in folgenden Kapiteln erläutert, wie und warum diesen Tools benutzt wurden, um entwickelte Netz zu untersuchen.

Kapitel 5 Erweiterte Petrinetze

Die allgemeinen Forderungen nach Eindeutigkeit, Erhöhung der Zuverlässigkeit und Produktivitätssteigerung sowie die steigende Komplexität von Kommunikationssoftware, Protokollen und Algorithmen verlangen Entwicklungsmethoden, die eine einheitliche, unmissverständliche, formale und präzise Darstellung erlauben. In diesem Kapitel wird eine formale Methode auf Basis gefärbter Petrinetze entwickelt, die die Grundlage der entwickelten Methode dargestellt. Als konkretes Beispiel wird ein Beispielalgorithmus aus einer Gruppe von Algorithmen, die in P2P-Netzen (Peer-to-Peer-Netzen) arbeiten und mit Hilfe so genannter Wanderer funktionieren, mit der entwickelten Methode auf Basis der gefärbten Petrinetze modelliert.

5.1 Definition des Netzes

Die bekanntesten Netzklassen mit unterscheidbaren Marken aus der Gruppe der High-Level-Petrinetze sind die gefärbten Petrinetze (engl. Coloured Petri Net, CPN) [Jen92][Fen93] und die Prädikat/Transitions-Netze (Pr/T-Netze) [Gen87][Gen,Lau81].
Außerdem gibt es zahlreiche Erweiterungen von Petrinetzen mit unterschiedlicher Modellierungsmächtigkeit und unterschiedlichem Niveau der Analysierbarkeit, z.B. die "Hierarchical Coloured Stochastic Deterministic Petrinets" (HCSDPN)[Kno95].
In diesem Kapitel wird ein erweitertes Petrinetz mit unterscheidbaren Marken definiert, welches notwendig ist, um die Struktur des Wanderers zu modellieren. Ein Wanderer ist wie oben erwähnt ein Baustein einer Gruppe von Algorithmen, die als Grundlage für die Entwicklung der Modellierungsmethode benutzt wurden.
Auf Basis der CPN, der Well-formed Nets (WN) [CHI 91, CHI93] und der HCSDPN werden in diesem Kapitel "High Level Coloured Petrinets with Structured Tokens" (HCPN-ST) unter Einbeziehung von prädikatenlogischen Anschriften (mit Formeln) eingeführt.
Die Definition der HCPN-ST erfolgt in mathematischen Begriffen und baut auf wesentliche Beschreibungsvorteile der Pr/T-Netze [Gen,Lau81] auf. Im Gegensatz zu diesen werden hier Formeln mit Prädikatenlogik als Kantenanschriften definiert.
Diese HCPN-ST unterscheiden sich von den CPN durch die prädikatenlogischen Kantenausdruck-Funktionen, die lokal an Transitionen definiert sind. Wie in anderen Typen von Petrinetzen gibt es in HCPN-ST u.a. eine Menge von Plätzen und eine Menge von Kanten. Jeder Platz kann mehrere Marken enthalten und jede von diesen trägt eine Datenstruktur, die eine komplexe Form hat oder auch ein Produkt aus verschiedenen Typen sein kann (wie auch in[CHI 91, CHI 93][Kno95]). Durch eine Kantenanschrift [Sta90], die in der Form einer Funktion definiert werden kann, wird die Schaltregel beeinflusst. Diese Funktion kann auch ein logischer Ausdruck sein [Rok92]. Die Transitionen sind schaltfähig, wenn die Vorbedingungen und Nachbedingungen gleichzeitig erfüllt sind. In den nachfolgenden Abschnitten werden alle Details erläutert.

Definition 5.1 Farbset

Es gibt zwei Farbtypen:

1. Elementares Farbset:

X := $\{x \mid x$ ist eine endliche natürliche Zahl größer als 2$\}$ entspricht der Anzahl der Elemente eines Plans und wird Länge des Plans genannt.

I := {1,2,...,m-1,m} eine endliche natürliche geordnete Indexmenge

Z := $\{z \mid z \in \mathbb{N}, 0 \le z < x\}$ Zeiger auf die Position i des Plans, ist immer kleiner oder gleich der Anzahl der Elemente eines Plans.

Knotenadressen := {n1, n2, n3,.....ni}, endliche geordnete Menge von Knotenadressen im Netzwerk mit $i \in I$

Knotenanzahl: = |Knotenadressen|

Leerposition := {e}, ein Symbol für einen leeren Platz in einem Plan.

Randomsendung := {r}, ein Symbol für einen Platz in einem Plan, an dem die Nachricht an einen zufälligen Empfänger geschickt wird.

A:={a| $a \in (Knotenadressen \cup Leerposition \cup Randomsendung)$ }, eine Position eines Plans ist entweder eine Knotenadresse, eine Leerposition oder eine Randomsendung.

Alle Elemente des elementaren Farbsets werden im Kontext dieser Arbeit als Farbmarke (fm) bezeichnet. Das heißt, dass eine Farbmarke fm eine Knotenadresse oder ein Zeiger oder die Länge eines Plans ist.

2. **Strukturiertes Farbset:** ist ein Farbset, das durch ein kartesisches Produkt aufgebaut wird. Die Menge kann auch Elemente enthalten, die wieder Mengen sind. Alle Elemente des strukturierten Farbsets werden im Kontext dieser Arbeit als Verbundfarbmarke (vm) [Kno95] bezeichnet.

   $Plan := \underbrace{A \times A \times \ldots A}_{i \ mal} = \{(a_1, a_2, \ldots, a_i) \mid a_i \in A, i \in I\}$, hier ist klar, dass die Länge des Plans gleich (i) ist. Das heißt, der Plan i hat eine Farbmarke. Dieses kann man wie folgt schreiben: $Plan := A^{|I|} = \{(a_1, a_2, \ldots a_i) \mid a_i \in A, i \in I\}$. Jedes Element der endlichen Menge 'Plan' ist ein i-Tupel und beschreibt gleichzeitig den Plan einer zirkulierenden Nachricht. (Dazu wird a ohne einen Index i als Wert (Knotenadresse, e, r) verstanden, und a mit Index i (also a_i) ist die Position Nr. i in einem Plan.)

   Wanderer := Z× X × Plan =

   $$= \{(z, x, (a_1, a_2, \ldots, a_i)) \mid z \in Z \wedge x \in X \wedge (a_1, a_2, \ldots, a_x) \in Plan \wedge \ z \leq x \ \wedge i \in I\}$$

   Ein Verbundtyp beschreibt eine Datenstruktur, wobei z ein Zeiger (natürliche Zahl) ist, der auf die aktuelle Knotenadresse zeigt, an welcher der Wanderer bearbeitet wird.

   X: Länge des Plans, eine natürliche endliche Zahl.

   $(a_1, a_2, \ldots, a_i)$ ein Plan mit i Positionen und z = i mod x. Jede Instanz eines Wanderer-Farbsets wird als Verbundfarbtyp (wm) bezeichnet. Dieser Verbundfarbtyp ist als semantisches Abbild der Zustände des Kommunikationsalgorithmus spezifizierbar.

Die Definition des Farbsets lässt eine Vielzahl unterschiedlicher Markentypen zu.

Diese Vielzahl ist nützlich, um Nachrichtentypen von Marken, die für die Modellierung des Verhaltens der Algorithmen benötigt werden, zu unterschieden.

Definition 5.2 HCPN-ST

Das Tupel HCPN-ST = (Farbset, P, T, F, C, V, K, $m_{0,}$) sei ein gefärbtes Petrinetz mit:

1. Farbset: Menge von Farben, nicht leere Menge, genannt Farbset; siehe Definition 5.1
2. P, T endliche nicht leere Mengen von Plätzen und Transitionen
3. $P \cap T = \emptyset$ (disjunkte Mengen)
4. $P \cup T = X \neq \emptyset$
5. F Menge aller Kanten, wobei:
   $F \subseteq (P \times T) \cup (T \times P)$ $Vorkanten(F_v) \subseteq (P \times T)$, $Nachkanten(F_n) \subseteq (T \times P)$
6. Für alle $t \in T$ gilt: tF: Menge der Nachplätze von t

Ft: Menge der Vorplätze von t

7. C: eine Abbildung, die jedem Knoten $x \in (P \cup T)$ eine endliche nicht leere Menge C (x) von Farben zuordnet.
8. V eine Abbildung, die jeder Kante $f \in F$ eine Abbildung V(f) von C(t) in die Menge aller Multimengen über c_p zuordnet, wobei t die Transition und P der Platz an der Kante und f eine Kantenanschrift ist, die in Form einer Funktion definiert sein kann. Diese Funktion kann ein logischer Ausdruck sein, der eine Verknüpfung Boolescher Funktoren enthält. Diese können in folgender Weise verknüpft sein: (2c1^ 1c3) ˇ 2c2 wobei 2 und 1 Kantengewichte sind und ^ und ˇ logische Verknüpfungen (and, or). Bei einer Nachkante ist ˇ (or) aufgrund ihrer schwierigen technischen Interpretierbarkeit nicht zugelassen[Kno95].
9. $((P \times T) \rightarrow Bool(c)) \cup ((T \times P) \rightarrow C_t(C))$
   V_v: $F_v \rightarrow Bool(C)$, F_v ist eine nicht leere Menge. Die Menge Bool (C) stellt hier die Menge von Booleschen Ausdrücken dar, die mit logischem AND (&), OR (|) und NOT (not) verknüpfbar und auch mit Klammern () zusammenfassbar sind.
10. V_n: $F_n \rightarrow Bool(C)$, F_n ist eine nicht leere Menge. Bool (C) sind hier Boolesche Ausdrücke (prädikatenlogische Formeln), die nur mit logischem AND verknüpft werden können. Die Kantenanschriften können kombinierte Anschriften sein. Diese können als Operanden bei Vorkanten bzw. in Termen bei Nachkanten, als Konstanten oder Variablen benutzt werden. Vorkanten und Nachkanten müssen entweder mit kombinierten Anschriften oder mit Funktionsanschriften bezeichnet werden, siehe Abschnitt 5.2.
11. K: Kapazitätsfunktion: $(P \times C) \rightarrow \mathbb{N}$ Abbildung, die jeder Farbe eines Platzes eine Kapazität zuordnet. Es gilt auch: $\forall (p \in P, c \notin C(p), c \in C) \quad K(p_c) = 0$
12. m_0 (Anfangsmarkierung): $(P \times C) \rightarrow \mathbb{N}$, Abbildung, die jeder Farbe eines Platzes eine Markierung zuordnet. Es gilt : $\forall p \in P, \forall c \in C(p): m_0(p_c) \leq K(p_c)$.
13. dt: $(t \times Farbset) \rightarrow \mathbb{N}$, Abbildung, die jeder Farbe einer Transition $t \in T$ einen Zeitverbrauch zuordnet.

Schaltregel in **HCPN-ST**

Die Schaltregel in HCPN-ST verwendet die allgemeine Definition aus [Sta90] für P/T Netze mit farbabhängigen Plätzen und Transitionen. Die nächste Definition beschreibt die Schaltregel für eine Transition unter Berücksichtigung der möglichen Erweiterungen.

Definition 5.3 Schaltregel :

Eine Transition $t \in T$ hat Konzession in der Farbe $c \in C(t)$ bei der Markierung m wenn gilt: $\forall C \in C(P), \forall c \in C(t) \quad V(P_c, t_c) \leq m(p_c) \quad \forall f \in F_v$. Wenn (t_c) Konzession bei m hat, darf t in c bei m schalten und aus m entsteht $\grave{m}$ mit:

$$\grave{m}(p_c) = \begin{cases} m(p_c) - V_v(p_c, t_c), & \forall p \in Ft \wedge p \notin tF \\ m(p_c) + V_n(p_c, t_c), & \forall p \in tF \wedge p \notin Ft \\ m(p_c) - V_v(p_c, t_c) + V_n(p_c, t_c), & \forall p \in Ft \wedge p \in tF \\ m(p_c), & \forall p \notin tF \wedge p \notin Ft \end{cases}$$

In den nachfolgenden Abschnitten werden Erweiterungen wie Farbkonvertierungsmechanismen und Sonderkanten sowie ihr Einfluss auf die Schaltregel untersucht.

Lokale Farbkonvertierungsvariablen an Transitionen

Die Kommunikationsaktivitäten werden durch Wanderer dargestellt, die verschiedene Verbundfarbtypen benötigen (wie Wanderer, Plan, Nachrichtentypen). Diese Wanderer haben einen fest determinierten Ablauf, wenn ihr Plan gefüllt ist und keine Operationen mehr auf dem Wanderer ausgeführt werden können. Dieser Ablauf kann sehr gut durch eine farbabhängige Transitionsschaltfolge mit festgelegter Farbumwandlung modelliert werden. In HCPN-ST bleibt diese Farbkonvertierung ausschließlich der Transition als aktivem Netzelement vorbehalten. Aus diesem Grund werden Farbkonvertierungsvariablen eingeführt, die nur in der lokalen Umgebung einer Transition Gültigkeit haben, wobei als Umgebung einer Transition die Vorplätze und Nachplätze sowie die Vorkanten und Nachkanten verstanden werden [Rokyta92][Kno95].
Die Farbkonvertierung wird durch eine Abbildung von Farben des Vorbereiches auf Farben des Nachbereiches mittels einer Konvertierungstabelle realisiert, wobei die Vorkantenanschriften und Nachkantenanschriften Variablen sind.

Farbmenge und Operation auf einer Farbmenge :

- Mit den **Farbkonvertierungsvariablen** wurde ein Mechanismus eingeführt, der mit Blick auf Analysemethoden eine theoretisch eindeutige Konvertierung von Markenfarben zulässt.
- Die Art und Weise der **Farbdefinition** hat erheblichen Einfluss einerseits auf die Beschreibungsmächtigkeit des definierten Netztyps und andererseits auch auf die Analysierbarkeit der beschrifteten Netzstruktur [Jen92][Sta90].
- Jensen definierte für Transitionen die Abbildung einer sogenannten Guard-Funktion wie folgt: „The **Guard function** maps each transition, t, to an expression of type boolean, i.e. a predicate.“. In HCPN-ST wird die Guard-Funktion wie folgt definiert:
  {f (t): Vv → Vn Es gibt eine Liste von Daten in einem Platz, die mit Farbmarken und Verbundfarbmarken dargestellt wird. Die Vorkantenanschrift Vv überprüft die Existenz von Marken in dem Platz P. Wenn diese Marke, die bestimmte Bedingungen erfüllen muss, existiert, schaltet die Transition t und diese Marke wird dem Platz P entgezogen. Danach werden entsprechend der Nachkantenanschrift Marken dem Nachplatz hinzugefügt, oder durch die Nachkantenfunktion werden neue Marken erzeugt, die zum Farbset des Nachplatzes gehören. }
- Die **Farbmengen** werden entsprechend Definition 5.1 deklariert. **Beispiel:**
  vm = (3,6,(n1,n4,n6,e,n2,r)). Dadurch ergibt sich der Farbraum des Farbsets aus der Menge aller deklarierten Farbmengen.
- Während Jensen die Farben eines Platzes lokal definiert, sind sie in HCPN-ST global verfügbar.
- Deklaration der **Farbmengenvariablen**: Die Deklaration von Farbmengenvariablen vermeidet es, lange Kantenbeschriftungen zu schreiben. Durch eine Variablendeklaration wird eine Variable basierend auf schon deklarierten Farbmengentypen eingeführt:
  Farbset Index: j,i ; j,i Index einer Position eines Plans.
  Farbset Zeiger: x ; x gibt die Nummer einer Position der Liste der Positionen an, die aktuell darauf zeigt.
  Farbset Länge: num ; num Anzahl der Positionen eines Plans.
  Farbset A : fm ; fm Werte einer Position eines Plans.
  : pos_i ; pos_i eine Position
  Farbset Knotenadresse: n
  Farbset Plan : vm ; vm Plan eines Wanderers.
  Farbset Wanderer: wm ; wm ein Wanderer

- **Operationen auf Farbmengen** werden in dem folgenden Abschnitt 5.2 erläutert.
- **Filteroperationen auf Verbundfarbtypen:**
  Durch diese Operationen wird eine besonders effektive Spezifikation von Schaltbedingungen für Transitionen erlaubt [Kno95], die nur von Elementen einzelner oder mehrerer Unterfarbmengen eines Verbundfarbtyps abhängt und nicht von dem gesamten Verbundfarbtyp. Das hat eine große Bedeutung für die Darstellung von Protokollen, Algorithmen und Nachrichten in einem Netz. Die folgende Abbildung (5.1) erläutert dieses Prinzip. Für die Verbundfarbtypvariable vm vom Typ Plan soll die Entscheidung der Schaltfähigkeit der Transition t1 nur von Position Nr.2 des Plans abhängig sein. Das bedeutet, t1 schaltet für alle Verbundmarken, die n3 in der Position 2 enthalten.

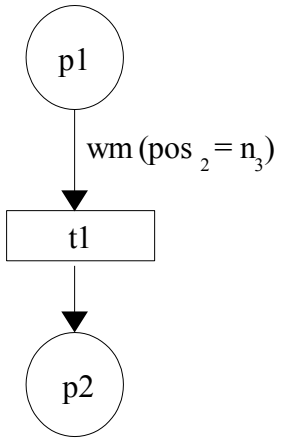


Abbildung 5.1: Farbtypselektion in Verbundfarbtypen

In dem folgenden Abschnitt werden eingehende Erweiterungen von Vorkantenanschriften und Nachkantenanschriften definiert und ihr Einfluss auf die Schaltregel für Normalkanten bzw. Sonderkanten untersucht. Diese Anschriften werden als Vorbedingung und Nachbedingung bezeichnet.

5.2 Kantenanschriftenfunktionen(Vorbedingungen und Nachbedingungen)

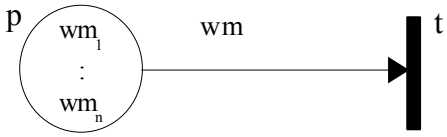
Deklaration der Farbmarkenvariablen

Farbset Index: j, i, m
Farbset Zeiger: z
Farbset Länge: x
Farbset A: fm Werte einer Position eines Plans.
pos $_i$ die Position Nr. i eines Plans.
Farbset Knotenadresse: n
Farbset Plan: vm Plan eines Wanderers
Farbset Wanderer: wm, wm_1, wm_2 strukturiertes wm (z, x, vm)

Normale Kanten

- **Vorkantenanschriften (Vorbedingungen)** $V(p_c, t_c) \leq m(p_c)$

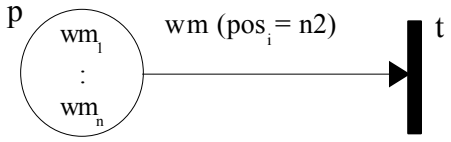
  1) V_v = wm $\forall wm \in Wanderer \exists wm : (m(p_{wm}) \geq 1)$



Schaltfähig: Es existiert im Platz P die Verbundmarke wm mindestens einmal.
Nach dem Schalten von t wird 1* wm entnommen.

2) wm (pos_i = n2)

$$\forall wm \in Wanderer \exists wm \in m(p): wm(pos_i = n2)$$

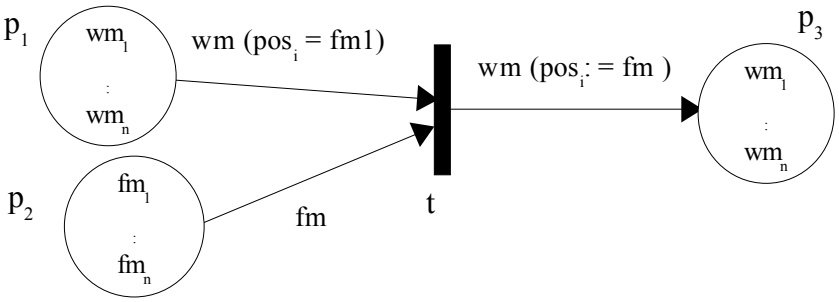


Schaltfähig: Es existiert mindestens eine Verbundmarke wm in m(p), die an der Position i des Plans der wm die Knotenadresse n2 enthält .
Nach dem Schalten von t wird den wm Verbundmarken, deren Position i die Knotenadresse n2 enthält, 1* wm entzogen.

3) wm(pos_i = fm1) wobei fm typischerweise eine Vorbedingung an der gleichen Kante oder einer anderen Vorkante der gleichen Transition ist**.**

$$(\forall wm \in Wanderer \wedge fm1 \in A \exists wm \in m(p_1): wm(pos_i = fm1))$$

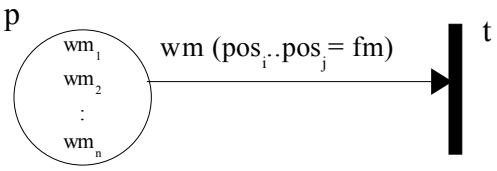
and $$\forall fm \in A \exists fm \in m(P_2): wm(pos_i := fm)$$



Schaltfähig: es existiert die Verbundmarke wm in $m(p_1)$, in der es möglich ist, ihre Position posi durch fm zu ersetzen.
Nach dem Schalten von t: es wird 1* wm entnommen, wobei die o.g. Bedingung gilt. Weiterhin gilt $wm(pos_i := fm)$, wobei fm eine Farbmarke ist, die dem Platz p_2 entzogen wird (1*fm) und die Position i von wm einnimmt.

4) wm (pos_i,…, pos_j = fm)

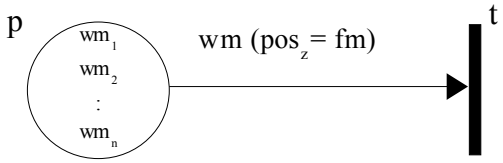
$$\forall wm \in Wanderer \exists (wm \in m(p) \wedge fm \in A \wedge x\ zeiger\ wm) : wm(pos_m = fm)\ \ für\ i \leq m \leq j \leq x \wedge m \in I$$



Schaltfähig: es existiert eine Verbundmarke wm in m (p) und auf den Positionen von $pos_i, \ldots, pos_j$ existiert mindestens einmal die Farbmarke fm.
Nach dem Schalten von t: es wird 1* wm entnommen, wobei diese mindestens einmal die Farbmarke fm in den Positionen zwischen i und j enthält.

5) wm(pos_zeiger = fm)

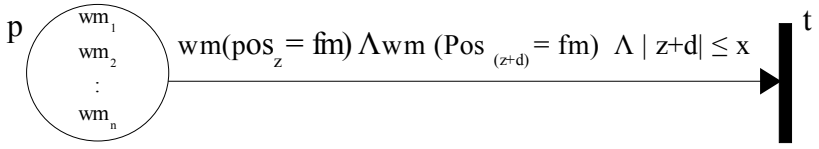
$$\forall\, wm \in Wanderer\, \exists (wm \in m(p) \wedge z\ zeiger(wm)) : wm(pos_z = fm)$$



Schaltfähig: es existiert eine Verbundmarke wm in m(p), wobei die aktuellen Position ihres Zeigers z fm enthält.
Nach dem Schalten von t: es wird 1* wm entnommen, wobei die o.g. Bedingung gilt.

6) wm (pos_Zeiger, ...pos_(Zeiger + d) = fm), |d| ≤ x-Zeiger

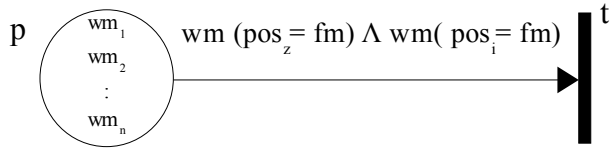
$$\forall\, wm \in Wanderer\, \exists\, wm \in m(p)\, und\ z\ Zeiger(wm), x\ lange\ wm$$
$$: wm(pos_z = fm) \wedge wm(pos_{(z+d)} = fm) \wedge 1 \le z + d \le x$$



Schaltfähig: es existiert eine Verbundmarke wm, und auf den Positionen in wm, auf die der Inhalt des Zeigers z und der Inhalt des Zeigers (z)+ im Abstand d (evtl. auch negativ) zeigen, existiert die Farbmarke fm.
Nach dem Schalten von t: es wird 1* wm entnommen, wobei die o.g. Bedingung gilt.

7) wm(pos_zeiger = fm Λ pos_i = fm)

$$\forall\, wm \in Wanderer\, \exists\, wm \in m(p):$$
$$(i\ index \in wm) \wedge (z\ \ Zeiger\ der\ wm) \wedge (pos_z = fm) \wedge wm(pos_i = fm)$$



Schaltfähig : es existiert eine Verbundmarke wm und auf Position pos i in wm existiert die

Farbmarke derjenigen Position, auf die der Zeiger z zeigt.
Nach dem Schalten von t: es wird 1*wm entnommen, wobei die o.g. Bedingung gilt.

Vorbedingungen sind mit AND (&), OR (|) und NOT (not) logisch verknüpfbar und auch mit Klammern () zusammenfassbar.
Andere Kombinationen von Farbvariablen, Farbmarken, Verbundmarkenvariablen und Verbundmarken, als oben angegeben, sind möglich.

z.B. $wm(pos_i = fm_j \wedge pos_k = fm_m)$, $wm_j(pos_i = fm_k) \vee wm_m(not\ pos_n = fm_1)$

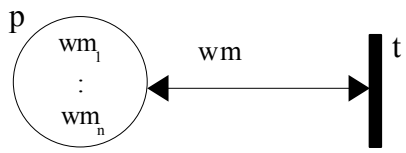
Vielfachheiten bei wm und fm sind möglich, keine Angabe bedeutet v=1 an der Kantenanschrift.

Vorkantenanschrift für Sonderkanten

- **Vorbedingungen (Test- und Inhibitorkanten)**

1a) V_v = wm

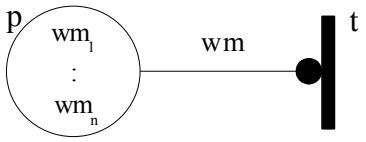
$\forall wm \in Wanderer \exists wm : (m(p_{wm}) \geq 1)$



Schaltfähig: Es existiert die Verbundmarke wm im Platz p mindestens einmal.
Nach dem Schalten: keine Auswirkung

1b) V_v = wm

$\forall wm \in Wanderer \exists wm : (m(p_{wm}) = 0)$



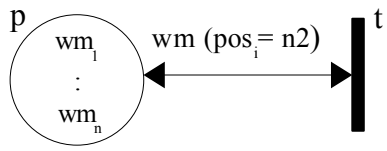
Schaltfähig: Es existiert keine Verbundmarke wm im Platz p.
Nach dem Schalten: keine Auswirkung

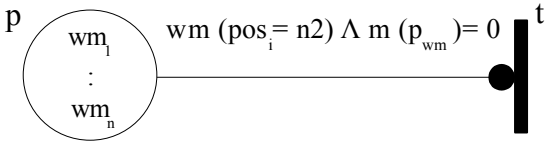
2a) wm (pos_i = n2) (Testkanten)

$\forall wm \in Wanderer \exists wm \in m(p) : vm(pos_i = n2)$

2b) not(wm (pos_i = n2)) (Inhibitorkanten)

2b) $\forall wm \in Wanderer \wedge wm(pos_i = n2) : m(p_{wm}) = 0$





Schaltfähig: es existiert eine Verbundmarke wm, deren Position i die Farbmarke n2 enthält.
Nach dem Schalten: keine Auswirkung

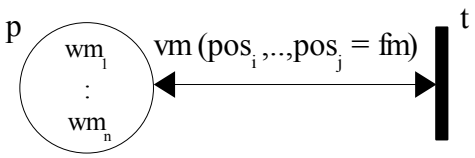
Schaltfähig: es existiert keine Verbundmarke wm, deren Position i die Farbmarke n2 enthält.
Nach dem Schalten: keine Auswirkung

3a) wm(pos_i ,..,**pos_j =fm)**

3b)not wm(pos_i ,..,pos_j = fm)

$\forall wm \in Wanderer \exists (wm \in m(p) \wedge fm \in A \wedge x\ zeiger\ wm)$
$: wm(pos_m = fm)\ \ für\ i \leqslant m \leqslant j \leqslant x \wedge m \in I$

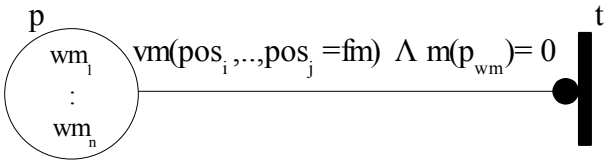
$$\forall\, wm \in Wanderer \wedge wm(\, pos_i = fm) : m(\, p_{vm}) = 0$$



Schaltfähig: es existiert eine Verbundmarke wm und auf den Positionen posi bis posj in wm existiert mindestens einmal die Farbmarke fm

Nach dem Schalten: keine Auswirkung

4a) wm(pos$_{zeiger}$ = fm)



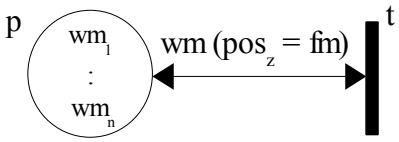
Schaltfähig: es existiert keine Verbundmarke wm, die auf einer der Positionen von posi bis posj eine Farbmarke fm enthält.

Nach dem Schalten: keine Auswirkung

4b) not wm(pos$_{zeiger}$ = fm)

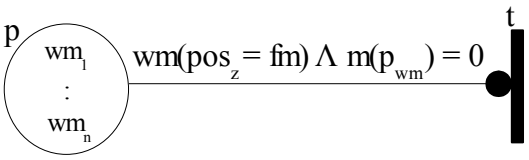
4a) $\forall\, wm \in Wanderer \exists\, wm \in m(\, p) : wm(\, pos_z = fm)$

4b) $\forall\, wm \in Wanderer \wedge wm(\, pos_z = fm) : m(\, p_{wm}) = 0$



Schaltfähig: es existiert eine Verbundmarke wm, die auf der Position ihres Zeiger z die Farbmarke fm enthält.

Nach dem Schalten: keine Auswirkung

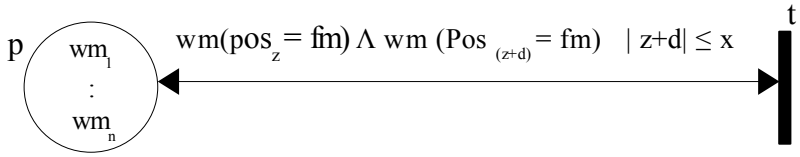


Schaltfähig: es existiert keine Verbundmarke wm, die auf auf der Position ihres Zeiger z die Farbmarke fm enthält.

Nach dem Schalten: keine Auswirkung

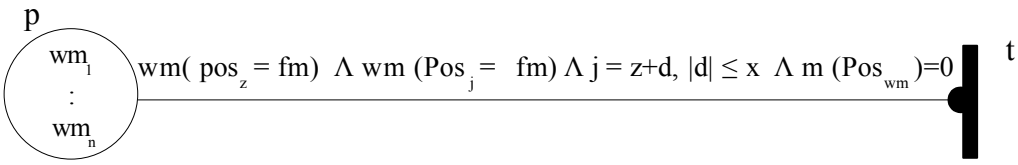
5a) wm (pos_Zeiger, ...pos_(Zeiger + d) = fm), |d| ≤ x-Zeiger

$$\forall\, wm \in Wanderer \exists\, wm \in m(\, p)\, und\, z\, Zeiger\, (wm),\, x\, lange\, wm$$
$$: wm(\, pos_z = fm) \wedge wm(\, pos_{(z+d)} = fm) \wedge 1 \le z + d \le x$$



5b) not wm (pos_Zeiger, ...pos_(Zeiger + d) = fm), |d| ≤ x-Zeiger

$$\forall\, wm \in Wanderer \exists\, wm \in m(\, p)\, und\, z\, Zeiger\, (wm),\, x\, lange\, wm$$
$$: wm(\, pos_z = fm) \wedge wm(\, pos_{(z+d)} = fm) \wedge 1 \le z + d \le x$$



Schaltfähig: es existiert eine Verbundmarke wm, und auf den Positionen in wm, auf die der Inhalt des Zeigers z und der Inhalt des Zeigers z+ Abstand d (evtl. auch negativ) zeigt, existiert die Farbmarke fm.
Nach dem Schalten: keine Auswirkung

Schaltfähig: es existiert keine Verbundmarke wm, die auf den Positionen in wm, auf die der Inhalt des Zeigers z und Inhalt des Zeigers z+ Abstand d (evtl. auch negativ) zeigt, die Farbmarke fm enthält.
Nach dem Schalten: keine Auswirkung

6a) wm (pos $_z$ = pos$_i$ =fm)

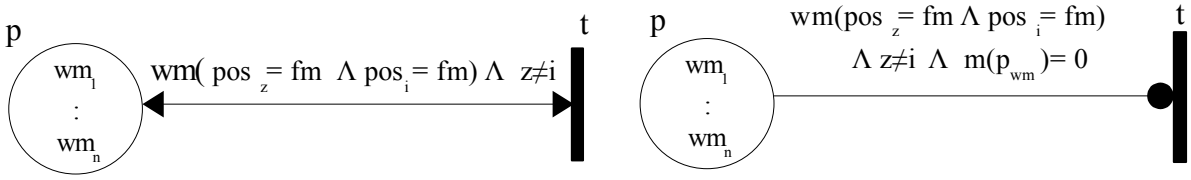
6b) not wm(pos $_z$ = pos$_i$ =fm)

6a)

$$\forall\, wm \in Wanderer\, \exists\, wm \in m(p):$$
$$(i\ index \in wm) \wedge (z\ Zeiger\ der\ wm) \wedge (pos_z = fm) \wedge wm(pos_i = fm)$$

6b)

$$\forall\, wm \in Wanderer\, \exists\, (wm(i\ index \in wm) \wedge (z\ Zeiger\ der\ wm) \wedge (pos_z = fm) \wedge wm(pos_i = fm)) : m(p_{wm}) = 0$$



Schaltfähig: es existiert eine Verbundmarke wm, und auf ihrer Position pos $_i$ existiert die Farbmarke fm, die auch an der Position z enthalten ist.
Nach dem Schalten: keine Auswirkung

Schaltfähig: es existiert eine Verbundmarke wm, und auf Position pos in wm existiert keine Farbmarke wie an der Position, auf die z zeigt.
Nach dem Schalten: keine Auswirkung

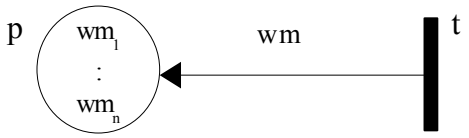
- **Nachkantenanschrift (Nachbedingungen):**

Eine Transition hat Konzession in der Farbe $c \in C(t)$ bei der Markierung m wie in [Starke90], [Knorr 94], wenn gilt :

$$\forall\, c \in C(p), \forall\, c \notin C(t) \quad V(p_c, t_c) \leq m(p_c), \quad \forall\, f \in F^+$$
$$m'(p_c) = m(p_c) + v(p_c, t_c) \qquad \forall\, P \in tF \wedge P \notin Ft$$

1) wm

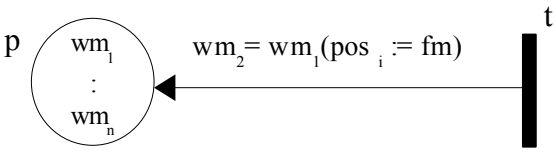
$$\forall\, wm \in Wanderer : m(p_{wm}) + 1 \leq k$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der Verbundmarke wm..
Nach dem Schalten: 1* wm hinzufügen, wobei die Zahl der Verbundmarke wm um 1 erhöht wird.

2) wm(pos $_i$:= fm)

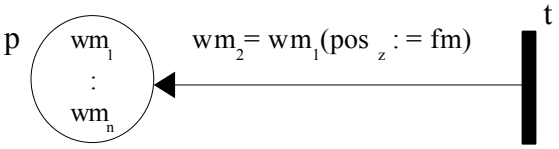
$$\forall wm_1 \in Wanderer \exists (wm_2 \in Wanderer : wm_2 = wm_1(pos_i := fm), i \in I) \wedge m(p_{wm2}) < k$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm_2.
Nach dem Schalten: 1* wm_2 hinzufügen, wobei in wm_1 die Position pos_i durch die Farbmarke fm belegt wird.

3) wm2(z : zeiger, pos $_z$:=fm)

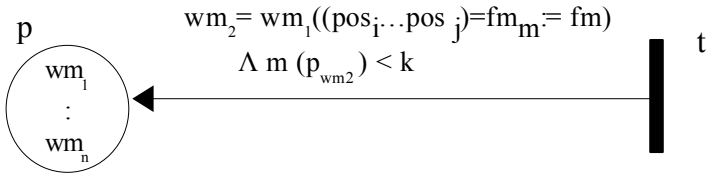
$$\forall wm_1 \in Wanderer \exists (wm_2 \in Wanderer : wm_2 = wm_1(pos_z := fm) \wedge m(p_{wm2}) < k)$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm_2.
Nach dem Schalten: 1* m_2 hinzufügen, wobei in wm_2 die Position pos_z, auf die der Zeiger z zeigt, durch die Farbmarke fm belegt wird.

4) wm$_2$((pos$_i$…pos $_j$)=fm$_m$:=fm)

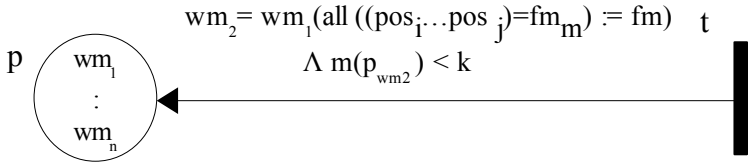
$$\forall wm_1 \in Wanderer \exists wm_2 \in Wanderer :$$
$$wm_2 = wm_1(((pos_i, \ldots, pos_j) = fm_m := fm)) \wedge m(p_{wm2}) < k, (i, j, m) \in I$$



Schaltfähig : Es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm_2.
Nach dem Schalten: 1* wm_2 hinzufügen, wobei in wm_1 die Position, die im Bereich pos $_i$ bis pos $_j$ die Farbmarke fm_m enthält, durch die Farbmarke fm belegt wird. Wenn mehrere Positionen möglich sind, ist das ein Nachkonflikt, der dadurch gelöst wird, dass nur die erste Position durch fm belegt wird.

5) wm((all(pos$_i$…pos $_j$)=fm$_m$):=fm)

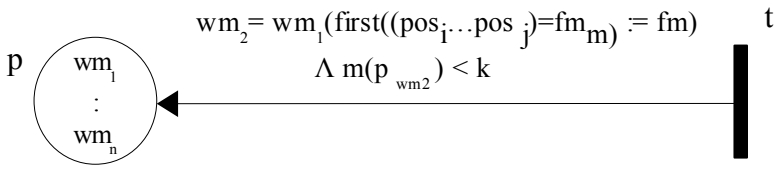
$$\forall wm_1 \in Wanderer \exists wm_2 \in Wanderer:$$
$$wm_2 = wm_1(all((pos_i, \ldots, pos_j) = fm_m := fm)) \wedge m(p_{wm2}) < k, (i, j, m) \in I$$



Schaltfähig: Es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm$_2$.
Nach dem Schalten: 1* wm$_2$ hinzufügen, wobei in wm alle Positionen im Bereich pos $_i$ bis pos $_j$, welche die Farbmarke fm $_m$ enthalten, durch die Farbmarke fm belegt werden.

6) wm$_2$ = wm $_1$ ((first(pos$_i$…pos $_j$)=fm$_m$):=fm)

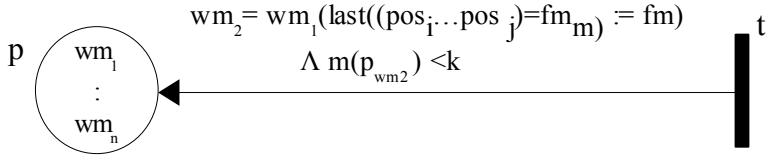
$$\forall wm_1 \in Wanderer \exists wm_2 \in Wanderer:$$
$$wm_2 = wm_1(first((pos_i, \ldots, pos_j) = fm_m) := fm) \wedge m(p_{wm2}) < k, (i, j, m) \in I$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm$_2$.
Nach dem Schalten: 1*wm$_2$ hinzufügen, wobei in wm$_1$ die erste Position im Bereich pos $_i$ bis pos $_j$, welche die Farbmarke fm $_m$ enthält, durch die Farbmarke fm belegt wird.

7) wm$_2$ = wm $_1$ ((last(pos$_i$…pos $_j$)=fm$_m$):=fm)

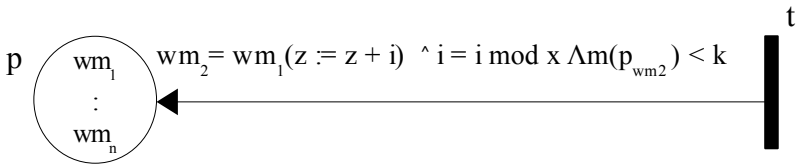
$$\forall wm_1 \in Wanderer \exists wm_2 \in Wanderer:$$
$$wm_2 = wm_1(last((pos_i, \ldots, pos_j) = fm_m) := fm) \wedge m(p_{wm2}) < k, (i, j, m) \in I$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm$_2$.
Nach dem Schalten: 1*wm$_2$ hinzufügen, wobei in wm$_2$ die letzte Position im Bereich pos $_i$ bis pos $_j$, welche die Farbmarke fm $_m$ enthält, durch die Farbmarke fm belegt wird.

8) wm$_2$ = wm $_1$ (z :=z + (i mod x))

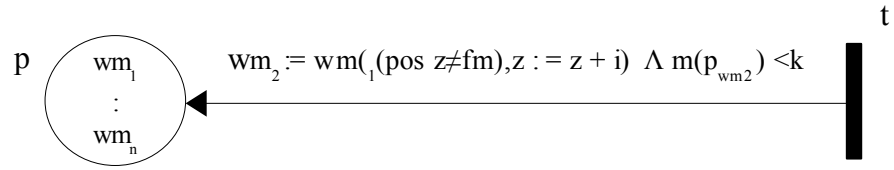
$$\forall wm_1 \in Wanderer \exists wm_2 \in Wanderer:$$
$$wm_2 = wm_1(z := z + (i \bmod x)) \wedge z \in Zeiger \wedge m(p_{wm2}) < k$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der neuen Verbundmarke wm_2. Der Zeiger wird in seinem Definitionsbereich um i modulo Länge des Definitionsbereichs erhöht. Negative i sind möglich.
Nach dem Schalten: 1*wm_2 hinzufügen, wobei der Zeiger in wm_2 durch die Farbmarke z:= (z + (i mod x)) belegt wird.

9) wm_2 := wm_1 (z :=z + i) wobei wm_1 (pos $_z \neq$ fm)

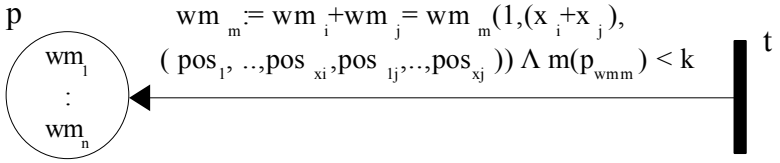
$$\forall\, wm_1 \in Wanderer \wedge wm_{(posz \neq fm)} \exists\, wm_2 \in Wanderer:$$
$$wm_2 = wm_1(z := z+i) \wedge m(p_{wm2}) < k$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der Verbundmarke wm_2. Der Zeiger wird in seinem Definitionsbereich um i modulo Länge des Definitionsbereichs erhöht. Dabei werden nur Positionen beim Erhöhen berücksichtigt, die nicht mit der Farbmarke fm belegt sind. Negative i sind möglich.
Nach dem Schalten: 1*wm2 hinzufügen, wobei wm2 eine Verbundmarke ist, die durch wm1(posz ≠ fm) erzeugt wird und deren Zeiger mit der Farbmarke (x+i) belegt wird.

10) $wm_m := wm_i + wm_j = wm_m (z_m, x_m, v_m)$

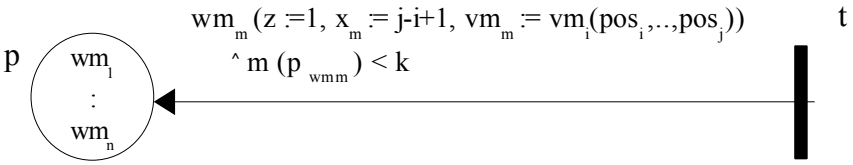
$$\forall\, wm_m \in Wanderer\, \exists (wm_i, wm_j) \in Wanderer : wm_m := wm_i + wm_j = wm_m(z_m, x_m, vm_m)\, wobei$$
$$(z_m := 1 \wedge x_m := x_i + x_j \wedge vm_m := vm_i + vm_j := (pos_{1,..}\, pos_i, pos_{(i+1)}, .., pos_m))$$
$$wobei\ pos_{(i+1)} := pos_1\, von\ \ vm_j,\ pos_{(i+2)} := pos_2\, von\ \ vm_j \ldots pos_m := letzte\, Position\, von\ \ vm_j$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der Verbundmarke wm_m (z_m :=1, x_m :=x_i+ x_j , wm $_m$:= pos$_1$,.., pos $_{xj}$) wobei pos$_1$,..,pos $_{xi}$ von wm $_i$, pos $_{(i+1)}$ bis pos $_{(xi+xj)}$:= pos $_1$,..,pos $_{xj}$ von wm $_j$ belegt wird.
Nach dem Schalten: 1* wm_m hinzufügen.

11) wm_m (z:=1, x_m := j-i+1, vm_m := (pos_i, …,pos_j))

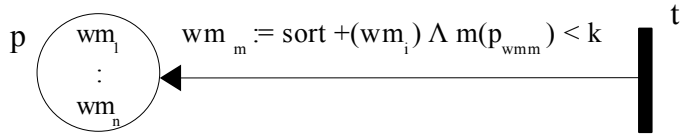
$$\forall wm_i \in Wanderer \exists wm_m \in Wanderer:$$
$$wm_m := wm_i(z_m := 1, x_m := j-i+1, vm_m := vm_i(pos_i, .., pos_j)) \wedge (i, j) \in I$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der Verbundmarke wm_m (z_m :=1, x_m :=j-i+1, wm_m := pos_i bis pos_j von wm_i).
Nach dem Schalten: 1* wm_m hinzufügen.

12) wm_m := sort + (wm_i)

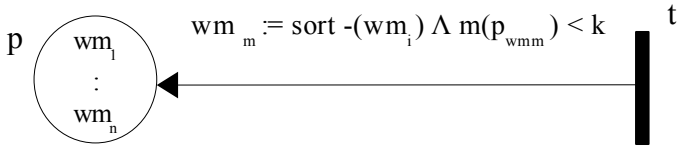
$$\forall wm_i \in Wanderer \exists wm_m \in Wanderer: wm_m := (z_m := z_i, x_m := x_i, vm_m := vm_i(pos_i, .., pos_j))$$
$$wobei \left|pos_i\right| \leq \left|pos_{(i+1)}\right| \wedge (i, j) \in I$$



Schaltfähig : es existiert freie Kapazität für die Aufnahme der Verbundmarke wm_m wobei z_m=1, x_m: = x_i , pos 1 bis pos x_m von wm_m : = aufsteigend sortiert aus den Positionen 1 bis pos x_i von wm_i stammen. Die Farbmarken auf diesen Positionen müssen aus einer geordneten Farbmenge stammen.
Nach dem Schalten: 1* wm_m hinzufügen**.**

13) wm_m := sort – (wm_i)

$$\forall wm_i \in Wanderer \exists wm_m \in Wanderer: wm_m := (z_m := z_i, x_m := x_i, vm_m := vm_i(pos_i, .., pos_j))$$
$$wobei \left|pos_i\right| \geq \left|pos_{(i+1)}\right| \wedge (i, j) \in I$$



Schaltfähig: es existiert freie Kapazität für die Aufnahme der Verbundmarke wm_m wobei z_m :=1, x_m := x_i , pos_1 bis pos_{xm} von wm_m := absteigend sortiert aus den Positionen 1 bis pos_{xi} von wm_i stammen. Die Farbmarken auf diesen Positionen müssen aus einer geordneten Farbmenge stammen.
Nach dem Schalten: 1* wm_m hinzufügen.

Nachbedingungen sind mit AND (&) und NOT (not) logisch verknüpfbar und auch mit Klammern (()) zusammenfassbar. In Vergleichen zur Positionsbestimmung ist auch OR (|) möglich. Andere Kombinationen von Farbvariablen, Farbmarken, Verbundmarkenvariablen und Ver-

bundmarken, als oben angegeben, sind möglich. Vielfachheiten bei wm und fm sind möglich, keine Angabe heißt v=1.

5.3 Verbale Beschreibung des Algorithmus

Die Cluster-Algorithmen [Ung, Wul02], die im Rahmen dieser Arbeit behandelt werden, konzentrieren sich nicht auf Daten und Typen der Nachrichten, sondern auf den Mechanismus der Kommunikation im Netzwerk. Sie wurden zur Unterstützung der Suche in P2P-Systemen entwickelt und werden für Anwendungsschichten für das Internet genutzt. Außerdem wird die Kommunikation und Verwaltung des Datenaustausches zwischen den Knoten in P2P-Netzen durch Wanderer organisiert.

Wanderer sind zirkulierende Nachrichten, die jeden Knoten eines Cluster innerhalb eines vorgegebenen Zeitrahmens besuchen können.

Die Wanderer arbeiten den Plan sequentiell ab und zirkulieren demzufolge zwischen Peers, deren Adressen im Plan gespeichert sind und somit die Cluster beschreiben.

Cluster ist ein Set der Knoten, die von demselben Wanderer besucht werden können. Er wird dynamisch durch die Wanderer gebildet.

Die Cluster-Algorithmen haben folgende Merkmale [Wul05]:
1. Sie erhalten eine stochastische Komponente.
2. Sie bauen mit Hilfe von Wanderern ihre Strukturen auf.
3. Sie sind nicht deterministische Algorithmen
4. Wanderer sind hier Nachrichten, die sich nach vorgegebenen Regeln im Netz bewegen, dabei Aufgaben der Strukturbildung übernehmen und zur Kommunikation zwischen den Peers verwendet werden.

In den Abschnitten 2.4.6 und 2.4.7 wurden die Anforderungen und die Funktionsweise der Algorithmen erläutert:
- Die besprochenen P2P-Systeme werden auf der Anwendungsschicht aufgebaut.
- Sie bestehen aus zwei Schichten, dem Internet (basierend auf dem Internetprotokoll) und dem überlagerten logischen P2P-Netz.
- Die Verbindungen zwischen den Peers sind logische Verbindungen.
- Die physikalischen Verbindungen sind vorgegeben und nicht veränderbar.

Eine Untersuchung solcher Algorithmen in dynamischen Umgebungen und großen verteilten Systemen ist schwierig und sehr aufwendig, außerdem ist eine globale Sicht auf reale Systeme unmöglich, und somit ist eine Überprüfung der Korrektheit der Funktion der Algorithmen schwierig. Ebenso erschwert die Inhomogenität neben der Dynamik realer P2P-Netze auch die Erstellung eines mathematischen Modells. Aus diesem Grund ist es günstig, solche Algorithmen mit den vorgestellten Petrinetzen zu modellieren und zu simulieren. Dadurch wird das Verhalten der Algorithmen untersucht. Die Anzahl an Wanderern im System im Zeitverlauf kann durch Simulation ermittelt werden. Außerdem wird die Zeit, die benötigt wird, um bestimmte Strukturen aufzubauen, betrachtet.

In dieser Arbeit wird der erste Cluster–Algorithmus als Beispiel untersucht. Einerseits wird die Effizienz der vorgestellten formalen Methode für P2P-Systeme und besonders für solche Arten von Algorithmen gezeigt. Andererseits wird der Ablauf des Algorithmus sichtbar und verifizierbar.

Verbale Beschreibung des Algorithmus der nicht disjunkten Cluster

Eine Algorithmus-Beschreibung erfolgt häufig noch in einer verbalen, informellen Form, die

die Bedingungen für einen Kommunikationsablauf beschreibt. Diese Art der Darstellung reicht dort aus, wo es im wesentlichen auf das prinzipielle Verständnis des allgemeinen Algorithmus-Ablaufs ankommt. Aus diesem Grund wird hier mit der verbalen Beschreibung begonnen.

In diesem Algorithmus werden die Knoten von mehr als einem Wanderer besucht. Infolge des mehrfachen Besuchs sind einige Knoten in mehr als einem der erzeugten Cluster enthalten. Diese Knoten werden zum Austausch der Information zwischen den Wanderern sowie den Clustern benutzt. Die folgende Abbildung zeigt Cluster, die durch unterschiedliche Wanderer erzeugt wurden.

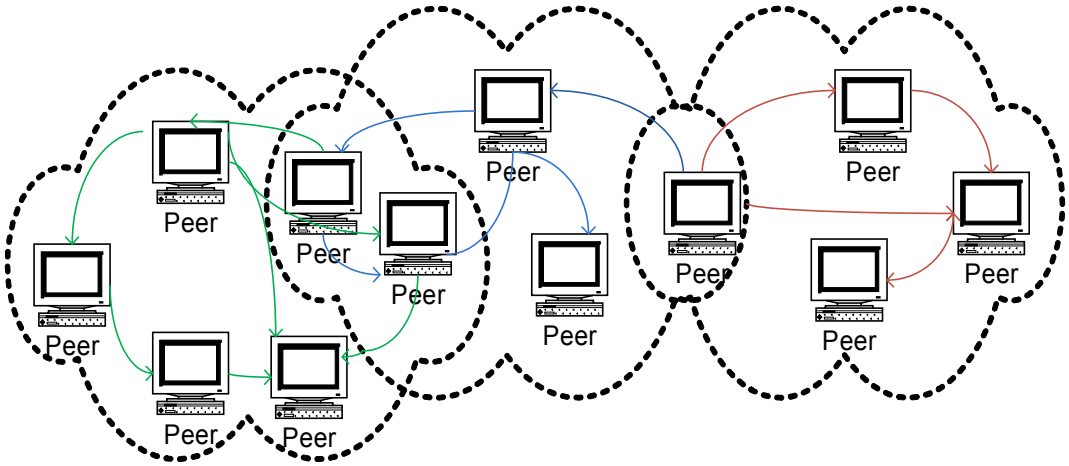


Abbildung 5.2: Durch Wanderer erzeugte nicht disjunkte Cluster. Jede Wolke stellt einen Cluster dar, jede gefärbte Linie einen Wanderer.

Mehrere Wanderer werden stets durch unterschiedliche Knoten erzeugt, weil jeder Knoten nur einen eigenen Wanderer erzeugen darf. Diese Wanderer bewegen sich entsprechend ihrer Pläne durch das Netz und werden auf den besuchten Knoten mit dem gleichen Mechanismus bearbeitet.

Wir nehmen an, dass ein Knoten n_1 seit einer bestimmten Zeit nicht von einem Wanderer besucht wurde. Deshalb wird er nun einen eigenen neuen Wanderer erzeugen.

Die Zeit $t_{avg,n1} + \Delta t$ gibt die Länge des Intervalls an, in dem der Knoten n1 von einem Wanderer besucht werden möchte.[1]
Am Anfang enthält der Plan des erzeugten Wanderer nur den Eintrag n_1, also den erzeugenden Knoten. Die übrigen Einträge sind leer. Zusätzlich speichert ein Wanderer folgendes:
t_{cycle} : Die durchschnittlich benötigte Zeit für einen Zyklus, berechnet aus den y letzten Zyklen.
z : Aktuelle Planposition
x : Länge des Plans. (ein zeitlich konstanter Wert)

1 Es gilt gemäß einer Ergänzung des Autors von [Wul05]: „Ein Knoten möchte in Intervallen t_avg,i besucht werden und duldet dabei eine Abweichung delta t. Mit ADD_TO_PLAN(l) und RMV_FROM_PLAN(l) kann der Wanderer sich in dem Plan vorbeikommender Wanderer ein- bzw. austragen und so Korrekturen an dem Besuchsintervall durchführen“.

Zu Anfang hat der Plan des neuen Wanderer leere Positionen. Jeder Plan hat eine bestimmte Anzahl an Leerpositionen reserviert, an denen immer eine zufällige Weiterleitung stattfindet, damit ein Wanderer das Netz erkunden kann.
Der Wanderer bewegt sich weiter zu nächsten Knotenadresse, die in ihrem Plan verzeichnet ist. Falls die nächste Position (z + 1) noch nicht besitzt ist, wird der Wanderer an einen Nachbarn des aktuellen Knotens gesendet, und auf den betroffenen Knoten wird eine der **folgenden Operationen durchgeführt:**

- **Forward:** Der Wanderer wird weiter an den nächsten Knoten entsprechend der erklärten Vorgehensweise geschickt. Der Zähler **t**, der die Zeit zwischen den Besuchen zweier Wanderer zählt, wird zurückgesetzt.
- **Kill_own**: Ein Knoten kann nur seinen eignen Wanderer löschen. Er tut das, wenn die Zeit t zwischen zwei Besuchen kleiner als $t_{(n_1, kill)}$ wird.
- **Set_Req:** Wenn ein Knoten nicht häufig genug von einem Wanderer besucht wird, bedeutet das, dass die Zeit t zwischen den letzten r Besuchen zweier beliebiger Wanderer länger ist als $t_{(request, n_1)}$. Dann wird das Flag REQUEST gesetzt, d.h. der Knoten versucht bei einem eintreffenden Wanderer die Operation Add_to_Plan auszuführen. Wenn dies nach gewisser Zeit $t_{(try, n_1)}$ nicht zum Erfolg führt, wird die Operation Generate_own ausgeführt. In beiden Fällen wird das Request_ Flag zurückgesetzt.
- **Generate_own:** Ein Knoten erzeugt einen neuen Wanderer, wenn er zu selten von einem fremden Wanderer besucht wird. Das Request_Flag ist gesetzt und die Zeit $t_{(try, n_1)}$, während derer der Knoten n_i versucht, sich in einem Wanderer einzutragen, ist abgelaufen.
  Dabei muss
  $$t_{(kill, n_i)} \ll t_{(avg, n_i)} \ll t_{(request, n_i)} < t_{(try, n_i)}$$
  gelten.
- **Add_to_Plan:** Der Knoten mit der Adresse n_i , der während der Zeit $t_{(request, n_i)}$ erfolglos auf einen ankommenden Wanderer gewartet hat, darf sich selbst in einen Wanderer eintragen.
- **Rmv_from_Plan:** Wenn ein Knoten in kurzer Zeit zweimal besucht wird (Zeit zwischen letztem Wanderer und aktuellem Wanderer $t \leq t_{(kill, n_i)}$), wird diese Knotenadresse aus dem Plan des aktuellen Wanderers gelöscht.
- **Check_Plan(ptr*).**

Mit diesen Operationen kann ein Knoten den Plan eines Wanderers so verändern, dass die Besuchsintervalle für diese Wanderer verlängert oder verkürzt werden.

Bei der Modellierung dieses Algorithmus mit erweiterten gefärbten Petrinetzen, die am Anfang dieses Kapitels definiert wurden, wird im Gegensatz zu [Kno95] die Kommunikation zwischen zwei Knoten im Netzwerk durch eine neue strukturierte Markenfarbe dargestellt, während das Verhalten des Algorithmus durch Veränderung der Markenfarben beschrieben wird.

Die Operationen, die der Algorithmus durchführt, werden mit Transitionen modelliert, und die Bedingungen werden durch Kantenausdruck-Funktionen überprüft. Die Transitionen werden nur geschaltet, wenn alle Vorbedingungen und Nachbedingungen, die sich in der Menge der Vor- und Nachplätze einer Transition $t \in T$ befinden, erfüllt werden.

5.4 Erweitertes TCPN-Modell des Algorithmus der nicht disjunkten Cluster

Der oben beschriebene Algorithmus soll jetzt mit einem TCPN modelliert werden. Abbildung 5.3 stellt dieses Modell dar. Die Beschreibung der Transitionen erfolgt in Tabelle 5.1 und die Beschreibung der Plätze des Netzes in Tabelle 5.2.

Tabelle 5.1 :Transitionsnamen des Netzmodells und ihre Bedeutung

Transition	Funktion
nettime	Modelliert die Verzögerung eines Wanderers während seines Zyklus im Netzwerk.
wr	Modelliert die Wanderer, die mit ihrer aktuellen Knotenadresse starten können.
timeover	Modelliert die abgelaufene Kill-Zeit, wonach die Knotenadresse vom Platz nodetoremove gelöscht werden kann. Das heißt, dass sie ab diesem Moment besucht werden kann.
nodetoadd	Modelliert die abgelaufene Wartezeit, wonach eine Knotenadresse besucht werden darf.
nodetoremove	Modelliert einen Controller, der alle Knotenadressen enthält, die vor kurzer Zeit besucht wurden.
wrgenarated	Modelliert einen Wanderer-Generator
random	Modelliert eine zufällige Sendung an einen Nachbar.
normforword	Modelliert eine normale Wanderer-Sendung zur nächsten Knotenadresse seines Plans.
noadd	Modelliert die Möglichkeit des Hinzufügens einer Knotenadresse, ohne das Knoten bereit dafür sind.
addtoplan	Modelliert das Hinzufügen vorhandener Knotenadressen zum ankommenden Wanderer.
addnoo	Modelliert den erfolglosen Versuch eines Knotens, sich in den ankommenden Wanderer einzutragen.
removefromplan	Löscht die aktuelle Knotenadresse aus dem Plan dieses Wanderers.
noremove	Fortsetzung, ohne den aktuellen Knoten zu löschen.
next	Setzt den Zeiger auf die nächste Position.
Kill	Löscht einen ankommenden Wanderer.

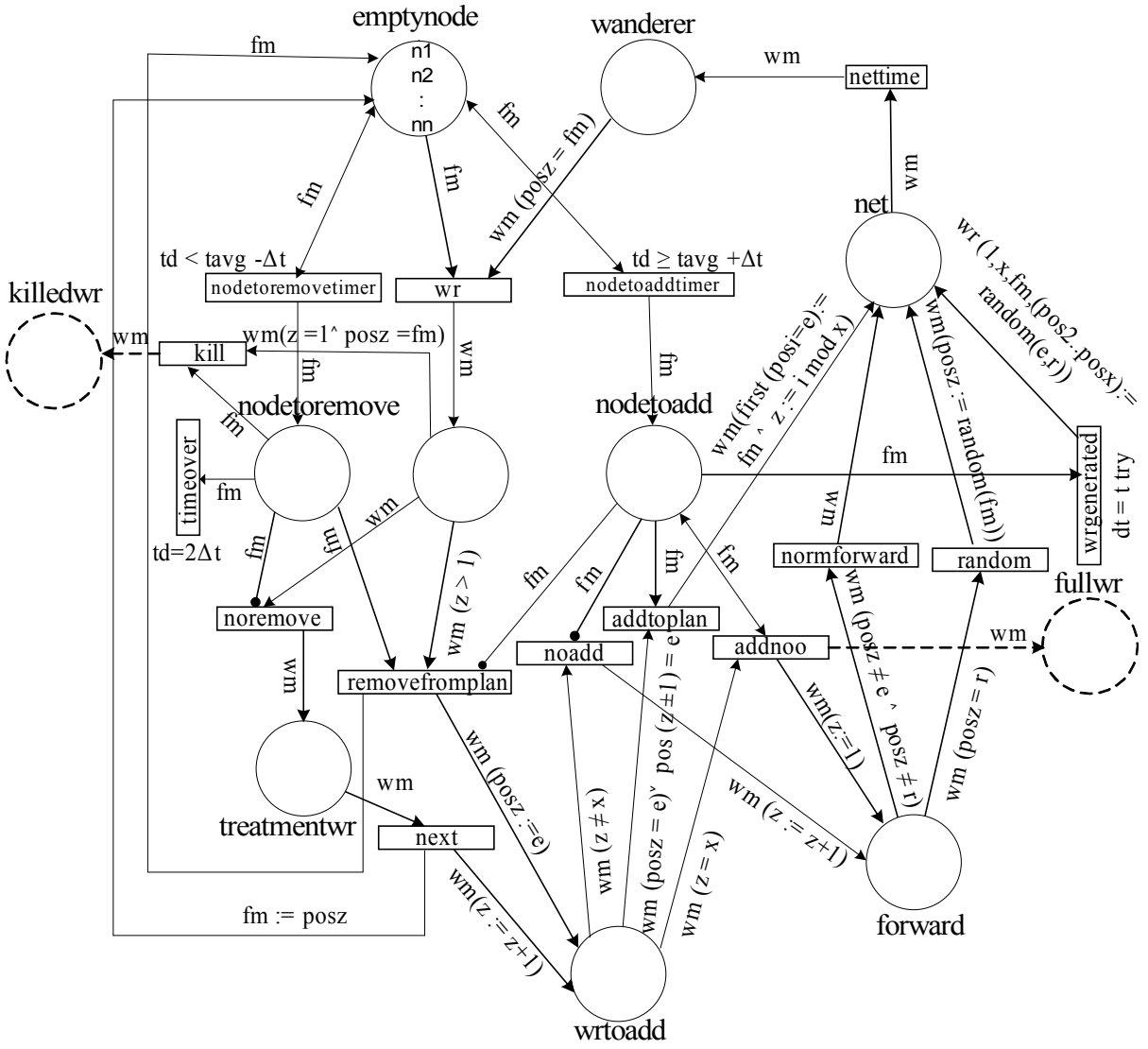


Abbildung 5.3: Modell des Algorithmus der nicht disjunkten Cluster

Tabelle 5.2 :Platznamen des Netzmodells und ihre Bedeutung

Platz	Beschreibung
emptynode	Adressen von Knoten, die keinen Wanderer in Bearbeitung haben.
wanderer	Enthält alle Wanderer mit der aktuellen Knotenadresse, hier wird an der Transition wr überprüft, ob es keinen Wanderer auf dieser Knotenadresse gibt.
net	Stellt alle Wanderer im Netz zum betrachteten Zeitpunkt dar.
nodetoadd	Alle Knotenadressen, die seit bestimmter Zeit nicht besucht wurden und sich in einen ankommenden Wanderer eintragen dürfen.
nodetoremove	Alle Knotenadressen, die in einem bestimmten Zeitraum vom gestarteten Wanderer gelöscht werden sollen.
wrstart	Enthält alle gestarteten Wanderer.
treatmentwr	Alle Wanderer, die in Bearbeitung sind.
wrtoadd	Alle Wanderer, die darauf überprüft wurden, ob sie hinzugefügt werden dürfen.

forward	Wanderer, die weitergeschickt werden sollen.
wrfull	Alle Wanderer, deren Plan voll ist.
killedwr	Alle gelöschten Wanderer

Spezifikation des Algorithmus-Modells:

Knotenadresse: $n_1, n_2, n_3, ... n_n$.
Wanderer: Wird als Verbundfarbmarke dargestellt. Zum Beispiel: (1,5, n1, n3, e, e, e, r)

Aufbau der Wanderer (Zeiger, Länge, Positionen des Plans)

Zeiger: Ist eine natürliche Zahl, die die Nummer der Position angibt, an der die aktuelle Knotenadresse enthalten ist, an der der Wanderer gerade bearbeitet wird.

Länge : Natürliche Zahl, beschreibt die Länge des Plans des Wanderers.

Positionen des Plans: Enthalten die besuchte Knotenadresse mit n und Index bezeichnet und freien Platz mit e bezeichnet und möglicherweise zufällige Sendung mit r bezeichnet.

Verhalten der Wanderer auf der Knotenadresse: Beschreibt, wie der Wanderer weitergeschickt und während der Bearbeitung geändert wird.

Wenn ein Knoten einen Wanderer empfängt, werden möglicherweise die folgenden Operationen auf diesem ausgeführt:

KILL_OWN:

Der Knoten n1 kann seinen eigenen Wanderer wm_{n1}(1,5, n1, n3, e, e, e, r) löschen, wenn die Besuchsintervalle zu kurz werden. Das heißt, die Zeit *t*, die zwischen den Besuchen zweier beliebiger Wanderer liegt, wird kleiner als *tkill*,*i* . Im TCPN Modell wird das dargestellt, indem n1 als Farbmarke im Platz **nodetoremove** steht. Wenn die Knotenadresse n1 ihren eignen Wanderer löscht, verschwindet einfach die Verbundfarbmarke durch Schalten der Transition **kill**. Wenn aber der Wanderer (2,5,n1,n3, e, e, e, r) als Marke im Platz **startwr** steht und gleichzeitig n3 als Farbmarke im Platz **nodetoremove** steht, wird die Transition **removfromplan** geschaltet. Dadurch ersetzt die Knotenadresse n3 eine Freiposition mit der Marke **e,** und der Wanderer (2,5,n1,e,e,e,r) kommt in den Platz **wrtoadd.** Mit dieser Vorgehensweise wird (RMV_FROM_PLAN) modelliert.

FORWARD:

Der Wanderer wird entsprechend seinem Plan weitergeschickt. Dazu wird zuerst der Zeiger um Eins erhöht, was durch das Schalten der Transition **next** realisiert wird. Danach wird die Position mit dem Index z+1 weiterbearbeitet. Die Stelle *z*+1 des Plans mit der Farbmarke **e** wird durch die Transition **addtoplan** mit einer Knotenadresse belegt und mit der Transition **normforward** an diese Knotenadresse geschickt. Im Sonderfall enthält die Stelle *z*+1 des Plans die Farbmarke **r,** dann soll der Wanderer zufällig an einen Nachbarn der zuletzt besuchten Knotenadresse geschickt werden, was durch die Transition **random** geschieht. Der Zähler *t*, der die Zeit zwischen zwei Wanderern zählt, wird zurückgesetzt. Das wird im Modell des Algorithmus durch das Eintragen der Knotenadresse auf dem Platz **emptynode** nach dem Ende der Wanderer-Bearbeitung des aktuellen Knotens dargestellt, wo die Zeit durch die Transition **nodetoaddtimer** gezählt wird.
Die Veränderung der Verbundfarbmarke **Wanderer** wird durch **Forward** beschrieben**,** wobei der Zeiger um Eins erhöht wird: z:= z +1 =1+1=2 . Da aber die Position 2 den Wert n3 ent-

hält, wird der Wanderer zur Knotenadresse n3 geschickt und die Verbundfarbmarke wird zu (2,5,n1,n3,e,e,r).

SET_REQ:
Diese Operation wird im Modell durch das Schalten der Transition **nodetoaddtimer** dargestellt, und die Knotenadresse, die das Flag REQUEST gesetzt hat, wird in den Platz **nodetoadd** kopiert, wobei sie auch im Platz **emptynode** bleibt. Aber dort wird sie zusätzlich durch die Transition **wrgenerated** beobachtet, die nach dem erfolglosen Ablauf der Zeit t_{try} (Zeit, innerhalb derer ein Knoten versucht einen ankommenden Wanderer einzutragen) für jede Farbmarke im Platz **nodetoadd** geschaltet wird und damit die Operation GENERATE_OWN ausführt, wobei neue Wanderer erzeugt werden. Z.B. wm (1,5,n4,e,e,e,r).

Nach einer gewissen Zeit sollten sich die Anzahl der Wanderer und deren Pläne stabilisieren, falls das zugrunde liegende P2P-Netz nicht mehr verändert wird. In einem dynamischen Netz, in dem ständig Knoten verschwinden und neue hinzukommen, werden auch die Wanderer und somit die Cluster ständigen Veränderungen unterliegen. Zum Simulieren dieses Modells werden zwei weitere Plätze eingefügt. Der eine, **fullwr,** repräsentiert komplette Wanderer mit stabilisierten Plänen und der andere, **killdedwr,** zeigt gelöschte Wanderer. Die beiden Plätze werden grafisch mit gestrichelten Linien gezeichnet. Nach einiger Zeit stabilisiert sich das Verhalten und es ist zu erkennen, dass sich der Wanderer nun auf einem recht stabilen Zyklus bewegt. Dabei erkennen wir Cluster, die durch die stabilisierten Pläne gebildet werden.

5.5 TCPN-Modell des Algorithmus der nicht disjunkten Cluster

5.5.1 Modellerstellung

Zunächst sollen die einzelnen verwendeten Farbmarken, Platzssymbole, Transitionssymbole und Funktionssymbole erläutert werden. Zum besseren Verständnis werden sie in den folgenden Tabellen erläutert:

Einschränkungen bzw. Vereinbarungen für Wanderer und Knoten

Es werden einige Einschränkungen definiert, deren Grund darin besteht, dass das Modell mit dem Werkzeug PENECA Chromos erzeugt werden soll und hier mit normalem Aufwand simulierbar sein soll.

3 Knoten: w1, w2,w3

- Länge des Wanderers: 4;
- Je Knoten ist ein Eintrag im Wanderer möglich, auf einem festen Platz (1…3), z.B.:

w1	w2	w3	p

- erster Platz, erzeugender Knoten;
- Vierter Platz: p, d.h. zufälliges Verschicken;
- Je Wanderer und Zeiger auf den Platz im Wanderer, auf den der Wanderer verschickt wird, entsteht je eine unterschiedliche Farbmarke. Der Zeiger kann nicht auf leere Plätze zeigen, sie werden deshalb übergangen. Für jeden Wanderer entsteht eine Farbgruppe, die den Wanderer und seinen Zeiger beschreibt. Siehe z.B. den Wanderer von oben:

v1,w1	x1,w2	y1,w3	z12,w2;z13,w3

Aufbau: Farbmarke, mögliche Zielknoten; bei p die beiden anderen Wanderer.
Bei leeren Plätzen im Wanderer wird die Farbgruppe entsprechend kleiner.

- Das Hinzufügen eines Knotens erfolgt nur bei den festen Plätzen im Wanderer, d.h. ein Knoten kann nur einmal im Wanderer vorkommen.

Tabelle 5.3: Wanderer und Knotenadressen

sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9	sp10	sp11	sp12
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2

Spalte 1 bis 4: Mögliche Wanderer im System mit 4 Plätzen
w1, w2 ,w3: Knotenadressen
l: leerer Platz
p: Platz für stochastisch gesuchten Nachfolger
Sp. 1: Erzeugerknoten des Wanderers

Spalte 5 bis 8: Mögliche Zeigerwerte für die Wanderer der gleichen Zeile, Zeiger auf die Knotenadresse, an die der Wanderer verschickt wird (Folgeknoten-adresse) (Bsp.: Zeiger = 3 für Wanderer w3, l, w2,p, Wanderer wird an Knoten w2 verschickt)

Spalte 9 bis 12: Zuordnung der Wanderer mit entsprechendem Zeiger zu Farben und Knotenadressen
(Bsp.: Zeiger=3, Wanderer w3,l,w2,p: g3,w2)
Besonderheit Spalte 12: durch zufällige Auswahl wird die Folgeknoten-adresse aus der Menge der anderen Knoten gebildet.
(Bsp.: Zeiger=4, Wanderer w3, l, w2,p, Farbe hs1: Folgeknotenadresse w1 oder Farbe hs2: Folgeknotenadresse w2)

In Abbildung 5.4 stellen die **blauen** Plätze mit rotem Rand Platzunternetze dar, während die **grünen** Plätze für das eigentliche Modell keine Rolle spielen. Sie dienen der Simulation und können dabei zeigen, wie oft Add to-Plan, Kill, Remove-from-Plan und Forward im Netz durchgeführt werden.

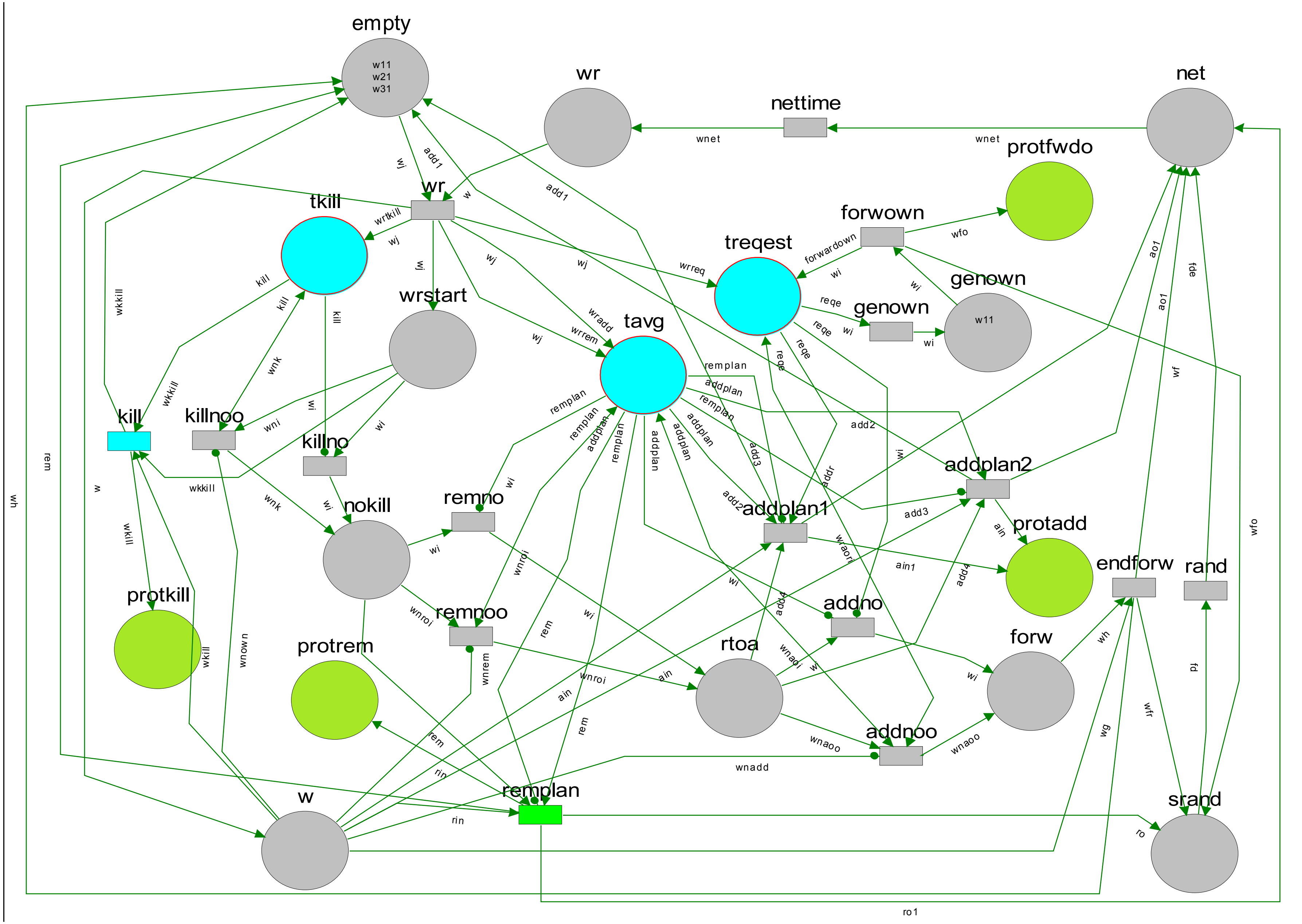


Abbildung 5.4 CPN Algorithmus-Modell mit Peneca Chromos

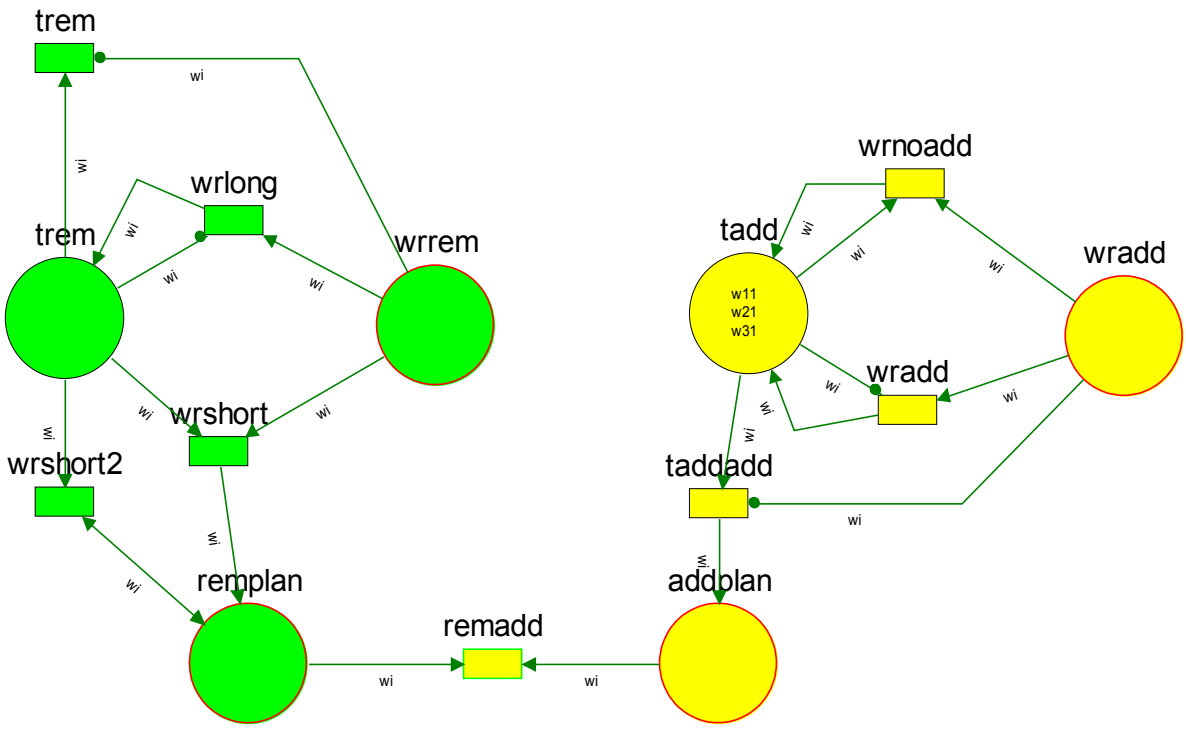


Abbildung 5.5: Unternetz Tavg

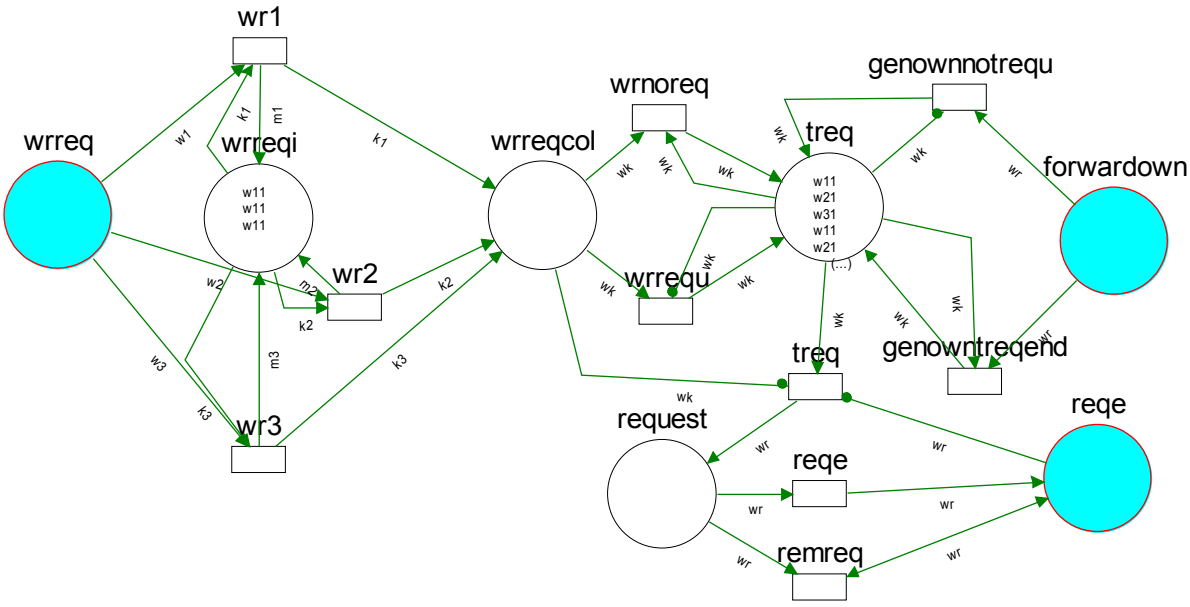


Abbildung 5.6: Unternetz T_request

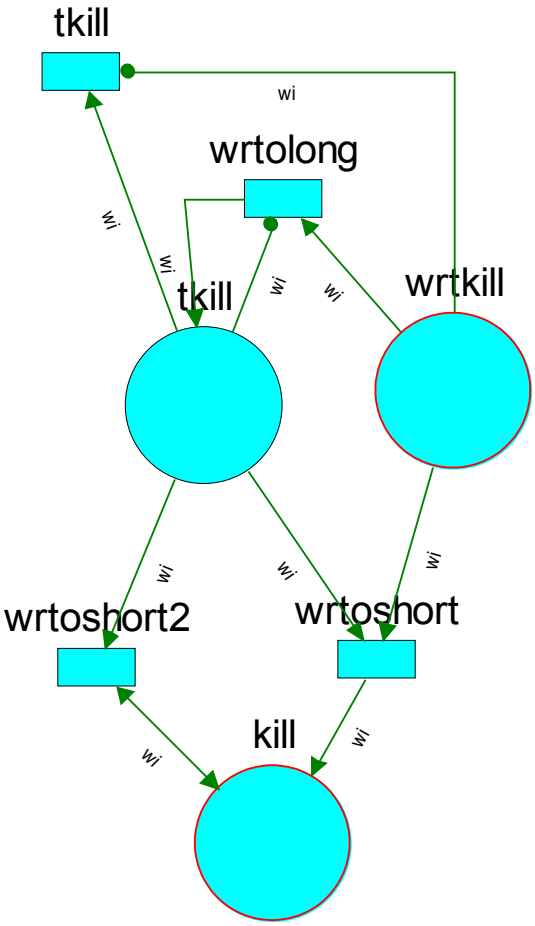


Abbildung 5.7: Unternetz T_kill

Die folgenden Tabellen 5.4 bis 5.11 beschreiben die Plätze und Transitionen des Chromos-Netzes.

Hauptnetz

Tabelle 5.4 : Plätze, Platzunternetze

empty	Freie Knoten, d.h. Knoten ohne Wanderer in Bearbeitung (empty nodes)
forw	Start des Verschickens eines Wanderers (sent forward)
genown	Generieren eines eigenen Wanderers (generate own)
net	Modelliert das Verbindungsnetz, mit Farbmarke modelliert es den Wanderer im Netz (wanderer in net)
nokill	Kein Wanderer zu löschen (no node to kill)
protadd	Protokoll der Wanderer, zu denen Knoten hinzugefügt wurden. Nur für Simulation! (protocol of add)
protfwdo	Protokoll der eigenen, neu generierten Wanderer. Nur für Simulation! (protocol of forward own)
protkill	Protokoll der entfernten Wanderer. Nur für Simulation! (protocol of kill)
protrem	Protokoll der Wanderer, aus denen Knoten entfernt wurden. Nur für Simulation! (protocol of remove)
rtoa	Vom Entfernen zum Hinzufügen (remove to add)
srand	Start des zufälligen Verschickens (start to sent random)
tavg,	Zwei Zeitgeber zur Behandlung von tavg, für das Entfernen und Hinzufü-

Unternetz	gen (timer for tavg)
tkill, Unternetz	Zeitgeber für das Löschen (timer for tkill)
trequest, Unternetz	Zeitgeber für trequest, für das Hinzufügen bzw. Generieren (timer for trequest)
W	Wanderer, die auf einem Knoten in Bearbeitung sind (wanderer)
Wr	Empfangene Wanderer (wanderer received)
wrstart	Knoten, Start der Bearbeitung der Wanderer (node start to do)

Tabelle 5.5: Transitionen

addno	Knoten soll nicht hinzugefügt werden, weil die Zeit tavg,i ist nicht groß genug für den Besuch des Wanderers ist
addnoo	Knoten soll hinzugefügt werden, was bei dem aktuellem Wanderer aber nicht möglich ist
addplan	Knoten zu aktuellem Wanderer hinzufügen (add to plan)
endforw	Ende des normalen (d.h. ohne Entfernen, ohne Hinzufügen, nicht zufällig) Versendens an einen Nachfolgeknoten (end forward)
forwown	Versenden des neu generierten Wanderes mittels zufälligem Generieren eines Nachfolgeknotens (forward own)
genown	Generieren eines eigenen Wanderers mit der Zeitverzögerung ttry (generate own)
kill	Aktuellen Wanderer löschen (kill)
killno	Löschen nicht angefordert (kill no)
killnoo	Löschen angefordert, aber der bearbeitete Wanderer ist kein eigener Wanderer, weshalb er nicht gelöscht werden darf (kill but no own, not allow)
nettime	Stochastische Verzögerung (mit einstellbarer Wahrscheinlichkeit) zwischen Senden und Empfangen von Nachrichten (net time)
rand	Zufälliges Generieren eines Nachfolgeknotens (random)
remno	Entfernen nicht angefordert (rmove no)
remnoo	Knoten soll entfernt werden, bei aktuellem Wanderer aber nicht möglich (remove but no possible)
remplan	Knoten aus aktuellem Wanderer entfernen (remove from plan)
Wr	Wanderer empfangen (wanderer received)

Platzunternetz tkill

Tabelle 5.6: Plätze

kill	Knoten, deren eigene Wanderer gelöscht werden sollen, Schnittstelle (kill)
tkill	Knoten, für die tkill dekrementiert wird (time for kill)
wrtkill	Wanderer empfangen für den Löschzeitgeber, Schnittstelle (wanderer received for kill)

Tabelle 5.7: Transitionen

tkill	Dekrementieren der Zeitgeber für Löschen, Zeitverzögerung tkill (timer kill)
wrtolong	Zeit zwischen zwei Wanderern groß genug (wanderer received too long)
wrtoshort	Zeit zwischen zwei Wanderern zu kurz, Löschen (wanderer received too short)
wrtoshort2	Zeit zwischen zwei Wanderern zu kurz, Löschen aber schon vorgesehen (wanderer received too short 2)

Platzunternetz tavg

Tabelle 5.8: Plätze

addplan	Knoten, die zu einem Wanderer hinzugefügt werden sollen, Schnittstelle (add to plan)
remplan	Knoten, die aus einem Wanderer entfernt werden sollen, Schnittstelle (remove from plan)
tadd	Knoten, für die tadd dekrementiert wird (time for add)
trem	Knoten, für die trem dekrementiert wird (time for remove)
wradd	Wanderer empfangen für den Hinzufügen-Zeitgeber, Schnittstelle (Wanderer received for add)
wrrem	Wanderer empfangen für den Entfernen-Zeitgeber, Schnittstelle (Wanderer received for remove)

Tabelle 5. 9: Transitionen

remadd	Gleichzeitig vorgesehenes Entfernen und Hinzufügen kompensieren (remove and add)
taddadd	Dekrementieren des Hinzufügen-Zeitgebers, Zeitverzögerung tavg+dt, bei Nulldurchgang Hinzufügen (timer add and add)
trem	Dekrementieren des Entfernen-Zeitgebers, Zeitverzögerung tavg-dt (timer remove)
wradd	Neustart des Hinzufügen-Zeitgebers (wanderer add new)
wrlong	Zeit zwischen zwei Wanderern groß genug (wanderer received too long)
wrnoadd	Zeit zwischen zwei Wanderern nicht zu groß (wanderer no add)
wrshort	Zeit zwischen zwei Wanderen zu kurz, Entfernen (wanderer received tooshort)
wrshort2	Zeit zwischen zwei Wanderen zu kurz, Entfernen aber schon vorgesehen (wanderer received to short 2)

Platzunternetz trequest

Tabelle 5.10: Plätze

forwardown	Knoten, die gerade einen eigenen Wanderer generiert haben, Schnittstelle (forward own)
reqe	Maximal 1 Anforderung je Knoten für das Generieren, Schnittstelle (request end)
request	Alle Anforderungen für das Generieren (request)
treq	Knoten, für die trequest dekrementiert wird (time for request)
wrreq	Wanderer empfangen für den Generieren-Zeitgeber, Schnittstelle (wanderer received for request)
wrreqcol	Wanderer empfangen, in r Spalten eingeordnet (wanderer for request column)
wrreqi	i-ten Wanderer empfangen (wanderer for request i)

Tabelle 5.11: Transitionen

genownnotrequ	Anfangswerte für treq bei fehlender Zeitmarke (generate own with no trequest)
genowntreqend	Rücksetzen von treq bei vorhandener Zeitmarke (generate own trequest end)
remreq	Löschen der anderen Requestanforderungen des gleichen Knotens nach Erzeugen einer Anforderung (remove request)
reqe	Erzeugen einer Requestanforderung (request end)
treq	Dekrementieren des Generieren-Zeitgebers, Zeitverzögerung trequest, bei Nulldurchgang Request setzten (timer request)
wr1,2,3	Wanderer i empfangen (i=1,2,3) (wanderer received 1,2,3)
wrnoreq	Zeit zwischen r Wanderern nicht zu groß (wanderer no request)
wrreq	Neustart des Generieren-Zeitgebers (wanderer request new)

Symboltabellen (Schaltmodus-Tabellen) aus Peneca Chromos

Diese Tabellen realisieren die Bedingungen für die Farbmarken, welche im Vorbereich (input) und im Nachbereich (output) einer Transition auftreten.

Die Schaltmodi sind untereinander mit UND oder ODER verknüpft, bei Nachkanten nur mit UND.
Die Variablen (erste Zeile, ab Spalte 2) sind an den entsprechenden Vor- und Nachkanten der Transition zu finden.
Die Schaltfähigkeit und das Schalten von Farbmarken durch die angegebenen Transitionen für das Netzmodell aus Abbildung (5.3) werden komplett in Anlage A dargestellt. Die folgende Tabelle ist dazu ein konkretes Beispiel.

Tabelle 5.12 Symboltabelle addnoo (aus PENECA Chromos)

Schaltmodi	wnaoi	wnaoo	wnadd	wraori
w3	w3 1	w3 1	d13 1 & m13 1 & d23 1 & h23 1	
w3a		w3 1	d13 1 & m13 1 & d23 1 & h23 1	w3 1
w1	w1 1	w1 1	d31 1 & h31 1 & d21 1 & m21 1	
w1a		w1 1	d31 1 & h31 1 & d21 1 & m21 1	w1 1
w2	w2 1	w2 1	d32 1 & m32 1 & d12 1 & h12 1	
w2a		w2 1	d32 1 & m32 1 & d12 1 & h12 1	w2 1

Die folgenden Tabellen 5.13 und 5.14 zeigen die möglichen Operationen add und rem für einige Knotenadressen. Für die übrigen Knotenadressen siehe Anlage A.

Tabelle 5.13 Add Knoten w3

Sp1	Sp2	Sp3	Sp4	Sp5	Sp6	Sp7	Sp8	Sp9	Sp10	Sp11	Sp12	Sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	g1,w3
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	y1,w3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	k2,w3
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	

w2	l	w1	p	1		3	4	e2,w2		g1,w3	h21,w1;h23,w3	x2,w3
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y1,w3	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3w3		g3,w2	h31,w1;h32,w2	
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	

Tabelle 5.14 rem Knoten w

Sp1	Sp2	Sp3	Sp4	Sp5	Sp6	Sp7	Sp8	Sp9	Sp10	Sp11	Sp12	Sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	d12,w2;d13,W3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	g1,W3
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3w3		g3,w2	h31,w1;h32,w2	d31,w1;d32,w2
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	m31,w1;m32,w2

5.5.2 Übersicht über das Gesamtmodell

- Der Algorithmus beschreibt, wie Wanderer zwischen Peers zirkulieren und welche Bearbeitung auf den Knoten durchgeführt wird.
- Die Peers und Wanderer werden mit Farbelementen modelliert.
- Die Operationen werden durch Plätze dargestellt.
- Die Zeitbearbeitungen werden in Unternetzen durchgeführt.
- Das Modell wird mit dem Werkzeug Peneca Chromos erzeugt und simuliert.
- Das Verhalten des Algorithmus kann durch Simulation des Modells gezeigt werden.

Beschreibung der Simulation des Algorithmus-Modells mit Peneca Chromos (Abbildung 5.4)

Mit Hilfe der Simulation lassen sich Abläufe des Netzes ermitteln, wie im Folgenden beschrieben wird. Damit wird das Verhalten des Algorithmus klar und sichtbar. Als konkretes Beispiel wird untersucht, wie sich der Unterschied zwischen zwei Modellen auswirkt. Eines dieser Modelle benutzt n Knotenadressen und Wanderer mit der Länge 5, das andere 3 Knotenadressen und Wanderer mit der Länge 4. Gleichzeitig wird deutlich, dass dabei der Modellierungsaufwand mit normalen CPN sehr groß wird. Dieser Unterschied wird im nächsten Kapitel weiter erklärt.

- **Normale Fälle:**

Mit Beginn der Simulation empfängt der Knoten w1 den Wanderer a1 (w1, l, l, p). Das wird als Anfangsmarkierung im Platz **wr** dargestellt.

Damit der Knoten w1 den Wanderer a1 empfangen kann, darf er keinen anderen Wanderer haben, entsprechend dem Algorithmus von Wulff [Wul05] S.46 .

Der Platz **empty** (freier Knoten) enthält die Farbmarke w1, was bedeutet, dass der Knoten w1 frei ist und es keinen Wanderer auf dem Knoten w1 in Bearbeitung gibt / Tabelle 5.3 /.

Dann schaltet die Transition **wr**, weil die Vorbedingungen und Nachbedingungen erfüllt sind, und der Wanderer wird empfangen.

Die Farbmarke w1 erscheint im Platz **wrstart**, was bedeutet, dass der Knoten die Wandererbearbeitung startet. Gleichzeitig erscheint a1 im Platz **w** (Wanderer sind auf einem Knoten in Bearbeitung).

Weiterhin schaltet die Transition **killno.** Das Löschen von w1 aus dem Plan des Wanderers ist nicht angefordert, weil die Zeit zwischen den Besuchen beliebiger Wanderer noch größer als tkill,i ist[Wul05].

Danach feuert die Transition **remno** und erzeugt die Farbmarke w1 im Platz **rtoa** (vom Entfernen zum Hinzufügen), was bedeutet, dass der Knoten w1 zum Plan des Wanderers hinzugefügt werden soll. Entsprechend dem Algorithmus ist w1 noch nicht im Plan enthaltem und die Zeit tavg,i ist noch gültig, weshalb w1 dem Plan zugeordnet werden kann.

Weiterhin schaltet die Transition **addno** (Knoten soll nicht hinzugefügt werden), was bedeutet, dass das Hinzufügen des Knotens w1 nicht erforderlich ist, weil die Zeit tavg,i (wird später erklärt) nicht groß genug zum Besuch des Wanderers ist.

w1 erscheint im Platz **forw.** Dieses Erscheinen modelliert den Start des Weiterschickens des Wanderers / Tabelle 5.4 /.

Die Simulation geht weiter mit der Aktivierung der Transition **endforw**, was das Ende des normalen Versendens an den Nachfolgeknoten w1 darstellt. Die Farbmarke w1 erscheint wieder im Platz **empty.** Es verschwindet die Farbmarke a1 vom Platz **w**, dazu erscheint die Farbmarke d1 im Platz **srand** (Start des zufälligen Verschickens) (Beginn des Sonderfalls).

- **Sonderfälle :**

In folgenden werden die Sonderfälle vorgestellt:

1. Der Wanderer hat in seinem Plan keinen Nachfolgeknoten, dann startet ein **z**ufälliges Verschicken, das durch den Platz s**rand** modelliert wird.

Die Farbmarke d1 erscheint im Platz **srand**, und die Transition **rand** kann jetzt schalten.

Das Schalten der Transition **rand** bedeutet, dass der Wanderer einen Nachfolgeknoten generiert /Tabelle 5.5/. Mit dem Erscheinen der Farbmarke d3 im Platz **net** wird dargestellt, dass ein Wanderer im Netz läuft. Gleichzeitig stellt die Farbmarke d3 ein zufälliges Verschicken zum Knoten w3 dar /Tabelle 5.3/.

Zunächst schaltet die Transition **nettime,** die die Verzögerung der Zeit zwischen Senden und Empfangen einer Nachricht modelliert.

Danach geht es weiter mit dem Mechanismus, der im normalen Fall erklärt wird.

2. Entsprechend dem Algorithmus gilt: „Wenn ein Knoten nicht häufig genug von einem Wanderer besucht wird, d. h. die Zeit *t* zwischen den letzten *r* Besuchen zweier beliebiger

Wanderer ist länger als *trequest,i*, dann wird das Flag REQUEST gesetzt. Dies bewirkt, dass dieser Knoten versuchen wird, bei einem eintreffenden Wanderer die Operation ADD_TO_PLAN(l) auszuführen“[Wul05].

Die Transition **addplan** (Knoten zu aktuellem Wanderer hinzufügen) schaltet nach dem Schalten der Transition **remno,** und die Farbmarke w3 wird vom Platz **rtoa** entfernt. Gleichzeitig erscheint im Platz **protadd**, der das Protokoll der Wanderer modelliert, die Farbmarke d3, / Tabelle5.13/.

Dazu erscheint die Farbmarke g1 im Platz **net**, der darstellt, dass es einen neuen Knoten im Netz gibt, der besucht werden möchte.

Die Transition **addnoo** schaltet, wenn die Vorbedingungen und Nachbedingungen erfüllt sind.

Der Platz **w** hat keine Farbmarken (d31,h31,d21,m21), und das Unternetz tavg hat die Marke w1 im Platz **addplan.** Im Unternetz **treqest** hat w1 aber keine Anforderung, das heißt, der Knoten w1 ist schon in der Liste dieser Wanderer, und das Request-Flag ist entsprechend dem Algorithmus zurückgesetzt. Nach dem Schalten von **addnoo** erscheint die Farbmarke w1 im Platz **forw** und es beginnt ein neues Verschicken, wozu die Transition **endforw** schaltet. Damit endet das normale Versenden an einen Folgeknoten, es beginnt aber ein zufälliges Versenden.

3. Wenn die Transition **Kill** schaltet, dann heißt das, dass die Zeit zwischen den Besuchen zweier beliebiger Wanderer kleiner wird als **tkill**. Das wird durch das Platzunternetz gesteuert. Entsprechend dem Algorithmus gilt: „Ein Knoten kann seinen eigenen Wanderer löschen, wenn die Besuchsintervalle zu kurz werden. Das heißt, die Zeit *t*, die zwischen den Besuchen zweier beliebiger Wanderer liegt wird kleiner als *tkill,i[Wul05]“*

Die Farbmarke des Wanderers erscheint im Platz **protkill**, der das Protokoll der gelöschten Wanderer modelliert.

Wenn die Transition **killnoo** schaltet und die Farbmarke w1 im Platz **nokill** erscheint, modelliert das, dass der Knoten w1 gelöscht werden muss, wobei w1 aber nicht der erzeugende Knoten ist. Deshalb schaltet danach entweder die Transition **(remno)** und es geht weiter wie im normalen Fall, oder es schaltet die Transition **remplan** (nächster Sonderfall).

4. Wenn alle Vorbedingungen und Nachbedingungen der Transition **remplan** erfüllt sind, schaltet diese Transition. Die Vorbedingungen stellen dar, dass die Zeit tavg, w1 nicht mehr reicht, damit w1 vom Wanderer besucht wird. Mit dem Schalten der Transition **remplan** erscheint die Farbmarke d1 im Platz **srand** (was der Start des zufälligen Verschickens ist, wie im ersten Sonderfall erwähnt).

Wenn die Transition **remnoo** schaltet, dann bedeutet das, dass der Knoten entfernt werden soll, was aber nicht möglich ist / Tabelle 5.4 /.

Wenn der Knoten noch nicht in der Liste ist, ist das Entfernen des Knotens w1 nicht möglich. Unter der Bedingung, dass die Zeit (tavg,w1) groß genug zum Besuch des Knotens w1 durch den Wanderer ist, schalten die Transitionen **addno** und **endforw** nacheinander.

Die Zeit ist ein wichtiger Faktor im Algorithmus, deshalb soll die Zeit jedes Wanderers kontrolliert werden. Das wird durch die Unternetze tkill, trequest und tavg modelliert.

Der Mechanismus dieser Unternetze wird im folgenden erklärt.

Unternetz tkill

Der Platz **wrtkill** empfängt Wanderer für den Löschzeitgeber /Tabelle 5.6 /.

Zuerst schaltet die Transition **(wrtolong)**. Dann verschwindet die Farbmarke der Knotenadresse w1, die den Wanderer empfängt, Es erscheint die gleiche Farbmarke im Platz (tkill), wodurch modelliert wird, dass dieser Knoten seine Zeit tkill dekrementieren soll / Tabelle 5.6 /.

Wie geht es weiter? Es gibt **drei Möglichkeiten:**

- Entweder schaltet die Transition **tkill** nach der Zeitverzögerung t > tkill, und es verschwindet die Farbmarke w1. Die Löschzeit für den Knoten w1 wird dekrementiert, was bedeutet, dass die Knotenmarke w1 im Platz **tkill** bleibt bis die Löschzeit abgelaufen ist.
- Oder es schaltet die Transition **(wrtoshort),** wenn die beide Plätze eine Marke haben. Das modelliert, dass w1 einen Wanderer zum zweiten mal empfängt. Um die Löschzeit zu bestimmen soll diese dekrementierte Zeit überprüft werden. Nach dem Schalten von **wrtoshort** erscheint die Marke w1 im Platz **kill**, d.h. wegen der zu kurzen Zeit zwischen den zwei Wanderern soll der Wanderer gelöscht werden / Tabelle 5.6 /.
- Schließlich können die beiden Plätze **wrtkill** und **tkill** unterschiedliche Marken haben. In der Folge schaltet die Transition **wrtolong** und es werden die beiden Farbmarken im Platz **tkill** erzeugt. Die Zeit für die beiden Wanderer wird dekrementiert und die Löschzeit für die beiden wird überprüft.

Unternetz trequest

Der Algorithmus sagt aus: [Wul05]**:**

„Wenn ein Knoten nicht häufig genug von einem Wanderer besucht wird, d. h. die Zeit *t* zwischen den letzten *r* Besuchen zweier beliebiger Wanderer ist länger als *trequest*,*i*, dann wird das Flag REQUEST gesetzt. Dies bewirkt, dass versucht wird, bei einem eintreffenden Wanderer die Operation ADD_TO_PLAN(l) auszuführen. Führt das innerhalb der Zeit *ttry*,*i* nicht zum Erfolg, wird die Operation GENERATE_OWN ausgeführt. In jedem Fall wird das REQUEST-Flag zurückgesetzt.“

Das Unternetz **trequest** erfüllt diese Aufgabe, indem es an den Ausgängen eine Farbmarke der Knotenadresse entweder zur Transition **addplan1** oder zur Transition **genown** liefert, damit der adressierte Knoten seinen eigenen Wanderer erzeugt.

In folgenden wird erklärt, wie das **Platzunternetz request** arbeitet:

- Mit Beginn der Simulation erhält das Unternetz die Farbmarke w1 im Platz **wrreq** (Wanderer empfangen für Generieren-Zeitgeber), wodurch modelliert wird, dass der Wanderer w1 empfangen wird, um die Zeit trequest zu generieren.
- Zuerst schaltet die Transition **wr1,** wodurch dargestellt wird, dass der Wanderer w1 zum ersten Mal empfängt. Dann erscheint die Farbmarke w1 im Platz **wrreqcol** (der das Empfangen eines Wanderer modelliert und diesen in der Spalte r einordnet), was bedeutet, dass der Wanderer w1 empfangen und in der Spalte 1 eingeordnet wird /Tabelle 5 .10 /. Gleichzeitig erscheint die Farbmarke w12 im Platz **wrrreqi,** und es verschwindet die Farbmarke w11. Der Platz **wrrreqi** arbeitet als Geber, der jedem ankommenden Wanderer eine Nummer (bis 3) gibt.
- Danach schaltet die Transition **wrnoreq,** wenn die Zeit zwischen den letzten beiden Wanderern nicht groß genug ist. Dabei verschwindet die Farbmarke w1 vom Platz **wrreqcol.** Im Nachbereich gibt es wegen der Testkante keine Wirkung. Im anderen Fall schaltet die Transition **wrrequ,** d.h. wenn die Zeit zwischen den letzten beiden Wande-

rern groß genug ist. Als Folge verschwindet die Farbmarke w1 vom Platz **wrreqcol,** und es scheint wieder die Farbmarke w1 im Platz **treq**.

Wenn die Zeit **t > t request** (die Zeit wird in der Simulation trequest = t gesetzt) ist und die beiden letzten Plätze **wrreqcol**, **req** (request end) keine Farbmarken haben, schaltet die Transition **treq** (Dekrementieren des Generieren-Zeitgebers) und im Platz **request** erscheint die Farbmarke w2. Danach feuert die Transition **reqe** (Erzeugen einer Requestanforderung), und als Folge erscheint die Farbmarke w2 im Platz **req.** Dadurch wird das Setzen des Flags REQUEST an der Knotenadresse w2 modelliert. Jetzt gibt es zwei Möglichkeiten: Entweder schaltet die Transition **addplan1** oder die Transition **genown**. Aber zuerst sollte die Transition **addplan1** schalten, wenn auch die andere die Bedingungen im Hauptnetz erfüllt. Die Transition **genown** schaltet, wenn die Zeitverzögerung $t_{request,i} < t_{try,i}$ ist, entsprechend der Aussage im Algorithmus: „Ein Knoten erzeugt einen neuen Wanderer, wenn er zu selten von fremden Wanderern besucht wird. Dies ist der Fall, wenn das Request-Flag gesetzt ist und die Zeit $t_{try,i}$ abgelaufen ist. Dabei muss: $t_{kill,i} << t_{avg,i} << t_{request,i} < t_{try,i}$ gelten.“ Das bedeutet, dass mit dem Schalten der Transition **addplan** der Versuch w2 erfolgreich durchgeführt wird. Wenn die Transition **addplan1** nicht schaltet und die Zeit $t_{try,I}$ (Zeitverzögerung) abgelaufen ist, schaltet die Transition **genown.** Das bedeutet, dass w2 seinen eigenen Wanderer erzeugt, was mit dem Erscheinen der Farbmarke w2 im Platz **genown** modelliert wird. Weiterhin schaltet die Transition **forwown,** und der Platz forwardown erhält die Marke w1. Als nächstes schaltet entweder die Transition **genownnotrequ** (Anfangswert für **treq** bei fehlender Zeitmarke (generate own with no trequest)) bei fehlender Farbmarke der Knotenadresse im Platz **treq**, oder die Transition **genowntreqende** (Rücksetzen von treq) /Tabelle 5.10 / bei vorhandener Farbmarke der Knotenadresse. Die Transition **remreq** (Löschen der anderen Request-Anforderungen des gleichen Knotens nach dem Erzeugen einer Anforderung) schaltet und es verschwindet die Farbmarke der Knotenadresse vom Platz **request**, welche im Platz **reqe** schon bereitsteht (entweder zum Hinzufügen zum Plan eines kommenden Wanderers, oder zum Generieren eines eigenen Wanderers). Aber sie bleibt im Platz **req** für das gleiche Ziel (realisiert durch die Testkante).

Platzunternetz (tavg)

“Die Zeit *tavg,i* gibt die Länge des Intervalls an, in dem der Knoten von einem Wanderer besucht werden möchte. Ein Knoten möchte in Intervallen t_avg,i besucht werden und duldet dabei eine Abweichung **Δ**t. Mit ADD_TO_PLAN(l) und RMV_FROM_PLAN(l) kann der Wanderer sich in dem Plan vorbeikommender Wanderer ein- bzw. austragen und so Korrekturen an dem Besuchsintervall durchführen. (Ergänzung aus der Mail vom 25.11.05)
Dieses Unternetz kontrolliert tavg,i und meldet dem Hauptnetz, welche Knotenadressen zu addieren und welche zu entfernen sind.

Die Plätze **wradd** und **wrrem** empfangen die Farbmarke w1, wodurch modelliert wird, dass Wanderer zum Einstellen der Hinzufügen-Zeit oder der Entfernen-Zeit empfangen werden /nach Tabelle 5.7-5.8 /. Dann schalten gleichzeitig die Transitionen (**wrnoadd)** und (**wrlong**), da alle Bedingungen erfüllt sind.

Möglichkeiten:

- Wenn die Zeit t > (tvg - Δt) ist und sich weder eine Farbmarke w1 noch eine Farbmarke w2 im Platz **wrrem** befindet, schaltet die Transition **trem** und es verschwindet die Farbmarke w1 vom Platz **trem.** Die Farbmarke w2 bleibt aber dort..
- Wenn die Zeit t < (tvg - Δt) ist und der Platz **wrrem** keine Farbmarke w1 enthält, dann schaltet die Transition **wrshort,** und die Farbmarke w1 wird im Platz

remPlan erzeugt.

1) Entweder wartet sie bis alle Bedingungen der Transition **remplan** im Hauptnetz erfüllt sind, und danach verschwindet die Farbmarke w1 mit dem Schalten der Transition **remplan,** oder es schaltet die Transition **remnoo,** je nachdem welche Bedingungen erfüllt ist.
2) Oder es erscheint keine Farbmarke **w1** im Platz **trem,** und es schaltet die Transition **wrshort2.** Damit verschwindet die Farbmarke w1 vom Platz **trem**, damit sie nicht zweimal entfernt wird.
3) Oder die Transition **remadd** schaltet, was bedeutet, dass die Farbmarke w1 in den beiden Plätzen **remplan** und **addplan** enthalten sein soll. Dadurch wird modelliert, dass die Knotenadresse w1 zum Plan des Wanderers hinzugefügt und gleichzeitig aus diesem entfernt werden soll. Die gleichzeitige Durchführung beider Operationen wird mit dieser Transition vermieden. Nach dem Schalten der Transition remadd verschwindet die Farbmarke w1.

- Wenn die Zeit t > (tvg + Δt) ist und der Platz **wradd** keine Farbmarke (z.B. w3) enthält, schaltet die Transition **taddadd2** und der Platz **addplan2** empfängt die Farbmarke w3. Das stellt dar, dass der Knoten w3 zu einem Wanderer hinzufügt werden soll. Nach Verzögerungszeit dt wird die Farbmarke der Knotenadresse zum Hauptnetz exportiert, wie es der Algorithmus verlangt. Nach dem Entfernen der Farbmarke w3 vom Platz **tadd** wird diese durch das Schalten der Transition **wradd** im Platz **tadd** ersetzt.

Kapitel 6 Transformation HCPN-ST in CPN

Modellierung wird als Vorgang verstanden, bei dem ein Modellierer, der einen Sachverhalt in der realen oder gedachten Welt wahrnimmt, auf Basis dieser Wahrnehmung ein Abbild dieses Sachverhaltes konstruiert[Hol99].
Repräsentationsformen für Modelle: die Modelle können entsprechend ihrer Arten repräsentiert werden. Z.B. bildhaft, symbolisch durch mathematische Formeln oder in einer formalen Sprache. In diesem Sinne wurden im vorigem Kapitel zwei Modelle in den formalen Sprachen HCPN-ST und CPN vorgestellt. Die Modelle entstehen als Ergebnis einer Modellierung, in deren Verlauf ausgewählte Aspekte des betrachteten Systems in das Modell übernommen werden. Diese richten sich nach dem Modellzweck.
Netze werden zum Ersten für die Analyse des Verhaltens der entwickelten Methode eingesetzt. Dies erfolgt durch Simulation und formale Analyseverfahren.
Als **Simulation** bezeichnet man das modellhafte Nachbilden eines Systems (z.B. durch ein Programm). Dabei sollen die durch die Simulation gewonnen Ergebnisse in interessierenden Punkten mit denen des originalen Systems übereinstimmen[Mül69].
Simulation von Systemen, die mit einer Klasse von PN modelliert werden, sind nicht nur zum Nachweis der Eigenschaften dienlich. Sie ist auch eine große Hilfe zum Verstehen des modellierten Systems und ist sinnvoll, wenn die Zeit für die Bewertung des Netzes betrachtet wird oder wenn Wissen des Systemablaufs in seiner Umgebung erwünscht ist.

Es gibt für die Analyse von gefärbten Netzen zwei Wege. Der Erste beginnt auf der unteren Ebene mit dem Entfalten des gefärbten Netzes zu einem Petrinetz. Dann wird es mit den Mitteln analysiert, die für Petrinetze bereitgestellt wurden. Die andere Analyse wird auf der höheren Ebene der gefärbten Netzen durchgeführt. Die Invariantenberechnung bildet das wichtigste Beispiel für eine Analyse gefärbter Netze ohne Entfaltung.

Trotz der Vorteile formaler Methoden werden sie nur beschränkt genutzt. Ein Grund dafür ist, dass diese nicht simulierbar oder sehr aufwendige Prozesse sind.
Die Spezifikation kann als Startpunkt für alle Analysemethoden dienen. Sie ist die Schnittstelle zwischen Modellierung der Aktivitäten und Analyseaktivitäten[Gir,Val03]. Wenn die Spezifikation nicht von der Analysemethode abhängig ist, ist die Analyse möglich, ohne das ganze System noch einmal zu modellieren. Aber manchmal sind zusätzliche Analyseaktivitäten der Preis dieser Flexibilität. Die Spezifikation muss in ein Modell transformiert werden, damit das Modell mit der passenden Analysemethode analysiert werden kann. Deshalb ist es wichtig, ein Tool zu finden, dieses Tool kann die Transformation unterstützen und ermöglicht die Transformation zwischen der Spezifikation, dem analytischen Modell und dem Resultat der Analyse des CPN. Das erlaubt die Vereinigung von Analysetechniken in verschieden Anwendungsdomainen.

Im Kapitel 5 wird eine formale Methode dargestellt, die erweiterten gefärbten Petrinetze (HCPN-ST) zum spezifizieren, analysieren und verifizieren verteilter paralleler Systeme nutzen zu können. Dann wird HCPN-ST am Beispiel erläutert. Danach wird mit einem CPN Beispiel ein System (mit nur drei Knoten) modelliert, weil die CPN Methode bei großen Systemen (höhere Zahl von Knoten) aufwendig bis unfähig wird. Erweiterte HCPN-ST haben die Fähigkeit und Sichtbarkeit und sind leichter zu verstehen. Aber um dieses Netz zu simulieren, analysieren und verifizieren soll ein Tool gefunden werden. Die existierenden Tools können einige Klassen von Petrinetzen analysieren, verifizieren und simulieren (bzw. Teile davon).

Als Ergebnis der Suche nach dem passenden Tool wurde herausgefunden, dass das PENECA chromos mit der Integration des Tools INA die Lösung sein kann. (Nach der Transformation HCPN-ST zu CPN).
Im folgenden wird der Algorithmus dieser Transformation vorgestellt. Danach wird der Algorithmus am Beispiel erläutert. Es folgt das Analysieren und Verifizieren dieses Modells mit den genannten Tools. Diese wurden in Kapitel 4 kurz vorgestellt.

6.1 Transformation eingeschränkter HCPN-ST in Peneca Chromos (gefärbtes Petrinetz)

Farbaufbau (Farbkonvertierung)

Das Farbset lautet wie folgt:
Zeiger = {1,2,3,4}
Länge = 4
Knoten Adresse = {n1,n2,n3}
Positionen des Plans = A = {n1,n2,n3,e,r}
Fm = {e,r,n1,n2,n3}

$$wm = (zeiger \times Länge \times Positionen)$$
$$= \{\, wm = \{(z, x, vm) \mid z \in zeiger, x \in Länge, vm \in Plan\}\,.$$

Es gibt 480 verschiedene Verbundfarbmarken

$|Zeiger| \times |Länge| \times |positionen| = 4 \times 1 \times 5! = 480$ das heißt, die Zahl der Verbundfarbe ist gleich S.

$$S = z \times 1 \times n \times (n+1)^{(x-2)} \times 1 = x \times n \times (n+1)^{(x-2)}$$

Wobei z= Maximum Werte z = x Länge der Wanderer.

Algorithmus der Transformation

- **Transformation strukturierte Farbe in CPN Farbe**

Für alle Verbundmarken mit dem Vektor (1,4, (ni,e,e,r)) wird die chromos Farbe 14nieer genommen. Es wird also von der Verbundfarbmarke die Chromos Farbmarke als Zeichenkette nach der Entfernung der Kommas und Klammern des Vektors verwendet.
Entsprechend dem Wanderer-Aufbau in diesem konkreten Beispiel dürfen die ersten Plätze nur die Knotenadresse, die Zweiten und Dritten entweder Knotenadresse besitzen oder e (frei) sein. Der Vierte kann nur r (zufällige Sendung) sein.

$$4 \times 1 \times 3 \times 4 \times 4 \times 1 = 192\, Verbundfarbmarke$$

wm (z, x, vm): für z gibt (4) Möglichkeiten, für x nur eine Möglichkeit (1), für die erste Position im Plan des Wanderers gibt es 3 Möglichkeiten, entweder n1 oder n2 sowie n3, für die zweite und dritte Position 4 Möglichkeiten, weil die Farbmarke e dazu kommt. Die vierte Position kann nur die Farbmarke r sein (entsprechen dem Wanderer).

Die folgende Tabelle dominiert die Chromos Farbmarken für unser vorheriges Beispiel:
Hier werden die Möglichkeiten eines Wanderers mit Länge 4 auf einem P2P-Netzwerk mit 3 Knotenadressen erklärt.
Wenn die erzeugte Knotenadresse z.B. n1 ist, ergibt sich die folgende Verbundmarke:

(1, 4, n1, n1, n1, r)	(1, 4, n1, n2, n2, r)	(1, 4, n1, n3, n3, r)	(1, 4, n1, e, e, r)
(1, 4, n1, n1,n2, r)	(1, 4, n1, n2, n1, r)	(1, 4, n1, n3, n2, r)	(1, 4, n1, e, n1, r)
(1, 4, n1, n1, n3, r)	(1, 4, n1, n2, n3, r)	(1, 4, n1, n3, n1, r)	(1, 4, n1, e, n2, r)
(1, 4, n1, n1, e, r)	(1, 4, n1,n2, e, r)	(1, 4, n1, n3, e, r)	(1, 4, n1, e, n3, r)

Z = 1, Pos1= n1 ergibt 16 Verbundfarbmarken
Z =2, Pos1= n1 ergibt 16 Verbundfarbmarken (diese sind gleich den oben dargestellten, die 1wird mit einer 2 ersetzt)
Z = 3, Pos1= n1 ergibt 16 Verbundfarbmarken (diese sind gleich den oben dargestellten, die 1 wird mit einer 3 ersetzt)
Z = 4, Pos1= n1 ergibt 16 Verbundfarbmarken (diese sind gleich den oben dargestellten, die 1 wird mit einer 4 ersetzt)
Es gibt daher 4 * 16 = 64 Verbundmarken für n1.
Nach der gleichen Methode werden für n2 und n3 die Wanderer Farben erstellt. Daraus folgen 3 * 64 = 192 unterschiedliche Verbundfarbmarken.

Chromos Farbaufbau entsprechend dem Algorithmus :
Die Farbmarken sollen transformiert werden, um sie in Peneca Chroms (CPN) darzustellen.

Tabelle 6.1 Transformation der Verbundfarbmarke

	Verbundfarbmarke	Chromos-Farbe
1	(1, 4, n1, n1, n1, r)	14n1n1n1r
2	(1, 4, n1, n1,n2 , r)	14n1n1n2r
3	(1, 4, n1, n1, n3, r)	14n1n1n3r
4	(1, 4, n1, n1,e , r)	14n1n1er
5	(1, 4, n1, n2, n2, r)	14n1n2n2r
6	(1, 4, n1, n2, n1, r)	14n1n2n1r
7	(1, 4, n1, n2, n3, r)	14n1n2n3r
8	(1, 4, n1,n2 ,e ,r)	14n1n2e r
9	(1, 4, n1, n3, n3, r)	14 n1n3n3r
10	(1, 4, n1, n3, n2, r)	14n1n3n2r
11	(1, 4, n1, n3, n1, r)	14n1n3n1r
12	(1, 4, n1, n3, e , r)	14n1n3er
13	(1, 4, n1, e, e, r)	14n1eer
14	(1, 4, n1,e ,n1,r)	14 n1en1r
15	(1, 4, n1,e ,n2, r)	14n1en2r
16	(1, 4, n1,e ,n3 ,r)	14n1en3r
17	(2, 4, n1, n1, n1, r)	24n1n1n1r
18	(2, 4, n1, n1,n2 , r)	24n1n1n2r
19	(2, 4, n1, n1, n3, r)	24n1n1n3r
20	(2, 4, n1, n1,e , r)	24n1n1er

21	(2, 4, n1, n2, n2, r)	24n1n2n2r
22	(2, 4, n1, n2, n1, r)	24n1n2n1r
23	(2, 4, n1, n2, n3, r)	24n1n2n3r
24	(2, 4, n1, n2, e, r)	24n1n2e r
:	:	:
:	:	:
192	:	:

Farbmarke in Peneca Chroms

Chromos-Marke	**Farbmarke**
n1 1	n1
n2 1	n2
n3 1	n3

- **Transformation der Symboltabellen Netzmodell (5.3)**

ni: Knotenadressen ,wm: Wanderer (z;x;pos1,pos2,.,,,..,r)
Posz= nz Zielknoten
pos1 : enthält Erzeugerknoten
pos2..pos(x-1) enthält entweder ni oder e (leerer Platz in Wanderer)
z: Zeiger auf Element in w =1…4
x: Länge der Wanderer: Anzahl Elemente in w = 5
r: Platz in dem Plan eines Wanderers für zufälliges Senden.

Symboltabelle (wr) in HCPN-ST

Schaltmodus	**fm**	**wm(posz=ni)**	**i**
ni	ni	wm(posz= ni)	1…n
	Knoten ist frei	Knoten ist Zielknoten	

Symboltabelle (wr) in CPN

Schaltmodus	**fm**	**wm(nz(ni))**	**i**
ni	ni	Zxn1..posz(ni)..r	1…n
	Knoten ist frei	Knoten ni ist Zielknoten	

Symboltabelle (kill) in HCPN-ST

Schaltmodus	**wm**	**wm(z=1&(posz=fm))**	**fm**	**i**
wm	wm	wm(z=1 & posz=ni)	ni	1..3
	Wanderer des Knotens löschen	Knoten ist Zielknoten und Erzeugerknoten		

Symboltabelle (kill) in CPN

Schaltmodus	**wm**	**wm(z(1)&posz(fm))**	**fm**	**i**
wm	14ni..r 1	14ni..r 1	ni	1..3

	Wanderer des Knotens löschen	Knoten ist Zielknoten und Erzeugerknoten		

Symboltabelle (nodetoremovetimer) Verzögerungszeit <tavg-Δt

Schaltmodus	**fm**	**i**
ni	ni	1..3
	Knoten zum entfernen, vor kurzer Zeit besucht wird	

Weil die Kante eine Testkante ist, beeinflusst dies den Platz emptynode nicht. Die Transition ist schaltfähig, wenn die Verzögerungszeit beendet ist. Aber in dieser Zeit verliert sie ihre Konzession nicht.

Symboltabelle (remov fromplan) in HCPN-ST

Schaltmodus	**wm(z>1&posz=ni)**	**fm**	**wm (posz:=e)**	**i**
wm	wm(z≠1 & posz=ni)	ni	wm(posz=e)	1..3
	Knoten ist Zielknoten und nicht Erzeugerknoten und in der aktuellen Position des Wanderers enthalten	Aktueller Knoten, vor kurzer Zeit besucht worden	Knoten aus Wanderer entfernen	

Symboltabelle (remov fromplan) in CPN

Schaltmodus	**wm(z(m1&posz(ni))**	**fm**	**wm (posz:=e)**	**i**
wm(2,4,n1,n2,n3,r) wm(2,4,n1,n3,e,r)	24,n1n2 n3r 1 24n1n3er 1 :	n2 1 n3 1 :	24n1en3r 1 24n1eer 1 :	2 3 :
wm(3,4,n1,n2,n3,r) wm(3,4,n1,n3,n2,r)	34,n1n2 n3r 1 34n1n3n2r 1 :	n3 1 n2 1 :	24n1en3r 1 24n1eer 1 :	3 2 :
	Knoten ist Zielknoten und nicht Erzeugerknoten und in dem aktuellen Wandererplatz enthalten	Aktueller Knoten, vor kurzer Zeit besucht worden	Knoten aus Wanderer entfernen	

Symboltabelle (nodetoaddtimer) Zeitdauer ≥tavg-Δt

Schaltmodus	**fm**	**i**
ni	ni	1..3
	Knoten (to add), sie werden als Requestflag gesetzt, und versuchen sich bei einem kommenden Wanderer einzutragen.	

Die Transition **nodetoaddtimer** ist eine zeitbewertete Transition.

Sie ist schaltfähig, wenn tavg-Δt beendet ist. Aber in dieser Zeit verliert sie ihre Konzession. Sie stört damit den kommenden Wanderer (mit aktueller Knotenadresse in gleicher Farbe) nicht. Weil die Kante eine Testkante ist beeinflusst dies den Platz **emptynode** nicht. Im Beispiel wird die Zeit als bestimmter Wert in jeder Simulation festgelegt.
Besser könnte diese Transition könnte als Transition mit einem Schaltintervall [tavg, tavg+Δt] definiert werden. Aber dann muss das Petrinetz-Tool für diese Klasse von Petrinetzen geeignet sein.

Symboltabelle (timerover) Zeitdauer = 2Δt

Schaltmodus	**fm**	**i**
ni	ni	1..3
	Knoten entfernen, der vor kurzer Zeit besucht wurde	

Die Transition **timerover** ist eine zeitbewertete Transition. Sie ist schaltfähig, wenn t= 2Δt beendet ist. Aber in dieser Zeit verliert sie ihr Konzession. Somit stört sie den kommenden Wanderer (mit aktueller Knotenadresse in gleicher Farbe) in der Transition removefromplan nicht. Die Transition lässt die Knotenadresse von Platz **nodetoremove** verschwinden. Weil diese Knotenadresse nach dieser Zeitdauer verbraucht wird, darf dieser Knoten nicht mehr vom Wanderer entfernt werden.

Symboltabelle (noremove) in HCPN-ST

Schaltmodus	**wm(posz=ni)**	**fm**	**i**
wm	wm(z,x,pos1..pos(z)=ni)	kein ni	1..3
	Der Knoten soll aus Wanderer nicht entfernt werden und der Wanderer geht weiter		

Symboltabelle (noremove) in CPN

Schaltmodus	**wm(posz=ni)**	**fm**	**i**
wm	24,n1n2 n3r 1 24n1n3er 1 :	kein n2 kein n3	2 3
	Der Knoten soll aus Wanderer nicht entfernt werden und der Wanderer geht weiter		

Symboltabelle (addtoplan, alternativ: Eintragen auf einen leeren Platz) in HCPN-ST

Schaltmodus	**wm(posz=e |pos(z±1)=e)**	**fm**	**wm(first (posi=e):=fm &z:=i mod x)**	**i**
wm	wm(z,x,..Posz(e)..,r)	ni	wm (z:=i mod x & first(e):=fm	1..3
	Aktueller Platz im Plan ist leer	Knoten in Wanderer hinzufügen aufgrund tavg	Zeiger i mod x ersetzt, Inhalt von Zeiger wird Zielknoten, einen leeren Platz durch Knoten ersetzen	

Symboltabelle (addtoplan, alternativ: Eintragen auf einen leeren Platz) in CPN

Schaltmodus	**wm(posz=e | pos(z±1)=e)**	**fm**	**wm(first (posi=e):=fm &z:=i mod x)**	**i**
wm(3,4,n1,e,e,r) wm(2,4,n1,e,n1,r) :	34 n1e er 1 24n1en1r 1 :	n2 1 n3 1	34n1n2er 1 24n1n3n1r 1 :	2 2
	Wanderer mit leerem Platz	Knoten in Wanderer hinzufügen aufgrund tavg	Zeiger i mod x ersetzt, Inhalt von Zeiger wird Zielknoten, einen leeren Platz durch Knoten ersetzen	

Symboltabelle (addno) in HCPN-ST

Schalt modus	**fm**	**wm(z=x & posi ≠e)**	**i,j,k**
wm	ni	wm(z=x,x,ni,nj,nk,ni) | wm(z,x, ni,.nj,nk,nj)	1..3
	Knoten in Wanderer nicht hinzufügen	voller Wanderer	

Symboltabelle (addno) in CPN

Schaltmodus	**fm**	**wm(z=x & posi ≠e)**	**i,j,k**
wm(1,4,n1,n3,n2,r) wm(4,4,n1,n3,n2,n1)	n1 1 n3 1	14n1n3n2r 1 44n1n3n2r 1	1..3
	Knoten in Wanderer nicht hinzufügen	voller Wanderer	

Symboltabelle (normforward) in HCPN-ST

Schaltmodus	**wm(posz≠r & posz ≠e)**	**wm**	**i**
wm	wm (posz = ni)	wm (posz=ni)	1..3
	Aktuelle Position in dem Plan der Wanderer ist mit einer Knotenadresse besetzt also nicht frei und wird nicht zufällig ermittelt	Weitersenden an Zielknoten .	

Symboltabelle (normforward) in CPN-ST

Schaltmodus	**wm(posz≠r & posz ≠e)**	**wm**	**i**
wm(1,4,n1,n3,n2,r) wm(3,4,n1,n3,n2,n1)	14n1n3n2r 1 34n1n3n2r 1 :	14n1n3n2r 1 34n1n3n2r 1 :	1..3
	Aktuelle Position in dem Plan der Wanderer ist mit einer Knotenadresse besetzt also nicht frei und wird nicht zufällig ermittelt	Weitersenden an Zielknoten .	

Symboltabelle (random) in HCPN-ST

Schaltmodus	**wm(z=r)**	**wm(posz:=random(ni)**	**i**
wm	wm(z=r)	wm(Posz = rando(ni))	1..3
	Wanderer, für den der Zielknoten zufällig gebildet werden soll.	wird zufällig an den Zielknoten (auf posz) geschickt	

Symboltabelle (random) in CPN

Schaltmodus	**wm(z=r)**	**wm(posz:=random(ni)**	**i**
wm(4,4,n1,e,n2,r) wm(4,4,n1,e,n2,r) wm(4,4,n3,n2,n1,r) wm(4,4,n3,n2,n1,r) :	44n1en2r 1 44n1en2r 1 44n3n2n1r 1 44n3n2n1r 1 :	44n1en2n1 1 44n1en2n3 1 44n3n2n1n3 1 44n3n2n1n21 :	1..3
	Wanderer, für den der Zielknoten zufällig gebildet werden soll.	wird zufällig an den Zielknoten (auf posz) geschickt	

Mit der Transition random wird durch verschiedene Schaltmode mit verschiedenen Wahrscheinlichkeit realisiert. Wobei eine höhere Wahrscheinlichkeit für die Knoten, die noch nicht im Plan danach kommt, die nicht öfter im Plan vorkommt und so weiter, definiert wird.

Symboltabelle (nettime) in HCPN-ST

Schaltmodus	**wm**
wm	wm
	Wanderer direkt weiterleiten (keine Funktion

Symboltabelle (nettime) in CPN

Schaltmodus	**wm**
Wm (1,4,n1,n1,n1, r)	14n1n1n1r 1
wm(1,4, n1,n1,n2,r) :	14n1n1n2r 1 :

nettime Transition ist eine Transition mit Zeitverzögerung. Diese verzögert das Zirkulieren der Wanderer im Netz, um die Verzögerungzeiten im realen System darzustellen.

Symboltabelle next in HCPN-ST

Schaltmodus	**wm**	**wm(z:=z+1)**
wm	wm (1,4,n1,n1,n1, r)	wm (2,4,n1,n1,n1, r)
wm :	wm(1,4, n1,n1,n2,r) :	wm (2,4,n1,n1,n1, r) :
	Wanderer wird bearbeitet	Wanderer zum nächsten Nachfolger weiterschicken

Symboltabelle next in CPN

Schaltmodus	**wm**	**wm(z:=z+1)**
wm (1,4,n1,n1,n1, r)	14n1n1n1r 1	24n1n1n1r 1
wm(1,4, n1,n1,n2,r) :	14n1n1n2r 1 :	24n1n1n2r 1

Symboltabelle (wrgenerated) in HCPN-ST

Schaltmodus	**wm**	**i**
ni	wm ; t >t try	1..3
	Der Konten ni geniert seine eigenen Wanderer	

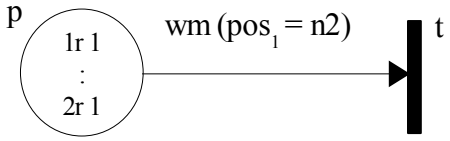
Die Transition **wrgenerated** ist eine zeitbewertete Transition. Sie ist schaltfähig, wenn t= t try beendet ist. In dieser Zeit verliert sie ihre Konzession, damit ein Knoten sich in einem kommenden Wanderer eintragen kann. Aber wenn die Zeit vorbei ist, schaltet die Transition und der Knoten generiert seinen eignen Wanderer.

Symboltabelle (wrgenerated) in CPN

Schaltmodus	**wm**
n1	14n1eer 1
n2	14n2eer 1
n3	14n3eer 1

Die Transformation der Schaltregel erfolgt wie bei der Schalttabelle in gleicher Art und Weise. Ich stelle das am Beispiel vor:

wm (pos_i = n2) $\forall wm \in Wanderer \exists wm \in m(p): wm(pos_1 = n2)$



Schaltfähig: Es existiert mindestens eine Verbundmarke wm in m(p), die an Position 1 des Plans der wm die Knotenadresse n2 enthält.
Nach dem Schalten der Transition t werden al1e Verbundmarken entzogen, deren Position 1 die Knotenadresse n2 enthalten.

Chromos Symboltabelle t

Schaltmodi	Pos1= n2
	14n2eer
	14n2n3er
	14n2en1r
	14n2n3n1r
	24n1n2er
	24n2n3n1r|
	34n2en1r
	34n2n3n1r
	44n2eer
	44n2eer|
	44n2n3er
	44n2n3er
	44n2en1r
	44n2en1r
	44n2n3n1r
	44n2n3n1r

Kapitel 7 Verifikation der HCPN-ST und CPN Modelle

7.1 Verifikation

Verifikation wird zum Überprüfen der Korrektheit eines Entwurfs (z.B. von verteilten Informationssystemen) und zum Nachweis von Fehlfunktionen eingesetzt. Dabei kann ein Modell des Systems durch Verifikation auf bestimmte Eigenschaften, z.B. die Existenz von Nebenläufgkeiten 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[VTR00]:
”Verifikation ist die Überprüfung des Wahrheitsgehaltes eines Modells gegen seine Spezifikation,“oder ” der Nachweis durch formale oder analytische Verfahren, dass die Spezifikation eines Systems oder Methode konsistent und vollständig ist.“
Es gibt verschiedene Arten und Möglichkeiten zur Verifikation von verteilten Informationssystem. Diese unterscheiden sich in Komplexität und Aussagekraft.
Dazu gehören Simulation, Test, deduktive Methoden (Theorem Proving) und Model Checking oder formale Verifikation an Hand von formalen Modellierungsmitteln wie z.B. Petrinetzen.
Unter Simulation versteht man „die Nachbildung eines dynamischen Prozesses in einem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.”. Simulation dient dem Zweck, eine Prognose über das Verhalten des modellierten Systems zu erhalten. Um eine 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.
In dieser Arbeit wird eine formale Methode, einer erweiterter Klasse von Petrinetzen am Beispiel erläutert. Danach wird dieses Beispielmodell von Hand simuliert und analysiert. Damit wird die Fähigkeit dieser Methode bei der Beschreibung verteilter Systemabläufe sowie des dynamischen Verhaltens gezeigt.
Zum Simulation der vorgestellten Modelle wird Peneca Chromos eingesetzt.

7.2 Ergebnisse der Simulation und Analyse INA Tool

1. Modell 5.4 :

- Das Model ist **konfliktfrei**. (statisch und dynamisch)
- **Lebendigkeit** : Das Modell ist lebendig. Selbst nach langer Zeit der Simulation und mehrmaligem Durchführen wird kein Meldung totes Netz in Peneca chromos gezeigt. Mit INA wird das Modellnetz mit schwach lebendig beschreibt. Das ist normal, weil die meisten gefärbten Petrinetze schwach lebendig sind, weil ein gefärbtes Netz dadurch gekennzeichnet ist, dass mehrere Farben zu einer Transition zusammengefasst sind und durch eine Farbe die Transition nicht lebendig ist.
- Beschränktheit: Das Netz ist beschränkt und unsicher.
- Erreichbarkeit : 75 Zustande zeigen keine unerwünschten Fälle.

2. **Modell 5.3:** Das Modell wird simuliert. Es verhält sich wie erwartet.
   - Das Modell ist **konfliktfrei (**statisch und dynamisch)
   - **Lebendigkeit** : Das Model ist lebendig. Nach langer Zeit der Simulation und mehrmaligem Durchführen wird kein Meldung totes Netz in Peneca chromos gezeigt. Mit INA wird das Modellnetz mit schwach lebendig beschrieben.
   - Beschränktheit: Das Netz ist beschränkt und unsicher.
   - Erreichbarkeit: Es sind mehr Zustände als in 5.4 vorhanden. Diese benötigt man später noch zur Prüfung des Zeitkonzeptes, Dieses wird im folgenden Abschnitt vorgestellt.

Analyseverfahren

Herkömmlich werden **Analyseverfahren für Petrinetze** wie folgt klassifiziert[Gir,Val03]:

1.Aufzählung (Enumeration):

Diese Methode basiert auf dem Aufbau des Erreichbarkeitsgraphen. Wenn das Netzsystem beschränkt ist, ist auch Erreichbarkeitsgraph endlich und unterschiedliche qualitative Eigenschaften können überprüft werden. Aber trotz ihrer Mächtigkeit ist diese oft schwer anzuwenden.

2.Transformation:

Ein Netz N wird transformiert in N1, um einige Eigenschaften zu verifizieren und weil das in N1 leichter ist als in N ist oder N1 vielleicht zu einer Unterklasse gehört, in der Zustände mit Aufzählung vermieden werden können. Die Verkleinerungsmethode ist eine spezielle Klasse der Transformationsmethode, wobei eine Sequenz von Netzsystemen aufgebaut wird und alle die Eigenschaften erhalten, die verifiziert werden müssen. Aber das transformierte Netzsystem ist kleiner (z.B. hat die Markierung weniger Marken) als das vorherige Netzsystem. Diese Methode ist begrenzt, wenn das Netzsystem nicht mehr reduzierbar ist. Wegen der Existenz elementarer Netzsysteme muss diese Methode mit anderen Methoden komplementär sein.

3. Strukturanalyse:

Diese Methode erforscht die Beziehung zwischen dem Verhalten des Netzsystem und seiner Struktur und benutzt die Markierung als Parameter. Diese Methode unterteilt sich in zwei Gruppen: 1. Linear Algebra basierend auf den Netzzustand-Gleichungen (Linear Invariante)
2. Technik basierend auf einem Graph (Erreichbarkeitsgraph).

7.3 Beschreibung zeitlicher Anforderungen mit formalen Mitteln

Der Einsatz formaler Beschreibungsmittel und Methoden beim Systementwurf kann die Zuverlässigkeit solcher Systeme erhöhen. Formal definierte Syntax und Semantik sind die Voraussetzungen für eine eindeutige Systembeschreibung. Mit Hilfe mathematischer Verfahren können Verhaltenseigenschaften verifiziert, d.h. mittels eines mathematischen Beweises nachgewiesen werden. Im Gegensatz dazu bezeichnet die Validierung lediglich eine Überprüfung, ob die Spezifikation oder das Programm die geforderten Eigenschaften erfüllen oder nicht.

Durch Verifikation oder Validierung einer Systemspezifikation können frühzeitig Fehler oder unvollständige Anforderungen aufgedeckt werden. Fehler, die nicht vor der Implementierung aufgedeckt werden, können später sehr hohe Kosten verursachen.

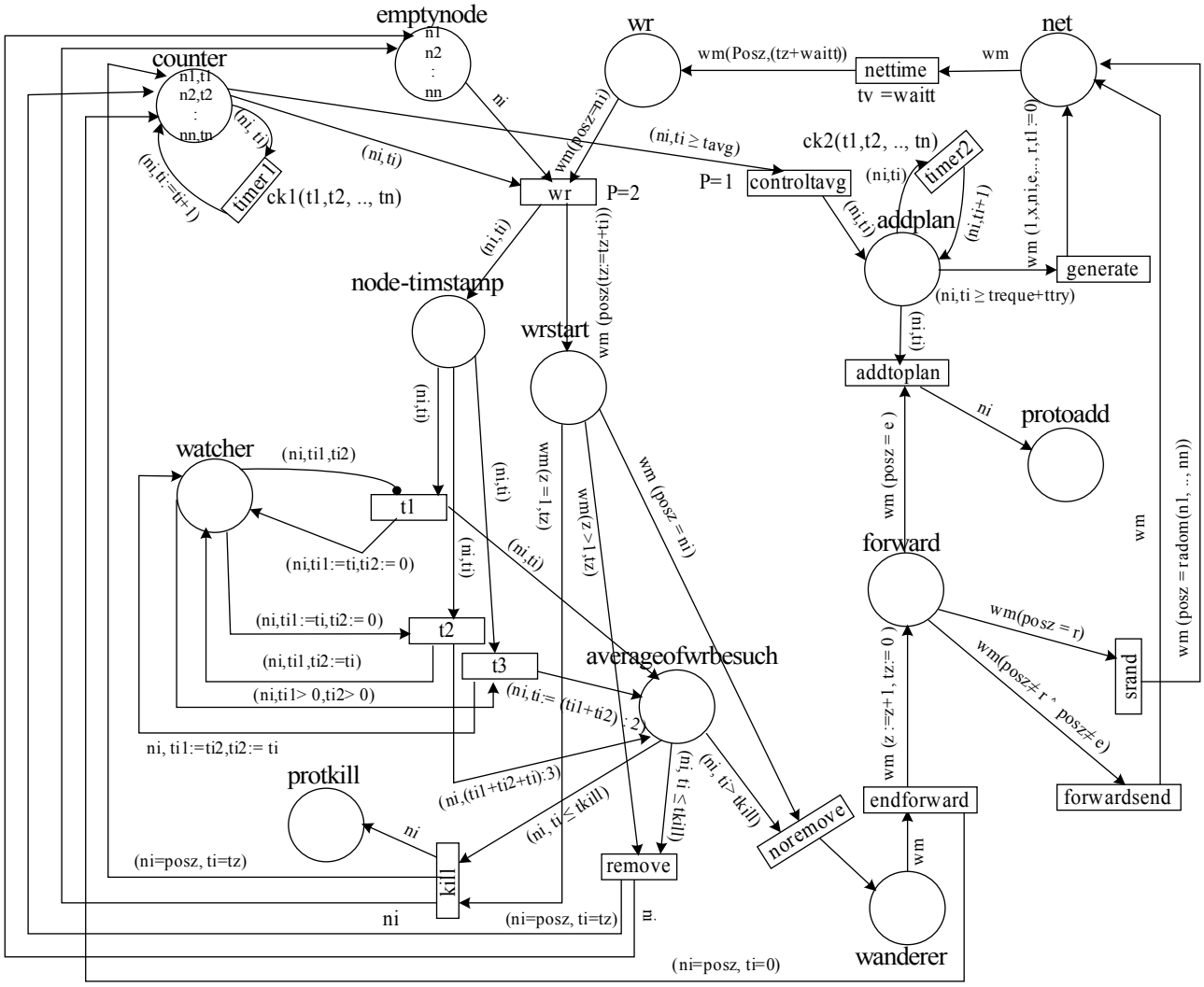


Abbildung 7.1:Modell des Algorithmus der nicht Disjunkten Cluster

Diese Arbeitet beschäftigt sich mit parallelen verteilten Systemen und insbesondere P2P Systeme. Besonders die Inhomogenität und Dynamik realer P2P-Netze erschweren die Erstellung eines mathematischen Modells für diese Systeme. Außerdem müssen viele Einflüsse berücksichtigt und somit ein Modell mit vielen Parametern entwickelt werden. Hinzu kommt, dass nicht nur Parameter auf P2P-Ebene die Funktion beeinflussen sondern auch die physikalischen Schichte. Alle genannten Parameter müssen in ein solches Modell einbezogen werden. Eine mögliche Lösung für dieses Problem ist eine Simulation, um das Verhalten der Strukturbildungsalgorithmen in einem großen Netz zu untersuchen. Anhand von Simulationen können kostengünstig große Systeme erzeugt und Parameter gezielt verändert werden. Die Zeit spielt eine große Rolle in diesem Algorithmus. Deshalb könnte eine formale, graphische, durchführbare Technik eine Alternative sein, um die Spezifikation und Analyse der nebenläufigen Umgebung des Verhaltens des Algorithmus in unterschiedlich großen Netzen genauer zu untersuchen.

Die meisten Anwendungen der gefärbten Petrinetze werden zur Überprüfung der logischen Korrektheit von Systemen benutzt. Das bedeutet, dass die dynamischen Eigenschaften und die Funktionalität berücksichtigt werden. Ferner wird die Performance des Systems mit gefärbten Petrinetzen untersucht, zum Beispiel die maximal benötigte Zeit zur Durchführung der Aktivitäten durchzuführen. Um diese Art von Analyse zu ermöglichen, soll das gefärbte

Petrinetz Modell mit einem Zeitkonzept erweitert werden. Das wird durch die Erweiterung jeder Marke um einen Zeitstempel neben der Farbe realisiert. Die Zeitstempel geben genau die Zeiten an, wann die Marken an der Transition bereit sind. Diese Zeit kann in alle Arten angeben werden (konstant oder Intervall). Sie beschreibt wann jede Marke an der Transition zur Verfügung steht. Nach dem Schalten der Transition werden neue Marken mit Zeitstempel erzeugt. Diese Zeitstempel hängen vom Kantenausdruck und der Transition ab, die die Marke erzeugt. Durch die Verwendung einer Simulationsumgebung können die gewonnenen Ergebnisse repliziert und verschiedene Messungen unter den gleichen Bedingungen durchgeführt werden.

Um bessere Ergebnisse zu bekommen, wird ein anderes Modell vorgestellt. Es basiert auf erweiterten HCPN-ST mit Zeitstempel und verschiedenen Konzepten. Folgend wird eine Simulation per Hand durchgeführt, um die Zeitkonzepte zu demonstrieren und gleichzeitig das Verhalten des Algorithmus zu untersuchen und Aufschluss über die genaue Arbeitsweise des Algorithmus zu liefern. Dazu gehören die Bewegung der Wanderer durch das Netz und die durch die Bewegung der Wanderer aufgebauten Strukturen (Clustern) .

Die Modellbeschreibung

Das Konzept: Es gibt einen Platz **Counter.** Dieser enthält alle Knoten des Netzwerkssystems. Seine Markenfarbe ist ein Tupel mit der Knotenadresse und dem inneren lokalen Zeitwert wie (n1,t1). T1 ist eine lokale Uhr und schaltet mit der Meldung dieser Knoten. Am Anfang ist t1=0 (mit Beginn der Simulation). Die Transition **Timer 1** schaltet jede Zeiteinheit für alle Knoten, die sich im Platz **Counter** befinden. Wenn der Knoten ni vom Platz **Counter** entfernt wird, schaltet sich seine Uhr aus. Dieser Timer erhöht mit jedem Schalten die Zeitwerte für zu treffende Knoten um 1. Diese Zeitwerte sind die Zeitstempel in diesem Modell. Timer 2 arbeitet nach dem gleichen Prinzip. Diese Zeitstempel begleiten jede Knotenmarke und werden im Platz **watcher** benutzt. Die Zeitwerte werden zu (t1,t2). Dabei wird t1 als Speicher für alte Zeitwerte und t2 für aktuelle Zeitwerte verwendet.

Eine Verbundmarke hat einen Zeitstempel, der zu der aktuellen Knotenadresse (Zielknoten) im Plan gehört.

Simulation für das ganze Modell (für Wanderer mit der Länge 5 und einer zufälligen Sendung): Wir nehmen an, dass ein Knoten n1seinen eigenen Wanderer generiert. Die Wanderer-Verbundmarke wm (1,4,n1,e,e,e,r,tz) wandert im Netz. Die Transition **nettime** mit der Zeitverzögerung tv=wait t ändert die ankommende Verbundmarke. Sie wird zu (1,4,n1,e,e,e,r,tz+wait t) und geht weiter bis zur ersten Kontrolle. Ist der Zielknoten (hier ist n1 der erzeugte Knoten und Zielknoten) frei, geht es weiter mit dem Schalten der Transition **wr**. Transition **wr** ändert die Verbundmarke zu wm (n1,e,e,e,e,r, tz=tz+t1).

t1 wird von der Marke (n1,t1) genommen. Gleichzeitig wird die Marke (n1,t1) zur Bearbeitung geschickt. Wenn n1 zum ersten Mal besucht wird (also kommt), schaltet die Transition **t1**, weil noch keine Marke im Platz **watcher** ist**.** Der Knoten ist schon angemeldet und hat im Wacher die Belegung (n1,t1,0). Aber beim nächsten Besuch schaltet die Transition **t2** und diesmal wird die Belegung auf (n1,t1,t2) gesetzt. Beim nächsten Besuch schaltet die Transition **t3** und es werden die Zeitwerte t1 mit den Zeitwerten t2 ersetzt und die neu ankommenden Zeitstempel werden in t2 gespeichert.

Mit diesen Mechanismen haben wir immer 3 Zeitwerte für die drei letzten Besuche. Aus diesen wird der Durchschnitt berechnet und zum Kontrollieren der Zeit des Wanderers benutzt. Dies sieht man in den Vorkantenanschriften der Transitionen **kill, remove, noremove.** Der weiter Verlauf der Simulation weiter ist wie bei Modell 5.3.

Mit dieser Simulation werden die Anforderungen an eine Simulation erfüllt.

In einer Simulation soll eine reale Umgebung nachgebildet werden. Es gibt viele Parameter, die in einem P2P-System eine Rolle spielen:

- Reale P2P-Systeme sind meistens heterogen aufgebaut. Dies hat insbesondere eine unterschiedliche Leistungsfähigkeit der einzelnen Peers zur Folge.
- Die Netzanbindung der einzelnen Peers ist oft sehr unterschiedlich. Es kann zu uneinheitlichen Verzögerungen der im System versandten Nachrichten kommen.
- Peers können das System verlassen oder es kommen neue hinzu, ohne dass es eine An- bzw. Abmeldeprozedur gibt.

Das Modell erfüllt die meisten Anforderungen. Was damit nicht realisiert wird, kann eine andere Methode oder die Erweiterung dieser Methode erfüllen.

Kapitel 8 Zusammenfassung und Ausblick

Petrinetze werden in vielen Bereichen als Modellierungstechnik verwendet. Die verschiedenen Einsatzgebiete und Modellierungsziele erfordern dabei unterschiedliche Klassen von Petrinetzen. Eine Petrinetzklasse kennzeichnen neben den üblichen Plätzen,Transitionen und Kanten eine Anzahl zusätzlicher, spezifischer Elemente sowie eine spezifische Schaltregel.
In der Literatur findet man zahlreiche verschiedene Petrinetz-Typen. Diese Vielfalt an Petrinetz-Typen lässt sich nicht ohne weiteres überblicken.
Einfachheit, Modularität und Flexibilität der Modellbildung sowie Anschaulichkeit und relativ leichte Erlernbarkeit zeichnen Petrinetze aus.
In dieser Arbeit werden verteilte Parallele System betrachtet. Diese Systeme gewinnen zunehmend an Bedeutung und sind ein aktuelles Forschungsgebiet. Besonders P2P Anwendungssysteme spielen für Kommunikationssysteme eine große Rolle. Die Algorithmen, Protokolle und Mechanismen sind der wesentliche Teil dieser Wissenschaft. Dieser Teil gewinnt zusätzlich an Bedeutung. Zur Entwicklung von Protokollen und Algorithmen fehlen immer noch formale Methoden zur Spezifikation und Verifikation.
Eine formale Beschreibung kann die Eindeutigkeit eines Spezifikationsmodells durch die Verwendung einer definierten Syntax und Semantik gewährleisten.Formale Beschreibungen können die frühzeitige Fehlerentdeckung ermöglichen, wenn sie einer rechnergestützten Untersuchung bzw. Verifikation zum Zweck des Eigenschaftsnachweises eines Modells zugänglich sind.

Im Vergleich zu einem Formalen Modell lassen höhere Programmiersprachen die Struktur einer zeitlich parallelen Kommunikation jedoch nicht oder nur sehr schwer erkennen.. Ein weiterer wesentlicher Nachteil ist jedoch, dass für diese Spezifikationsmodelle mit Programmiersprachen keine Analyseverfahren angewendet werden können, die einen Nachweis von für Kommunikationssysteme wichtigen Eigenschaften (z.B.Verklemmungsfreiheit) liefern. Ein passende Modellierungssprache wie das Petrinetz kann diese Nachteile beseitigen.

Aufgrund dieser gesteigerten Anforderungen an die Modellierung ist die Entwicklung einer formalen Sprache Ziel dieser Arbeit.

Die vorliegende Arbeit stellt eine formale Spezifikationsmethode für verteilte Systeme vor.

Diese Methode wurde entwickelt als formale Methode zur Modellierung der Algorithmen, die auf Kommunikationssystemen arbeiten. Sie kann verschiedene Arten von Nachrichtenstrukturen modellieren und zugleich das Zeitverhalten darstellen. Dadurch kann das Verhalten der Algorithmen betrachtet werden.
Da die höheren Petrinetze sich gut zur Formalisierung des Verhaltens verteilter eingebetteter Systeme mit dem Zweck der Verifikation von Kausalitäten eignen folgt die Behandlung von bekannten Verifikationsmethoden als Grundlage zur späteren Erweiterung und Integration in den gesamten Analysesprozess. Abschließend wird die Notwendigkeit und Motivation zur Erweiterung von dynamischen Modellierungsansätzen für die Analyse von verteilten eingebetteten Systemen behandelt und die Zielstellung der Arbeit abgeleitet.

In Kontext dieser Arbeit wird die erweiterte gefärbte Petrinetzeklasse HCPN-ST vorgestellt. Danach wird die Anwendung am Beispielalgorithmus mit dem Ziel erläutert, diese Methode zu demonstrieren zu analysieren und zu verifizieren.

Nach der mathematischen Definition wird ein Algorithmus modelliert, der auf Wanderern basiert und auf P2P Systemen arbeitet. Danach wird dieses Modell verbal beschrieben. Ein anderes Modell mit CPN für Netze mit 3 Knoten wird zum Vergleich vorgestellt. Es zeigt sich die Unanschaulichkeit und Unzweckmäßigkeit von CPN bei großer Anzahl von Knotenadressen. Danach werden beide Modelle von Hand simuliert . Dann wird passende Software Tools werden gesucht. Damit können die beiden Modelle simuliert analysiert und verifiziert werden.

Mit dem CPN Algorithmus Modell ist die Simulation ebenso problemlos möglich wie die Analyse mit dem Tool INA. Das andere Modell erfordert eine Transformation, um die Software -Tools einsetzen zukönnen.

Peneca Chromos wurde als Tool ausgewählt, weil INA viele Eigenschaften analysieren und verifizieren kann. So z.B. Erreichbarkeitsgraph und T-Invarianten sowie P- Invarianten. Peneca Chromos kann eine Datei erzeugen, die auch INA bearbeiten kann.

Als Ergebnis wurde gezeigt, dass die vorgestellte Methode nutzbar, simulierbar und verifizierbar ist. Sie kann viele Arten von Nachrichten darstellen und verfügt über leicht verständliche Modelle.

Eine Weiterbearbeitung, der in dieser Arbeit beschriebenen Themen wäre in folgende Richtungen wünschenswert und ein Ausblick sein.

- Erstens kann für die vorgestellte Methode in ein Software -Tools implementiert werden.
- Mit Zeit-Petrinetzen können unterschiedliche Erweiterungen vorgenommen werden.
- Die Transitionen in Petrinetzen können effektiver realisiert werden. Durch Funktionen, die zusätzliche Bedingungen in einem System oder dessen Verhalten darstellen bzw. beim System Ablauf erfüllen können. Dabei können neue Petrinetzeklassen definiert werden.
- Ein gemeinsames Software-Tool sollte die neuen Petrinetzeklassen simulieren und analysieren können. Es arbeitet mit einer Input Schnittstelle. Entsprechend der Input Datei wird die richtige Prozedur abgerufen.
- Für P2P Systeme können noch unterschiedliche Anwendungen im Internet aufgebaut werden.
- Software -Tools können die Farbmarken erzeugen und abhängig von der Art der erzeugten Funktion Schalttabellen automatisch generieren.

Literaturverzeichnis

[AMC87] Ajmone Marsan, Marco ; Chiola, Giovanni, On Petri Nets with Deterministic and Exponentially Distributed Firing Times.Advances in Petri Nets In: (Lecture Notes in Computer Science(LNCS) 266,Rozenberg,Grzegorz (Hrsg.), Berlin, Heidelberg, NewYork Springer, 1987, S. 22

[Bau,Kri96] F. Bause, P.S. Kritzinger, Stochastic Petri Nets: An Introudution to the Theory, Vieweg, 1996

[Bau92] F. Bause, Funktionale Analyse zeitbehafteter Petri-Netze, 1992

[CHI91] G.Chiola, G. Franceschinis, A Structural Coloured Simplificati on in Well-Formed Coloured Net, IEE CS Press, 1991

[Cli05] The Gnutella Protocol Specification v 0.4http://clip2.com/Genutellaprotocolo4.pdf, Aufruf 15.07.2005,

[Clo, Dol, Kin02] G. Coulouris, J. Dollimore, T. Kindberg,Verteilte Systeme, Konzepte und Design , Pearson Studium, München, 2002

[Clo, Dol, Kin05] G. Cloulouris, J. Dollimore, T. Kindberg,Distributed Systems,concepts and Design , Addison Wesley,USA, 2005

[Day, Zim83] J. Day, H. Zimmerman, The OSI refernce model.Proceedings of the IEEE, 1983

[Dia09] M. Diaz, Petri Netze: Fundmental Models, Verfication and Applications, Wiley,2009

[Dra99] R. Drath, Modellierung hybrider Systeme auf der Basis modifizierter Petri-Netze, 1999

[Ens78] P. H. Enslow, What is a distributed system?, 1978

[Evg 07] B. Evgeniya, Entwicklung einer Modellierungsmethode mit höheren Petrinetzen, 2007

[Fen05] W. Fengler, Vorlesungen:Technische Applikation von Petri-Netzen, 2005

[Fen93] W. Fengler, A Colored Petri Netz Interpretation for Modelling and Control in Textile Processing.,CSCW & Petri Net Workshop, 14th International Conference- Applikation and theory of Petri Nets, 1993

[Fre90] J. Freudenmann,Transformation von Protokollspezifikationen in Kommunikationssoftware , Dissertation- Karlsruh, 1990

[Gen,Lau81] H.G.Genrich, K. Lautenbach, System Modelling with high level Petri Nets, theoretical Computer Science 1981

[Gen87] H.J. Genrich, , Predicate/transition Nets. Lecture Notes in Computer Science Nr 254-Petri Nets :Central Models and their Properties.Advances in Petri nets ,Springer, Berlin Heidelberg New York ,1987

[Ger06] Gerdes, Elke, Modellierung von Organisations- und Kommunikationsmechanismen in P2P Systemen mit erweiterten Statecharts, 2006

[Gir,Val03] C. Girault,R. Valk, Petri Nets for System Engineering: A Guide to Modelling, Verifikation, and Application , Springer,2003

[Gnu05] Gentuella homepage gnutella.com, 2005, Abruf: 23.7.2007

[Gru88] U. Grude, Theorie und Anwendungen von Petri-Netzen, Berlin,1988

[Ham05] U. Hammerschall, Verteilte Systeme und Anwendungen, Pearson Studium München, 2005

[Han,Lau98] Hanisch, H.M., Lautenbach, K., Simon, C., Thieme, J.: Zeitstempelnetze in technischen Anwendungen. Fachberichte Informatik, Universität Koblenz-Landau, 1998.

[Han92] H.-M Hanisch, Petri-Netze in der Verfahrenstechnik. Modellierung und

Steuerung verfahrenstechnischer Systeme, 1992
[Hau, Dus05] M. Hauswirth, S. Dustdar, Peer-to-Peer: Grundlagen und Architektur, dpunkt,2005
[Hei, PoP97] M.Heiner,L. Popova-Zeugmann, Worst-case Analysis of ConcurrentSystems with Duration Interval Petri Nets. In: . (Hrsg.): , 1997
[Hol99] R.Holten, Entwicklung von Führungsinformationssystemen. Ein methodenorientierterAnsatz, 1999
[Irt] http://irt.de/en/activities/digital-television/peer.html, Aufruf 20.08.2010,
[Jen92] K.Jensen ,Coloured Petri Nets , Springer-Berlin, Heidelberg, New York,Tokyo,1992
[Jen95] K. Jensen, Coloured Petri Nets: Basic concepts, Anaysis Methods and Practical Use , Springer Berlin Heidelberg Newyork,1995
[Kas,Kle08] Uwe Kasten, Hans Kleine Büning, Modellierung,Grundlagen und Formale Methoden, Hanser,2008
[Kaz05] http://kazaa.com, 2005 Abruf: 10.02.2006
[Kie97] U. Kiencke, Ereignisdiskrete Systeme, Modellierung und Steuerung verteilter Systeme, Oldenbourg,Wien, München, 1997
[Kno95] R. Knorr, Spezifikation, Verfikation, Leistungsbewertung und Implementierung von Kommunikationsprotokollen mit hierarchischen High-Level-Netzen, Shaker, 1995
[Kön, Quä88] R. König, L. Quäck, Petri-Netze in der Steuerungstechnik, Oldenbourg Verlag, 1988
[Kön87] H. König, , Kommuniktionsprotokolle-Prinzip und Entwicklung, Dissertation,Universität Dresden, 1987
[Kru85] H. Krumm, Spzifikation, Implementierung und Verfikation von Kommunikationsdiensten für verteilte Daten Verbreitung, Uni-Karlsruhe,1985
[Lic04] T. Licht, Ein Verfahren zur zeitlichen Analyse von UML-Modellen beim Entwurf von Automatisierungssystemen, 2004
[Lv, Cao, Coh, Li,She02]Q. Lv, P. Cao,E.Cohen, K.Li, S. Shenker, Search and Replication in Unstructured Peer- to-Peer Networks, Proceedings of 16th ACM International Conference on Supercomputing, 2002
[Mah, Sch07] P. Mahlmann, C.Schindelhauer , Peer to Peer Netzwerke:Algorithmen und Methoden, Springer,2007
[Mer74] P. Merlin, A Study of the Recoverability of Communication Protocols, 1974
[Mil,Dej,Kal,Raj02] Milojicic,S. Dejan, V.Kalogeraki,R. Lukose, K.Nagaraja, J.Pruyne, B.Richard, S. Rollins und Z.Xu , Peer to Peer Computing, 2002
[Mis, Hae, Pos, Dru05] A. Mislove, A.Haeberlen,A. Post, P. Druschel, Peer to Peer Systems and Application, Springer, 2005
[Mül69] Müller,P, Moderne Industrie München,1969
[NaP00] http://napster.com, 2000, Abruf: 23.12.2006
[P2P] http://P2P-next.org, Aufruf 15.07.2010,
[POS03] Forschungsgruppe, April 2003., http://www.uni-koblenz.de/agpn/html/projekte/poseidon.html , 2003,
[Pri, Wim08] L.Priese, H. Wimmel, Petri-Netze, Springer, 2008
[Ram74] C.Ramchandani, Analysis of Asynchronous Concurrent Systems by TimedPetri Nets, 1974
[Rei85] W. Reisig, Systementwurf mit Petri Netzen, Springer,1985

[Roc,str01] S.Roch, P. Starke. Stand 2001http:, Online Handbuch INA - Integrierter Netz AnalysatorVersion 2.1, 2001,
[Rok92] P.Rokyta, Theoretische Grundlagen für Tools für gefärbte Petri-Netze, 1992
[Roz87] G.Rozenberg, Behaviour of Elementary Net Systems, 1987

[Rupr,Hah,Que, ..05] C.Rupp, J.Queins, M. Jeckle, B. Zengler, UML2 glasklar: Praxiswissen für die UML-Modellierung und Zertifizierung, Hanser,2005
[Sch, Spr07] A. Schill, T. Springer, Verteilte Systeme, Springer, Dresden ,2007
[Sta90] P. Starke, Analyse von Petri-Netz-Modellen, Teubner Stuttgart,1990
[Stu90] M. Stumm, S.Zhou, Algorithms Implementing Distributed Shared Memory , ComputerBd. 23, Nr. 5,1990
[Tan, Ste03] A.Tannenbaum,M. Van Steen, Verteilte Systeme, Grundlagen und Paradigmen, Pearson Studium, München,2003
[Thi 87] P. Thiagarajan, S.: Elementary Net Systems. 1987

[Tui97] Tu- Ilmenau PENECA chromos , http://tin.tu-ilmenau.de/ra/skripte/pn-_chr/, 1997
[Ung, Wul02] H. Unger, M. Wulff, Cluster-building in P2P-Community Networks, 2002
[uni10] Uni- Hamburg , http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html, 2010

[wikipedia.org] *de.wikipedia.org/wiki/**Endlicher**_Automat* , Aufruf 10.07.2010
[VTR00] Verification & Testing Research Group, http://www.dcs.shef.-ac.uk/research/groups/vt/mission.html, 2000
[Wul05] M. Wulff, Untersuchungen Zur Strukturbildung in P2P -Netzen durch " Random Walker",Uni Rostock 2005

Anlage A

CPN-Wanderer Modellierung

Tabellen der Modellerstellung :

Tabelle 1: Wanderer und Knotenadressen

Sp1	Sp2	Sp3	Sp4	Sp5	Sp6	Sp7	Sp8	Sp9	Sp10	Sp11	Sp12
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2

Plätze und Transitionen

Die folgenden Tabellen 2 bis 9 eschreiben Plätze und Transitionen des nicht jdisjunkte Cluster Algorithmus CPN Chromos-Netzes.

Hauptnetz

Tabelle 2 : Plätze, Platzunternetze

empty	freie Knoten, d.h. ohne Wanderer in Bearbeitung (empty nodes)
Erand	Ende des zufälligen Verschickens (end random)
Forw	Start des Verschickens eines Wanderer (sent forward)
genown	Generieren eines eigenen Wanderers (generate own)
Net	Modelliert das Verbindungsnetz ,aber mit Farbmarke modelliert der Wanderer im Netz (wanderer in net)
Nokill	kein Wanderer zu löschen (no nod to kill)
protadd	Protokoll der Wanderer, zu denen Knoten hinzugefügt wurden, nur für Simulation! (protocol of add)
protfwdo	Protokoll der eigenen, neu generierten Wanderer, nur für Simulation!

	(protocol of forward own)
protkill	Protokoll der entfernten Wanderer, nur für Simulation! (protocol of kill)
protrem	Protokoll der Wanderer, aus denen Knoten entfernt wurden, nur für Simulation! (protocol of remove)
Rtoa	vom entfernen zum hinzufügen (remove to add)
Srand	Start des zufälligen Verschickens (start to sent random)
tavg, Unternetz	zwei Zeitgeber für Behandlung von tavg, für Entfernen und Hinzufügen (timer for tavg)
tkill, Unternetz	Zeitgeber für Löschen (timer for tkill)
trequest, Unternetz	Zeitgeber für trequest, für Hinzufügen bzw. Generieren (timer for trequest)
W	Wanderer, die auf einem Knoten in Bearbeitung sind (wanderer)
Wr	empfangene Wanderer (wanderer received)
wrstart	Knoten, Start der Wandrerbearbeitung (nod start to do)

Tabelle 3: Transitionen

addno	Knoten soll nicht hinzugefügt werden, weil die Zeit tavg,i ist nicht groß genug zum Besuch des Wanderers
addnoo	Knoten soll hinzugefügt werden, bei aktuellem Wandrer aber nicht möglich (add but no possible)
Addplan1,2	Knoten zu aktuellem Wanderer hinzufügen (add to plan)
endforw	Ende des (normalen: ohne Entfernen, ohne Hinzufügen, ohne zufällig) Versendens an einen Nachfolgeknoten (end forward)
forwown	Versenden des neu generierten Wanderes über zufälliges Generieren eines Nachfolgeknotens (forward own)
genown	Generieren eines eigenen Wanderers, Zeitverzögerung ttry (generate own)
Kill	aktuellen Wanderer löschen (kill)
Killno	Löschen nicht angefordert (kill no)
killnoo	Löschen angefordert, aber kein eigenen Wanderer in Bearbeitung, deshalb darf nicht gelöscht werden (kill but no own, not allow)
nettime	Verzögert (stochastisch mit einstellbarer Wahrscheinlichkeit) Nachrichten zwischen Senden und Empfangen (net time)
Rand	zufälliges Generieren eines Nachfolgeknotens (random)
remno	Entfernen nicht angefordert (rmove no)
remnoo	Knoten soll entfernt werden, bei aktuellem Wandrer aber nicht möglich (remove but no possible)

remplan	Knoten aus aktuellem Wanderer entfernen (remove from plan)
Wr	Wanderer empfangen (wanderer received)

Platzunternetz tkill

Tabelle 4: Plätze

Kill	Knoten, deren eigene Wanderer gelöscht werden sollen, Schnittstelle (kill)
Tkill	Knoten, für die tkill dekrementiert wird (time for kill)
wrtkill	Wanderer empfangen für Löschzeitgeber, Schnittstelle(wanderer received for kill)

Tabelle 5: Transitionen

Tkill	Dekrementieren der Zeitgeber Löschen, Zeitverzögerung tkill (timer kill)
wrtolong	Zeit zwischen zwei Wandrern groß genug (wanderer received to long)
wrtoshort	Zeit zwischen zwei Wanderen zu kurz, Löschen (wanderer received toshort)
wrtoshort2	Zeit zwischen zwei Wanderen zu kurz, Löschen aber schon vorgesehen(wanderer received to short 2)

Platzunternetz tavg

Tabelle 6: Plätze

addplan	Knoten, die zu einem Wanderer hinzugefügt werden sollen, Schnittstelle (add toplan)
remplan	Knoten, die aus einem Wanderer entfernt werden sollen, Schnittstelle (remove from plan)
Tadd	Knoten, für die tadd dekrementiert wird (time for add)
Trem	Knoten, für die trem dekrementiert wird (time for remove)
Wradd	Wanderer empfangen für Hizufügenzeitgeber, Schnittstelle (Wanderer received for add)
Wrrem	Wanderer empfangen für Entfernenzeitgeber, Schnittstelle (Wanderer received for remove)

Tabelle 7: Transitionen

remadd	gleichzeitiges vorgesehenes Entfernen und Hinzufügen kompensieren (remove and add)
taddadd	Dekrementieren der Zeitgeber Hinzufügen, Zeitverzögerung tavg+dt, bei Nulldurchgang Hinzufügen (timer add and add)
Trem	Dekrementieren der Zeitgeber Entfernen, Zeitverzögerung tavg-dt (timer remove)
wradd	Neustart Zeitgeber Hinzufügen (wanderer add new)

wrlong	Zeit zwischen zwei Wanderern groß genug (wanderer received to long)
wrnoadd	Zeit zwischen zwei Wanderern nicht zu groß (wanderer no add)
wrshort	Zeit zwischen zwei Wanderen zu kurz, Entfernen (wanderer received to short)
wrshort2	Zeit zwischen zwei Wanderen zu kurz, Entfernen aber schon vorgesehen(wanderer received to short 2)

Platzunternetz trequest

Tabelle 8: Plätze

forwardown	Knoten, die gerade einen eigenen Wanderer generiert haben, Schnittstelle (forward own)
Reqe	maximal 1 Anforderung je Knoten für Generieren, Schnittstelle (request end)
request	alle Anforderungen für Generieren (request)
Treq	Knoten, für die trequest dekrementiert wird (time for request)
Wrreq	Wanderer empfangen für Generierenzeitgeber, Schnittstelle (wanderer received for request)
wrreqcol	Wanderer empfangen, in r Spalten eingeordnet (wanderer for request column)
wrreqi	i-ter Wanderer empfangen (wanderer for request i)

Tabelle 9: Transitionen

genownnotrequ	Anfangswerte für treq bei fehlender Zeitmarke (generate own with no trequest)
genowntreqend	Rücksetzen für treq bei vorhandener Zeitmarke (generate own trequest end)
Remreq	Löschen der anderen Requestanforderungen des gleichen Knotens nach Erzeugen einer Anforderung (remove request)
Reqe	Erzeugen einer Requestanforderung (request end)
Treq	Dekrementieren der Zeitgeber Generieren, Zeitverzögerung trequest, bei Nulldurchgang Request stezten (timer request)
Wr1,2,3	Wanderer i empfangen (i=1,2,3) (wanderer received 1,2,3)
Wrnoreq	Zeit zwischen r Wanderern nicht zu groß (wanderer no request)
Wrreq	Neustart Zeitgeber Generieren (wanderer request new)

Symboltabellen in Chromos

(Schaltfähigkeit und Schalten von Farbmarken durch die angegebenen Transitionen)

Die Schaltmodi sind untereinander ODER-verknüpft.

Die Variablen (erste Zeile, ab Spalte 2) sind an den entsprechenden Vor- und Nachkanten der

Transition zu finden.

Einträge sind wie folgt zu lesen (Bsp): w1 1: Farbmarke w1; Vielfachheit 1;

a1 & i1: Farbmarke a1 UND i1; je mit Vielfachheit 1 notwendig (Vorkante) bzw. zu erzeugen (Nachkante) a | i1: Farbmarke a1 ODER (nicht EXOR!) i1; je mit Vielfachheit 1 notwendig (nur an Vorkante möglich)

Die Schaltmodi Tabelle für alle Transitionen der Hauptnetze (erzeugente mit PENECA Chromos Tools):

Chromossymboltabelle addno, remno, killno, genown,

Schaltmodi	wi
w3	w3 1
w1	w1 1
w2	w2 1

Chromossymboltabelle forwown

Schaltmodi	wi	wfo
w3	w3 1	d3 1
w1	w1 1	d1 1
w2	w2 1	d2 1

Chromossymboltabelle addnoo

Schaltmodi	wnaoi	wnaoo	wnadd	wraori
w3	w3 1	w3 1	d13 1 & m13 1 & d23 1 & h23 1	
w3a		w3 1	d13 1 & m13 1 & d23 1 & h23 1	w3 1
w1	w1 1	w1 1	d31 1 & h31 1 & d21 1 & m21 1	
w1a		w1 1	d31 1 & h31 1 & d21 1 & m21 1	w1 1
w2	w2 1	w2 1	d32 1 & m32 1 & d12 1 & h12 1	
w2a		w2 1	d32 1 & m32 1 & d12 1 & h12 1	w2 1

Chromossymboltabelle addplan1

Schaltmodi	ain	add2	addr	add4	add3	ao1	add1	ain1
w3d13a	d13 1	w3 1	w3 1	w3 1	w3 1	g1 1	w3 1	d13 1
w3m13a	m13 1	w3 1	w3 1	w3 1	w3 1	y1 1	w3 1	m13 1
w3d23a	d23 1	w3 1	w3 1	w3 1	w3 1	k2 1	w3 1	d23 1
w3h23a	h23 1	w3 1	w3 1	w3 1	w3 1	x2 1	w3 1	h23 1
w1d31a	d31 1	w1 1	w1 1	w1 1	w1 1	k3 1	w1 1	d31 1
w1h31a	h31 1	w1 1	w1 1	w1 1	w1 1	x3 1	w1 1	h31 1
w1d21a	d21 1	w1 1	w1 1	w1 1	w1 1	g2 1	w1 1	d21 1
w1m21a	m21 1	w1 1	w1 1	w1 1	w1 1	y2 1	w1 1	m21 1
w2d32a	d32 1	w2 1	w2 1	w2 1	w2 1	g3 1	w2 1	d32 1
w2m32a	m32 1	w2 1	w2 1	w2 1	w2 1	y3 1	w2 1	m32 1
w2d12a	d12 1	w2 1	w2 1	w2 1	w2 1	k1 1	w2 1	d12 1
w2h12a	h12 1	w2 1	w2 1	w2 1	w2 1	x1 1	w2 1	h12 1

Chromossymboltabelle addplan2

Schaltmodi	wh	wg	wf	wfr
a1	w1 1	a1 1		d1 1
a2	w2 1	a2 1		d2 1
a3	w3 1	a3 1		d3 1
i1	w1 1	i1 1	k1 1	
i2	w2 1	i2 1	k2 1	
i3	w3 1	i3 1	k3 1	
k1	w2 1	k1 1		m1 1
k2	w3 1	k2 1		m2 1
k3	w1 1	k3 1		m3 1
e1	w1 1	e1 1	g1 1	
e2	w2 1	e2 1	g2 1	
e3	w3 1	e3 1	g3 1	
g1	w3 1	g1 1		h1 1
g2	w1 1	g2 1		h2 1
g3	w2 1	g3 1		h3 1
v1	w1 1	v1 1	x1 1	
v2	w2 1	v2 1	x2 1	
v3	w3 1	v3 1	x3 1	
x1	w2 1	x1 1	y1 1	
x2	w3 1	x2 1	y2 1	
x3	w1 1	x3 1	y3 1	
y1	w3 1	y1 1		z1 1
y2	w1 1	y2 1		z2 1
y3	w2 1	y3 1		z3 1
d12	w2 1	d12 1	a1 1	
d13	w3 1	d13 1	a1 1	
d21	w1 1	d21 1	a2 1	
d23	w3 1	d23 1	a2 1	
d31	w1 1	d31 1	a3 1	
d32	w2 1	d32 1	a3 1	
h12	w2 1	h12 1	e1 1	
h13	w3 1	h13 1	e1 1	
h21	w1 1	h21 1	e2 1	
h23	w3 1	h23 1	e2 1	
h31	w1 1	h31 1	e3 1	
h32	w2 1	h32 1	e3 1	
m21	w1 1	m21 1	i2 1	
m23	w3 1	m23 1	i2 1	
m12	w2 1	m12 1	i1 1	
m13	w3 1	m13 1	i1 1	
m31	w1 1	m31 1	i3 1	
m32	w2 1	m32 1	i3 1	
z12	w2 1	z12 1	v1 1	
z13	w3 1	z13 1	v1 1	
z21	w1 1	z21 1	v2 1	
z23	w3 1	z23 1	v2 1	
z31	w1 1	z31 1	v3 1	
z32	w2 1	z32 1	v3 1	

add4	ao1	add1
3 1	g1 1	w3 1
3 1	y1 1	w3 1
3 1	k2 1	w3 1
3 1	x2 1	w3 1
1 1	k3 1	w1 1
1 1	x3 1	w1 1
1 1	g2 1	w1 1
1 1	y2 1	w1 1
2 1	g3 1	w2 1
2 1	y3 1	w2 1
2 1	k1 1	w2 1
2 1	x1 1	w2 1

Chromossymboltabelle endforw*Chromossymboltabelle killnoo*

Schaltmodi	wni	wnown	wnk
w1	w1 1	a1 1 & i1 1 & e1 1 & v1 1	w1 1
w2	w2 1	a2 1 & i2 1 & e2 1 & v2 1	w2 1
w3	w3 1	a3 1 & i3 1 & e3 1 & v3 1	w3 1

Chromossymboltabelle kill

Schaltmodi	wkill	wkkill
w1a1	a1 1	w1 1
w1i1	i1 1	w1 1
w1e1	e1 1	w1 1
w1v1	v1 1	w1 1
w2a2	a2 1	w2 1
w2i2	i2 1	w2 1
w2e2	e2 1	w2 1
w2v2	v2 1	w2 1
w3a3	a3 1	w3 1
w3i3	i3 1	w3 1
w3e3	e3 1	w3 1
w3v3	v3 1	w3 1

Chromossymboltabelle rand

Schaltmodi	fd	fde
d12	d1 1	d12 1
d13	d1 1	d13 1
d21	d2 1	d21 1
d23	d2 1	d23 1
d31	d3 1	d31 1
d32	d3 1	d32 1
m12	m1 1	m12 1
m13	m1 1	m13 1
m21	m2 1	m21 1
m23	m2 1	m23 1
m31	m3 1	m31 1
m32	m3 1	m32 1
h12	h1 1	h12 1
h13	h1 1	h13 1
h21	h2 1	h21 1
h23	h2 1	h23 1
h31	h3 1	h31 1
h32	h3 1	h32 1
z12	z1 1	z12 1
z13	z1 1	z13 1
z21	z2 1	z21 1
z23	z2 1	z23 1
z31	z3 1	z31 1
z32	z3 1	z32 1

Chromossymboltabelle remnoo

Schaltmodi	wnrem	wnroi
w3	g1 1 & y1 1 & k2 1 & x2 1	w3 1
w1	k3 1 & x3 1 & g2 1 & y2 1	w1 1
w2	g3 1 & y3 1 & k1 1 & x1 1	w2 1

Chromossymboltabelle nettime

Schaltmodi	wnet
a1	a1 1
a2	a2 1
a3	a3 1
d12	d12 1
d13	d13 1
d21	d21 1
d23	d23 1
d31	d31 1
d32	d32 1
i1	i1 1
i2	i2 1
i3	i3 1
k1	k1 1
k2	k2 1
k3	k3 1
m12	m12 1
m13	m13 1
m21	m21 1
m23	m23 1
m31	m31 1
m32	m32 1
e1	e1 1
e2	e2 1
e3	e3 1
g1	g1 1
g2	g2 1
g3	g3 1
h12	h12 1
h13	h13 1
h21	h21 1
h23	h23 1
h31	h31 1
h32	h32 1
v1	v1 1
v2	v2 1
v3	v3 1
x1	x1 1
x2	x2 1
x3	x3 1
y1	y1 1
y2	y2 1
y3	y3 1
z12	z12 1
z13	z13 1
z21	z21 1
z23	z23 1
z31	z31 1
z32	z32 1

Chromossymboltabelle remplan

Schaltmodi	rem	rin	ro	ro1
g1	w3 1	g1 1	d1 1	
y1	w3 1	y1 1	m1 1	
k2	w3 1	k2 1	d2 1	
x2	w3 1	x2 1		g2 1
k3	w1 1	k3 1	d3 1	
x3	w1 1	x3 1		g3 1
g2	w1 1	g2 1	d2 1	
y2	w1 1	y2 1	m2 1	
g3	w2 1	g3 1	d3 1	
y3	w2 1	y3 1	m3 1	
k1	w2 1	k1 1	d1 1	
x1	w2 1	x1 1		g1 1

Chromossymboltabelle wr

Schaltmodi	w	wj
a1	a1 1	w1 1
a2	a2 1	w2 1
a3	a3 1	w3 1
e1	e1 1	w1 1
e2	e2 1	w2 1
e3	e3 1	w3 1
g1	g1 1	w3 1
g2	g2 1	w1 1
g3	g3 1	w2 1
i1	i1 1	w1 1
i2	i2 1	w2 1
i3	i3 1	w3 1
k1	k1 1	w2 1
k2	k2 1	w3 1
k3	k3 1	w1 1
v1	v1 1	w1 1
v2	v2 1	w2 1
v3	v3 1	w3 1
x1	x1 1	w2 1
x2	x2 1	w3 1
x3	x3 1	w1 1
y1	y1 1	w3 1
y2	y2 1	w1 1
y3	y3 1	w2 1
d12	d12 1	w2 1
d13	d13 1	w3 1
d21	d21 1	w1 1
d23	d23 1	w3 1
d31	d31 1	w1 1
d32	d32 1	w2 1
m12	m12 1	w2 1
m13	m13 1	w3 1
m21	m21 1	w1 1
m23	m23 1	w3 1
m31	m31 1	w1 1
m32	m32 1	w2 1
h12	h12 1	w2 1
h13	h13 1	w3 1
h21	h21 1	w1 1
h23	h23 1	w3 1
h31	h31 1	w1 1
h32	h32 1	w2 1
z12	z12 1	w2 1
z13	z13 1	w3 1
z21	z21 1	w1 1
z23	z23 1	w3 1
z31	z31 1	w1 1
z32	z32 1	w2 1

Unternetze tkill, tavg

Chromossymboltabelle tkill, wrtolong, wrtoshort, wrtoshort2, trem, wrlong, wrshort, wrshort2, remadd, taddadd, wrnoadd, wradd

Schaltmodi	wi
w3	w3 1
w1	w1 1
w2	w2 1

Unternetz trequest

Chromossymboltabelle reqe, remreq

Schaltmodi	wr
w3 1	w3 1
w3 2	w3 1
w3 3	w3 1
w11	w1 1
w12	w1 1
w13	w1 1
w21	w2 1
w22	w2 1
w23	w2 1

Chromossymboltabelle genowntreqend, genowntreqe, treq

Schaltmodi	wr	wk
w3 1	w3 1	w31 1
w3 2	w3 1	w32 1
w3 3	w3 1	w33 1
w11	w1 1	w11 1
w12	w1 1	w12 1
w13	w1 1	w13 1
w21	w2 1	w21 1
w22	w2 1	w22 1
w23	w2 1	w23 1

Chromossymboltabelle wr1

Schaltmodi	w1	k1	m1
w31	w3 1	w31 1	w32 1
w11	w1 1	w11 1	w12 1
w21	w2 1	w21 1	w22 1

Chromossymboltabelle wr2

Schaltmodi	k3	w3	m3
w31	w33 1	w3 1	w31 1
w13	w13 1	w1 1	w11 1
w23	w23 1	w2 1	w21 1

Chromossymboltabelle wr3

Schaltmodi	k3	w3	m3
w31	w33 1	w3 1	w31 1
w13	w13 1	w1 1	w11 1
w23	w23 1	w2 1	w21 1

Tabelle 10 add w3

Sp1	Sp 2	Sp3	Sp 4	Sp5	Sp6	Sp 7	Sp8	Sp9	Sp10	Sp11	Sp12	Sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	g1,w3
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	y1,w3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	k2,w3
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	
w2	l	w1	p	1		3	4	e2,w2		g1,w3	h21,w1;h23,w3	x2,w3
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y1,w3	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3w3		g3,w2	h31,w1;h32,w2	
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	

Sp. 12: Wanderer, Knoten, bei dem W3 eingetragen werden kann

Sp. 13: Wanderer, Knoten, nach Eintrag von W3

Tabelle 11: rem W3

sp 1	sp 2	sp 3	sp 4	sp 5	sp 6	sp 7	sp 8	sp9	sp10	sp11	sp12	sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	d12,w2;d13,w3
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	m12,w2;m13,w3
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	d21,w1;d23,w3
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	g2,w1
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2	
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	

Sp. 12: Wanderer, Knoten, bei dem W3 entfernt werden kann

Sp. 13: Wanderer, Knoten, der nach Entfernung von W3 verschickt wird

Tabelle 12: add w1

sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9	sp10	sp11	sp12	sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	g2,w1
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	y2,w1
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	k3,w1
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2	x3,w1
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	

Sp. 12: Wanderer, Knoten, bei dem w1 eingetragen werden kann

Sp. 13: Wanderer, Knoten, nach Eintrag von w1

Tabelle 13: rem w1

sp1	sp2	sp3	sp4	sp5	sp6	sp 7	sp 8	sp9	sp10	sp11	sp12	sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	d21,w1;d23,w3
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	m21,w1;m23,w3
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	d31,w1;d32,w2
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2	
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	g3,w2

Sp. 12: Wanderer, Knoten, bei dem w1 eingetragen werden kann

Sp. 13: Wanderer, Knoten, der nach Entfernung von w1 verschickt wird

Tabelle 14: add w2

sp1	sp2	sp3	sp4	sp5	sp6	sp7	sp8	sp9	sp10	sp11	sp12	sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	k1,w2
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	x1,w2
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	g3,w2
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	y3,w2
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2	
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	

Sp. 12: Wanderer, Knoten, bei dem w2 eingetragen werden kann

Sp. 13: Wanderer, Knoten, nach Eintrag von w2

Tabelle 15: rem w2

sp 1	sp 2	sp 3	sp 4	sp 5	sp 6	sp 7	sp 8	sp9	sp10	sp11	sp12	sp13
w1	l	l	p	1			4	a1,w1			d12,w2;d13,w3	
w1	w2	l	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3	d12,w2;d13,w3
w1	l	w3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3	
w1	w2	w3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3	g1,w3
w2	l	l	p	1			4	a2,w2			d21,w1;d23,w3	
w2	w3	l	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3	
w2	l	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3	
w2	w3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3	
w3	l	l	p	1			4	a3,w3			d31,w1;d32,w2	
w3	w1	l	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2	
w3	l	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2	d31,w1;d32,w2
w3	w1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2	m31,w1;m32,w2

Sp. 12: Wanderer, Knoten, bei dem w2 eingetragen werden kann

Sp. 13: Wanderer, Knoten, der nach Entfernung von w2 verschickt wird

Tabelle 16: kill w3, w1, w2

Sp1	Sp2	Sp3	Sp4	Sp5	Sp6	Sp7	Sp8	Sp9	Sp10	Sp11	Sp12
w1	L	L	p	1			4	a1,w1			d12,w2;d13,w3
w1	W2	L	p	1	2		4	i1,w1	k1,w2		m12,w2;m13,w3
w1	L	W3	p	1		3	4	e1,w1		g1,w3	h12,w2;h13,w3
w1	W2	W3	p	1	2	3	4	v1,w1	x1,w2	y1,w3	z12,w2;z13,w3
w2	L	L	p	1			4	a2,w2			d21,w1;d23,w3
w2	W3	L	p	1	2		4	i2,w2	k2,w3		m21,w1;m23,w3
w2	L	w1	p	1		3	4	e2,w2		g2,w1	h21,w1;h23,w3
w2	W3	w1	p	1	2	3	4	v2,w2	x2,w3	y2,w1	z21,w1;z23,w3
W3	L	L	p	1			4	a3,w3			d31,w1;d32,w2
W3	W1	L	p	1	2		4	i3,w3	k3,w1		m31,w1;m32,w2
W3	L	w2	p	1		3	4	e3,w3		g3,w2	h31,w1;h32,w2
W3	W1	w2	p	1	2	3	4	v3,w3	x3,w1	y3,w2	z31,w1;z32,w2

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.
Insbesondere habe ich hier für nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen.
Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.
Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.
Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zu Folge hat.

(Ort, Datum) *(Unterschrift)*