Erstellung von Testplänen für verteilte Systeme durch stochastische Modellierung

Der technischen Fakultät der
Universität Erlangen-Nürnberg

zur Erlangung des Grades

D O K T O R – I N G E N I E U R

vorgelegt von

Stefan Dalibor

Erlangen – 2000

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	1. Dezember 2000
Tag der Promotion:	13. Juli 2001
Dekan:	Prof. Dr. Harald Meerkamm
Berichterstatter:	Prof. Dr. Mario Dal Cin
	Prof. Dr. Reinhard German

Meinen lieben Eltern

Danksagung

Herzlich bedanken möchte ich mich bei Prof. Dr. Dal Cin für die Erstbegutachtung und die freundliche und geduldige Unterstützung während der Erstellung dieser Arbeit, sowie bei Prof. Dr. German für die bereitwillige Übernahme des Zweitgutachtens.

Mein Dank gilt auch allen Kolleginnen und Kollegen am IMMD III für die gute Zusammenarbeit und die freundschaftliche Atmosphäre, die die Arbeit am Lehrstuhl zu einer schönen Zeit gemacht haben.

Sehr dankbar bin ich schließlich meinen Freunden – allen voran Kerstin und Birgit – für ihre Unterstützung.

Diese Arbeit wurde im Rahmen des Sonderforschungsbereichs 182 (»Multiprozessoren und Netzwerkkonfigurationen«) von der Deutschen Forschungsgemeinschaft unterstützt.

Kurzfassung

Verteilte Systeme – im Sinne von vernetzten Rechnern, die zusammen Dienste so erbringen daß sie nach außen als *ein* System wirken – werden heute in der Informationsverarbeitung überall dort eingesetzt, wo zuverlässig und skalierbar hohe Leistung benötigt wird. Den Vorteilen dieser System-Architektur stehen jedoch auch potentielle Probleme gegenüber: Die für einen reibungslosen Betrieb notwendige schnelle und präzise Diagnose im Fall von Störungen ist von erheblicher Komplexität.

Durch die modulare Struktur der verteilten Systeme ist eine große Anzahl von Komponenten zu überprüfen, zwischen denen viele Abhängigkeiten bestehen; gleichzeitig unterliegt ein Test solcher Anlagen im praktischen Einsatz zumeist Einschränkungen, die durch die Notwendigkeit bedingt sind, die Güte der erbrachten Dienste nicht zu beeinträchtigen. Deshalb ist für verteilte Systeme schon die (der eigentlichen Diagnose von Störungen vorgelagerte) Aufgabe der Erstellung von Plänen zum Test der Anlagen eine anspruchsvolle Aufgabe, die wegen der schwierigen Rahmenbedingungen heute zumeist nur von Experten auf Basis von heuristischen Ansätzen ad hoc bewältigt wird.

In dieser Arbeit wird ein Verfahren entwickelt, mit dem die Erstellung von Plänen zum Test verteilter Systeme auf der Basis eines strukturierten und formalisierten Ansatzes durchgeführt werden kann. Dazu wird eine Methode beschrieben, mit der ein formal definiertes Modell der für die Testplanerstellung relevanten Aspekte der zu diagnostizierenden Anlagen aufgebaut werden kann. Aus dem Modell können dann mittels mathematischer Analyseverfahren Testpläne abgeleitet werden, die im Mittel optimal bezüglich einer die Kosten und den potentiellen Informationsgewinn der Tests beschreibenden Bewertungsfunktion sind.

Als Modellierungsverfahren werden stochastische, bewertete Petrinetze verwendet, die so erweitert wurden, daß sie zur Analyse auf Markov-Entscheidungsmodelle abgebildet werden können. Mit den so definierten ***Rekonfigurations-Petrinetzen*** konnten die Vorteile der Modellierung mit Petrinetzen (formal definierte Semantik mit Algorithmus zur Entfaltung der Zustandsräume, gute Visualisierung der Modelle) mit denen der Markov-Entscheidungsmodellierung (effiziente numerische Lösungsverfahren zur Berechnung im Mittel optimaler Entscheidungen auch für große Modelle) verbunden werden.

Mit der Formalisierung sowie einer geeignet gewählten Hierarchisierung der Petrinetze wird es möglich, die Erstellung, Parametrisierung und Auswertung der Modelle weitgehend zu automatisieren: Nach der Angabe der Systemkomponenten mit ihren Abhängigkeiten, der Menge der zur Verfügung stehenden Tests sowie deren Klassifizierung in verschiedenen Kategorien nach Kosten und Nutzen durch den Modellierer werden die Testpläne durch das in der vorliegenden Arbeit beschriebene Werkzeug (***Diagnosis Support System***) ohne weitere manuelle Einwirkung berechnet.

Die Anwendbarkeit des neuen Verfahrens zur Testplan-Generierung auf praxisbezogene verteilte Systeme wird schließlich an einem Beispiel gezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einführung

1.1 Motivation

Verteilte Rechensysteme, deren Größe und Verarbeitungsleistung mit dem Bedarf der Anwendungen wächst, werden in zunehmendem Maße in Bereichen eingesetzt, die bisher klassischen Großrechenanlagen vorbehalten waren (s. [Har97] für eine Fallstudie). Dies gilt nicht nur für technisch-wissenschaftliche Anwendungen, sondern auch für Transaktionssysteme, die Tausende von Teilnehmern in komplexen Netzwerken verbinden und ihnen Dienste anbieten. Skalierbare Leistung wird dabei heutzutage meist nicht durch massiv parallele Rechner, sondern (wegen der günstigeren Kosten sowie der einfacheren Erweiterbarkeit) durch über dedizierte Hochleistungsverbindungen vernetzte Standard-Systeme (sog. *Server-Cluster*) bereitgestellt [Ave98a, Ave98b]. Dabei entstehen vielschichtige, heterogene Architekturen (*multi-tier architecture*), die in hohem Grad an die speziellen Bedürfnisse der zu erbringenden Dienstleistung angepaßt sind.

Mit dem breiten Einsatz solcher Anlagen im kommerziellen Bereich (z.B. internationales Finanz- und Wertpapiergeschäft, globaler elektronischer Handel) immer wichtiger wird eine hohe Verfügbarkeit der Cluster-Systeme – sie stellen einen der technologischen Basisbausteine der »New Economy« dar und müssen daher häufig bei großen Investitionskosten wichtige Dienste erbringen. In der Folge können selbst kurze Ausfallzeiten schon extreme wirtschaftliche Schäden verursachen.

Um für solche missionswichtigen (*mission critical*) Installationen die erforderliche Verfügbarkeit gewährleisten zu können, ist es vor allem notwendig, das System – Hardware *und* Software – schnell und gründlich diagnostizieren zu können, sobald sich Anhaltspunkte für einen Fehler oder Ausfall in einem der Teile ergeben. Nur dann lassen sich Schäden mit der notwendigen Geschwindigkeit beheben sowie die für eine Fehler tolerierende Behandlung von Teil-Ausfällen nötigen Maßnahmen rechtzeitig einleiten.

Eine Diagnose derart umfangreicher Systeme läßt sich aber nicht bewerkstelligen ohne Vorab-Information über mögliche Vorgehensweisen – die Durchführung von Tests als Grundlage der Lokalisierung von Fehlern muß sorgfältig geplant

werden. Dabei muß eine Vielzahl von Informationen berücksichtigt werden, z.B. über die Art der zur Verfügung stehenden Tests und deren gegenseitige Abhängigkeiten, sowie über die Struktur der zu testenden Anlage und den Einfluß der Tests auf deren Betriebszustand.

Die Fehlersuche in verteilten Systemen ist also eine komplexe Aufgabe, die unter hohen, teilweise in Widerspruch zueinander stehenden Anforderungen stattfindet:

- Aufgetretene Fehler sind in verteilten Systemen wesentlich schwieriger zu lokalisieren als in konventionellen Rechnern: Neben der Existenz von mehreren verschiedenen Daten- oder Steuer-Pfaden zu Systemkomponenten involvieren viele Vorgänge mehrere, gleichzeitig aktive Teile des Gesamtsystems. In der Folge erschweren Synchronisierungsprobleme und Zeitbeschränkungen die Fehlersuche erheblich.

- Defekte und ihre Folgen sind in verteilten Systemen oft nur indirekt oder über lange Kausalketten gekoppelt, d.h. die Auswirkungen eines Ausfalls machen sich oft in strukturell getrennten Teilen des Systems und/oder erst mit erheblicher Zeitverzögerung bemerkbar.

- Leistungsfähige verteilte Systeme haben (selbst auf hoher Abstraktionsebene) hunderte von testbaren Komponenten (Hardware der Knotenrechner, aktive und passive Netzwerkverbindungen, Betriebssystem- und Applikations-Software-Dienste etc.) mit der entsprechenden kombinatorischen Explosion der Zahl von möglichen Vorgehensweisen bei der Durchführung von Tests als Folge.

- Für die sinnvolle Ausführung von Tests sind oft zahlreiche logische Abhängigkeiten gegeben: Einige Tests müssen erfolgreich abgeschlossen worden sein, bevor andere gestartet werden können; einige Tests können in zeitlicher Parallelität durchgeführt werden. Evtl. vorhandene redundante Pfade sollten falls möglich ausgenützt werden.

- Stochastische Aspekte sind zu berücksichtigen: Die Aussichten auf erfolgreiche Ausführung und die voraussichtliche Dauer von Tests sowie der erwartete Gewinn an Information durch die Testergebnisse kann oft nur geschätzt oder mit stochastischer Unsicherheit behaftet angegeben werden.

- Der Betreiber ist natürlich an einer möglichst schnellen und präzisen Lokalisierung der Ursache von Fehlern interessiert, um die Eskalation der Fehler-Auswirkungen zum Totalausfall des Systems unter allen Umständen zu verhindern.

- Gleichzeitig dürfen jedoch laufende Applikationen durch die Fehlersuche möglichst wenig beeinträchtigt werden, um die Güte der erbrachten Dienste nicht abfallen zu lassen: Ausgedehnte Betriebsunterbrechungen für Testläufe (sog. *Offline*-Tests) können zumeist nur als letztes Mittel hingenommen werden, wenn sich Fehlerursachen anders nicht eingrenzen lassen; für missionswichtige oder unter Echtzeit-Beschränkungen operierende Applikationen müssen oft alle zusätzlichen Belastungen für das System (d.h. jeglicher Verbrauch von Ressourcen wie z.B. Netzwerk-Bandbreite durch applikationsfremde Verursacher) weitgehend minimiert werden.

Unter den oben beschriebenen Randbedingungen wird schon die der eigentlichen Diagnose vorgelagerte Aufstellung von Testplänen (also die Festlegung welche der vorhandenen Tests in welcher Reihenfolge auszuführen sind) zu einer anspruchsvollen Aufgabe: Aus einer großen Zahl von Optionen müssen Entscheidungen gewählt werden – dabei sind komplexe Abhängigkeiten zu beachten, und viele der Entscheidungskriterien sind nur als Wahrscheinlichkeiten gegeben.

Zur Zeit werden Tests von verteilten Systemen üblicherweise (wenn überhaupt) von den Operateuren der Anlagen allein nach heuristischen Gesichtspunkten und ohne definierte Methodik geplant; hierzu ist große Erfahrung und umfangreiches Hintergrundwissen im Umgang mit Rechnern und Netzwerken nötig, sowie eine genaue Kenntnis der (oft ebenfalls nicht formal dokumentierten) Struktur und des Betriebszustandes des spezifischen Systems.

1.2 Ziel der Arbeit

In der vorliegenden Arbeit soll untersucht werden, wie mit Mitteln der stochastischen Modellierung ein Planungsunterstützungs-System für das Testen verteilter Rechenanlagen (**D**iagnosis **S**upport **S**ystem, **DSS**)[1] entwickelt und (prototypisch) implementiert werden kann.

Das DSS soll – auf der Basis einer formal definierten Methodik – die Auswahl von Diagnoseverfahren für verteilte Rechenanlagen unter Betonung der Kosten/Nutzen-Aspekte unterstützen, und zwar unter der Annahme, daß Tests überwiegend parallel zum normalen Betrieb des Systems durchgeführt werden (sog.

[1] Das DSS kann als ein Spezialfall (für technische Anwendungen) der als Planungshilfe für betriebswirtschaftliche Entscheidungen zunehmende Verbreitung findenden »Decision Support Systems« betrachtet werden; die für das DSS vorgesehenen Zielsysteme (d.h. große verteilte Rechenanlagen) werden auch oft zum Betrieb von Decision Support Systems eingesetzt. Allerdings unterscheidet sich das DSS in Problemstellung und Lösungsansatz erheblich von den derzeit im Einsatz befindlichen Decision Support Systems (deren Aufgabe im Wesentlichen aus der Extraktion von Wissen aus riesigen Datenbeständen besteht).

Online-Tests). Die Aufgabe des DSS ist es, für die jeweilige Systemkonfiguration eine Vorgehensweise zum Testen des Systems festzulegen. Diese soll:

- Abhängigkeiten und Einschränkungen (bezüglich Auswahl und Reihenfolge der Tests) berücksichtigen, die durch die Struktur des zu diagnostizierenden Systems sowie die Ausführungsbedingungen der zur Verfügung stehenden Tests und/oder durch Strategien zur Fehler-Diagnose gegeben sind.

- Die Grundlage für eine effiziente Fehlerdiagnose schaffen; dabei soll »Effizienz« in dem Sinne erzielt werden, daß der durch das Testen verursachte Verbrauch an Zeit und anderen Ressourcen bedarfsgerecht anpaßt werden kann (z.B. von möglichst geringer zusätzlicher Last auf das getestete System bis zu maximalem Informationsgewinn in möglichst kurzer Zeit).

- Den gesamten Vorgang der Testplan-Erstellung so weit wie möglich automatisieren; außer der Angabe der Struktur des zu testenden Systems sowie der zur Verfügung stehenden Tests, einer vergleichenden Bewertung von Nutzen und Einfluß der einzelnen Tests auf das System und der Zielsetzung der Effizienz-Optimierung sollte für die Erstellung von Testplänen durch das DSS keine Interaktion mit dem Anwender notwendig sein.

Voraussetzungen für die Anwendung des DSS

Es wird davon ausgegangen, daß die zu testenden Systeme in allen Bereichen (d.h. Hardware, Betriebssystem- und Anwendungssoftware) aus verschiedenen Komponenten bestehen, deren Zusammenspiel die für das Gesamtsystem erforderliche Funktionalität ermöglicht. Gleichzeitig wird das Vorhandensein von Testverfahren vorausgesetzt, mit denen die Betriebsbereitschaft der Komponenten selektiv überprüft werden kann. Die Tests werden nach der durch sie getesteten Funktionalität in *Moduln* gruppiert; es wird praxisnah ein weitgespanntes Fehlermodell angenommen, von Hardware-Ausfällen auf niedriger Ebene über Software-Defekte bis hin zu Konfigurations- und Bedienungs-Fehlern.

1.3 Stand der Technik und verwandte Arbeiten

Die vorliegende Arbeit stützt sich auf Vorarbeiten aus zwei Bereichen ab: Während für die Modell-Bildung Ideen aus der Modellierung mit stochastischen, bewerteten Petrinetzen aufgegriffen sowie zur Analyse Verfahren aus der Markov-Entscheidungsmodellierung auf die Ebene der Petrinetze transportiert werden, ist das Konzept der Modellierung von System-Tests an sich sowie der Ansatz zur Parametrisierung der Modelle von Arbeiten aus der Systemdiagnose geprägt.

1.3.1 Stochastische Modellierung

Verallgemeinerte stochastische Petrinetze bieten ein leistungsfähiges, theoretisch gut abgesichertes Instrumentarium zur Untersuchung von Fragestellungen im Zusammenhang mit nicht-deterministischen, durch Konkurrenz geprägten Systemen. Ajmone Marsan, Balbo e.a. geben in [AMBC86, AMBC$^+$95] eine umfassende Darstellung von Theorie und Anwendung stochastischer Petrinetze in der Informatik; in [AMBC$^+$95] (Kap. 10) wird auch die Generierung von Petrinetz-Modellen aus anderen Beschreibungen am Beispiel von parallelen Anwendungen gezeigt. German definiert in [Ger97] eine durchgängig formalisierte, anwendungsbezogene Spezifizierungssprache zur Beschreibung hierarchisch strukturierter Petrinetz-Modelle sowie der Bewertungsgrößen, mit denen der Modellierer die bei der Auswertung relevanten Fragestellungen in Bezug auf das Petrinetz kodieren kann – eine reduzierte Form dieser Spezifizierungssprache [All98, Des98] ist im für die DSS-Modellierung verwendeten Werkzeug PANDA implementiert worden und ermöglicht für das DSS eine automatisierte Generierung der Bewertungsstruktur der Modelle aus Parametern der zu testenden Systeme.

Markov-Entscheidungsmodelle sind seit längerem zur Behandlung von betriebswirtschaftlichen Problemen und zur Optimierung von Fertigungs- und Produktionsplanungssystemen bekannt; eine Darstellung der mathematischen Grundlagen dieses Modellierungsverfahrens mit einigen Anwendungsbeispielen aus dem Operations Research findet sich z.B. in [Tij86]. de Meer und Mauser bauen dieses Verfahren in [dMM91, dM92] aus: Es werden effiziente Algorithmen für die Analyse der Modelle (insbesondere im transienten Fall) eingeführt und in einem Werkzeug [dMŠ97] implementiert. Außerdem wird hier die Grundlage für eine formalisierte Abbildung des Zustandsraums einer erweiterten Klasse von stochastischen Petrinetzen auf Markov-Entscheidungsmodelle geschaffen – die Modelle werden auf Markov-Ebene so definiert, daß sie in allen außer den die Entscheidungen modellierenden Elementen isomorph zu der Klasse von Markov-Modellen sind, in die der Zustandsraum von stochastischen Petrinetzen zur Analyse eingebettet wird. In [Weg96] wird schließlich der naheliegende Schritt der Erweiterung von stochastischen, bewerteten Petrinetzen zur Entscheidungsmodellierung vollzogen; de Meer e.a. setzen diese neuen Petrinetz-Modelle zur Optimierung des Ressourcen-Managements bei Multimedia-Diensten ein [dMD96, dMDF99].

1.3.2 Systemdiagnose

Die Systemdiagnose ist ein eigenständiges Feld mit langjähriger Tradition innerhalb der Informatik. In den bekannten Arbeiten wird i.A. versucht, die Diagnose von komplexen Systemen als algorithmisches Problem der Auswahl von Tests bzw. des Ziehens von Schlußfolgerungen aus Testergebnissen zu lösen.

Ziel dabei ist, Testfolgen zum Auffinden eines Fehlers in einem gegebenen System zu bestimmen, die mit geringstem Testaufwand eine möglichst genaue Eingrenzung von potentiellen Fehlerquellen erreichen. Daneben wird auch versucht, die Testbarkeit (*testability*) im Sinne der »Diagnostizierbarkeit« eines Systems schon während dessen Entwicklung günstig zu beeinflussen (d.h. ein System möglichst schon so zu entwerfen, daß in ihm auftretende Fehler leicht und mit geringem Testaufwand zu finden sind).

Informationsfluß-Modelle

Wesentlichen Einfluß auf die vorliegende Arbeit hatten (trotz der weiter unten detaillierter beschriebenen grundlegenden Unterschiede in Zielrichtung und Lösungsansatz) die Arbeiten von Simpson und Sheppard, die in mehreren Artikeln [SS91b, SS91a, SS92c, SS92b, SS93a, SS93b, SS96] und einem Buch [SS94] eine umfassende Theorie zur Systemdiagnose sowie der Berücksichtigung von Diagnostizierbarkeit beim System-Entwurf entwickeln.

Dazu wird von Simpson und Sheppard (angeregt durch ähnliche Ansätze aus den Bereichen der Künstlichen Intelligenz und der Informationsfusion) ein sog. Informationsfluß-Modell (*information-flow model*) eingeführt: Basiselemente dieses Modells sind Tests und die aus ihrer Ausführung möglichen Schlußfolgerungen, die als »zu fusionierende« Informationsquellen betrachtet werden.

Tests und Schlußfolgerungen können in *replaceable unit groups*, *test groups* und *failure groups* zusammengefaßt werden; außerdem können den Tests Gewichtungsparameter zugewiesen werden, die Eigenschaften wie z.B. Zeitbedarf, Kosten oder für die Ausführung nötige Kenntnisse repräsentieren. Beziehungen zwischen Tests und Schlußfolgerungen werden graphisch in Abhängigkeitsgraphen (*dependency graphs*) und *logic diagrams* dargestellt; mathematisch werden diese Abhängigkeiten in 2 Matrizen beschrieben, (jeweils eine für Abhängigkeiten Test-von-Test- bzw. Test-von-Schlußfolgerung).

Diese Matrizen werden als die Wissensbasis des Gesamtsystems betrachtet; sobald sie in verschiedenen Aspekten (Transitivität, Freiheit von logischen Zirkeln etc.) abgeschlossen sind, kann das Modell analysiert werden. Durch die Analyse sollen Problembereiche in Bezug auf die Diagnostizierbarkeit (isolierte Bereiche, Test-Redundanz etc.) identifiziert werden. Maße auf Mengen von Tests (in Bezug auf Wirksamkeit, Eindeutigkeit, übermäßige oder unzureichende Beaufschlagung usw.) können verwendet werden, um das modellierte System zu bewerten und Schwachstellen zu finden. Aufbauend auf die Auswertung dieser Maße können dann existierende Tests verbessert oder neue Tests eingeführt werden, um die Testbarkeit des modellierten Systems zu erhöhen.

Nach der Optimierung der Menge der zur Verfügung stehenden Tests können aus den Matrizen Diagnosebäume (*diagnostic* oder *fault trees*) aufgebaut wer-

den, indem der Wert des Zuwachses an Information (*information value*) berechnet wird, der durch die Ausführung der einzelnen Tests erzielt werden kann. Dieser Informationswert hängt ab sowohl von der Zahl der Schlußfolgerungen, die implizit durch die Ausführung des Tests gezogen werden können, als auch von der Menge an Ungewißheit über den Zustand des getesteten Systems, die durch den Test beseitigt werden kann. Unter Verwendung des Informationswertes und der Gewichtung, die den Tests zugewiesen wurde, können die Diagnosebäume in Bezug auf verschiedene Zielsetzungen hin optimiert konstruiert werden; schließlich können durch eine geeignete Traversierung des Diagnose-Baumes effiziente Strategien zur Auswahl von Tests sowie zur Ableitung von Ausfallursachen aus den Testergebnissen gewonnen werden.

Ein Beispiel für die Anwendung des Informationsfluß-Modells auf ein hypothetisches Waffensystem wird in [SS91a, SS92c, SS92b, SS93a, SS93b] angegeben; andere Beispiele sind in [SS94] zu finden.

Aufbauend auf dem Informationsfluß-Modell schlagen Sheppard und Shombert eine formale Sprache für die Beschreibung des Verhaltens von zu testenden Systemen vor: Das in [SS99] propagierte sog. *behaviour information model* ist abgeleitet von objektorientierten Techniken und zielt darauf, wiederverwendbare und portable Test-Programme zu entwickeln.

Fehlerverzeichnis-Modelle

Ein anderer Ansatz im Bereich der Systemdiagnose ist die Verwendung von Fehlerverzeichnissen (*fault dictionary*): Hier wird eine Matrix aufgebaut, die alle durch Tests aufdeckbaren Fehler auf die entsprechenden Testergebnisse, d.h. die sog. Test-Signatur (*test signature* oder *test pattern*) abbildet; verschiedene Fehlerverzeichnis-Modelle und ihre Anwendung werden z.B. in [CMM70] beschrieben. Diese Matrizen dienen als Grundlage, um aus Testergebnissen auf Ausfallursachen zu schließen – bei geeignetem Aufbau des Fehlerverzeichnisses kann aus ihm und der Test-Signatur eine Diagnose des Systems direkt abgelesen werden.

Die Fehlerverzeichnis-Matrizen können entweder manuell erstellt, aus physikalischen Experimenten mit dem zu testenden System mit Fehler-Injektion abgeleitet, oder durch rechnergesteuerte Fehler-Injektion in ein Computermodell des Zielsystems simulativ ermittelt werden.

Verschiedene Maße zur Bewertung von Tests und Fehlerverzeichnissen wurden von Chang definiert: In [Cha65] werden Tests nach der Zahl der Fehler bewertet, die sie aufdecken können (und daraus wird ein Algorithmus zum Finden einer optimalen Diagnose-Strategie entwickelt); in [Cha68a] wird dieses Konzept mit der Einführung eines Maßes zur *fault distinguishability* verfeinert. In [Cha68b] werden schließlich Fehlerverzeichnisse in Kategorien wie Genauigkeit (*accuracy*) und Auflösung (*resolvability*) bewertet.

Mandelbaum wendet in [Man64] die von Shannon [Sha48] definierte Entropie-Funktion aus der Informationstheorie als Maß für die Effizienz eines Tests an: Ähnlich wie bei den durch Chang eingeführten Maßen wird auch hier die Zahl der Komponenten, die ein Test als betriebsbereit oder fehlerhaft identifiziert, als Grundlage für die Bewertung verwendet.

Sowohl die von Chang als auch von Mandelbaum definierten Maße werden unter dem Begriff Überdeckung (*test coverage* bzw. *fault coverage*) zusammengefaßt; Überdeckungsmaße quantifizieren die Menge an diagnostischer Information, die die Ausführung eines Tests in Bezug auf eine Menge von Fehlern liefern kann – dies kann entweder durch die Zahl der möglichen Fehler gegeben sein, die durch einen Test aufgedeckt werden können, oder auch durch die Zahl der Vermutungen, die durch den Test ausgeschlossen werden können (in diesem Sinne können auch die von Simpson und Sheppard eingeführten Maße als Überdeckung interpretiert werden).

Auch Abraham und Agarwal verwenden in einer Arbeit zur Qualität von Tests [AA86] die Überdeckung als ein Bewertungskriterium; weiterhin wird eine Mißerfolgsrate für das Testen (*field reject rate*) definiert, die mißt wieviele der getesteten Einheiten nach Abschluß der Tests noch unerkannte Defekte aufweisen (dieses Maß kann aber nur durch Schätzungen ermittelt werden und ist statistisch mit der Fehlerüberdeckungsrate korreliert).

In [SS96] beschreiben Simpson und Sheppard eine Strategie, um unsichere und uneindeutige Testergebnisse im Zusammenhang mit Fehlerverzeichnissen aufzulösen: Wenn eine unbekannte Test-Signatur auftritt, wird diese heuristisch (d.h. unter Anwendung statistischer Verfahren) mit allen bekannten Signaturen verglichen, um den aufgetretenen Fehler einzukreisen.

Kime entwickelt in [Kim86] andere Repräsentationen von Fehlerverzeichnissen sowie Verfahren, um diese zu erstellen: Die Modelle werden durch Bool'sche Ausdrücke und Digraphen beschrieben, und es wird gezeigt wie aus diesen Darstellungen Diagnose-Bäume abgeleitet werden können. Außerdem wird (ausgehend von einer hierarchischen Struktur des zu testenden Systems) eine Hierarchie auf der Menge der Tests sowie der Fehler definiert, die von Ausfällen in Geräten bis zum Versagen von Netzwerken reicht. Ein aus der Informationstheorie abgeleiteter, heuristischer Ansatz zur Generierung von Diagnose-Bäumen wird schließlich von Hartmann e.a. in [HVMG82] vorgeschlagen.

Expertensysteme

Hier wird versucht, die Erfahrung und das Fachwissen eines menschlichen Experten nachzubilden, indem regelorientierte, wissensbasierte Werkzeuge aufgebaut werden [DCP88, Pup87]. Zum Beispiel basiert das in [EP90] entwickelte System zum Testen eines massiv parallelen Rechners auf einer kommerziel-

len Expertensystem-Shell, wobei die Wissensbasis die Test-Erfahrungen bei der Entwicklung des Systems reflektieren; für DIAMONT [Phi91] wurde eine eigene, objektorientierte Entwicklungsumgebung auf der Grundlage eines PROLOG-Interpreters geschaffen. Auch Abramovici, Breuer und Friedman beschreiben in [ABF90] die Grundlagen des Einsatzes von Expertensystemen zur Systemdiagnose, stellen aber gleichzeitig auch die Einschränkungen der üblichen, regelbasierten Expertensysteme in puncto Erweiterbarkeit der Wissensbasis sowie der Integration von empirischem Wissen heraus.

Andere Quellen

Eine Klassifikation von Fehlern und Ausfällen in verteilten Betriebssystemen findet sich in [Tan95]: Es werden die Komponenten in einem verteilten System beschrieben, die ausfallen können, und den Ausfälle werden Kategorien (*temporary*, *periodically* und *permanent*) zugeordnet. Außerdem wird auf die übliche Weise zwischen »normalem« und »byzantinischem« Ausfallverhalten unterschieden: Byzantinische Fehler äußern sich in einem (für die Außenwelt) scheinbar funktionierendem Verhalten der betroffenen Einheit, wobei aber falsche Resultate geliefert werden. In der vorliegenden Arbeit werden (im Wesentlichen zur Reduzierung der Modellkomplexität) nur permanente Fehler betrachtet; außerdem wird davon ausgegangen, daß durch die zur Verfügung stehenden Tests auch byzantinisches Fehlverhalten detektiert werden kann.

McRee schlägt in [McR95] schließlich einen offenen Standard für eine Test-Infrastruktur innerhalb von Netzwerk-Management-Systemen vor, der in den von der International Organisation for Standardisation bzw. der International Telecommunications Union definierten Standard zur *Test Management Function* (TMF) integriert ist. Innerhalb des TMF wird zwischen gesteuerten und ungesteuerten Tests zur Überprüfung von Einheiten über ein Netzwerk unterschieden, und es werden Test-Kategorien und -Parameter für beide Arten von Tests definiert.

1.3.3 Diskussion der existierenden Ansätze

Die bekannten Arbeiten zur Systemdiagnose beschäftigen sich zumeist mit dem Problem des Ableitens von diagnostischer Information aus den Ergebnissen von Tests; Gesichtspunkte wie die Kosten des Testens spielen (wenn überhaupt) eine untergeordnete Rolle. Es wird vorausgesetzt, daß das zu diagnostizierende System für ausreichende Zeit exklusiv zum Testen zur Verfügung steht – dies kann aber bei wichtigen Systemen, die Dienstleistungen unterbrechungsfrei erbringen müssen, nicht mehr ohne Weiteres angenommen werden.

In bisherigen Ansätzen liegt der Schwerpunkt zudem im Wesentlichen auf der Diagnose von Hardware-Fehlern, d.h. es wird davon ausgegangen, daß die im

Einsatz befindliche Software stets spezifikationsgemäß arbeitet und auch korrekt installiert und konfiguriert betrieben wird. Gerade letztere Annahme beeinflußt bei den bekannten modellbasierten Verfahren die gewählten Methoden und Maße nachhaltig, ist aber für große verteilte Systeme heutzutage so nicht mehr zu rechtfertigen: Erfahrungsgemäß spielen hier Hardware-Defekte eine eher zweitrangige Rolle als Ursache von Systemausfällen im Vergleich zu Software-Fehlern (d.h. Implementierungsdefekten sowie falscher Bedienung und Konfiguration von System- und Anwendungs-Programmen).

Beim derzeitigen Stand der Technik ist die Hardware, die bei wichtigen verteilten Systemen zum Einsatz kommt, zumeist schon redundant oder sogar fehlertolerant ausgelegt – dagegen trifft dies auf die eingesetzte Software bisher nur in äußerst eingeschränktem Maß zu. Ein Grund hierfür ist sicher, daß leistungsfähige Hardware immer preisgünstiger zur Verfügung steht, während die Kosten der Erstellung qualitativ hochwertiger Software ständig steigen. In der Folge ist zur Zeit die (der redundanten Auslegung von Hardware entsprechende) mehrfache unabhängige Implementierung von Software-Komponenten nur in Bereichen mit extremen Anforderungen an Zuverlässigkeit oder sehr hohem Verlust-Potential (wie z.B. Luft- und Raumfahrt) üblich.

Sollen aber auch Tests für Software-Komponenten berücksichtigt werden, sind die auf Hardware zugeschnittenen Maße zur Beurteilung des Informationswertes von Tests (wie z.B. Überdeckung) allein nicht mehr aussagekräftig genug: Wenn bei bei der Planung von Tests nicht mehr nur die direkte Gewinnung von diagnostischen Informationen im Vordergrund steht, sondern auch die Auswirkungen der Tests auf den Betrieb des Systems berücksichtigt werden sollen, wird die Wahl von anderen Maßen zur Beurteilung der Tests möglich (und auch notwendig) – die bekannten diagnostischen Maße müssen zwar nach wie vor in die Klassifizierung der vorhandenen Tests und die Modellierung des Testablaufs einbezogen werden, reichen aber nicht mehr aus. Zusätzliche, spezifisch auf die Struktur und den Einsatzzweck des Systems bezogene Informationen müssen ausgewertet werden.

Auf Expertensystemen basierende Verfahren haben verschiedene Probleme: Neben der teilweise unzureichenden Effizienz durch den hohen Verarbeitungsaufwand und dem Problem, genügend umfangreiche und detaillierte Wissensbasen zu generieren kann auch nur implizites Wissen über das zu diagnostizierende System eingebracht werden. Außerdem sind mit Wissensbasis und Regelsystem wesentliche, grundlegende Elemente des Diagnosewerkzeugs speziell auf ein einziges System zugeschnitten und damit weitgehend nicht portabel.

Dagegen wird bei modellbasierten Verfahren externes Wissen über die Struktur und das Verhalten des zu diagnostizierenden Systems in eine Form gebracht, die mittels standardisierter Modellierungsverfahren bearbeitbar ist. Jedoch können die meist auf der Repräsentation von Information über das zu diagnostizierende

System in Form 2-dimensionaler Matrizen basierenden, bekannten Verfahren nur schlecht wesentliche Eigenschaften verteilter Systeme modellieren, wie z.B. die zum großen Teil nur stochastisch beschreibbaren Parameter der Tests, oder daß viele Komponenten über verschiedene Verbindungen erreichbar bzw. testbar sind – diese hohe Konnektivität kann mittels Test-Abhängigkeits-Matrizen oder Fehlerverzeichnissen nur schwer dargestellt werden.

1.4 Lösungsansatz und Übersicht

Zur Modellierung von Systemen, die durch komplexe Abhängigkeiten, stochastisches Verhalten sowie große diskrete Zustandsräume gekennzeichnet sind, haben sich stochastische bewertete Petrinetze bewährt – das Ziel der Modellierung mit diesem Verfahren ist jedoch die Bestimmung des (erwarteten) *Verhaltens* des analysierten Systems.

Mit der Markov-Entscheidungsmodellierung existiert aber auch eine komplette mathematische Theorie zur Optimierung von *Entscheidungen* in speziellen (d.h. durch die Markov-Eigenschaft näher beschriebenen) stochastisch geprägten Kontexten, in denen auch eine Kontrolle über das Verhalten des modellierten Systems ausgeübt werden kann – das Verfahren ist jedoch bisher nur wenig angewendet worden, da die Modell-Bildung direkt auf Markov-Ebene mühsam und fehleranfällig ist.

Modellierungsmethode

In der vorliegenden Arbeit soll (mit Bezug auf die auf S. 2 geschilderte Problematik) der folgende Ansatz zum Aufbau von Modellen mit dem Ziel der Bestimmung von Testplänen für verteilte Systeme untersucht werden:

1. Das DSS ist modellbasiert, d.h. der Vorgang des Testens eines verteilten Systems wird – auf der Basis einer zur Entscheidungsmodellierung passend erweiterten Definition von stochastischen bewerteten Petrinetzen (die in Kap. 3 näher beschrieben wird) – in einem Zustands-/Übergangs-Modell nachgebildet.

2. Durch die Mittel der logischen Verknüpfungen von Petrinetz-Elementen werden im Modell die Struktur des zu testenden Systems und der zur Verfügung stehenden Tests, die funktionalen Abhängigkeiten der Systemkomponenten bzw. Tests untereinander sowie evtl. durch übergeordnete Strategien zur Systemdiagnose gegebene Einschränkungen bei der Auswahl und Reihenfolge von Tests repräsentiert.

3. Diejenigen Parameter und Randbedingungen der System-Tests, die nur mit Unsicherheit behaftet beschrieben werden können, werden über entsprechende stochastische Petrinetz-Parameter (Schaltwahrscheinlichkeiten und -raten) modelliert.

4. Für die Repräsentation der möglichen Entscheidungsaktionen (d.h. ob und wann Tests ausgeführt werden sollen) im Modell wird eine zusätzliche Klasse von Transitionen (sog. *Rekonfigurationstransitionen*) verwendet, deren Schalten (über den Erreichbarkeitsgraphen des Modells) auf kontrollierte Übergänge in einem Markov-Entscheidungsmodell abgebildet werden kann.

5. Die möglichen Optionen bei der Auswahl von Tests werden so modelliert, daß der Algorithmus zur Entfaltung des Zustandsraums von Petrinetzen verwendet werden kann, um aus einer kompakten Darstellung der möglichen Kombinationen von einzelnen Tests im Modell die Menge aller potentiell darstellbaren Sequenzen abzuleiten.

6. Die verfügbaren Tests werden einer bewertenden Klassifizierung unterzogen, die alle Tests sowohl bezüglich ihres zu erwartenden Nutzens (d.h. dem Gewinn an diagnostischer Information nach der Ausführung) als auch bezüglich der Kosten (d.h. dem zusätzlichen Verbrauch an Ressourcen durch das Testen) in einen einheitlichen Rahmen einordnet.

7. Auf Petrinetz-Ebene wird eine Bewertungsstruktur definiert, die die aus der Klassifizierung abgeleiteten kosten-/nutzenbezogenen Parameter der Tests in entsprechende Maße für Zustände und Übergänge im Modell übersetzt; diese Maße werden bei der Generierung des Erreichbarkeitsgraphen in Bewertungen auf Markov-Ebene umgewandelt und dann als Zielgröße der Entscheidungs-Optimierung eingesetzt.

Damit enthält der Erreichbarkeitsgraph des Petrinetz-Modells alle Informationen, um als Markov-Entscheidungsmodell interpretiert und analysiert zu werden; die Ergebnisse der Analyse können in einen (im Mittel) auf die Anwenderspezifizierten Ziele (bezüglich Systembelastung und Informationsgewinn) hin optimierten Plan zum Testen des modellierten Systems umgerechnet werden

Auf diese Weise können die Vorteile von Petrinetzen für die Modell-Bildung genutzt werden: Einfache Modellierung nebenläufiger Vorgänge, Überflüssigkeit der expliziten Angabe großer Zustandsräume, anwendungsbezogene Angabe von Bewertungsmaßen, Visualisierbarkeit der Modelle über die graphische Darstellung der Netzpläne.

Über die Interpretation des Erreichbarkeitsgraphen der Netze können diese Vorzüge auf einfache Weise mit den für Markov-Entscheidungs-Modelle existierenden analytischen Verfahren zur dynamischen Strategieoptimierung für die Auswertung der Modelle kombiniert werden.

Erstellung der Testplan-Modelle

Zur Generierung eines Testplans mit dem DSS werden zunächst die funktionalen Abhängigkeiten der Komponenten des zu testenden Systems in einem Petrinetz-Modell nachgebildet (sog. *Abhängigkeitenmodell*). Dieses Modell bildet auch die logisch möglichen Testabläufe ab, da es beschreibt, welche Einzel-Tests als Voraussetzung für den Start anderer Tests erfolgreich absolviert worden sein müssen oder welche Test-Sequenzen unabhängig voneinander (und damit möglicherweise parallel) ausgeführt werden können.

Spezielle Transitionen im Abhängigkeitenmodell werden hierzu als mögliche Ausführung von Testmoduln interpretiert. Im nächsten Schritt werden diese Transitionen zu Subnetzen verfeinert, um Parameter sowie Bewertungsmaße (Ressourcenverbrauch während der Ausführung, möglicher Informationsgewinn nach Beendigung etc.) für das jeweilige Testmodul detailliert spezifizieren zu können (das zur hierarchischen Strukturierung der Petrinetze verwendete Verfahren wird in Kap. 4 erläutert).

Dabei werden alternative Test-Sequenzen identifiziert und ihre Einsprungstellen im Petrinetz durch Rekonfigurationstransitionen markiert; das so entstandene erweiterte Petrinetz modelliert die verschiedenen möglichen Test-Sequenzen unter durch die Bewertungsmaße gegebenen Randbedingungen. Die Vorgehensweise bei der Erstellung der Testplan-Modelle wird in Kap. 5 beschrieben.

Nach der Parametrisierung des Petrinetz-Modells und der Transformation des Erreichbarkeitsgraphen des Netzes zum Markov-Entscheidungsmodell resultiert dessen Analyse in (im Mittel) optimalen Schaltzeiten für die Rekonfigurationstransitionen. Die Ergebnisse der Analyse werden in einem weiteren Schritt (der in Kap. 7 beschrieben wird) in einen (ebenfalls im Mittel) optimalen Testplan umgerechnet.

Parametrisierung und Bewertungsstruktur der Modelle

Als Grundlage für die Strategieoptimierung auf Markov-Ebene dienen Bewertungsmaße, die in das zur Testplan-Erstellung gebildete erweiterte Petrinetz eingebracht werden müssen (Optimierungskriterium bei der Markov-Entscheidungs-Analyse ist der Erwartungswert eines Bewertungsmaßes unter variierender Strategie). Diese Bewertungsmaße müssen die Kosten bzw. den Nutzen der zur Option stehenden Testpläne reflektieren, und sie müssen (den die Ausführung der

Tests modellierenden) Zuständen und Zustandsübergängen im Markov-Entscheidungsmodell zugeordnet werden. Um auch die Bewertungsmaße auf Petrinetz-Ebene angeben zu können (d.h. hierfür nicht auf Elemente des Erreichbarkeitsgraphen zugreifen zu müssen), wurde eine Beschreibungssprache implementiert, deren Struktur und Abbildung auf Markov-Ebene in Kap. 2 und App. A beschrieben wird. Zusätzlich zu den Bewertungsmaßen müssen Petrinetz-Elementen in den Subnetzen, mit denen die Ausführung von Testmoduln modelliert wird, stochastische Parameter zugewiesen werden (z.B. geht die für die Ausführung eines Moduls durchschnittlich benötigte Zeit als Schaltrate einer Transition ein).

Sowohl zur Bildung der Bewertungsmaße als auch zur Festlegung der stochastischen Parameter für die Petrinetz-Modelle muß eine Vorgehensweise gefunden werden, um Tests vergleichend klassifizieren zu können. Durch die Unterschiedlichkeit der Testmoduln (Hardware, Software) und die Vielfalt der möglichen Fehler bedingt, muß diese Klassifizierung sehr flexibel gestaltet werden, um allen Testmoduln eines Systems in einem einheitlichen Rahmen Bewertungsparameter zuweisen zu können; gleichzeitig müssen die Petrinetz-Bewertungsmaße und -Parameter auf formalisierte Weise aus den bei der Klassifizierung gefundenen numerischen Werten berechenbar sein, damit dieser Vorgang in einem Werkzeug automatisierbar wird. Zu diesem Zweck wird eine Testmodul-zu-Systemkomponenten-Matrix aufgestellt und deren Einträge nach verschiedenen Kriterien numerisch gewichtet werden; dabei können die konkreten Daten von MTTF-Angaben der Hardware-Hersteller bis zu auf Betriebserfahrungen basierenden heuristischen Werten über die Wahrscheinlichkeit von Software-Fehlern variieren (das angewendete Verfahren wird in Kap. 6 vorgestellt).

Im Resultat wird jedem Testmodul ein Kosten- (Ressourcenverbrauch während der Ausführung) sowie ein Nutzenmaß (bezogen auf den möglichen Informationsgewinn) je nach Ausgang des Tests zugewiesen; die Maße aller Testmoduln (entsprechend aller Subnetze des Testplan-Modells) ergeben dann die Zielfunktion der Entscheidungs-Optimierung. Die Zielrichtung der Optimierung hinsichtlich eines möglichst raschen (d.h. auf maximalen Informationsgewinn abzielenden) oder gering belastenden (also auf minimalen Ressourcenverbrauch durch Testen angelegten) Testplans kann gesteuert werden, indem die Gewichtung des Nutzenmaßes gegenüber dem Kostenmaß entsprechend angepaßt wird.

Struktur des DSS

Eine Übersicht zum Aufbau des DSS ist in Abb. 1.1 dargestellt: Es wurde eine mehrschichtige Architektur gewählt, wobei in Schicht 1 die direkt auf das modellierte System bezogenen Eingabedaten in einem Format (Petrinetz zur Entscheidungsmodellierung mit Bewertungsmaßen) angegeben werden, das in Schicht 2 von einem Analyse-Werkzeug bearbeitet wird (Generierung eines Markov-Ent-

scheidungs-Modells und Optimierung). In der Präsentations-Schicht 3 werden die Ergebnisse der Entscheidungs-Optimierung in Anweisungen zur Ausführung der in Schicht 1 angegebenen Testmoduln umgerechnet.

Die Parametrisierung und Aufstellung der Bewertungsmaße für die Petrinetze wurde in einem Werkzeug implementiert, so daß die Generierung von Testplänen direkt aus den Bewertungen der Testmoduln und dem verfeinerten Abhängigkeitenmodell erfolgen kann. In der vorliegenden prototypischen Implementierung wird dabei die Struktur des zu testenden Systems sowie die Diagnose-Strategie in Form eines Petrinetzes (in einem von PANDA verarbeitbaren Format) beschrieben, während die Menge der Testmoduln mit ihren Bewertungsparametern als Datensätze in einer interpretierten Programmiersprache [WCS96] vorliegen.

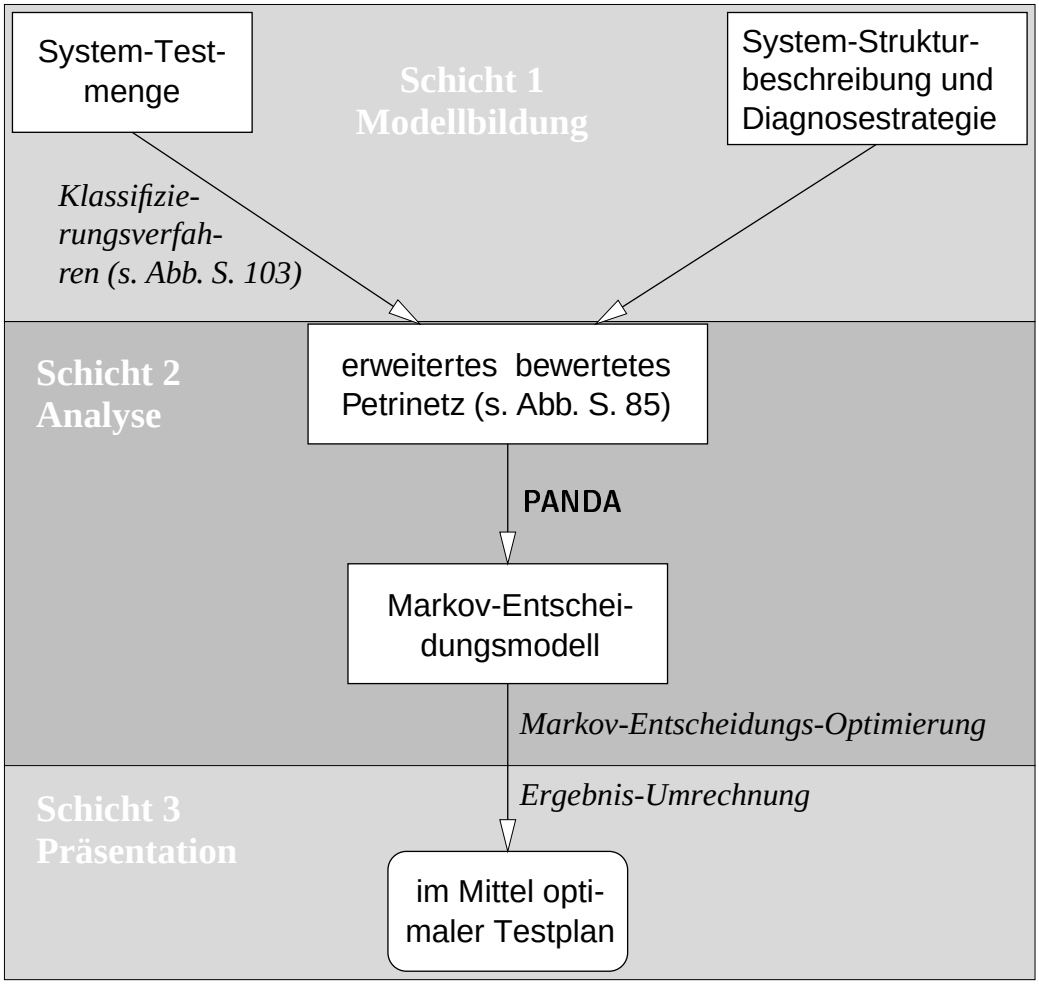


Abbildung 1.1: Übersicht der DSS-Architektur

1.5 Einbettung in PANDA

Da am Lehrstuhl Informatik III schon Kompetenz in der Entwicklung eigener Werkzeuge zur Zuverlässigkeits-Modellierung vorhanden war, lag es nahe, diese bei Untersuchungen im Bereich der Fehlertoleranz bewährten Modellierungs-Werkzeuge auch für die Erstellung von optimalen Testplänen für verteilte Systeme einzusetzen.

Für quantitative Bewertungen von fehlertoleranten Rechnern wird am Lehrstuhl der »**P**etri net **An**alysis and **D**esign **A**ssistant« (PANDA) zur Modellierung und Analyse auf der Basis von verallgemeinerten stochastischen Petrinetzen entwickelt. Für die Implementierung des DSS wurden an diesem Werkzeug folgende Erweiterungen vorgenommen:

Hierarchische Modelldarstellung

Beim Einsatz von großen Petrinetz-Modellen für reale Anwendungen schlägt der Vorteil der (für kleine Netze) leicht verständlichen, standardisierten graphischen Darstellungsweise mit zunehmendem Umfang der Modelle oft in das Gegenteil um – die Netzpläne werden unübersichtlich, und da immer nur Teile gleichzeitig dargestellt werden können, werden Zusammenhänge schlecht erkennbar.

Aus diesem Grund wurde PANDA um die Möglichkeit zur hierarchischen Darstellung von Netzplänen erweitert. Um dem Modellierer höchstmögliche Flexibilität zu erlauben, wurde dies in Form von kantenberandeten Subnetzen implementiert; diese können in beliebiger Hierarchisierungstiefe geschachtelt werden und erlauben eine modulare Visualisierung von komplexen Netzplänen. Durch geeignete Erweiterungen der Operationen des Graphik-Editors von PANDA können Subnetze einfach und schnell erstellt, kopiert und bei Bedarf auch wieder aufgelöst werden.

Beim Aufbau der Petrinetze für das DSS wird diese hierarchische Darstellung angewendet, indem jedes Testmodul als ein Subnetz modelliert wird.

Bewertungsmaße

In PANDA wurde eine Beschreibungssprache (mit zugehörigem Interpreter) realisiert, die es gestattet auf Petrinetz-Ebene beliebige Zustände oder Zustandsübergänge im Erreichbarkeitsgraphen des Netzes zu markieren und (mittels arithmetischen oder logischen Verknüpfungen) zu Bewertungs-Funktionen zusammenzufassen. Für diese Funktionen können dann bei der Analyse je nach Bedarf automatisch stochastische Ergebnisgrößen (wie z.B. Erwartungswert) berechnet werden. Damit können mit PANDA *Stochastic Reward-Nets* (SRN) modelliert und ausgewertet werden.

Die Bewertungsmaße für die DSS-Modelle werden in dieser Beschreibungssprache generiert und an PANDA übergeben.

Entscheidungsmodellierung

Speziell für den Einsatz im Rahmen des DSS wurde in PANDA die Möglichkeit integriert, erweiterte SRN auf Markov-Entscheidungs-Modelle abzubilden. Hierzu können durch die Einführung einer neuen Klasse von Transitionen Entscheidungsalternativen modelliert werden, die (im Zusammenhang mit Bewertungsmaßen für Modellzustände und -übergänge) die Formulierung von optionalen Strategien unter vorgegebenen Optimierungs-Randbedingungen erlauben: Modelliert werden durch das Schalten von Rekonfigurationstransitionen *kontrollierte* Aktionen, die – im Gegensatz zu den durch stochastische Gesetzmäßigkeiten bestimmten Abläufen, die mit den »normalen« Transitionen der SRN assoziiert sind – unter der Bedingung der Optimierung des Erwartungswertes eines Bewertungsmaßes stattfinden.

Dazu wird bei der Generierung des Erreichbarkeitsgraphen des Petrinetzes das Feuern der neuen Transitionen auf Entscheidungs-Übergänge auf Markov-Ebene abgebildet; im Zusammenhang mit der Übertragung der oben genannten Bewertungsmaße aus dem erweiterten Petrinetz auf das Markov-Entscheidungsmodell kann dann ein vorhandenes AnalyseWerkzeug [dMŠ97] verwendet werden, um im Mittel optimale Schaltzeiten für die Rekonfigurationstransitionen zu berechnen.

In den für das DSS aufgebauten Modellen ist für jedes Testmodul die Option zur Ausführung durch eine Rekonfigurationstransition repräsentiert.

Hinweis für den Leser

Definierte Begriffe sind im Index (S. 139) vermerkt und an der Stelle ihrer Einführung ***kursiv fett*** gesetzt.

Teil I

Grundlagen

Kapitel 2

Bewertungsmaße für Petrinetze

In diesem Kapitel wird ein Verfahren beschrieben, das es erlaubt, Bewertungsmaße für quantitative Petrinetz-Modellierung auf Netz-Ebene zu spezifizieren. Mit diesen Bewertungsmaßen werden später die für das DSS erstellten Modelle parametrisiert sowie die Randbedingungen für die Optimierung angegeben.

2.1 Definition von SRN

Das Ziel jeder Art von quantitativer Modellierung ist eine Bewertung der modellierten Systeme. Bei der klassischen Modellierung mit stochastischen Petrinetzen wird für die quantitative Analyse ein dem Petrinetz (in Bezug auf die stochastischen Charakteristika) isomorpher Semi-Markov-Prozess ausgewertet [Mol81]. Die dabei anfallenden Ergebnisse sind Wahrscheinlichkeiten von Markov-Zuständen bzw. durchschnittliche Raten für Übergänge zwischen den Zuständen des Prozesses; sie werden als »Standardresultate« (d.h. Erwartungswerte für die Zahl von Token pro Stelle bzw. den Durchsatz von Transitionen) auf das Ausgangs-Petrinetz zurückgerechnet.

Für die Beantwortung von komplexeren Fragen an das stochastische Modell sind diese Ergebnisse jedoch oft zu indirekt auf die für den Modellierer relevanten Resultate bezogen, oder die Petrinetz-Modelle müßten aufwendig so konstruiert werden, daß die interessierenden Größen als Standardresultate anfallen.

Wünschenswert sind daher Beschreibungsverfahren, die es erlauben, dem Analyse-Prozess in Begriffen des spezifischen Modells genau die Größen anzugeben die berechnet werden sollen. Dabei müssen diese von der stochastischen Analyse zu berechnenden Ergebnisse spezifiziert werden können, ohne daß für ihre Formulierung Kenntnisse über den aufgefalteten Zustandsraum des Petrinetzes oder über mathematische Details der Markov-Analyse nötig sind.

Im Rahmen der Petrinetz-Modellierung sind Formalismen, die solche erweiterten Abfragen an die Modelle gestatten, als ***stochastische bewertete Petrinetze*** (*Stochastic Reward Nets, SRN*) definiert worden. Eine gute Darstellung von Definition und Auswertungsverfahren findet sich in [CBC+93]; die dort verwendete Terminologie wird für die vorliegende Arbeit übernommen.

2.1.1 Stochastische bewertete Petrinetze

Da stochastische bewertete Petrinetze der Ausgangspunkt des für das DSS verwendeten Modellierungs-Paradigmas sind, wird an dieser Stelle eine komplette Definition wiedergegeben. Ein ***SRN*** wird definiert als ein 11-Tupel:

$$\mathbf{A} = \{P, T, D^-, D^+, D^\circ, \mu_0, g, >, \lambda, w, M\}$$

Dabei sind:

1. $P = \{p_1, \ldots, p_{|P|}\}$ und $T = \{t_1, \ldots, t_{|T|}\}$ mit $P \cap T = \emptyset$ die (endlichen) Mengen der Stellen und Transitionen des Netzes, wobei jede Stelle p eine (ganzzahlige) Anzahl von Token (Notation: $\sharp(p)$) größer oder gleich Null beinhalten kann.

2. D^+, D^- bzw. D° die die ***Multiplizität*** beschreibenden Funktionen für die als Teilmenge von $P \times T$ definierten Eingangs- (D^+), Ausgangs- (D^-), bzw. inhibitorischen (D°) Kanten. Diese Funktionen werden in Form von Adjazenz-Matrizen angegeben: Es existiert genau dann eine entsprechende Kante zwischen einer Stelle p und einer Transition t, wenn die jeweilige – von der aktuellen Markierung abhängige – Multiplizitätsfunktion $D^\star_{p,t} : \mathbb{N}^{|P|} \to \mathbb{N}$ (mit $\star \in \{+, -, \circ\}$) zu einer positiven Zahl auswertet. Für Markierungen, in denen ihre Multiplizität zu 0 wird, werden Kanten als nicht vorhanden betrachtet.

   Eine Transition $t \in T$ wird in einer Markierung $\mu \in \mathbb{N}_0^{|P|}$ als ***kantenbedingt feuerbereit*** betrachtet genau dann, wenn

   $$D^-_{p,t}(\mu) \leq \sharp(p,\mu) \wedge (D^\circ_{p,t}(\mu) > \sharp(p,\mu) \vee D^\circ_{p,t}(\mu) = 0) \ (\forall p \in P,\, t \in T)$$

   Nach dem Feuern einer Transition t in Markierung μ_i erfüllt die neue Markierung μ_j die Bedingung:

   $$\sharp(p,\mu_j) = \sharp(p,\mu_i) - D^-_{p,t}(\mu_i) + D^+_{p,t}(\mu_i) \ (\forall p \in P,\, t \in T)$$

   Die Multiplizitätsfunktionen $D^-_{p,t}$ bzw. $D^+_{p,t}$ für μ_i werden hier ausgewertet *bevor* t feuert.

3. $\mu_0 \in \mathbb{N}_0^{|P|}$ die ***Anfangsmarkierung*** des Netzes. Eine Markierung μ heißt von μ_0 aus ***erreichbar*** (Notation: $\mu_0 \xrightarrow{\star} \mu$), falls es Transitionen $t_1, \ldots, t_n$ gibt, so daß μ durch das Feuern von $t_1, \ldots, t_n$ aus μ_0 entsteht.

   Die ***Erreichbarkeitsmenge*** des Petrinetzes kann dann definiert werden als die Menge aller von μ_0 aus erreichbaren Markierungen:

   $$\mathfrak{S} = \{\mu \in \mathbb{N}_0^{|P|} : \mu_0 \xrightarrow{\star} \mu\}$$

   $\mathfrak{S}$ bildet die Knoten des ***Erreichbarkeitsgraphen*** von **A**, der dem vollständigen ***Zustandsraum*** des Petrinetzes entspricht. Die Kantenmenge $\mathfrak{E}$ des Erreichbarkeitsgraphen ist gegeben als die Menge der Übergänge zwischen den Zuständen aus $\mathfrak{S}$ (jeder dieser Zustandsübergänge repräsentiert das Feuern einer Transition):

   $$\mathfrak{E} \subseteq \mathfrak{S} \times \mathfrak{S} \text{ mit } \mu_i, \mu_j \in \mathfrak{S}, t \in T \wedge \mu_i \xrightarrow{t} \mu_j \;\Rightarrow\; \mu_i \xrightarrow{t} \mu_j \in \mathfrak{E}$$

4. $g_t : \mathfrak{S} \to \{1,0\}$ $(\forall t \in T)$ die ***Inhibitorfunktion*** (*guard function*), wobei t blockiert ist in jeder Markierung $\mu \in \mathfrak{S}$ in der gilt $g_t(\mu) = 0$ (auch wenn t kantenbedingt feuerbereit wäre).

5. $>$ eine irreflexive, transitive Relation auf T, durch die den Transitionen ***Prioritäten*** zugeordnet werden.

   Für alle Transitionen $t_i \in T$ gilt: t_i ist in einer Markierung $\mu \in \mathfrak{S}$ ***markierungsbedingt feuerbereit*** genau dann, wenn t_i kantenbedingt feuerbereit ist, $g_{t_i}(\mu) = 1$ gilt und es keine Transition $t_j \in T$ gibt mit $t_j > t_i$.

6. $\lambda_t : \mathfrak{S} \to \mathbb{R}_\infty^+$ $(\forall t \in T)$ die (markierungsabhängigen) ***Schaltraten*** der exponentiell verteilten Zufallsvariablen, die die Verzögerungs-Zeiten beim Feuern der Transitionen modellieren. Dabei gibt der Modellierer für das Petrinetz die (endliche) Rate für eine isolierte Transition (d.h. für eine, die mit Wahrscheinlichkeit 1 feuert) an. Beim Aufbau der zum Petrinetz isomorphen Semi-Markov-Kette während der stochastischen Analyse werden die isolierten Raten mit der Feuerwahrscheinlichkeit der jeweiligen Transition (für die evtl. konkurrierende andere Transitionen berücksichtigt werden müssen) zur ***effektiven Schaltrate*** verrechnet.

   Zur Vereinfachung des Formalismus wird hier (wie in [CBC+93], abweichend von der ursprünglichen GSPN-Definition z.B. in [AMBC84]) $\lambda_t = \infty$ gesetzt für ***zeitlose*** Transitionen, die ohne jede Verzögerung feuern sobald sie markierungsbedingt feuerbereit sind (*immediate transition*), im Gegensatz zu den ***zeitbehafteten*** Transitionen (*timed transition*) mit endlicher Feuerrate.

Mit dieser Definition kann die übliche Klassifizierung der Markierungen, d.h. eine disjunkte Zerlegung von $\mathfrak{S} = \mathfrak{S}_V \cup \mathfrak{S}_T$ (mit $\mathfrak{S}_V \cap \mathfrak{S}_T = \emptyset$) vorgenommen werden: Eine Markierung $\mu \in \mathfrak{S}_V$ wird als ***zeitlos*** (*vanishing*) bezeichnet, falls es eine markierungsbedingt feuerbereite Transition t gibt mit $\lambda_t(\mu) = \infty$; andernfalls wird $\mu \in \mathfrak{S}_T$ ***zeitbehaftet*** (*tangible*) genannt.

Über die unendlichen Schaltraten wird auch eine ***implizite Priorisierung*** der zeitbehafteten gegenüber den zeitlosen Transitionen eingeführt: Eine Transition t ist in einer Markierung $\mu \in \mathfrak{S}$ ***feuerbereit*** genau dann, wenn t markierungsbedingt feuerbereit ist und entweder μ zeitbehaftet ist oder $\lambda_t(\mu) = \infty$ gilt. Das bedeutet, daß in zeitlosen Markierungen μ alle Transitionen t mit $\lambda_t(\mu) < \infty$ blockiert sind (gleich ob sie markierungsbedingt feuerbereit sind oder nicht)[1].

7. $w_t : \mathfrak{S} \to \mathbb{R}_+$ $(\forall t \in T)$ die (markierungsabhängigen) ***Schalt-***oder ***Feuerwahrscheinlichkeiten*** der Transitionen t in Markierung μ sobald $\lambda_t(\mu) = \infty$. Die effektive Wahrscheinlichkeit, daß die Transition t feuert wenn sie in Markierung μ feuerbereit ist, ergibt sich damit zu:

$$w_t(\mu)\left(\sum_{i \in I} w_{t_i}(\mu)\right)^{-1} \quad (I \subseteq \{1,\ldots,|T|\} \text{ mit } t_i \text{ feuerbereit in } \mu)$$

8. $M = \{(\boldsymbol{\rho}_i, \boldsymbol{r}_i\}$ $(i = 1,\ldots,|M|)$ die (endliche) Menge der ***Bewertungsmaße***. Dabei sind $\boldsymbol{\rho} : \mathfrak{S} \to \mathbb{R}$ ***Zustandsmaße***, d.h. für $\mu \in \mathfrak{S}$ ist $\boldsymbol{\rho}(\mu)$ die Rate, mit der sich das akkumulierte Bewertungsmaß des Netzes in Markierung μ ändert. $\boldsymbol{r}_t : \mathfrak{S} \to \mathbb{R}$ sind (markierungsabhängige) ***Impulsmaße*** für Transitionen $t \in T$, die die Änderung des akkumulierten Bewertungsmaßes des Netzes beim Feuern von Transition t in Markierung μ angeben[2].

   Mit den Zustandsmaßen können alle Elemente von $\mathfrak{S}$ (sowie mit den Impulsmaßen alle Elemente von $\mathfrak{E}$) und damit der komplette Zustandsraum stochastischer Petrinetze bewertet werden.

2.1.2 Markierungs- und Bewertungsprozess

1. bis 5. ist die klassische Definition von Petrinetzen; 6. und 7. erweitern diese Definition zu den von Molloy [Mol81] eingeführten ***verallgemeinerten stochastischen Petrinetzen*** (*Generalized Stochastic Petri Nets, GSPN*, s. [AMBC84]).

[1] Diese Einschränkung ist notwendig um zu gewährleisten, daß der Erreichbarkeitsgraph des SRN dem des eingebetteten zeitlosen Petrinetzes äquivalent ist (und in der Folge alle Resultate über qualitative Eigenschaften von zeitlosen Petrinetzen auf SRN übertragen werden können).

[2] Die so (d.h. markierungsabhängig) definierten Impulsmaße bewerten die Elemente von $\mathfrak{E}$ – sind die Impulsmaße markierungsunabhängig, können sie auch einfacher allein auf T definiert werden.

Durch GSPN wird ein diskreter Semi-Markov-Prozess $\{(t^{[n]}, \theta^{[n]}, \mu^{[n]}),\ n \in \mathbb{N}\}$ definiert, der sog. ***Markierungsprozess*** von **A**: Für $n > 0$ ist $t^{[n]} \in T$ die n-te Transition die zum Zeitpunkt $\theta^{[n]}$ feuert, was den Prozess in die Markierung $\mu^{[n]}$ überführt (zum Zeitpunkt 0 soll sich der Prozess definitionsgemäß mit Wahrscheinlichkeit 1 in Markierung μ_0 befinden). Die Verweilzeiten in den Markierungen sind dabei entweder exponentiell verteilt (für zeitbehaftete Markierungen) oder Null (bei zeitlosen Markierungen).

Mit der Einführung von M in 8. wird es möglich, für jedes Bewertungsmaß $(\boldsymbol{\rho}, \boldsymbol{r}) \in M$ auf dem Markierungsprozess einen ***akkumulierten Bewertungsprozess***

$$Y^{[n]} \;=\; \int_0^{\theta^{[n]}} \boldsymbol{\rho}(\mu(u))\ \mathrm{d}u \;+\; \sum_{i=1}^{n} \boldsymbol{r}_{t^{[i]}}(\mu^{[i-1]})$$

zu definieren; $Y^{[n]}$ gibt dann für $n \in \mathbb{N}$ das bis zum Zeitpunkt $\theta^{[n]}$ akkumulierte Bewertungsmaß des Markierungsprozesses an (dabei bezieht sich das Integral auf die bis zum jeweiligen Zeitpunkt aufgelaufenen Zustandsmaße, während der zweite Summand die Impulsmaße akkumuliert).

2.1.3 Darstellung von Petrinetzen

Ein Vorteil von Petrinetzen bei der Modellierung ist ihre standardisierte Notation zur Visualisierung der Netzpläne; der Aufbau und das Verständnis von Petrinetz-Modellen kann dem Modellierer durch den Einsatz der graphischen Notation erheblich erleichtert werden, falls entsprechende Werkzeug-Unterstützung (d.h. graphische Editoren für die Eingabe und Darstellung der Netze wie z.B. in PANDA) vorhanden ist. Die Visualisierung entstammt dabei einer Sichtweise auf Petrinetze, die mit dem Feuervorgang von Transitionen den Transport von Token zwischen Stellen assoziiert.

In Abb. 2.1 ist die auch in dieser Arbeit verwendete Notation beispielhaft dargestellt: Kreise symbolisieren die Stellen, ausgefüllte bzw. nicht ausgefüllte Rechtecke die zeitlosen bzw. exponentiell verzögerten Transitionen (die hier ebenfalls gezeigte Rekonfigurationstransition wird in Kap. 3.3 eingeführt). Normale Kanten werden durch Pfeile und inhibitorische Kanten durch Linien mit ausgefüllten Kreisen an der Spitze dargestellt. Stellen und Transitionen können mit Namen belegt werden; in den Stellen kann zusätzlich die Belegung durch Token in der Anfangsmarkierung sowie an den Transitionen die Rate oder Schaltwahrscheinlichkeit notiert werden.

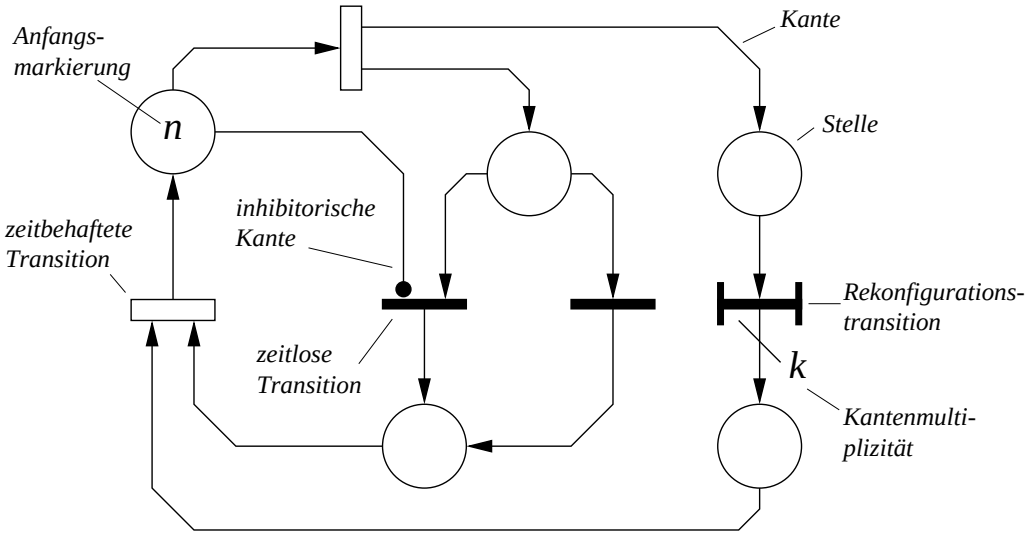


Abbildung 2.1: Beispiel für die graphische Notation von Petrinetzen

2.1.4 Auswertung von SRN

Die bei »klassischen« (d.h. zeitlosen bzw. keine stochastischen Parameter aufweisenden) Petrinetzen übliche Untersuchung der »qualitativen« Eigenschaften wie Beschränktheit, Lebendigkeit usw. tritt bei der Auswertung von SRN in den Hintergrund (bzw. wird meist nur zur Validierung von Modell-Eigenschaften durchgeführt) – hier interessiert die »quantitative« Analyse, d.h. die Berechnung der stochastischen Momente von Bewertungsmaßen. Diese Größen können sowohl stationär (d.h. für den Gleichgewichts-Zustand des durch das SRN beschriebenen stochastischen Prozesses nach unendlich langer Zeit) als auch transient (d.h. in ihrer Entwicklung innerhalb eines endlichen zeitlichen Intervalls) ermittelt werden; alle in der Praxis interessierenden Ergebnisse lassen sich dabei aus der Kenntnis des akkumulierten Bewertungsprozesses ableiten.

Die Bestimmung von $Y^{[n]}$ aus 2.1.2 geschieht normalerweise ausschließlich rechnergestützt und besteht im Wesentlichen aus drei Schritten (da die numerische Analyse[3] von SRN nicht Gegenstand dieser Arbeit ist, wird sie hier nur skiz-

[3] Neben der numerischen Analyse können SRN auch simulativ mit Monte-Carlo-Verfahren ausgewertet werden. Dies bietet potentiell verschiedene Vorteile wie sehr gute Speicherplatz-Effizienz sowie die Möglichkeit, auch Transitionen mit beliebig (nicht-exponentiell) verteilten Schaltverzögerungs-Zeiten zuzulassen. Die Ergebnisse sind jedoch statistischer Natur, und in der Praxis müssen zur Erzielung von Resultaten mit ausreichend kleinen Konfidenz-Intervallen oft unzumutbar lange Simulationszeiten in Kauf genommen werden (besondere Probleme macht dies, falls in den Modellen »seltene« Ereignisse auftreten können, d.h. um Größenordnungen differierende Schaltraten oder -wahrscheinlichkeiten vorkommen). Für PANDA wurde eine Komponente zur simulativen Auswertung von GSPN realisiert [HK95], aber u.a. aus den obengenannten Gründen nicht weiterentwickelt.

ziert – für detaillierte Beschreibungen der Berechnungsschritte sei nochmals auf [CBC+93] verwiesen):

Erreichbarkeitsgraph-Generierung: Zunächst wird aus der SRN-Spezifikation der Erreichbarkeitsgraph des Modells generiert (dies ist natürlich nur für SRN mit endlichem Erreichbarkeitsgraphen möglich). Dazu werden unter Anwendung der in 2.1.1 angegebenen Regeln zur Feuerbereitschaft zu jedem Knoten die direkten Nachfolger untersucht und mit ihren Übergängen in den Graphen integriert (falls sie nicht schon vorher besucht und gespeichert wurden). Wichtig für die Implementierung ist hier der Aufbau von Datenstrukturen, die Speicherplatz gut ausnützen sowie ein sehr effizientes Durchsuchen des schon aufgebauten Teil-Graphen ermöglichen.

Nach dem Aufbau des Graphen wird er durch Übertragung der stochastischen Parameter des SRN modifiziert: Abhängig von der Art des Anfangszustandes der Kanten werden an ihnen (entsprechend den Markierungsübergängen des stochastischen Prozesses) die Übergangsraten (bei zeitbehafteten Markierungen) oder Schaltwahrscheinlichkeiten (bei zeitlosen Markierungen) vermerkt.

Markov-Analyse: Der modifizierte Erreichbarkeitsgraph kann jetzt als zeitkontinuierliche Semi-Markov-Kette interpretiert und (stationär oder transient) analysiert werden, d.h. es werden die Erwartungswerte für die Verweilzeiten in Markierungen sowie für die Anzahl und Frequenz von Markierungsübergängen berechnet. Dies läuft im stationären Fall direkt auf die Lösung von großen, aber dünn besetzten linearen Gleichungssystemen hinaus; für die transiente Analyse wird eine als Uniformization bekannte Technik zur Reihen-Entwicklung über die Zeitachse verwendet, die ebenfalls in linearen Gleichungssystemen (i.e. eines für jeden Zeitpunkt zu dem analysiert werden soll) resultiert. Als sehr rechenzeiteffiziente Methode zur Behandlung der auftretenden Gleichungssysteme haben sich hier Mehrebenen-Verfahren erwiesen, wie sie auch zur Lösung partieller Differential-Gleichungssysteme angewendet werden (ein solcher Algorithmus [Hor95] ist in PANDA implementiert).

Bewertungsberechnung: Die Erwartungswerte für die Bewertungsmaße können nun aus den Ergebnissen der Markov-Analyse einfach durch Linear-Kombination gewonnen werden (für Zustandsmaße aus den Erwartungswerten für Verweilzeiten bzw. für Impulsmaße aus den Erwartungswerten der Zahl der entsprechenden Zustandsübergänge) und müssen nur noch den entsprechenden SRN-Elementen zugeordnet werden.

Neben der stationären oder transitiven Bestimmung der stochastischen Momente von Bewertungsmaßen kann mit der Definition des Bewertungsprozesses

auch noch die (parametrische) Sensitivitätsanalyse eingeführt werden: Betrachtet wird dabei die Änderung eines Bewertungsmaßes in Abhängigkeit von der Änderung eines (reellwertigen) Parameters des Bewertungsmaßes oder des SRN (Schaltrate oder Feuerwahrscheinlichkeit einer Transition). Dazu muß die Ableitung des Bewertungsmaßes nach dem abhängigen Parameter berechnet werden, was die Lösung von Systemen gewöhnlicher Differential-Gleichungen bedeutet. Üblicherweise wird auch hier die Technik der Uniformization angewendet; Verfahren zur Sensitivitätsanalyse sind z.B. in SPNP [CMT89] implementiert.

Die in PANDA realisierten Algorithmen zur SRN-Analyse sind für Parallelrechner (mit gemeinsamem oder verteiltem Hauptspeicher) implementiert; das Werkzeug eignet sich daher auch für die effiziente Analyse sehr großer Modelle [AKH97, ACDK97, ADK97, AK99].

2.2 Umsetzung bei der Modellierung

Mit dem in 2.1.1 beschriebenen stochastischen Prozess kann jede Kombination von Zuständen und Zustandsübergängen im Erreichbarkeitsgraphen des Netzes mit Bewertungsmaßen belegt werden. Auch die »klassischen« Ergebnisse der GSPN-Analyse wie durchschnittliche Zahl von Token in Stellen oder Durchsatz von Transitionen lassen sich leicht mit Hilfe von $Y^{[n]}$ formulieren.

Bei der Definition der Bewertungsmaße wurde jedoch direkt Bezug auf Elemente von $\mathfrak{S}$ genommen – dies bedeutet, daß der Modellierer die Maße in Form von Markierungen und Übergängen im Erreichbarkeitsgraphen des Netzes angeben müßte. Wünschenswert wäre es jedoch, die Bewertungsmaße direkt bezogen auf Elemente des SRN-Modells ausdrücken zu können; auch sollte die Formulierung der stochastischen Ergebnisgrößen, die als Analyse-Resultat gewünscht werden, bei der SRN-Modellierung flexibler möglich sein als bei der (auf Erwartungswerte der Standard-Resultate beschränkten) Auswertung der klassischen GSPN-Analyse.

2.2.1 Entwurfsziele für eine SRN-Spezifikationssprache

Die auf S.24 angegebene Definition von Bewertungsmaßen muß also für die Anwendung bei der Petrinetz-Modellierung in eine Beschreibungssprache umgesetzt werden, in der der Modellierer die SRN spezifiziert, und die von einem Werkzeug interpretiert und ausgewertet werden kann. Bei der konkreten Implementierung einer solchen Beschreibungssprache für den Einsatz in PANDA waren mehrere Entwurfsziele zu beachten:

Einfache Anwendbarkeit: Vom Modellierer müssen ohne großen Aufwand die für die Bewertung relevanten Elemente des Zustandsraums selektiert werden können, und es müssen daraus flexibel Ausdrücke aufgebaut werden können, die die direkte, vollständige Berechnung der für das jeweilige Modell interessierenden Ergebnisse durch das Werkzeug gestatten (d.h. es sollte i.A. keine numerische Nachbearbeitung der vom Werkzeug gelieferten Resultate mehr nötig sein).

In der Konsequenz werden neben einfach anzuwendenden Funktionen für den Zugriff auf Elemente des Zustandsraums der SRN sowie zur Angabe der gewünschten stochastischen Ergebnisse auch arithmetische Operatoren benötigt, aus denen die Bewertungsmaße konstruiert werden können. Dabei sollten sowohl für Bewertungsmaße als auch für eine der eigentlichen stochastischen Auswertung folgende Weiterverarbeitung der Analyse-Resultate komplexere Ausdrücke aus den zustandsraumselektierenden und stochastischen Operatoren gebildet werden können.

Um die Erlernung des Umgangs mit PANDA zu erleichtern, sollte zudem der Teil der Beschreibungssprache, der zur Auswahl von Zuständen aus der Erreichbarkeitsmenge $\mathfrak{S}$ dient, auch an anderen Stellen der Petrinetz-Spezifikation einsetzbar sein – bei der Angabe von Inhibitorfunktionen sowie zustandsabhängigen Kantenmultiplizitäten und Schaltraten muß auf die gleichen Daten aus $\mathfrak{S}$ zugegriffen werden wie bei der Formulierung von Bewertungsmaßen. Ebenfalls aus Gründen der Ergonomie sollte die Spezifizierung solcher Funktionen in die graphische Benutzungs-Schnittstelle des Werkzeugs integriert sein, um den gesamten Vorgang der Modell-Erstellung in einer konsistenten Umgebung zu erlauben.

Verständlichkeit: Die Bewertungsmaße sollten verständlich und gut strukturiert spezifizierbar sein – einmal definierte Ausdrücke sollten wiederverwendbar sein, um die Beschreibung der Maße kompakt zu halten; komplexere Ausdrücke sollten in einer Form dargestellt werden können, die die Mittel einer strukturierten Programmiersprache (Funktionsblöcke, Variablen etc.) bietet.

Das bedeutet, daß Klammerung und Gruppierung von Ausdrücken sowie programmiersprachliche Konstrukte (z.B. für bedingte Verzweigungen beim Aufbau der Bewertungsmaße, Gestaltung der Bewertungsmaßdefinitionen als verschachtelbar aufrufbare Funktionsblöcke) benötigt werden.

Auch sollte für die Spezifikation der GSPN-Standardresultate kein zusätzlicher Aufwand nötig sein, d.h. für gewisse kanonische Bewertungsmaße sowie deren stochastische Evaluierung sollten komplett vordefinierte Spezifikationen im Werkzeug vorhanden sein, die der Modellierer nur noch anwählen muß. Der Parser, der die Beschreibungssprache implementiert sollte einfach erweiterbar sein, um z.B. zusätzliche vordefinierte Spezifikationen bei Bedarf leicht integrieren zu können.

Überprüfbarkeit: Die Spezifikationen der Bewertungsmaße und die Anwendung der stochastischen Operatoren auf sie sollten auf syntaktische Korrektheit überprüft werden können, bevor die eigentliche Analyse durch das Werkzeug beginnt, um die Umlaufzeit bei der Modellierung gering zu halten – eine Fehlermeldung wegen eines Syntaxfehlers in der Angabe eines Bewertungsmaßes muß sofort nach der Eingabe erfolgen, und nicht erst beim Aufbau des Erreichbarkeitsgraphen als Beginn der Analyse der SRN.

Weiter sollten die zur Auswertung der Spezifikationen nötigen Funktionen komplett in PANDA integriert sein, um eine einfache Installation des Werkzeugs (ohne Abhängigkeiten von anderen Programmen wie Entwicklungsumgebungen) zu gewährleisten.

In der Folge wurde das bisher in PANDA angewendete wie auch bei anderen Werkzeugen zur Modellierung mit SRN (wie z.B. SPNP, vgl. [CMT89]) übliche und von der Auswertungsgeschwindigkeit her optimale Verfahren als ungeeignet betrachtet: Hier erfolgt die Angabe der Bewertungsmaße in einer Programmiersprache (normalerweise C), mit anschließender Übersetzung und Montage durch eine Enwicklungsumgebung (Compiler, Binder etc.) sowie der Ausführung der Auswertung der SRN direkt in Maschinencode. Eine Software-ergonomisch wünschenswerte Rückmeldung von Fehlern sowie eine zumindest teilweise Überprüfung auf semantische Fehler (z.B. der Bezug auf Elemente von $\mathfrak{E}$ in Zustandsmaßen) sofort bei der Spezifizierung ist auf diese Weise jedoch nicht realisierbar, und daneben ist für die Einsatzfähigkeit des Werkzeugs auch immer eine installierte Entwicklungsumgebung Voraussetzung.

Andererseits darf durch die Auswertung der Bewertungsmaße die Berechnungszeit für die Analyse nicht übermäßig nach oben getrieben werden; die direkte Implementierung der Spezifikation und Auswertung in einer interpretierten Programmiersprache (wie sie z.B. für qualitative Analyse bzw. Simulation von farbigen Petrinetzen in DesignCPN [Met93] unter Verwendung von ML vorgenommen wurde) ist für die Generierung und Analyse großer Zustandsräume bei der quantitativen Petrinetz-Modellierung zu langsam.

2.2.2 Integration in PANDA

Um die oben beschriebenen Ziele zu erreichen, wurde aufbauend auf der SPNL [Ger97] bei der Implementierung in PANDA für die Spezifikation von SRN mit [Des98] ein mehrstufiges Verfahren realisiert:

1. Zuerst werden die Netztopologie (über eine graphische Benutzungs-Schnittstelle) und danach die GSPN-Parameter wie Raten und Feuerwahrscheinlichkeiten von Transitionen, Inhibitorfunktionen und Kantenmultiplizitäten

angegeben; dabei werden ggf. markierungsabhängige Parameter mittels der unten für charakterisierende Funktionen angegebenen Konstrukte und Syntax spezifiziert.

2. Mit Hilfe von ***charakterisierenden Funktionen***, die Zustände und Übergänge im Erreichbarkeitsgraphen des Netzes indizieren, können die den Modellierer interessierenden Elemente des Zustandsraums ausgewählt werden. PANDA stellt dafür folgende Funktionen zur Verfügung:

   - Für Zustandsmaße gibt $mark : \mathfrak{S} \times P \to \mathbb{N}$ die Zahl von Token in einer Stelle und $enabled : \mathfrak{S} \times T \to \{0,1\}$ die Feuerbereitschaft einer Transition an (1 falls feuerbereit, andernfalls 0), jeweils bei gegebener Markierung.
   - Für Impulsmaße indiziert $fire : \mathfrak{S} \times T \to \{0,1\}$ das Schalten einer Transition (die Funktion evaluiert zu 1 falls eine Transition einen Zustandsübergang auslöst, andernfalls zu 0). Auf die effektive Schaltrate einer (zeitbehafteten) Transition kann mit $rate : \mathfrak{S} \times T \to \mathbb{R}^+$ zugegriffen werden, $probfire : \mathfrak{S} \times T \to [0,1]$ liefert die (effektive) Wahrscheinlichkeit, daß eine Transition in einer gegebenen Markierung feuert.

   Bei der Anwendung der charakterisierenden Funktionen werden vom Modellierer nur die Namen der relevanten Petrinetz-Stellen bzw. -Transitionen verwendet – der Zugriff auf die entsprechenden Elemente des Erreichbarkeitsgraphen des Netzes wird vom Werkzeug durchgeführt. Anders ausgedrückt werden durch den Modellierer also nur »potentielle« (evtl. leere) Teilmengen von $\mathfrak{S}$ formuliert und für die Bewertung vorgesehen; vom Werkzeug werden dann dafür (erst nach der Generierung von $\mathfrak{S}$) die *tatsächlich* existierenden Elemente des Zustandsraums substituiert. Damit wird die Spezifikation der Bewertungsmaße in der Terminologie des Petrinetz-Modells und ohne genaue Kenntnis des Zustandsraums möglich.

   Mit den charakterisierenden Funktionen als Bausteine können die auf S. 24 definierten Bewertungsmaße spezifiziert werden: Charakterisierende Funktionen können mit arithmetischen Operatoren sowie bedingten Verzweigungen verknüpft und den so aufgebauten Termen Bezeichner zugewiesen werden. Diese im weiteren als ***Bewertungsfunktionen*** bezeichneten Ausdrücke können auch ineinander verschachtelt aufgerufen werden, um die Spezifikationen verständlich und kompakt zu halten.

3. Durch die Anwendung von stochastischen Operatoren auf Bewertungsfunktionen wird angegeben, welche Größen aus den Ergebnissen der Markov-Analyse auf das Petrinetz zurückgerechnet werden sollen; als Operator ist

in der derzeitigen Version von PANDA nur der Erwartungswert implementiert. Den Ergebnisse der stochastischen Auswertung können ebenfalls Bezeichner zugewiesen und die so entstandenen ***Ergebnisfunktionen*** zusätzlich noch mit arithmetischen Verknüpfungen weiterverarbeitet werden, um auch komplexere stochastische Resultate direkt vom Werkzeug berechnen zu lassen.

Für Standard-Ergebnisse der GSPN-Analyse (durchschnittliche Zahl von Token bzw. Wahrscheinlichkeit für mindestens ein Token pro Stelle sowie Durchsatz von Transitionen) sind vordefinierte Ergebnisfunktionen vorhanden, so daß beim Modellaufbau zur Spezifizierung dieser Größen keine zusätzliche Arbeit für den Modellierer anfällt.

Die Beschreibungssprache wurde als kontextfreie LALR(1)-Grammatik formuliert, so daß der benötigte Parser aus dieser mit Hilfe von Generatoren [MB90] automatisch erstellt werden konnte. Der so erzeugte Parser erwies sich jedoch als zu langsam, um komplexere Bewertungs- und Resultatfunktionen unmittelbar damit auswerten zu lassen – gegenüber direkt in Maschinencode übersetzten Funktionen ergab sich ein ca. zehnfacher Zeitbedarf. Daher wurde dem Parser ein Interpreter nachgeschaltet: Nach der Analyse auf syntaktische Korrektheit werden die Spezifikationen von Bewertungs- und Ergebnisfunktionen vom Parser in einen Syntaxbaum transformiert, der dann abgespeichert und nach erfolgter Markov-Analyse vom Interpreter ausgewertet wird – der Zeitbedarf der interpretierten SRN-Auswertung liegt nur um ca. 30% über dem der in Maschinencode ausgeführten [Des98].

Der Spezifikations- und Analyse-Prozess von SRN in PANDA ist in Abb. 2.2 zusammengefaßt; eine Grammatik für die Beschreibung von Multiplizitäten, Inhibitor-, Bewertungs- und Ergebnisfunktionen sowie Beispiele für die Anwendung der Beschreibungssprache sind in App. A angegeben.

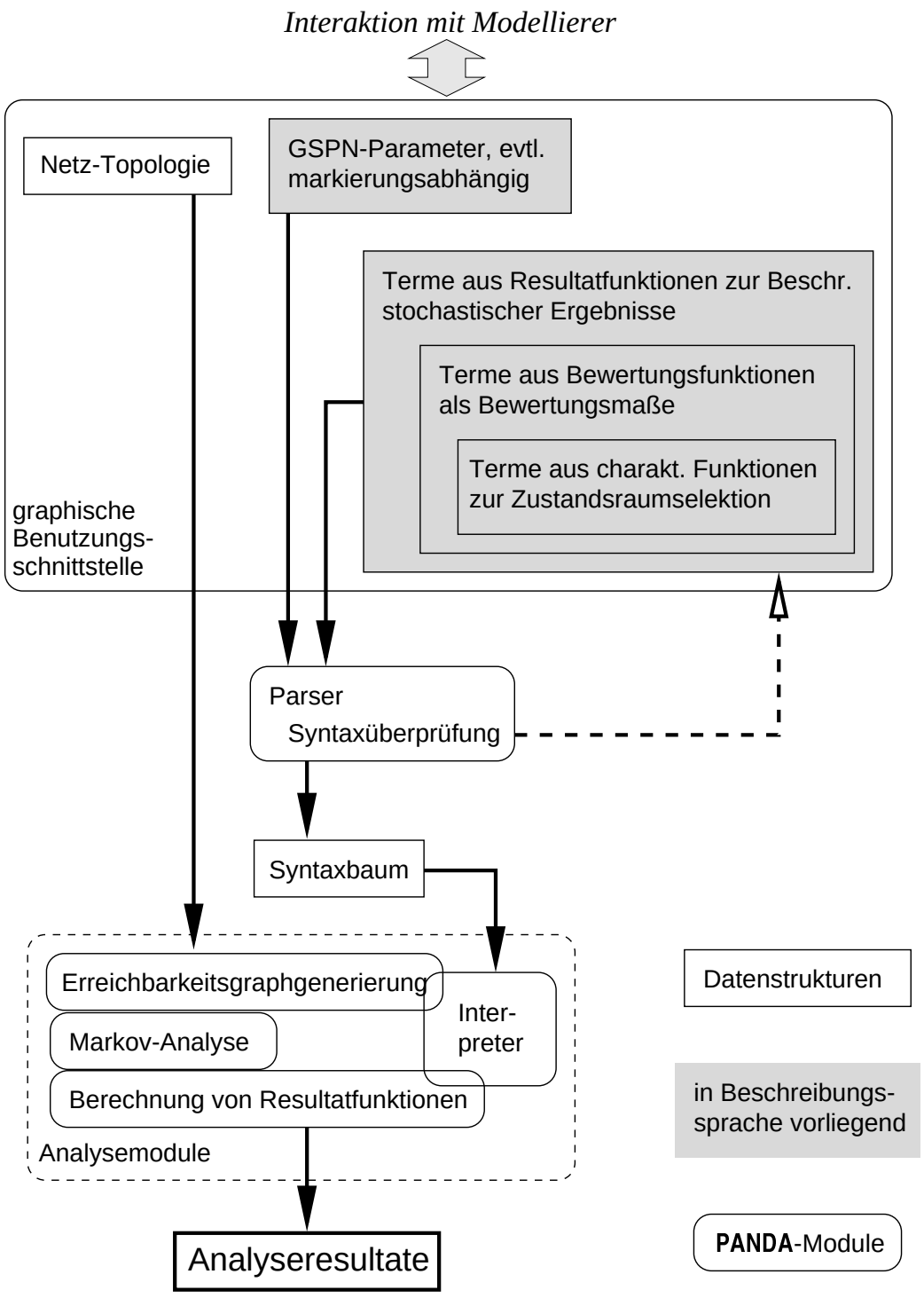


Abbildung 2.2: Übersicht der SRN-Spezifikation und -analyse in PANDA

Kapitel 3

Stochastische Petrinetze mit dynamischer Rekonfiguration

In diesem Kapitel wird eine für das DSS wesentliche Erweiterung des Modellierungs-Paradigmas der SRN eingeführt: Stochastische, bewertete Petrinetze zur Entscheidungsmodellierung, mit denen auch das kontrollierte Umschalten zwischen verschiedenen möglichen Varianten eines Netzes unter Optimierungsmaßgaben modelliert werden kann.

Das neue Modellierungsverfahren wird hier schrittweise aus den bekannten Markov-Ketten abgeleitet und an einem Beispiel-Modell dem schon existierenden Verfahren zur Entscheidungsmodellierung auf Markov-Ebene gegenübergestellt.

3.1 Entscheidungsmodelle

Die im letzten Kapitel beschriebene SRN-Modellierung liefert als Analyseergebnis Antworten auf jeweils einen Parameter-Eingabesatz; variiert werden bei der Auswertung eines SRN i.A. nur die GSPN-Parameter (Raten, Schaltwahrscheinlichkeiten etc.) und die Bewertungsmaße – die Topologie der Netze im Verlauf der Experimente von Hand zu ändern ist mühsam und würde in einem enorm großen Parameter-Raum resultieren, der dann mit einzelnen Auswertungen punktweise abgetastet werden müßte. Die Auswertung der SRN erfolgt also meist bei fester Topologie der Netze und variiert nur die numerischen Parameter.

Für das DSS benötigt werden jedoch Modellierungsmethoden mit entsprechenden Analyseverfahren, die alternative Struktur-Varianten in den Modellen nachbilden sowie Entscheidungen über diese Strukturänderungen zusätzlich optimieren können (und damit in der Lage sind, zur Entscheidungsfindung beizutragen). Die Fragestellung bei der für das DSS durchgeführten Modellierung besteht letztlich darin, *ob* (oder wann) bestimmte Aktionen durchgeführt werden sollen – und nicht darin, die Einstellung von System-Parametern zu optimieren. Die Wahl

von in gewissem Sinn guten oder sogar optimalen Entscheidungen für die gegebene modellierte Situation sollte unterstützt und falls möglich automatisiert getroffen werden.

Für die DSS-Modellierung müssen also andere Paradigmen eingesetzt werden als für die quantitative Analyse von Systemen, deren Verhalten rein stochastisch beschrieben werden kann: Bei der für das DSS angestrebten Entscheidungsmodellierung soll ein dynamischer Prozess optimiert werden, im Sinne des kontrollierten Treffens von Entscheidungen innerhalb eines dynamischen, zufälligen äußeren Einflüssen und Anforderungen unterliegenden Systems. Dagegen sollen bei der üblichen quantitativen Modellierung zumeist optimale Werte für Parameter gefunden werden, die einmal eingestellt und dann nicht mehr verändert werden.

3.2 Markov-Entscheidungsmodelle

Ein aus dem Operations Research bekanntes Verfahren zur optimierenden Entscheidungsmodellierung sind Markov-Entscheidungsmodelle [Tij86]. Diese entstehen, indem in bewertete Markov-Prozesse zusätzlich zu den rein stochastisch bestimmten Zustandsübergängen die Möglichkeit zur Modellierung von weiteren, kontrolliert gesteuerten Übergängen integriert wird. Mit speziellen iterativen Analyseverfahren können dann im Mittel (und in Bezug auf die Bewertungsstruktur) günstige Folgen von Entscheidungen berechnet werden.

An dieser Stelle werden nur die wichtigsten Definitionen und Ergebnisse der Theorie zur Markov-Entscheidungsmodellierung zusammengefaßt; für eine detaillierte Darstellung und Beweise sei auf [Tij86] und [dM92] verwiesen.

3.2.1 Definitionen

Markov-Modelle: Ausgegangen wird von einem zeitkontinuierlichen[1] und homogenen Markov-Prozess bzw. seinem ***Zustandsprozess*** $\{(\omega^{[t]}),\, t \in \mathbb{R}^+\}$:

1. $\Omega = \{\omega_1, \ldots, \omega_N\}$ sei der Zustandsraum. Da dieser endlich und diskret ist, wird er auch als ***Markov-Kette*** bezeichnet.

2. Die Zustandsübergangs-Wahrscheinlichkeiten seien ausschließlich vom aktuellen Zustand ω abhängig, d.h. nicht von der bisherigen »Geschichte« des Prozesses – also den vorher aufgetretenen Zuständen (diese Unabhängigkeit wird als ***Markov-Eigenschaft*** bezeichnet) – und auch nicht vom Zeitpunkt zu dem der Übergang stattfindet (Homogenität).

[1] Für die Analyse werden die zeitkontinuierlichen Markov-Ketten i.d.R. auf isomorphe zeitdiskrete Ketten abgebildet; solche lassen sich stets als sog. eingebettete Markov-Ketten finden [AMBC86].

3. $q_{i,j} \in \mathbb{R}$ seien die Raten für die Zustandsübergänge von ω_i nach ω_j, wobei sich die Wahrscheinlichkeiten $p_{i,j}$ für die Übergänge zu $q_{i,j} / \sum_{k=1}^{N} q_k$ berechnen.

Obige Annahmen führen zu exponentiell verteilten Aufenthalts-Zeiten in den Zuständen, d.h. die Verteilungsfunktion für die Aufenthalts-Zeiten kann für alle ω_i mit $F_i(t) = 1 - e^{-\sum_{j \in J} q_{i,j} t}$ angegeben werden (wenn mit $J \subseteq \{1, \ldots, N\}$ die Indices der möglichen Nachfolge-Zustände von ω_i bezeichnet ist).

Durch die Zuweisung von Bewertungen an die Markov-Zustände entstehen die ***bewerteten Markov-Prozesse*** (*Markov reward process*) – hier wird jeder Zustand $\omega \in \Omega$ zusätzlich auf ein Zustands-Bewertungsmaß $\boldsymbol{\rho}_\omega \in \mathbb{R}$ abgebildet. Dieses beschreibt als Bewertungsrate (*rate-reward*) den Gewinn oder die Kosten, die proportional zur Zeitdauer anfallen, die sich der Prozess im Zustand ω befindet.

Damit ist auch die Bewertungsrate $\boldsymbol{\rho}_{\omega^{[t]}}$ (d.h. die instantane Bewertungsfunktion des Prozesses $\{(\omega^{[t]}\})$ festgelegt und es kann (analog zu den SRN, vgl. Definition auf S. 25) aus dem Zustandsprozess ein akkumulierter Bewertungsprozess $\{Y^{[t]},\ t \in \mathbb{R}^+\}$ mittels Integration von $\boldsymbol{\rho}_{\omega^{[t]}}$ definiert werden, mit $Y^{[t]} = \int_0^t \boldsymbol{\rho}_{\omega^{[\tau]}}\, \mathrm{d}\tau$.

Markov-Entscheidungsmodelle: Beim Markov-Modell finden die Übergänge zwischen den Zuständen nach rein stochastischen Gesetzmäßigkeiten statt. Dabei gibt das Modell für einen aktuellen Zustand eine Menge aus mehreren möglichen Folge-Zuständen vor – in welchen von diesen das System dann übergeht, ist aber vollständig vom Zufall bestimmt. Für den Ausbau zum Entscheidungsmodell werden jetzt zusätzlich Entscheidungs- oder Auswahlmöglichkeiten zugelassen, die den Folgezustand (und damit auch die Änderung des Bewertungsmaßes) beeinflussen können, in den der Prozess übergeht.

Dazu wird jedem Zustand $\omega_i \in \Omega$ eine (evtl. leere) Menge von ***Entscheidungsalternativen*** $A_i \subseteq \Omega$ zugeordnet mit der Möglichkeit, das System – sofern es sich im aktuellen Zustand ω_i befindet – entweder im Zustand ω_i zu belassen (bis es den nächsten zufällig bestimmten Übergang durchläuft) oder aber es *kontrolliert* (d.h. eben nicht zufällig bestimmt) in einen der Folge-Zustände $a(\omega_i) \in A_i$ aus der Menge der Entscheidungsalternativen zu überführen.

Eine konkrete Auswahl aus den Alternativen (d.h. die Zuordnung eines konkreten Folge-Zustands) modelliert so eine ***Entscheidungsaktion***: Wird im Zustand ω_i die Aktion $a(\omega_i)$ durchgeführt, so ändert sich das Bewertungsmaß des Modells um $\boldsymbol{\rho}_i(a(\omega_i))$ und der Prozess geht mit Wahrscheinlichkeit $p_{ij}(a(\omega_i))$ in den Folgezustand ω_j über (dabei müssen sich alle Übergangs-Wahrscheinlichkeiten aller möglichen Aktionen zu 1 summieren, d.h. $\sum_{j \in J} p_{ij}(a(\omega_i)) = 1$).

Die Durchführung einer Entscheidungsaktion wird auch (bedingt durch den Einsatz der Markov-Entscheidungsmodelle bei Entwurf und Bewertung fehlertoleranter Rechenanlagen) ***Rekonfiguration***, und die den möglichen Aktionen ent-

sprechenden Transitionen werden (bezogen auf die Interpretation der Modelle als Zustands-Übergangs-Graphen) ***Rekonfigurationskanten*** genannt.

Eine Menge von Entscheidungsaktionen $\mathbf{S} = \{a_1, \ldots, a_K\} \subseteq \Omega$ wird im Zusammenhang der Markov-Entscheidungsmodelle auch als ***Strategie***[2] (*policy*) bezeichnet, wenn durch $\mathbf{S}$ für jede mögliche Entscheidung (d.h. für alle $\omega_i \in \Omega$ mit $A_i \neq \emptyset$) eine Aktion festgelegt wird. Zu unterscheiden ist dabei zwischen ***stationären*** und ***zeitabhängigen*** Strategien: Bei Ersteren hängen die Entscheidungen ausschließlich vom Zustand des Modells ab in dem die Entscheidung getroffen wird, wohingegen bei Letzteren zusätzlich noch der Zeitpunkt, zu dem die Entscheidungen getroffen werden, die jeweiligen Aktionen beeinflußt.

Die Markov-Entscheidungsmodelle können so auch als die kompakte Formulierung *mehrerer verschiedener* bewerteter Markov-Prozesse (die im Folgenden auch als ***Modellvarianten*** oder ***-instanzen*** bezeichnet werden) in *einem* Modell betrachtet werden: Während dabei jede stationäre Strategie genau einen spezifischen Prozess vollständig bestimmt (falls für alle Zustände jeweils eine Aktion festgelegt ist, resultiert ja eine normale bewertete Markov-Kette), wird durch transiente Strategien ein zusätzlich von der Zeit abhängiges Umschalten zwischen den möglichen Modellvarianten beschrieben.

Erweiterte Markov-Entscheidungsmodelle: In [dM92] wird schließlich durch die Einführung von ***Verzweigungs-Zuständen*** (*branching state*) noch zusätzlich die Möglichkeit geschaffen, auch Zustandsübergänge zu modellieren die keine Zeit benötigen (entsprechend den zeitlosen Transitionen bei Petrinetzen).

Der Zustandsraum der auf diese Weise definierten ***Extended Markov Reward Models*** (*EMRM*) setzt sich aus zwei disjunkten Mengen Ω_M (Markov-Zustände) und Ω_V (Verzweigungs-Zustände) zusammen: Dabei gelten für Zustände aus beiden Klassen die für Markov-Prozesse angegebenen Definitionen bezüglich Übergangsraten und -wahrscheinlichkeiten – während aber den Zuständen aus Ω_M wie oben beschrieben exponentiell verteilte Aufenthaltsdauern zugeordnet sind, sind die Verweilzeiten für Zustände aus Ω_V definitionsgemäß gleich Null. Außerdem sind von den Verzweigungs-Zuständen aus keine Rekonfigurationsübergänge erlaubt.

Die Definition der Bewertungsstruktur wird für die EMRM dahingehend erweitert, daß den Zuständen in Ω_V (genauso wie denen in Ω_M) Bewertungsmaße $\boldsymbol{r}_\omega \in \mathbb{R}$ zugewiesen werden können. Diese werden für die Verzweigungs-Zustände aber nicht als Rate der Änderung des Bewertungsmaßes, sondern als Impulsmaß interpretiert – sobald der Zustand eingenommen (und im selben Augenblick wie-

[2] Diese ist nicht zu verwechseln mit der in 5.1.3 beschriebenen *Diagnose-Strategie*: Letztere beschreibt den diagnostischen Ansatz bei der Fehlersuche und liefert so den Rahmen für die zu optimierenden Entscheidungsfolgen (entsprechend den hier definierten Strategien auf Entscheidungsmodell-Ebene).

der verlassen) wird, ändert sich der Bewertungszustand des EMRM um $\boldsymbol{r}_\omega$. Die Verzweigungs-Zustände können so auch in der Bewertungsstruktur der Modelle berücksichtigt werden; für die Definition des akkumulierten Bewertungsprozesses muß dabei (ebenfalls wieder analog zu den SRN) zusätzlich zur Integration der Zustandsmaße die Summe der aufgelaufenen Impulsmaße berücksichtigt werden.

Die oben definierten EMRM und ihre im weiteren beschriebenen Verfahren zur Analyse (d.h. Strategieoptimierung) bilden die mathematische Grundlage der DSS-Modellierung.

3.2.2 Darstellung von EMRM

Für den Aufbau konkreter Modelle ist die Darstellung von EMRM in Matrizen-Form wenig geeignet. Aus diesem Grund existiert (ähnlich wie für SRN) auch für EMRM eine graphische Notation, mit der die Topologie der Modelle beschrieben werden kann. Abb. 3.1 ist ein einfaches Beispiel für ein EMRM mit seinen Modellinstanzen: Markov-Zustände werden als Kreise und Verzweigungs-Zustände als ausgefüllte Kreise, Markov-Übergänge als Pfeile und Rekonfigurationskanten als Pfeile mit unterbrochenen Linien dargestellt:

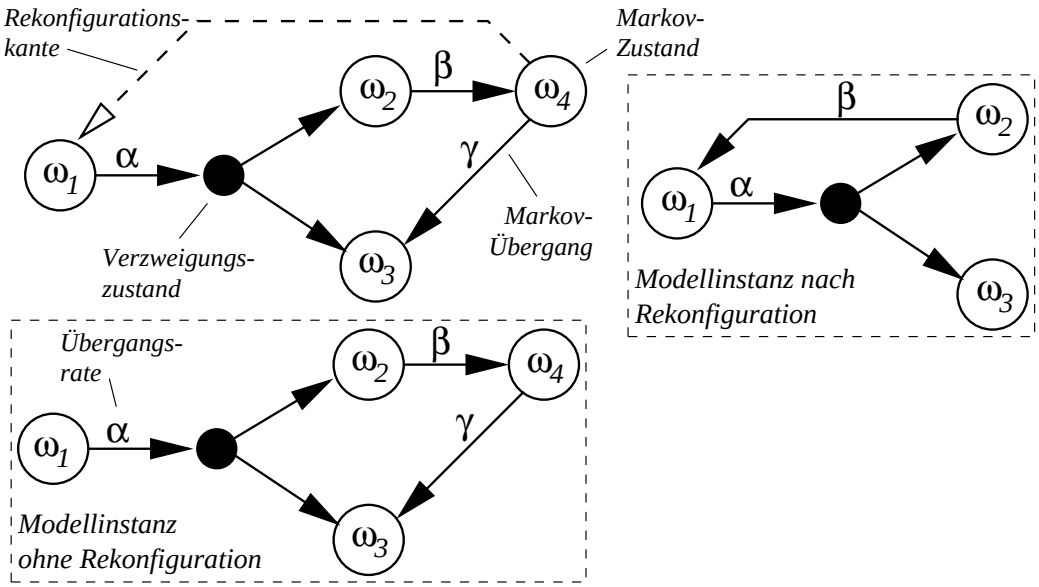


Abbildung 3.1: Beispiel für graphische Notation von EMRM mit Modellinstanzen: $A_4 = \{\omega_1\}$ und $A_i = \emptyset$ für $i \in \{1,2,3\}$ (s. S. 37)

3.2.3 Auswertung von EMRM

Für die in 3.2.1 eingeführten Entscheidungsmodelle hat eine reine Markov-Analyse im Sinne der Berechnung von Zustands-Wahrscheinlichkeiten und Momenten von Bewertungsmaßen – sei es transient (also im zeitlichen Verlauf) oder stationär (d.h. für den Gleichgewichts-Zustand des Prozesses nach unendlich langer Zeit) – natürlich wenig Sinn, denn die auszuwertenden Prozesse sind ja nur in Abhängigkeit von der gewählten Strategie vollständig definiert.

Aus diesem Grund tritt für die Auswertung von EMRM die Bestimmung von Strategien in den Vordergrund. Die Analyseziele (d.h. die zu berechnenden Größen) verschieben sich: Während es bei der Markov-Analyse darum geht, das Verhalten eines von zufälligen Ereignissen geprägten und als stochastischer Prozess modellierten Systems zu bestimmen, muß bei der EMRM-Analyse eine günstige Strategie für die Steuerung eines als Zustands-Übergangs-Modell beschriebenen Systems ermittelt werden, das von stochastisch geprägter Unsicherheit beeinflußt wird.

Optimalitätskriterien

Bezogen auf die geänderten Analyseziele für EMRM muß festgelegt werden, was eine »günstige« oder »optimale« Strategie ist. Naheliegenderweise wird dies unter Berücksichtigung der Bewertungsstruktur geschehen – allgemein wird hier eine Bewertungsgröße G abhängig von der Strategie $\mathbf{S}$ als Eingangs-Parameter betrachtet; eine Strategie $\dot{\mathbf{S}}$ heißt dann optimal, wenn $G(\dot{\mathbf{S}}) \geq G(\mathbf{S})$ gilt für alle Strategien $\mathbf{S}$ über Ω.

Für die Bestimmung von optimalen stationären Strategien wird dabei für G als Zielgröße das im (langfristigen, d.h. bei unendlich langer Beobachtung des zu analysierenden Systems) zu erwartende Bewertungsmaß pro Zeiteinheit betrachtet; berechnet wird also eine zeitunabhängige Strategie, die für den Gleichgewichts-Zustand des Prozesses (d.h. in der Terminologie der Entscheidungsmodellierung bezogen auf einen unendlichen Planungshorizont) in einem im Mittel optimierten Bewertungsmaß pro Zeiteinheit resultiert ([Tij86], S. 160).

In [dM92] werden für die EMRM zusätzlich Algorithmen entwickelt, die auch die Berechnung von im Mittel optimalen transienten Strategien (bezogen auf einen endlichen Planungshorizont und mit zeitabhängigen Entscheidungen) erlauben. Hier wird als zu optimierende Funktion G das über ein (endliches) Zeitintervall $[0,\ldots,T] \subseteq \mathbb{R}^+$ (auch als ***Planungshorizont*** oder ***Missionszeit*** bezeichnet, in dem das modellierte System betrachtet werden soll) aufgelaufene, akkumulierte Bewertungsmaß eingesetzt. Bestimmt wird eine Folge $\mathbf{S}(t)$ von Aktionen sowie zusätzlich die Zeitpunkte, an denen diese Aktionen ausgeführt werden müssen (d.h. das modellierte System rekonfiguriert werden muß) um G zu optimieren.

Optimierungsalgorithmen

Die Analyse von Markov-Entscheidungsmodellen erfolgt numerisch durch iterative Verfahren. Dafür werden aus der Dynamischen Programmierung bekannte Methoden an die stochastischen Lösungsalgorithmen der Markov-Analyse gekoppelt, um optimale Strategien zu ermitteln. Mathematisch betrachtet führt dies (sowohl im stationären als auch im transienten Fall) für jeden Iterationsschritt zur Lösung großer, dünn besetzter linearer Gleichungssysteme über der Dimension des Zustandsraums Ω.

Stationäre Analyse: Für die Bestimmung stationärer Strategien existieren zwei Varianten:

- Bei der ***Strategie-Iteration*** (*policy iteration*) wird von einer zufällig ausgewählten Anfangsstrategie ausgehend diese in jedem Iterationsschritt so variiert, daß sich eine direkte Verbesserung des zu erwartenden Bewertungsmaßes pro Zeiteinheit ergibt. Das Verfahren konvergiert in der optimalen stationären Strategie [Tij86].

- Da bei der Strategie-Iteration in jedem Iterationsschritt ein lineares Gleichungssystem der Dimension $|\Omega|$ gelöst werden muß, kann das Verfahren für große Systeme zu zeitaufwendig werden. Eine Alternative bietet unter bestimmten Voraussetzungen (i.e. die Aperiodizität sowie einfache Ergodizität der als Modellinstanzen vorliegenden Markov-Ketten) die ***Werte-Iteration*** (*value iteration*), die darauf beruht, eine mit weniger Zeitaufwand zu berechnende Approximation (die »Wertefunktion«) der Zielfunktion durch Variation der Strategie zu optimieren.

Transiente Analyse: Die Berechnung zeitabhängiger Strategien für EMRM erfolgt ebenfalls iterativ. Dabei wird die Markov-Eigenschaft ausgenutzt; für die Berechnung wird rückwärts entlang der Zeitachse vorgegangen – ausgehend vom Endpunkt des Planungshorizont-Zeitintervalls wird (nach einer Diskretisierung der Zeitachse) zu jedem Zeitpunkt, an dem eine Rekonfiguration möglich ist, diejenige Modellinstanz ausgewählt, die in einer Optimierung des Erwartungswertes des akkumulierten Bewertungsmaßes resultiert. Die Markov-Eigenschaft garantiert, daß diese Auswahl möglich ist und in einer (im Mittel) optimalen Strategie resultiert [Mau90]. Die Wertefunktion des erwarteten akkumulierten Rewards zwischen den für die zeitliche Diskretisierung verwendeten Zeitpunkten wird durch eine Taylor-Entwicklung approximiert[3].

[3] Bei der Auswertung der EMRM in der Praxis ist eine günstige Wahl der Diskretisierungs-Schrittweite für schnelle bzw. ausreichend genaue Approximationen ausschlaggebend; hier kann der Algorithmus sowohl durch eine adaptive Festlegung der Zeitintervalle als auch durch passende Wahl des Grades der Taylor-Polynome angepaßt werden (vgl. [dM92], S. 56 – 58).

Werkzeug-Unterstützung

In der Praxis ist die Analyse von Markov-Entscheidungsmodellen (bedingt durch die aufwendigen, iterativen Algorithmen sowie die bei realitätsnahen Modellen fast immer sehr großen Zustandsräume) nur mit Rechnerunterstützung sinnvoll durchführbar. Dabei hängt die Leistung aller Analyse-Verfahren stark von Implementierungsdetails ab – heuristisch günstige Auswahl der Anfangsstrategie bei der stationären Analyse, die Qualität der für die Lösung der dünn besetzten, linearen Gleichungssysteme verwendeten Algorithmen sowie die geeignete Wahl numerischer Parameter wie Diskretisierungs-Intervalle, Konvergenz-Schranken etc. können gerade bei großen oder steifen (d.h. von Übergangsraten extrem unterschiedlicher Größenordnung geprägten) Modellen den Zeitaufwand für die Analyse erheblich beeinflussen.

Für die Implementierung des DSS wurde PENELOPE [dMŠ97] als EMRM-Löser eingesetzt; er bietet sowohl stationäre als auch transiente Analyseverfahren mit effizienter Rechenleistung. PENELOPE verarbeitet EMRM als Eingabe, die auch mit Hilfe einer zusätzlich verfügbaren graphischen Benutzungs-Schnittstelle in der in Abb. 3.1 dargestellten Form spezifiziert werden können; das Programm errechnet sowohl die im Mittel optimalen Strategien als auch die aus deren Anwendung resultierenden Erwartungswerte der Bewertungsmaße.

3.2.4 Anwendungen der Markov-Entscheidungsmodellierung

Mit der Markov-Entscheidungsmodellierung steht eine leistungsfähige Theorie zur Verfügung, um ein so komplexes Problem wie die dynamische Steuerung von Systemen unter dem Einfluß von zufälligen Ereignissen zu planen und sogar zu optimieren. In Gestalt der EMRM (bzw. der entsprechenden Werkzeug-Unterstützung) existiert auch ein mächtiges Modellierungsverfahren, um die Markov-Entscheidungsmodellierung umzusetzen: Modelle können aufgebaut und daraus die (im Mittel) optimalen Steuerungsstrategien berechnet werden, ohne daß dazu detailierte Kenntnisse über die zur Analyse angewendeten Algorithmen notwendig sind.

Dennoch hat die Anwendung von Markov-Entscheidungsmodellen – bezogen auf das Potential dieses Modellierungsverfahrens – in der Informatik bis jetzt relativ wenig Verbreitung erfahren. Ein Grund hierfür ist sicherlich, daß sich die Bildung bzw. Bewertung der Modelle auf Zustandsebene aufwendig und schwierig gestalten kann: Die EMRM können nicht abstrakt aus charakteristischen Klassen von Elementen und Übergängen des modellierten Systems aufgebaut werden, sondern es muß direkt auf die »atomaren« Zustände und Zustandsübergänge von Teilen dieses Systems Bezug genommen werden. Auch muß zu diesem Zweck der Zustandsraum des Modells explizit und vollständig vom Modellierer aufgebaut

werden; dies kann – selbst wenn für die Eingabe eine komfortable, graphische Benutzungs-Schnittstelle zur Verfügung steht – bei den in der Praxis auftretenden Systemen mit ihren zumeist sehr großen Zustandsräumen ein erhebliches Problem darstellen.

Die bisherigen Einsatzgebiete der EMRM liegen im Wesentlichen in der Performability-Modellierung, d.h. der integrierten Modellierung von Leistungs- und Zuverlässigkeitsaspekten von parallelen, fehlertoleranten Rechenanlagen ([dM92, dMDT94], s.a. 3.3.3). Weitere Anwendungen (die meisten ebenfalls aus dem Gebiet der Optimierung von Wartungs- und Reparatur-Prozessen) werden in [HF96] aufgeführt, wo zeitdiskrete Markov-Entscheidungsmodelle zur Dienstgüte-Optimierung in Telekommunikations-Netzwerken benützt werden.

3.3 Rekonfigurations-Petrinetze

Es liegt nahe, den in 3.2.4 aufgeführten Problemen der Entscheidungsmodellierung auf Markov-Ebene analog zu begegnen wie beim Übergang von der Markov- zur Petrinetz-Modellierung: In der Tat stellt es sich heraus, daß es mit wenig Aufwand möglich ist, die in 2.1.1 vorgestellte Konstruktion der SRN so zur Modellierung von Entscheidungen auszubauen, daß damit:

1. Entscheidungsmodelle auf Petrinetz-Ebene beschrieben sowie gleichzeitig

2. die Analyseverfahren der EMRM zur Optimierung von Entscheidungen mit den auf diese Art erweiterten SRN genutzt werden können.

3.3.1 Definition

In [dMM91, Weg96, dMDF99] wurde eine Erweiterung von stochastischen Petrinetzen vorgeschlagen, die eine Abbildung des modifizierten Erreichbarkeitsgraphen auf ein EMRM nach [dM92] gestattet.

Abgeleitet wird dies aus der bekannten Abbildung des Erreichbarkeitsgraphen eines stochastischen Petrinetzes auf eine zeitkontinuierliche Semi-Markov-Kette: In Petrinetzen werden Zustandsübergänge als Wechsel zwischen Markierungen repräsentiert und durch das Feuern von Transitionen beschrieben. Diese Übergänge finden sich als Kanten im Erreichbarkeitsgraphen des Netzes wieder und werden für die stochastische Analyse zu Übergängen in der Markov-Kette (respektive Übergangskanten in dem die Kette repräsentierenden Graphen).

Andersherum betrachtet (d.h. falls ein EMRM als Erreichbarkeitsgraph eines neuen Typs von SRN aufgefaßt wird) können die durch die Rekonfigurationskanten dargestellten, optionalen Übergänge der EMRM als das Feuern einer neuen

Klasse von Transitionen auf Petrinetz-Ebene interpretiert werden. Für die Erweiterung der SRN zur Entscheidungsmodellierung wird also zusätzlich zu den exponentiell verzögerten und zeitlosen Transitionen eine dritte Klasse definiert: Die ***Rekonfigurationstransitionen***.

Um eine konsistente Integration in die Definition der SRN zu gewährleisten, werden für Rekonfigurationstransitionen folgende Festlegungen getroffen:

Priorisierung: Den Rekonfigurationstransitionen wird dieselbe implizite Priorität wie zeitbehafteten Transitionen zugeordnet – sie können nur mit zeitbehafteten Transitionen konkurrieren bzw. falls eine Rekonfigurationstransition gleichzeitig mit zeitlosen Transitionen feuerbereit ist, feuert immer eine der zeitlosen Transitionen zuerst. Durch diese Verteilung der Prioritäten wird sichergestellt, daß die durch das Feuern von Rekonfigurationstransitionen ausgelösten Zustandsübergänge nur in zeitbehafteten Markierungen des Netzes auftreten und somit auch nur von Markov-Zuständen des aus dem Petrinetz generierten EMRM ausgehen können (entsprechend der Definition von Markov-Entscheidungsmodellen auf S. 38).

Schaltverzögerung: Das Feuern einer Rekonfigurationstransition wird im Gegensatz dazu – anders als bei den zeitbehafteten Transitionen und analog zur Durchführung der Entscheidungsaktionen im EMRM – als ein Vorgang betrachtet, der keine Zeit benötigt (da er ja nur das Fällen einer Entscheidung modellieren soll).

Feuerbedingung: Die Regeln für das Feuern einer Petrinetz-Transition werden für die neuen Transitionen geändert – zusätzlich zu den logischen Bedingungen (d.h. der Feuerbereitschaft, s. S. 24) wird anstatt von stochastisch ausgelösten Ereignissen bei zeitbehafteten Transitionen (also der Auswertung einer Verteilungsfunktion für die Schaltverzögerung) für das Schalten einer Rekonfigurationstransition die Optimierung eines Bewertungsmaßes durch den ausgelösten Zustandsübergang verlangt.

Dies bedeutet, daß den Rekonfigurationstransitionen explizit (d.h. durch den Modellierer) weder Schaltraten noch -wahrscheinlichkeiten zugewiesen werden müssen – ihr Zweck ist es ausschließlich, eine *optionale Rekonfiguration* des modellierten Systems zu repräsentieren bzw. im Modell eine mögliche Auswahl zwischen zwei Instanzen darzustellen (dies entspricht genau den Rekonfigurationskanten der EMRM).

In der Folge werden die durch möglicherweise konkurrierende Rekonfigurationstransitionen entstehende Konflikte während der Auswertung auch nicht durch die Bestimmung von stochastischen Größen (wie Zustands-Wahrscheinlichkeiten oder mittleren Schalt-Frequenzen) aus entsprechenden stochastischen Parametern

(also Schaltwahrscheinlichkeiten oder Verteilungen von Schalt-Verzögerungen) gelöst, sondern durch die Auswertung von Optimierungsbedingungen: Es wird zu jedem Zeitpunkt, an dem eine Rekonfiguration vorgenommen werden kann, genau diejenige der möglichen Modellinstanzen ausgewählt (d.h. die entsprechende Rekonfigurationstransition geschaltet), die den Erwartungswert eines Bewertungsmaßes maximal verbessert.

Die implizite Priorisierung der Rekonfigurationstransitionen entsteht dabei zunächst aus der Notwendigkeit, eine automatische Abbildung des Erreichbarkeitsgraphen der erweiterten Petrinetze auf EMRM derart anzupassen, daß eine Analyse mit den vorhandenen Verfahren und Software-Paketen vorgenommen werden kann.

Allerdings ist eine solche Priorisierung auch vom modellierungstechnischen Standpunkt her zweckmäßig, zumindest falls – wie in der vorliegenden Arbeit – reale Systeme modelliert werden sollen: Über die Rekonfiguration eines solchen Systems kann nur dann sinnvoll entschieden werden, wenn das System sich in einem »realen« Zustand befindet, der auch eine physikalische Entsprechung haben muß (d.h. über eine meßbare Zeitspanne hinweg eingenommen wird). Dagegen repräsentiert das Feuern von zeitlosen Transitionen eben keine im modellierten System wirklich stattfindenden Zustandsübergänge, sondern nur die stochastischen Aspekte der *Auswirkung* von real aufgetretenen Zustandsübergängen (d.h. mit welcher Wahrscheinlichkeit ein Zustandsübergang welche Konsequenzen hat). Insofern ist es auch gerechtfertigt, in solchen zeitlosen »Zwischenzuständen« keine Rekonfigurationen zuzulassen[4].

Die wie oben beschrieben auf der Grundlage der SRN bzw. der EMRM definierten Petrinetze mit Rekonfigurationstransitionen wurden in [Weg96] als *Erweiterte GSPN Reward-Modelle* und in [dMD96, dMDF99] bzw. [dMM91] als *Controlled Stochastic Petrinets (COSTPN* bzw. *COST*) bezeichnet; sie werden in dieser Arbeit ***Rekonfigurations-Petrinetze*** (*RPN*) genannt und stellen das für das DSS verwendete Modellierungsverfahren dar.

3.3.2 Auswertung von RPN

Die in 3.3.1 definierten RPN sollen natürlich auch ausgewertet werden können, und zwar (analog den EMRM) mit dem Ziel der Bestimmung von optimalen Stra-

[4]Die implizite Priorisierung stellt auch in der Praxis keine Einschränkung bei der Modellierung dar – weil die Zustandsräume der erweiterten Petrinetze automatisch generiert werden, muß die Priorisierung der Rekonfigurationstransitionen nicht vom Modellierer beachtet werden (da sie vom Werkzeug umgesetzt wird); sollten Rekonfigurationstransitionen in einem Modell durch die implizite Priorisierung niemals schalten können, kann vom Werkzeug auch schon während der Erreichbarkeitsgraph-Erzeugung eine entsprechende Warnung an den Modellierer ausgegeben werden.

tegien für das Schalten der Rekonfigurationstransitionen. Dazu wird aus der RPN-Spezifikation automatisch ein EMRM erzeugt, das dann mit den bekannten Verfahren der Markov-Entscheidungsmodellierung analysiert werden kann.

Generierung von EMRM

Der erste Schritt zur Abbildung eines RPN auf ein EMRM ist die Entfaltung des Erreichbarkeitsgraphen aus der RPN-Spezifikation. Zunächst wird dazu der Graph mittels eines üblichen Tiefen- oder Breitensuch-Algorithmus[5] generiert – im Falle eines RPN müssen dabei jedoch bei der Generierung neuer Markierungen für Rekonfigurationstransitionen zusätzlich die erweiterten Bedingungen zur Feuerbereitschaft (d.h. die implizite Priorisierung) berücksichtigt werden.

Im nächsten Schritt wird dann (für jedes Element $(\boldsymbol{\rho}, \boldsymbol{r})$ aus der Menge der Bewertungsmaße des RPN) die Umwandlung des Erreichbarkeitsgraphen in ein EMRM – ähnlich wie beim Übergang vom SRN zu einer Semi-Markov-Kette (vgl. S. 27) – nach folgender Transformationsvorschrift durchgeführt:

1. Zeitbehaftete Markierungen, Markov-Zustandsübergänge mit ihren Raten sowie die Zustandsmaße der als Zielgröße für die Optimierung dienenden Bewertungsfunktion werden direkt (d.h. genauso wie bei der Generierung eines Markov-Reward-Modells aus einem SRN) übertragen.

2. Die zeitlosen Markierungen im Erreichbarkeitsgraphen werden zu Verzweigungs-Zuständen[6] im EMRM.

3. Die Kanten/Zustandsübergänge, die das mögliche Feuern einer Rekonfigurationstransition im Erreichbarkeitsgraphen des RPN repräsentieren, werden auf Rekonfigurationskanten abgebildet, die die möglichen Entscheidungsaktionen im EMRM modellieren.

4. Für die Impulsmaße der Bewertungsfunktion werden in die jeweils relevanten Kanten aus $\mathfrak{E}$ (aus technischen Gründen[7]) zusätzliche zeitlose Zustände eingetragen, die den entsprechenden Zustandsübergang bewerten.

[5] Ein solcher Algorithmus ist z.B. in [CBC+93] (S. 166) angegeben und kann – mit geringfügigen Anpassungen der Feuerregeln – für die RPN-Auswertung direkt übernommen werden.

[6] Auf die Darstellung der in den anderen Arbeiten zur Definition der RPN [Weg96, dMD96, dMDF99] ausgeführten Konstruktion zur Eliminierung von transienten Schleifen zeitloser Zustände wird hier aus Platzgründen verzichtet, da solche Schleifen in praktisch relevanten Modellen selten vorkommen (und insbesondere bei den für die vorliegende Arbeit erstellten DSS-Modellen mit azyklischen Erreichbarkeitsgraphen nicht auftreten).

[7] Diese zusätzlichen Zustände sind nur deshalb notwendig, da der für das DSS eingesetzte Löser Impulsmaße im Modell ausschließlich in der Form von »Bewertungsraten« an zeitlosen Zustände einlesen kann. In der Definition der EMRM [dM92] sind die Impulsmaße daher direkt den Verzweigungs-Zuständen zugeordnet, jedoch werden sie bei der Herleitung des Algorithmus

Die im letzten Schritt beschriebene Erweiterung des Erreichbarkeitsgraphen ist ohne Beeinträchtigung der Übertragbarkeit der analytischen Lösung des EMRM auf das RPN möglich, da der modifizierte Erreichbarkeitsgraph als Markov'scher Entscheidungsprozess dem ursprünglichen Prozess (in Bezug auf die Strategieoptimierung) äquivalent ist: Weil jeder der neuen Zustände ausschließlich ohne Konkurrenz eingenommen bzw. wieder verlassen wird und in den neuen Zuständen auch weder Zeit verbracht wird noch Entscheidungsaktionen möglich werden, ändern sie nichts an den Erwartungswerten der Bewertungsmaße des vom Erreichbarkeitsgraphen beschriebenen Entscheidungprozesses (obwohl die zusätzlichen Zustände im eigentlichen Erreichbarkeitsgraphen des RPN nicht enthalten sind).

Alle oben aufgeführten Schritte zur Umwandlung eines RPN in ein EMRM können natürlich von einem Werkzeug automatisch (d.h. ohne Zutun des Modellierers) durchgeführt werden.

Analyse und Ergebnis-Umrechnung

Das so entstandene EMRM kann als in dem Sinn isomorph zu dem RPN aus dem es generiert wurde betrachtet werden, daß es den Erreichbarkeitsgraphen des RPN mit den Rekonfigurations-Möglichkeiten genauso eindeutig auf ein EMRM abbildet, wie ein SRN von den bekannten Algorithmen zur Analyse auf eine Semi-Markov-Kette abgebildet wird.

Umgekehrt können die Ergebnisse der Analyse des aus dem RPN erzeugten EMRM auch (analog den Ergebnissen der Markov-Analyse eines SRN) auf das RPN zurückbezogen werden: Optimale Zeitpunkte für das Einleiten eines Rekonfigurationsüberganges im EMRM entsprechen genau den optimalen Zeitpunkten für das Schalten von Rekonfigurationstransitionen im RPN.

Daher können die Algorithmen zur Strategieoptimierung von EMRM (sowohl im transienten als auch im stationären Fall) zur Analyse[8] von RPN herangezogen werden; für die Umrechnung der Ergebnisse der EMRM-Analyse müssen lediglich die Rekonfigurationstransitionen im RPN identifiziert werden, deren Feuern mit Übergängen an den Rekonfigurationskanten im EMRM assoziiert ist. Die Vorteile der Modellierung mit RPN gegenüber EMRM (d.h. die automatische Entfaltung der Zustandsräume) bleiben dabei – wieder analog dem Verhältnis SRN-gegenüber Markov-Modellierung – erhalten.

zur Analyse der EMRM (analog zur SRN-Definition) ebenfalls auf die Kanten übertragen (vgl. [dM92], S. 60).

[8]Die simulative Auswertung der generierten EMRM wäre ebenso möglich (der für das DSS verwendete Löser hat auch eine Simulationskomponente), aber mit den gleichen Problemen behaftet die bei der Simulation von SRN auftreten (d.h. sehr lange Simulationszeiten bei Modellen mit seltenen Ereignissen, vgl. Fußnote auf S. 26).

Integration in PANDA

Wie bei SRN ist auch die Modellierung mit RPN ohne die Unterstützung durch leistungsfähige Werkzeuge nicht sinnvoll anwendbar. Für die DSS-Modellierung wurde daher in PANDA die Möglichkeit integriert, zusätzlich zu SRN auch RPN spezifizieren und auswerten zu können:

- Mittels der graphischen Benutzungs-Schnittstelle können vom Modellierer in den Petrinetzen auch Rekonfigurationstransitionen deklariert werden.
- Soll ein Petrinetz ausgewertet werden, erkennt das Werkzeug vorkommende Rekonfigurationstransitionen automatisch, und wandelt den Erreichbarkeitsgraphen nach der Erzeugung in ein EMRM um. Dabei wird auch je eine Impuls- und Zustands-Bewertungsfunktion des RPN (mit dem PANDA-Interpreter) in Zustands- und Impulsmaße für das EMRM umgerechnet, die als Zielfunktion an die EMRM-Strategieoptimierung übergeben werden.
- Danach wird von PANDA der PENELOPE-Löser [dMŠ97] gestartet, dem das aus dem RPN generierte EMRM als Eingabe zur Analyse zugeführt wird.

3.3.3 Beispiel

Um die Umsetzung von RPN auf EMRM sowie das Potential der Entscheidungsmodellierung auf Petrinetz-Ebene zu demonstrieren, wird hier der Aufbau eines RPN beschrieben, das zur Analyse auf aus der Literatur bekannte EMRM abgebildet wird: In [dMDT94] werden Entscheidungen über die Reparatur eines fehlertoleranten, parallelen Rechensystems bei Ausfall von Teilen des Systems mit Hilfe eines EMRM optimiert (in Abhängigkeit von der Ausfallrate, der verbleibenden Missionszeit nach dem Ausfall-Ereignis bzw. der angewendeten Reparatur-Methode)[9].

Ausgangssituation

Modelliert werden soll ein System mit i parallel arbeitenden Prozessoren; das Ausfallverhalten der einzelnen Prozessoren sei dabei mit einer Exponential-Ver-

[9] Das Beispiel soll in der vorliegenden Arbeit die Äquivalenz (in Bezug auf die Strategieoptimierung) des RPN-Modells zu dem in der Literatur beschriebenen EMRM aufzeigen und die Vorteile der RPN-Modellierung verdeutlichen. Deshalb wird hier auch auf die Ergebnisse der Auswertung des Reparatur-EMRM unter mathematischen und Zuverlässigkeits-Gesichtspunkten nicht näher eingegangen – eine ausführliche Darstellung mit verschiedenen Experimenten (bei denen neben einigen Modell-Parametern auch die Reparaturstrategie und die Bewertungsmaße variiert werden) sowie eine detaillierte Diskussion der Analyse-Resultate ist in [dMDT94] zu finden.

teilung zur Rate γ beschreibbar (d.h. als mittlere Rate für den Ausfall eines Prozessors ergibt sich bei $1 \leq j \leq i$ betriebsbereiten Prozessoren $j \cdot \gamma$). Das Gesamtsystem soll fehlertolerant sein in der Hinsicht, daß es nach dem Ausfall eines Prozessors mit den verbleibenden Prozessoren weiterarbeiten kann (mit verminderter Leistung sowie verringerter Redundanz) solange noch mindestens ein Prozessor einsatzfähig ist.

Zum Wiederaufsetzen nach dem Fehlerfall wird ein Fehlererkennungs-Verfahren mit der Überdeckung c eingesetzt – fällt von j Prozessoren einer aus und war die Fehlererkennung erfolgreich, so kann das Gesamtsystem innerhalb kurzer Zeit rekonfiguriert werden (der Zeitbedarf dafür wird als exponentiell zur Rate α verteilt angenommen) und danach mit $j-1$ Prozessoren weiterarbeiten. Scheitert die Fehlererkennung (mit der Wahrscheinlichkeit $1-c$), muß ein Neustart des Gesamtsystems erfolgen. Der Zeitbedarf für den Neustart soll ebenfalls exponentiell verteilt sein, aber erheblich höher mit der Rate $\beta \ll \alpha$.

Weiter existiert eine Reparatur-Einrichtung für defekte Prozessoren: Jedes Mal wenn das System nach einem Ausfall wieder den Betrieb aufnimmt (also noch mindestens ein Prozessor betriebsbereit ist), kann entschieden werden, ob einer der ausgefallenen Prozessoren repariert werden soll. Ein totaler System-Ausfall kann jedoch nicht hingenommen werden: Sobald der letzte Prozessor ausfällt, muß auf jeden Fall sofort eine Reparatur erfolgen (da das System sonst keine Leistung mehr erbringen würde).

Wird die Entscheidung für eine Reparatur getroffen, so kann nach einer (wieder exponentiell zur Rate δ verteilt angenommenen) Zeitspanne der betreffende Prozessor erneut den Betrieb aufnehmen (d.h. er wird in das Gesamtsystem eingegliedert, das daraufhin mit um einen Prozessor erhöhter Leistung weiterarbeitet).

Es wird davon ausgegangen, daß das System innerhalb einer endlichen Missionszeit (die deutlich größer sein soll als die mittlere Zeitdauer $\frac{1}{\gamma}$ bis zum Ausfall eines oder mehrerer Prozessoren) eine Aufgabe zu erfüllen hat; die durch die Entscheidungsmodellierung zu untersuchende Fragestellung ist, wann und unter welchen Gesichtspunkten ausgefallene Prozessoren repariert werden sollen bzw. wann es günstiger erscheint, das System (unter Ausnutzung der Redundanz) bis zum Ende der Missionszeit mit verringerter Prozessor-Zahl weiter zu betreiben.

Modellierung mit EMRM

In Abb. 3.2 ist das EMRM aus [dMDT94] dargestellt, das die oben beschriebene Problemstellung umsetzt. Dabei haben die Elemente des EMRM folgende Funktion (vgl. Abb. 3.1 auf S. 39 zur Erklärung der graphischen Notation):

Markov-Zustände: Die Zustände N_j $(0 \leq j \leq i)$ modellieren die Zahl der intakten Prozessoren – befindet sich das System im Zustand N_j, arbeitet es mit

j Prozessoren. Die Zustände RC_j und RB_j (mit $1 \leq j \leq i$) stellen das System nach dem Ausfall eines Prozessors und erfolgter Fehlererkennung dar, wobei RC_j (*Recover*) für die Wiederaufsetz-Phase nach erfolgreicher und RB_j (*Reboot*) für den Neustart nach gescheiterter Fehlererkennung (mit jeweils j Prozessoren) stehen. Mit den Zuständen R_j $(2 \leq j \leq i)$ werden die Reparatur-Phasen modelliert: Im Zustand R_j sind $j-1$ Prozessoren intakt und einer wird repariert.

Verzweigungs-Zustände: Die Zustände Cov_j (*Coverage*, $2 \leq j \leq i$) repräsentieren das System nach dem Ausfall eines Prozessors.

Zeitlose Ubergänge: Die von jedem Verzweigungs-Zustand ausgehenden beiden Kanten (mit ihren die Überdeckung des Fehlererkennungs-Verfahrens repräsentierenden Wahrscheinlichkeiten c bzw. $1-c$) modellieren ohne Zeitverzögerung das zufällig bestimmte Ergebnis der (erfolgreichen bzw. erfolglosen) Fehlererkennung.

Zeitlich verzögerte Übergänge: Die Zustandsübergänge mit exponentiell verteilter Zeitverzögerung zu den Raten α bzw. β und δ bzw. γ repräsentieren die Zeitdauern für Wiederaufsetzen bzw. Neustart des Systems nach einem Ausfall, sowie den Zeitbedarf für die Reparatur bzw. die durchschnittliche Ausfallrate eines einzelnen Prozessors. Dabei muß die Ausfallrate an den Übergängen von den Zuständen N_j bzw. R_j (für $1 \leq j \leq i$) mit der Anzahl der jeweils intakten Prozessoren multipliziert werden, um die höhere Rate (für die voneinander unabhängigen Ausfälle) korrekt in das Modell umzusetzen.

Rekonfigurationskanten: Die Übergänge zwischen den Zuständen N_j und R_{j+1} (mit $1 \leq j \leq i-1$) modellieren die möglichen Entscheidungsaktionen des Systems – wann immer sich das Modell in einem der Ausgangszustände der Rekonfigurationskanten befindet, kann die Entscheidung zur Reparatur getroffen (d.h. das Modell in den zugehörigen Reparatur-Zustand überführt) werden. Ausgenommen werden müssen die Zustände N_1 und R_2: Tritt in diesen ein Ausfall-Ereignis ein, muß sofort repariert werden (da der dem Totalausfall des Gesamtsystems entsprechende Zustand N_0 in jedem Fall so schnell wie möglich wieder verlassen werden muß).

Bewertungsmaß: Hier wird nur die Verfügbarkeit des Systems bewertet – genau diejenigen Zuständen, in denen mindestens ein Prozessor intakt ist (d.h. N_j für $1 \leq j \leq i$ sowie R_l für $2 \leq l \leq i$) werden mit dem Zustandsmaß 1 belegt, alle anderen mit 0 (das Bewertungsmaß differenziert also nicht nach der von der Zahl der arbeitenden Prozessoren abhängigen Leistung des Systems).

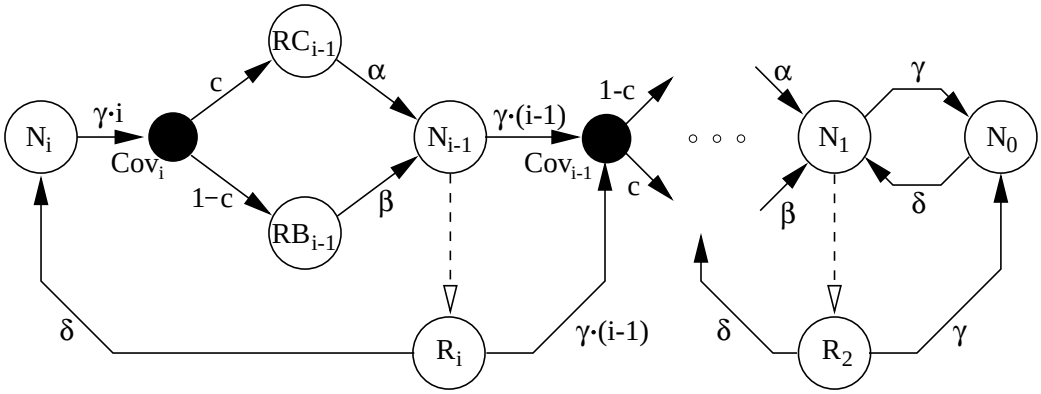


Abbildung 3.2: Beispiel EMRM – Reparaturstrategie-Modellierung für *i* redundante Prozessoren (nach [dMDT94], S. 182)

Modellierung mit RPN

In Abb. 3.3 sind Netzplan und Inhibitorfunktionen eines RPN angegeben, das dieselbe Problemstellung modelliert wie das EMRM aus Abb. 3.2 – die Stellen des Netzes repräsentieren hier die Zustände, die *ein* Prozessor durchlaufen kann, d.h. die Prozessoren des Systems werden durch Token dargestellt. Die Elemente des RPN haben dabei folgende Funktion (vgl. Abb. 2.1 auf S. 26 zur Erklärung der graphischen Notation):

Stellen: P1:Up dient zur Modellierung der intakten Prozessoren; die Zahl der Token in dieser Stelle gibt an, wieviele der im System vorhandenen Prozessoren betriebsbereit sind (für P1:Up ist daher auch die Anfangsmarkierung von *i* Token eingetragen). P2:Failed stellt den Ausfall von Prozessoren vor der Fehlererkennung dar; P3:Recover bzw. P4:Reboot bilden den Zustand der Prozessoren nach erfolgreicher bzw. gescheiterter Fehlererkennung nach, P5:Down bzw. P6:Repair modellieren die ausgefallenen bzw. in Reparatur befindlichen Prozessoren.

Zeitlich verzögerte Transitionen: Die exponentiell verzögerte Transition e1 repräsentiert das Ausfallverhalten der Prozessoren – ihre Rate ist daher abhängig von der Zahl der in der Stelle P1:Up befindlichen Token (und in der in PANDA implementierten und in 2.2.2 bzw. App. A beschriebenen Syntax als mark(P1:Up)·γ angegeben, wobei γ wieder die Ausfallrate eines einzelnen Prozessors ist). Die Transitionen e2, e3 sowie e4 haben markierungsunabhängige Raten α, β bzw. δ und modellieren die Zeitverzögerungen, die beim Wiederaufsetzen, dem Neustart des Systems sowie bei der Reparatur eines Prozessors anfallen.

Zeitlose Transitionen: Die in direkter Konkurrenz zueinander schaltenden Transitionen t1 bzw. t2 bilden mit ihren Schaltwahrscheinlichkeiten c bzw. $1-c$ die Überdeckung des Fehlererkennungs-Verfahrens nach, während t3 und t4 die Logik für die sofortige Einleitung der Reparatur nach dem Ausfall des letzten Prozessors modellieren (dabei ist t3 bzw. t4 für den Fall zuständig, daß sich zum Ausfall-Zeitpunkt des letzten Prozessors schon ein bzw. noch kein Prozessor in Reparatur befindet).

Rekonfigurationstransition: Die Transition r modelliert die Entscheidungsaktion über die Reparatur der ausgefallenen Prozessoren – sobald sich das System nach einem Ausfall wieder in stabilem Zustand befindet (d.h. die Fehlerbehandlung beendet ist, was durch den Transport eines Token in P5:Down im Modell umgesetzt ist) kann entschieden werden, ob r geschaltet (also ein Reparatur-Vorgang gestartet) wird. Die Ausfall-Ereignisse konkurrieren dabei sowohl mit der Entscheidungsaktion als auch mit der evtl. stattfindenden Reparatur-Phase.

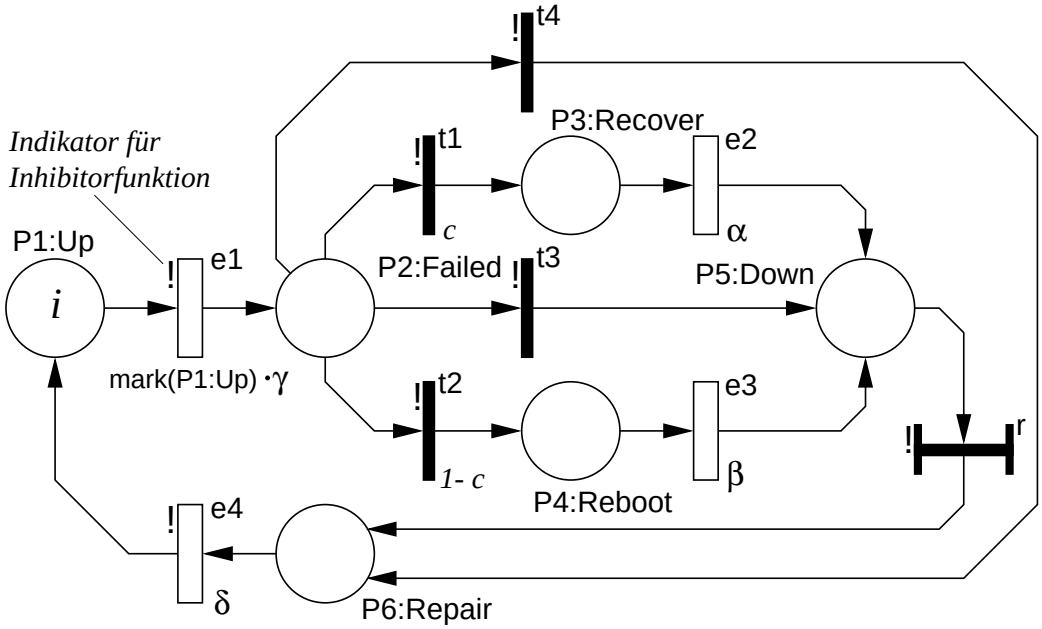
Bewertungsfunktion: Ein SRN-Zustandsmaß, das dieselbe Semantik wie das auf S. 50 für das EMRM-Modell angegebene realisiert, kann unter der Verwendung der in PANDA implementierten Syntax z.B. so formuliert werden:

```
IF ((mark(P3:Recover) = 0) AND (mark(P4:Reboot) = 0)
    AND (mark(P1:Up) > 0)
) THEN (
          1
) ELSE (
          0
)
```

Hier werden nur die Zustände mit dem Maß 1 belegt, in denen Token in Stelle P1:Up, aber keine in den Stellen P3:Recover oder P4:Reboot vorhanden sind (und alle anderen Zustände mit 0) – diese Zustände im Modell entsprechen genau den Zuständen des Systems, in denen es Nutzleistung erbringt.

Neben den üblichen SRN-Elementen enthält das RPN in Abb. 3.3(a) zusätzlich die in Tabelle 3.3(b) aufgeführten Inhibitorfunktionen[10]; diese – im direkten Vergleich zum EMRM aus Abb. 3.2 erhöhte – Komplexität entsteht dadurch, daß

[10] Ein RPN mit demselben Erreichbarkeitsgraphen könnte natürlich auch durch die Einführung von zusätzlichen (inhibitorischen) Kanten, d.h. ohne Inhibitorfunktionen und ausschließlich unter Verwendung der in 2.1.3 gezeigten graphischen Petrinetz-Notation beschrieben werden; um das Modell mit einem übersichtlicheren Netzplan darstellen zu können, sind in Abb. 3.3(a) jedoch nur diejenigen Kanten graphisch angegeben, über die im RPN auch Token transportiert werden.



(a) Netzplan für Beispiel RPN

Transition	Inhibitorfunktion
t1 t2	mark(P1:Up) > 0
t3	mark(P1:Up) = 0 AND mark(P6:Repair) > 0
t4	mark(P1:Up) = 0 AND mark(P6:Repair) = 0
e1 e2	mark(P3:Recover) = 0 AND mark(P4:Reboot) = 0
r	mark(P3:Recover) = 0 AND mark(P4:Reboot) = 0 AND mark(P6:Repair) = 0

(b) Inhibitorfunktionen für RPN aus Abb. 3.3(a) (Syntax wie in App. A angegeben)

Abbildung 3.3: Beispiel RPN – Reparaturstrategie-Modellierung wie in [dMDT94] für *i* redundante Prozessoren

viele (nach dem Netzplan bzw. den Feuerregeln) im Erreichbarkeitsgraphen des RPN mögliche Zustandsübergänge ausgeschlossen werden müssen: Während der Wiederaufsetz-Phasen darf kein Ausfall stattfinden, auch müssen die Rekonfigurations- sowie die die Reparatur modellierende Transition während dieser Zeit gesperrt werden.

Es muß aber betont werden, daß das RPN (im Gegensatz zum EMRM) sehr einfach für eine beliebige Zahl von Prozessoren des modellierten Rechensystems parametrisiert werden kann: Es reicht dazu aus, die Zahl der in der Anfangsmarkierung (in Stelle P1:Up) vorhandenen Token derjenigen der zu modellierenden Prozessoren anzupassen – das entsprechende EMRM wird dann über den Algorithmus zum Aufbau von Erreichbarkeitsgraphen für RPN automatisch erzeugt. Dagegen muß im Falle des EMRM für jede Anzahl von Prozessoren vom Modellierer ein neues Modell generiert werden; die allgemeine Form ist auf Markov-Ebene (wie auch schon an der graphischen Darstellung in Abb. 3.2 zu erkennen) nicht geschlossen darstellbar.

In Abb. 3.4 ist der erweiterte Erreichbarkeitsgraph des RPN aus Abb. 3.3 für ein System mit insgesamt 3 Prozessoren dargestellt:

- Sind in einer Markierung Rekonfigurations- oder zeitbehaftete Transitionen feuerbereit ist sie als Rechteck, andernfalls als Rechteck mit abgerundeten Ecken abgebildet.

- Für jede Markierung ist die Belegung der Stellen des RPN mit Token gezeigt (wobei die Reihenfolge in der Markierung der Nummerierung der Stellen in Abb. 3.3(a) entspricht); außerdem ist der Bezeichner des entsprechenden Zustands im EMRM (bezogen auf Abb. 3.2) angegeben. Dem EMRM-Zustand N_0 werden hier zwei zeitlose Markierungen[11] zugeordnet, die als Folge des Schaltens der zeitlosen Transitionen t4 bzw. t3 entstehen und die Sonderfälle des sofortigen Übergangs zur Reparatur-Phase (ohne Fehlerbehandlung und Wiederaufsetzen) nach dem Ausfall des letzten intakten Prozessors im RPN berücksichtigen.

- Die Übergänge zwischen den Markierungen sind mit den Namen der jeweils schaltenden Transition sowie ggf. der Rate oder Schaltwahrscheinlichkeit indiziert (Bezeichner ebenfalls wie in Abb. 3.3(a)).

Da der durch Anwendung der in 3.3.2 beschriebenen Transformation entstandene Erreichbarkeitsgraph des RPN zur Reparaturstrategie-Modellierung für jede Anfangsmarkierung genau dem EMRM für die entsprechende Zahl von modellierten Prozessoren entspricht (im Falle von 3 Prozessoren ist dies auch aus dem direkten Vergleich der Abbildungen 3.4 und 3.2 ersichtlich), können mit dem RPN dieselben Experimente und Analysen durchgeführt werden wie in [dMDT94], und die dort angeführten Ergebnisse sind direkt auf das RPN übertragbar.

[11] Diese zusätzlichen Markierungen sind im EMRM in [dMDT94] nicht enthalten; sie ändern jedoch nichts daran, daß durch den Erreichbarkeitsgraphen des RPN ein diesem EMRM (in Bezug auf die Erwartungswerte von Bewertungsmaßen) äquivalenter Markov'scher Entscheidungsprozess beschrieben wird (vgl. die Anmerkungen zur Einfügung zusätzlicher zeitloser Markierungen in den Erreichbarkeitsgraphen bei der EMRM-Generierung für RPN auf S. 47).

Jedoch wird bereits an diesem einfachen Beispiel deutlich, welche vorteilhaften Möglichkeiten die Kombination aus Markov-Entscheidungsmodellierung und bewerteten stochastischen Petrinetzen (bei entsprechender Werkzeug-Unterstützung zur automatischen Generierung der EMRM) bietet: Neben der Befreiung des Modellierers von der mühsamen und fehlerträchtigen Tätigkeit des Aufbaus großer und komplexer Zustandsräume kann die RPN-Modellierung in abstrakteren (d.h. von den einzelnen System-Zuständen gelösten) Begriffen erfolgen als auf Markov-Ebene. Zusätzlich können auch die Bewertungsmaße für die Modelle flexibler und auf höherem Abstraktionsniveau formuliert werden. Dabei wird es gleichzeitig möglich, transparent und ohne detaillierte Kenntnis der Theorie der Markov'schen Entscheidungsprozesse die leistungsfähigen Algorithmen zur EMRM-Analyse auch für die Auswertung von RPN einzusetzen.

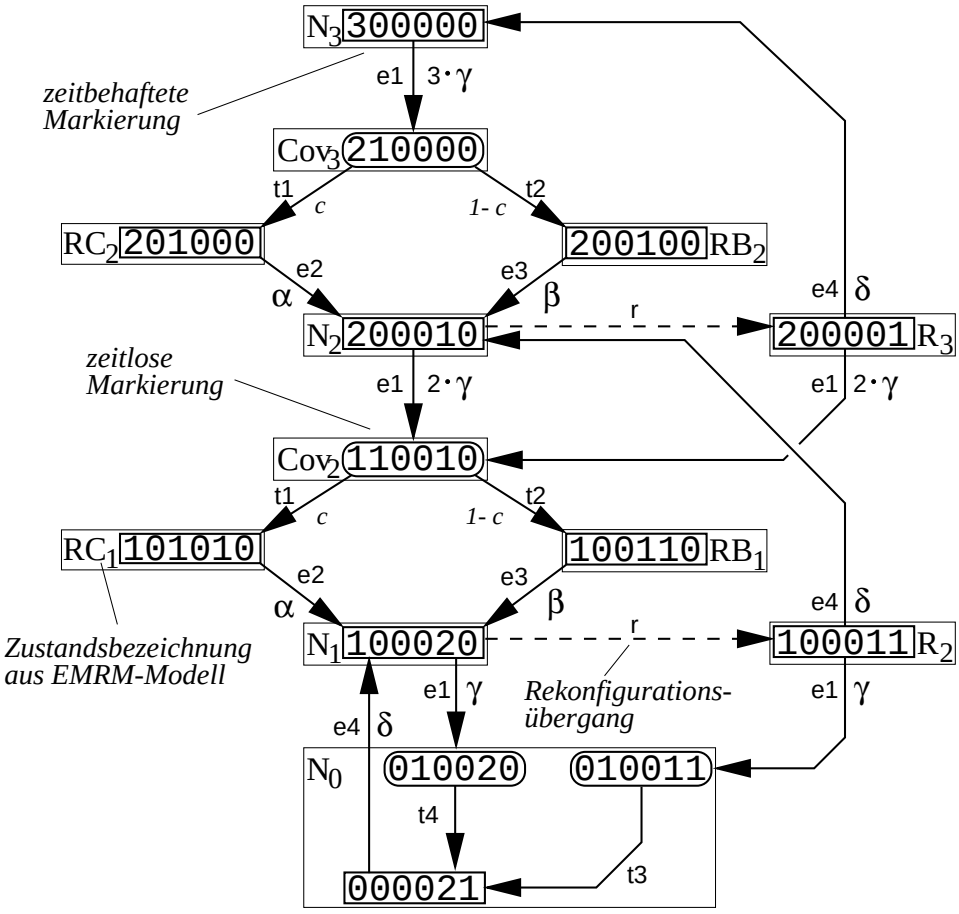


Abbildung 3.4: Beispiel Erweiterter Erreichbarkeitsgraph (Reparaturstrategie-Modellierung nach [dMDT94]) für 3 redundante Prozessoren

Kapitel 4

Hierarchische Darstellung von Petrinetzen

Beim Einsatz von Petrinetzen zur Modellierung komplexer Systeme stößt die für kleine Netze sehr vorteilhafte, anschauliche Visualisierung der Netze rasch an ihre Grenzen: Ab einem gewissen Umfang werden die graphischen Netzpläne unübersichtlich und Zusammenhänge schwer verständlich.

Da die für das DSS entwickelten Petrinetze realistische verteilte Systeme modellieren sollen, wurden in das zum Aufbau der Netze verwendete Werkzeug PANDA Möglichkeiten zur graphischen Hierarchisierung der Darstellung der Netzpläne integriert. Diese hierarchische Darstellung wird hier beschrieben, da sie für das Verständnis der für das DSS erstellten Modelle wichtig ist.

4.1 Hierarchisierung von Petrinetzen

Das Konzept der Hierarchisierung ist ein gängiger Ansatz zur Beherrschung von Komplexität; dabei werden größere Einheiten in über- und untergeordnete Teile mit definierten Schnittstellen und Übergängen zerlegt. Dies erlaubt Modularisierung und Abstraktion: Komponenten eines Systems können unabhängig von anderen entworfen und dargestellt werden. Das Gesamtsystem kann dann entweder durch Verfeinerung (*top-down*, ausgehend von den oberen Hierarchie-Ebenen) oder mittels Synthese (*bottom-up*, aus den unteren Hierarchie-Ebenen) aufgebaut werden.

Im Falle von Petrinetzen werden die Teilmodelle als ***Subnetze*** (*subnets* oder *pages*) bezeichnet. Jedes Subnetz kann sowohl normale Petrinetz-Elemente (Stellen, Transitionen, Kanten) als auch wiederum (rekursiv) andere Subnetze enthalten. Die Einbindung von Subnetzen in Petrinetze kann dabei auf unterschiedliche Weise realisiert werden:

Gemeinsame Stellen (*fusion* oder *common places*): Hier werden die Subnetze über Petrinetz-Stellen verbunden. Die gemeinsamen Stellen sind dabei in mehreren Subnetzen sichtbar, verhalten sich aber semantisch wie eine Stelle. Dadurch kann neben einer Modularisierung großer Netze auch eine übersichtliche Darstellung von Stellen mit sehr vielen ein- und ausgehenden Kanten erreicht werden. Dieses Konzept wurde in den Werkzeugen UltraSan [San95] und Design/CPN [Met93] implementiert.

Verfeinerte Transitionen (*substitution transitions*): Transitionen können mit Ersatz-Schaltbildern assoziiert werden, die den durch die Transition repräsentierten Subnetzen entsprechen. Der Transport von Token zwischen den Hierarchie-Ebenen geschieht durch spezielle Stellen, die im übergeordneten Teilnetz als *Port* und im untergeordneten als *Socket* bezeichnet werden. Design/CPN realisiert auch dieses Konzept.

***Schnittstellenkanten*:** Im Gegensatz zu den beiden vorgenannten Konzepten werden hier die Subnetze über Kanten miteinander verbunden. Dies führt dazu, daß die isolierten Subnetze keine eigenständigen Petrinetze mehr bilden (da die Petrinetz-Definition keine Kanten ohne Endknoten vorsieht). Kantenberandete Subnetze sind sowohl für die in [Ger97] vorgeschlagene Stochastic Petri Nets Language vorgesehen als auch im für die DSS-Modellierung verwendeten Werkzeug PANDA [AD97] implementiert.

Es muß betont werden, daß die Hierarchisierung sowohl bei Design/CPN, UltraSan als auch PANDA ausschließlich eine Unterstützung zur Visualisierung für den Modellierer darstellt – für die Analyse werden die Subnetze zu einem normalen, »flachen« Petrinetz mit Standardsemantik zusammengefaßt[1].

4.2 Implementierung in PANDA

Für PANDA wurden aus folgenden Gründen kantenberandete Subnetze in einer baumartigen Hierarchie realisiert [Mat96]:

- Sie erlauben die größtmögliche Freiheit beim Modellaufbau, da sowohl Stellen als auch Transitionen zu Subnetzen verfeinert werden können.

- Weder die visuelle noch die interne Repräsentation für klassische Petrinetz-Elemente mußten für die Implementierung verändert werden.

[1] Die Spezifikationen in [Ger97] sehen auch die Möglichkeit zur modularisierten Lösung über Fixpunktiteration vor, allerdings ist hierzu z.Zt. noch keine Werkzeug-Implementierung bekannt.

- Die Subnetze sind mit einem eigenen Symbol für den Modellierer gut erkennbar.

Die so erreichte Zerlegung der Petrinetze entspricht (bezogen auf das Gesamtnetz) einem »Aufschneiden« entlang ausgewählter Kanten. Wie schon erwähnt haben die auf diese Art definierten Subnetze keine eigenständige Semantik (wegen der »offenen« Kanten); sie können nur im Zusammenhang des kompletten Netzes vollständig interpretiert werden.

4.2.1 Struktur der Subnetz-Hierarchie

Da die Subnetze wiederum Subnetze enthalten dürfen, kann durch die Hierarchisierung dem Petrinetz erneut eine Graph-Struktur auferlegt werden. Da dies in voller Allgemeinheit in der Praxis leicht zu sehr unübersichtlichen Modell-Hierarchien führen könnte, wurden für PANDA Einschränkungen vorgenommen, die zu einer streng baumartigen Struktur der Subnetze untereinander und einer eindeutigen hierarchischen Gliederung der Modelle führen:

1. Es existiert stets genau ein ***Wurzelnetz*** an der Spitze der Hierarchie.

2. Jedes Subnetz hat immer maximal ein übergeordnetes Netz und ist als Subnetz immer nur in maximal einem anderen Subnetz enthalten (nur das Wurzelnetz ist in keinem anderen Subnetz enthalten).

3. Die Schnittstellenkanten zwischen den Subnetzen liegen entweder auf einer Hierarchie-Ebene (d.h. sie verbinden Subnetze, die im gleichen übergeordneten Netz enthalten sind), oder aber sie queren genau eine Hierarchiegrenze (d.h. sie führen direkt vom übergeordneten Netz in ein Subnetz bzw. umgekehrt).

Abb. 4.1 zeigt die Möglichkeiten der in PANDA realisierten Strukturierung der Subnetz-Hierarchie.

4.2.2 Handhabung von Subnetzen

Bei der Integration der Subnetze in die Benutzungs-Schnittstelle von PANDA wurde für die Subnetze ein neues grafisches Symbol eingeführt (s. Abb. 4.1). Es wurde darauf geachtet, dem Modellierer weitestmögliche Freiheit beim Aufbau der hierarchischen Modelle zu lassen:

- Für jedes Subnetz wird ein eigenes Editor-Fenster geöffnet. Der Modellierer kann beliebig viele dieser Fenster gleichzeitig anzeigen lassen und so mehrere Hierarchie-Ebenen im Zusammenhang darstellen.

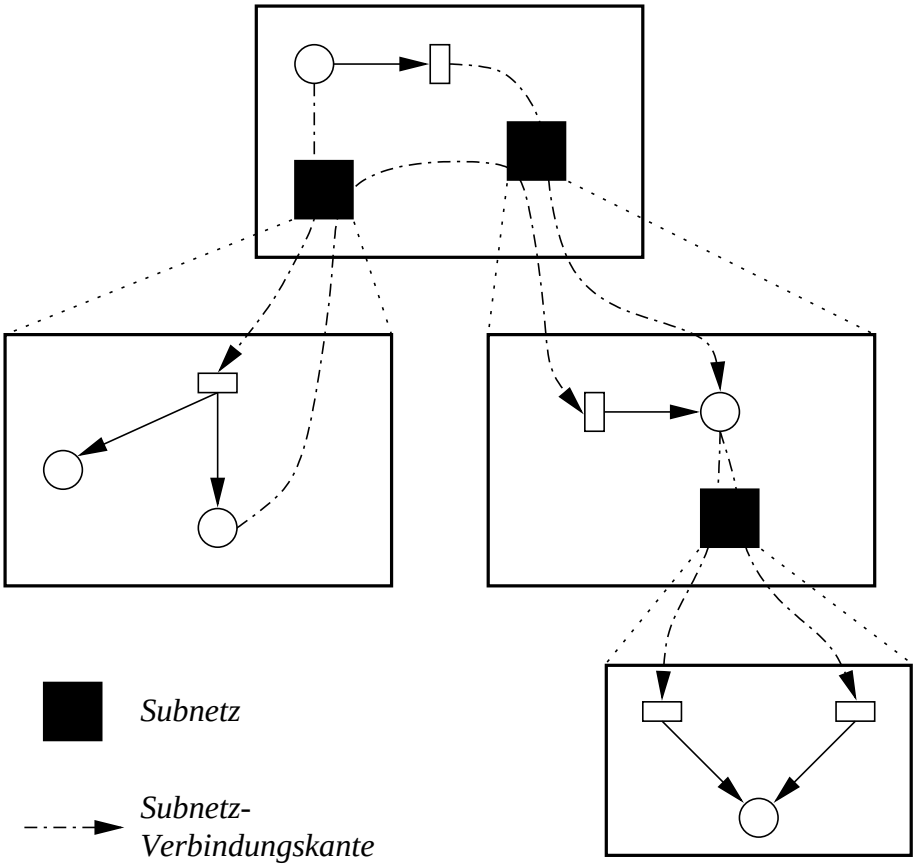


Abbildung 4.1: Mögliche Subnetz-Struktur für PANDA (schematisch)

- Der Subnetz-Strukturbaum kann explizit visualisiert werden.
- Vorhandene Standard-Petrinetz-Elemente können graphisch markiert und dann in einem Subnetz zusammengefaßt (d.h. auf eine niedrigere Hierarchie-Ebene umgelagert) werden. Angelegte Subnetze können auch wieder aufgelöst werden (d.h. ihre Elemente werden in die übergeordnete Hierarchie-Ebene eingegliedert).
- Schnittstellenkanten werden graphisch visuell gesondert gekennzeichnet. Beim Ziehen von neuen Schnittstellenkanten wird der Modellierer durch ein Menü-System unterstützt.
- Für Subnetze werden gesonderte Bearbeitungs-Funktionen zur Verfügung gestellt: Der Modellierer kann die Subnetze komplett (d.h. mit allen in ihnen enthaltenen weiteren Subnetzen) manipulieren (d.h. kopieren, ausschneiden und an anderer Stelle wieder einfügen etc.).

Teil II

Modellierung

Kapitel 5

Abbildung der Testplan-Erstellung auf Petrinetz-Modelle

Ziel dieses Kapitels ist es, zu beschreiben wie das Problem der Erstellung von Plänen zum Test verteilter Systeme in Begriffe der in Kap. 3 eingeführten Rekonfigurations-Petrinetze umgesetzt werden kann; die Modellierung soll dabei so erfolgen, daß die konstruierten Modelle zur Berechnung im Mittel optimaler Test-Anweisungen verwendet werden können.

Ein derartiger Modellierungsprozeß muß naturgemäß in hohem Maß an das betrachtete System angepaßt werden und wird daher hier auch nicht in voller Allgemeinheit methodisch formalisiert oder algorithmisch gefaßt. Deshalb (sowie um das für die Auswertung der vorliegenden Arbeit verwendete Modell detailliert zu beschreiben) wird das für das DSS eingesetzte Verfahren zur Modell-Erstellung sowie die verwendete Notation an Hand einer Fallstudie eingeführt.

5.1 Gegenstand und Aufgabe der Modellierung

Zunächst werden die über die zu modellierenden Systeme gemachten Annahmen sowie das mit den RPN-Modellen zu optimierende Problem näher beschrieben.

Die konkrete Auswahl des hier für die Fallstudie verwendeten Systems erfolgte aus Gründen der Verfügbarkeit einer solchen Anlage am Lehrstuhl Informatik III sowie weil derartige Installationen aus gekoppelten Server-Rechnern stark zunehmend eingesetzt werden; die für die Anwendbarkeit der beschriebenen Vorgehensweise zur Modell-Bildung angenommenen Voraussetzungen (d.h. im Wesentlichen Strukturierbarkeit in abhängige und unabhängige Komponenten sowie die Existenz von auf diese Komponenten bezogenen Tests für wichtige Teile der Funktionalität) sind aber allgemein genug gehalten, um auf einen weiten Bereich von Datenverarbeitungsanlagen zuzutreffen und keine schwerwiegende Einschränkung für die breite Anwendbarkeit des DSS darzustellen.

5.1.1 Zielsystem

Wie schon in 1.1 beschrieben, wird als zu diagnostizierendes System – im Folgenden auch als ***Zielsystem*** bezeichnet – eine verteilte Rechenanlage angenommen: Mehrere (eigenständige) Rechner oder ***Knoten*** (*node*) sind über verschiedene Netzwerkverbindungen zu einem Gesamtsystem gekoppelt; alle Knoten partizipieren an einer Applikation, deren Software die Hardware-Ressourcen der kompletten Installation (CPUs, RAM, Festplatten, Netzwerkbandbreite usw.) verwendet um volle Funktionalität zu erreichen (dies kann auch die Nutzung evtl. vorhandener Redundanz zur Erhöhung von Leistung und/oder Zuverlässigkeit beinhalten). Für solche Systeme hat sich in der industriellen Anwendung der Begriff ***Cluster***[1] durchgesetzt.

Zerlegbarkeit und Hierarchisierung

Es wird davon ausgegangen, daß sich die Zielsysteme sowohl physikalisch als auch logisch in ***Komponenten*** unterteilen lassen (bzw. bei der Entwicklung aus diesen konstruiert werden): Dies können klassische Hardware-Baugruppen wie z.B. Festplatten, Netzwerkschnittstellen oder CPUs, aber auch Software-Moduln wie Betriebssystem-Dienste oder Anwendungsprogramme sein (d.h. es wird angenommen, daß sich auch der durch das Gesamtsystem erbrachte Dienst aus dem Zusammenwirken von – durch Komponenten erbrachte – »Teildiensten« ergibt).

Die Strukturierung wird in der Regel hierarchisch aufgebaut sein, d.h. Komponenten können wiederum in anderen enthalten oder aus anderen zusammengesetzt sein; auch diese Art der Beziehung soll sowohl bei physikalischen Bauteilen (z.B. in Knoten eingebaute Festplatten) als auch logischen Komponenten (d.h. auf Ebene der Software-Funktionalität, z.B. im Fall von in Betriebssystem-Diensten enthaltenen Dateisystem-Diensten) möglich sein.

Komponenten-Abhängigkeiten

Außerdem werden die Komponenten untereinander oftmals ***Abhängigkeiten*** aufweisen, d.h. das System enthält Komponenten, die ihre Funktion nicht erfüllen können falls gewisse andere Komponenten nicht verfügbar sind (z.B. benötigt ein

[1] Ein »Cluster« wird hier im Unterschied zur klassischen parallelen Rechenanlage (bei der die Einzelknoten nur als Teil eines Gesamtsystems eingesetzt werden oder lauffähig sind) dadurch gekennzeichnet, daß die Cluster-Knoten durchaus auch genauso als singuläre (Server-)Systeme eingesetzt werden, d.h. über ein komplettes Betriebssystem und alle notwendigen Hardware-Ressourcen für den Einzelbetrieb verfügen. Der Grad der Koppelung innerhalb des Clusters kann aber genauso eng sein wie bei dedizierten Parallelrechnern (z.B. direkte Speicherkoppelung), und gegenüber der Außenwelt präsentiert sich das Cluster ebenso als *ein* System (*»Single system view«*).

Datenbank-Dienst normalerweise immer einen Festplattenspeicher zur Ablage der Daten, Applikationen greifen auf Betriebssystem-Dienste zu, für deren Funktion ein laufender Betriebssystem-Kern Voraussetzung ist usw.).

Für jede Systemkomponente $\mathbf{K}_n$ kann also auf Komponenten-Ebene die (möglicherweise leere) ***Abhängigkeitsmenge*** $\mathrm{D}^{\mathrm{K}}(\mathbf{K}_n)$ definiert werden als die Menge derjenigen Komponenten $\{\mathbf{K}_1, \ldots, \mathbf{K}_{n-1}\}$, auf deren Verfügbarkeit $\mathbf{K}_n$ zur Erreichung der vollen Funktionalität *direkt* angewiesen ist. $\mathrm{D}^{\mathrm{K}}(\mathbf{K}_n)$ muß nicht abgeschlossen sein in dem Sinn, daß die Komponenten in dieser Menge ihrerseits wieder von anderen abhängen können – die Abhängigkeitsmengen sollen die Struktur des Zielsystems aus einer »lokalen« Sichtweise repräsentieren (um so die modulare Modellierung komplexer Anlagen zu erleichtern).

Durch die Komponenten-Abhängigkeiten und die Hierarchisierung läßt sich dem Gesamtsystem sowohl eine physikalische als auch eine logische Struktur auferlegen – diese in den generierten Testplänen zu reflektieren stellt den Rahmen für die hier vorgenommene Modell-Bildung dar.

5.1.2 Testmoduln

Eine Voraussetzung für die Anwendung des DSS ist die Existenz von Tests für das Zielsystem – es wird angenommen, daß für alle wichtigen Komponenten Tests vorliegen, mit denen die Funktionalität der Komponenten überprüft werden kann. Solche Tests werden im Regelfall aus der Ausführung von Programmen (automatisch gesteuert oder von System-Operateuren gestartet) bestehen, können aber auch physikalische Überprüfungen von Kenngrößen, Bauteilen oder Kabelverbindungen sein.

Die einzelnen Tests können vom Komponenten-Lieferanten oder bei der Systemintegration erstellt werden; für die Zwecke des DSS wird ein Test, der eine bestimmte Funktion des Zielsystems überprüft, als ***Testmodul*** zu dieser Funktionalität bezeichnet (dabei kann auch die Ausführung mehrerer einzelner Tests zu einem Testmodul zusammengefaßt werden, falls sich dies für die Modellierung als günstig erweist).

Erreichbarkeit von Knoten

Die Testmoduln können dabei auf dem Knoten ausgeführt werden zu dem die getestete Komponente gehört, u.U. aber auch auf einem anderen Knoten des Clusters, falls eine geeignete Netzwerkverbindung zum getesteten Knoten besteht (oder falls die Verfügbarkeit einer netzwerkbasierenden Funktionalität geprüft werden soll).

Wird die Durchführung von Tests als System-Funktionalität aufgefaßt, kommt also im Fall der Ausführung eines Moduls auf einem entfernten Knoten zu den in 5.1.1 erwähnten Abhängigkeiten als Voraussetzung für die Verfügbarkeit noch die ***Erreichbarkeit*** des Knotens hinzu, auf dem der Test gestartet werden soll (d.h. die Verfügbarkeit einer geeigneten Netzwerkverbindung zum diesem Knoten).

Kosten der Tests

Wird ein Testmodul ausgeführt, so wird dafür eine bestimmte Zeitspanne benötigt, innerhalb derer durch die Tests gewisse Systemressourcen (Rechenzeit, Haupt- und Plattenspeicherkapazität, Netzwerkbandbreite) sowohl auf dem testenden als auch auf dem getesteten Knoten gebunden sind; diese Zusatzbelastung durch das Testen kann die Güte der vom Zielsystem erbrachten Dienste von kleineren Leistungseinbußen bis hin zum völligen Ausfall (falls etwa laufende Applikationen ganz oder in Teilen für die Durchführung der Tests unterbrochen werden müssen) beeinträchtigen.

Es wird angenommen, daß die (in der Praxis oft schwer exakt zu bestimmenden) Werte sowohl für die erwartete Dauer der Tests als auch für die Kosten-Parameter selbst in der Mehrzahl probabilistisch quantifiziert sind, d.h. z.B. durch Wahrscheinlichkeitsverteilungen angegeben werden oder statistischen Schwankungen unterliegen.

Um die Markov-Eigenschaft (als Grundvoraussetzung für die Anwendbarkeit der in 3.2.3 angeführten Algorithmen zur Strategieoptimierung) für die aufgebauten Modelle zu gewährleisten, müssen hier die Werte für die erwartete Dauer von Tests als exponentiell verteilt angenommen werden. Dies stellt sicherlich eine Einschränkung bei der Modellierung dar (z.B. wäre zumindest die Möglichkeit zur Angabe eines deterministisch bestimmten Zeitintervalls für Tests wünschenswert); inwieweit die Abweichung von in der Realität auftretenden Verteilungen die Brauchbarkeit der Ergebnisse des DSS beeinträchtigt, kann nur in Validierungs-Studien an realen Systemen zu überprüft werden.

Eindeutigkeit des Testausgangs

Nach der Ausführung eines Tests sollen Ergebnisse zurückgeliefert werden; dabei wird im Folgenden der Ausgang eines Tests als »erfolgreich«, »bestanden« oder »positiv« bezeichnet, falls dadurch *kein* Fehler gefunden wird, wogegen die Aufdeckung eines Fehlers als »fehlgeschlagene«, »gescheiterte« oder »negative« Testausführung gewertet wird.

Welchen konkreten Ausgang die Ausführung eines Tests nimmt soll im Voraus wie für die Kosten nur mit einer gewissen Wahrscheinlichkeit vorhersagbar sein; auch die Zuverlässigkeit der Testergebnisse wird als statistische Größe aufgefaßt.

Die Ergebnisse selbst werden aber auf jeden Fall als binär (d.h. stets ausschließlich in einem von zwei möglichen Werten resultierend) und in dem Sinne beurteilbar betrachtet, daß jeder Test nach seiner Ausführung entweder als bestanden oder gescheitert angesehen werden kann – auch das Fehlen jeder Reaktion der getesteten Komponente (*»fail silent«*) wird nach einer gewissen Zeit (*timeout*) als fehlgeschlagener Test gewertet.

5.1.3 Diagnose-Strategie

Bevor die eigentliche Testplan-Erstellung begonnen werden kann, muß zunächst eine grundlegende Strategie zur Systemdiagnose bestimmt werden: Es muß festgelegt werden, auf welche Weise aus den Ergebnissen von System-Tests diagnostische Informationen gewonnen werden soll.

Generell sind hierzu verschiedene Ansätze denkbar[2]; im Rahmen der in der vorliegenden Arbeit durchgeführten Fallstudie wurde vorerst die Optimierung der Ausführung von Testmoduln unter einer festen Diagnose-Strategie untersucht. Dabei wurde von einer *»Bottom-Up«*-Strategie zur Diagnose (bezogen auf die Abhängigkeiten der zu testenden Systemkomponenten) ausgegangen: Getestet werden sollen zuerst diejenigen Komponenten, deren Verfügbarkeit bzw. Erreichbarkeit die Voraussetzung für den Test anderer ist (so soll z.B. die Netzwerkverbindung zu einem Knoten getestet werden, bevor die Funktionalität eines auf diesem Knoten laufenden Netzwerk-Dienstes geprüft wird).

Hintergrund für diese Strategie ist, daß damit ein Fehler nach dem ersten fehlgeschlagenen Test auf der Ebene der als defekt identifizierten Komponente schon wirklich lokalisiert ist – um eine genauere Diagnose zu erreichen, müßten ggf. nur noch die Sub-Komponenten weiter untersucht werden, aus denen die als fehlerhaft identifizierte Komponente aufgebaut ist. Falls alle verfügbaren Tests in die Modellierung einbezogen wurden, kann der Testlauf dann nach dem ersten gefundenen Fehler abgebrochen werden, da der Fehler (bezogen auf die zur Verfügung stehenden Tests) schon mit größtmöglicher Genauigkeit diagnostiziert wurde.

Würde dagegen ohne Berücksichtigung der Abhängigkeiten (bzw. Erreichbarkeit) von Komponenten getestet, müßten auch nach einer gescheiterten Testausführung weitere Tests erfolgen, um einen Fehler in der Abhängigkeitsmenge (bzw. in den Verbindungspfaden zu) der – in diesem Fall nur potentiell fehlerhaften – Komponente zu lokalisieren.

[2] Das DSS könnte natürlich auch zur Bestimmung einer optimalen Diagnose-Strategie verwendet werden – hierzu könnte z.B. das im Folgenden vorgestellte, für die Fallstudie aufgebaute RPN als ein Subnetz in einem größeren RPN integriert werden, bei dem die optionale Ausführung mehrerer Diagnose-Strategien modelliert wird. In der vorliegenden Arbeit wurde die Diagnose-Strategie jedoch hauptsächlich dazu verwendet, die Größe des Zustandsraums der generierten Modelle zu verringern (d.h. zwecks Vereinfachung der Analyse eine Vorauswahl aus der vollen kombinatorischen Menge der möglichen Testpläne zu treffen).

5.1.4 Abhängigkeiten unter Testmoduln

Die Bottom-Up-Diagnose-Strategie wird bei den für das DSS aufgebauten Modellen dadurch umgesetzt, daß für den Start eines Testmoduls Bedingungen erfüllt sein müssen, und zwar in Form der fehlerfrei abgeschlossenen Ausführung anderer Testmoduln. Hierzu werden die in 5.1.1 eingeführten Abhängigkeiten bzw. die Erreichbarkeit von Komponenten (bezogen auf das Testen dieser Komponenten) als Voraussetzung für die Ausführung der Testmoduln aufgefaßt – sobald eine von einem Modul zu testende Komponente von anderen Komponenten abhängig ist, wird diese Abhängigkeit auf das Testmodul übertragen.

Die Abhängigkeiten unter den Komponenten des Zielsystems führen so zu Abhängigkeiten unter den Testmoduln zu diesen Komponenten; da sich die Testmoduln direkt auf die Funktionalität der Komponenten des Zielsystems beziehen, können die Testmoduln-Abhängigkeiten auch aus den Abhängigkeiten der Systemkomponenten abgeleitet werden.

Die ***Testbarkeit*** einer Komponente **K** bzw. die ***Ausführbarkeit*** des Testmoduls T zur Funktionalität von **K** ist für die Zwecke des DSS damit gegeben falls:

1. alle Komponenten aus der Abhängigkeitsmenge der zu testenden Komponente verfügbar sind (und dies mit den Testmoduln zu den Komponenten aus $\mathrm{D}^{\mathrm{K}}(\mathbf{K})$ überprüft worden ist) sowie

2. die zu testende Komponente erreichbar ist, d.h. alle Komponenten (mit den entsprechenden Testmoduln überprüft) verfügbar sind, die für die Verbindung zu **K** benötigt werden.

Die Ausführung eines Testmoduls wird bei der Erstellung der Testpläne also erst dann als Option zugelassen, wenn diese Vorbedingungen erfüllt sind: Ist ein Testmodul ausführbar, so können sich alle dadurch potentiell lokalisierten Fehler ausschließlich in der getesteten Komponente befinden.

Der Abhängigkeitsmenge $\mathrm{D}^{\mathrm{K}}(\mathbf{K})$ auf der Ebene der Systemkomponenten entspricht damit in Bezug auf die Testmoduln für jedes Modul T eine ***Testmodul-Abhängigkeitsmenge*** $\mathrm{D}^{\mathrm{T}}(T)$, bestehend aus denjenigen Moduln deren erfolgreiche Ausführung die Ausführbarkeit von T ermöglicht. Im Unterschied zur Abhängigkeitsmenge der Komponenten kommt hier jedoch zusätzlich das Kriterium der Erreichbarkeit hinzu; außerdem müssen die Testmodul-Abhängigkeiten in dem Sinn »erfüllt« sein, daß alle in $\mathrm{D}^{\mathrm{T}}(T)$ enthaltenen Moduln schon erfolgreich ausgeführt worden sein müssen, bevor der Start von T möglich wird.

Zur Darstellung der Testmodul-Abhängigkeiten wird folgende Notation verwendet:

- $T_1 \vee \ldots \vee T_k \;\rightsquigarrow\; T_{k+1} \mid \ldots \mid T_n$ bedeutet, daß *mindestens eines* der Testmoduln T_1 bis T_k erfolgreich ausgeführt worden sein muß, bevor eines der Moduln T_{k+1} bis T_n gestartet werden darf.

- $T_1 \wedge \ldots \wedge T_k \quad \rightsquigarrow \quad T_{k+1} \mid \ldots \mid T_n$ bedeutet daß *alle* Testmoduln T_1 bis T_k erfolgreich ausgeführt worden sein müssen, bevor eines der Moduln T_{k+1} bis T_n gestartet werden darf.

Dabei werden die Testmodul-Abhängigkeiten als *transitiv* aufgefaßt, d.h. $T_1 \rightsquigarrow T_2$ und $T_2 \rightsquigarrow T_3$ impliziert $T_1 \rightsquigarrow T_3$.

Diese Transitivität reflektiert die in realen Systemen auftretenden Ketten von Abhängigkeiten unter Komponenten; außerdem wird dadurch ein schrittweiser, modularer Aufbau der Petrinetz-Modelle aus der Kenntnis der »lokalen« Testmodul-Abhängigkeiten (d.h. ausschließlich aus Informationen von direkten Abhängigkeiten zwischen einzelnen Komponenten) ermöglicht, da weniger Abhängigkeiten explizit modelliert werden müssen.

5.1.5 Testpläne

Aufgabe des DSS ist es, ***Testpläne*** für die Zielsysteme aufzustellen – unter dem bestehenden Verdacht, daß (mindestens) eine der Komponenten des Systems von der spezifizierten Funktionalität abweicht, sollen Anweisungen erstellt werden, in welcher Reihenfolge welche Testmoduln ausgeführt werden sollen, um defekte Komponenten zu lokalisieren.

Die Fehler werden als permanent angenommen, und die Testpläne werden zur Lokalisierung von einem Fehler pro Ausführung aufgestellt[3].

Ein Testplan soll auf der Menge der zur Auswahl stehenden Testmoduln eine Auswahl bzw. zeitliche Reihenfolge definieren, in der verschiedene Testmoduln gestartet werden müssen um – im Mittel – optimale Testläufe zu erhalten. Die Definition von »optimal« kann dabei verschiedene Kriterien reflektieren (die in den Bewertungsmaßen der Testmoduln abgebildet werden, s. 6.6):

- Möglichst vollständige Informationen über das getestete System zu erhalten.
- Brauchbare Testergebnisse in möglichst kurzer Zeit zu erhalten.
- Testläufe mit minimaler Belastung des Zielsystems durchzuführen (um die Güte der erbrachten Dienste nicht abfallen zu lassen).

[3] Diese Einschränkungen dienen hier nur zur Vereinfachung der aufgebauten Modelle bzw. ihrer Auswertung – so reicht es unter den genannten Bedingungen aus, jedes Testmodul einmal auszuführen um komplette diagnostische Informationen zu gewinnen, und die Testausführung kann nach dem ersten lokalisierten Fehler beendet werden. Es lassen sich aber natürlich auch RPN-Modelle aufbauen, die die wiederholte Ausführung von Tests oder den Start weiterer Testmoduln nach der Lokalisierung der ersten defekten Komponente erlauben.

5.2 Petrinetz-Modellierung und Fallstudie

Nach der Bestimmung von Struktur und Abhängigkeiten des Zielsystems sowie der Identifizierung der zur Verfügung stehenden Testmoduln wird der eigentliche Aufbau der RPN-Modelle für das DSS in mehreren Schritten durchgeführt:

1. Die Abhängigkeiten der Testmoduln untereinander werden in ein zeitloses Petrinetz abgebildet, das sog. ***Abhängigkeitenmodell***. Die Testmoduln werden darin (vorerst) durch zeitlose Transitionen repräsentiert. Die für die Ausführung eines Testmoduls notwendigen Bedingungen – d.h. die erfolgreiche Ausführung der anderen Tests, die als Voraussetzung schon abgeschlossen sein müssen (entsprechend den Systemkomponenten, die zur Ausführung des Tests betriebsbereit sein müssen) – werden als Zustände an den Eingangskanten dieser Transitionen modelliert.

2. Danach werden die Transitionen zu kantenberandeten Subnetzen (***Testsubnetzen***) verfeinert (wie in Kap. 4 definiert): Jedes dieser Subnetze modelliert die Ausführung eines Testmoduls und beinhaltet Stellen und Transitionen, die den Ablauf der mit dem Modul assoziierten Tests im Petrinetz repräsentieren. Die Eingangs-Transitionen der Testsubnetze werden mit Rekonfigurationstransitionen besetzt – damit wird die Option zur Ausführung des jeweiligen Testmoduls modelliert bzw. es werden die Entscheidungsalternativen für die Markov-Entscheidungs-Optimierung markiert.

3. Das so entstandene Rekonfigurations-Petrinetz wird parametrisiert: Die Elemente der Testsubnetze werden mit Schaltraten und Schaltwahrscheinlichkeiten belegt, die die durchschnittliche Ausführungsdauer sowie die Wahrscheinlichkeiten für erfolgreiche bzw. gescheiterte Ausführung des Testmoduls repräsentieren.

4. Danach werden für jedes Testmodul aus dem Ressourcenverbrauch bzw. dem möglichen Informationsgewinn, der mit der Ausführung des Moduls assoziiert ist, Bewertungsmaße (mit Bezug auf die Zustände und Transitionen des dem Modul zugeordneten Testsubnetzes) bestimmt; mit diesen Maßen werden die Zielgrößen für die Optimierungsphase festgelegt.

5.2.1 Fallstudie

Der Aufbau eines Modells für das DSS soll an einer Fallstudie demonstriert werden – zum Vorbild wurde hierzu ein am Lehrstuhl Informatik III installiertes Cluster aus Rechnern des Typs Digital 2100/500 MP verwendet. Dabei entspricht die Hardware und Betriebssystem-Software des modellierten Systems dem realen

Cluster; bei der Datenbank-Applikations-Software wurden dagegen einige Komponenten im Modell durch andere Programme mit reduzierter Funktionalität ersetzt, um den Installationsaufwand und die Kosten (Lizenzgebühren) für verteilte Datenbank-Systeme einzusparen.

Hardware-Architektur (Abb. 5.1(a))

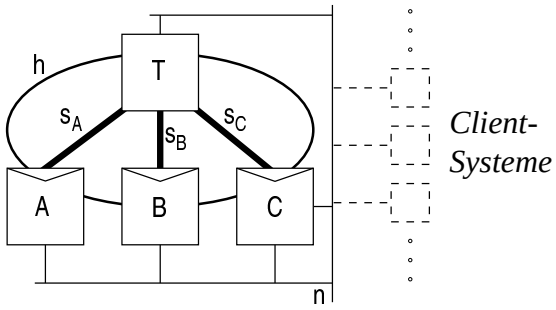
- Das Cluster besteht aus den Knoten **T**, **A**, **B** und **C**, die mittels der Netzwerke **n**, **h** und **s** gekoppelt sind.
- Dabei ist **h** ein Hochgeschwindigkeitsnetz, über das die Knoten des Clusters Daten für eine verteilte Anwendung mit hohem Durchsatz austauschen können.
- **n** ist ein Standard-LAN, mit dem die Außenanbindung des Clusters für interaktiven Zugriff auf die verteilte Applikation, allgemeine Ein- und Ausgabe-Operationen sowie Verwaltung des Clusters realisiert ist.
- **s** besteht aus Punkt-zu-Punkt-Verbindungen niedriger Bandbreite von **T** zu **A**, **C** und **B** mit speziellen Schnittstellen, die es erlauben, von **T** aus erweiterte Kontroll- und Test-Maßnahmen durchzuführen; dies beinhaltet die Möglichkeit, Reboots und ausführliche Offline-Hardware-Tests (über die ROM-Monitoren der getesteten Rechner) auszulösen. **T** wird daher als im Folgenden als ***Kontroll-*** und die anderen Knoten als ***überwachte Rechner*** bezeichnet.

Software-Architektur (Abb. 5.1(b))

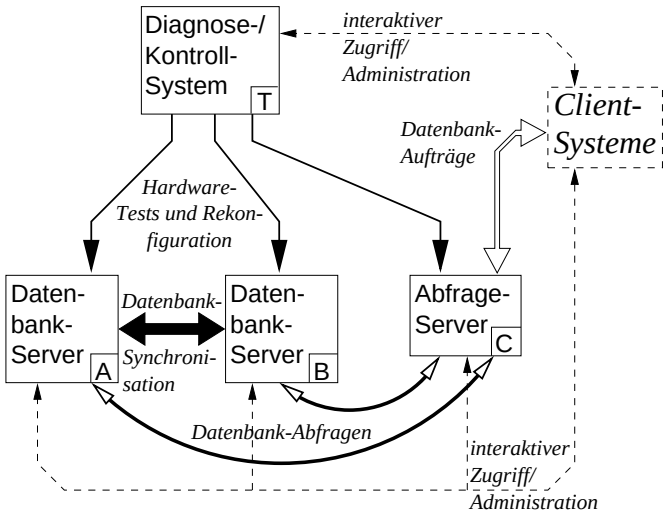
- Auf den Knoten **A** und **B** wird eine verteilte Datenbank betrieben; jeder Knoten hält dabei eine Kopie der gesamten Datenbestände, und bei schreibenden Zugriffen auf die Kopie eines Knotens wird die Kopie des jeweils anderen Knotens synchronisiert. Falls beide Knoten funktionsfähig sind, sollen die Zugriffe auf die Daten (zur Leistungssteigerung des Gesamtsystems) parallel von **A** und **B** aus möglich sein. Fällt einer der Knoten aus, erfolgen die Zugriffe nur durch den intakten Rechner (zur Erhöhung der System-Verfügbarkeit).
- Knoten **C** dient als Vermittlungsstelle des Systems nach außen: Aufträge an das System werden nur an **C** geschickt und dort (je nach Last und Verfügbarkeit) als Datenbank-Anfrage an einen der Datenbank-Server weitergeleitet (**C** fungiert als sog. Transaktions-Manager, s. [Ave98b]). Um auf die Datenbank mit einem standardisierten Protokoll zugreifen zu können, wird auf

C ein HTTP-Server betrieben, der die Aufträge entgegennimmt und auch die Ergebnisse der Anfragen in einem mit den üblichen Web-Browsern darstellbaren Format präsentiert.

- Die Kommunikation mit der Außenwelt zur Ausführung der Aufträge an das System wird von **C** über das Netz **n** abgewickelt; der interne Datenaustausch zwischen **C**, **A** und **B** für die Datenbank-Abfragen sowie zwischen **A** und **B** zur Synchronisation und Spiegelung der Datenbestände wird über **h** durchgeführt. Interaktiver und administrativer Zugriff auf das System von außen findet über **n** statt, während die Verbindungen über **s** zur Durchführung von Rekonfigurations-Maßnahmen bzw. bei der Ausführung einiger Testmoduln eingesetzt werden.



(a) Hardware-Struktur



(b) Software-Struktur

Abbildung 5.1: System-Architektur des Fallstudien-Clusters (schematisch)

Testbare Komponenten

Die Existenz folgender Tests zur Überprüfung der Funktionalität von Systemkomponenten wird angenommen (die in Klammern angegebenen Bezeichner der Testmoduln entsprechen den Namen der zugeordneten Transitionen im späteren Abhängigkeitenmodell bzw. den Namen der die Testmodul-Ausführung modellierenden Subnetzen im vollständigen RPN):

- Interne Funktionalität von **T** (Transition T_i) – dies beinhaltet die Überprüfung von Hardware (CPU, Festplatten, RAM etc.) sowie von Betriebssystem-Software (Kern, System- und Netzwerk-Dienste) und der Applikationssoftware zur Testausführung und -steuerung.
- Funktionalität der Hardware für jeden überwachten Knoten (Transitionen A_{HW}, B_{HW}, C_{HW}).
- Funktionalität der Betriebssystem-Software für jeden überwachten Knoten (Transitionen A_{OS}, B_{OS}, C_{OS}).
- Verfügbarkeit der Netzwerkschnittstelle für jeden überwachten Knoten und jede Netzwerk-Anbindung dieses Knotens (Transitionen A_n, B_n, C_n, A_h, B_h, C_h, A_s, B_s, C_s).
- Verfügbarkeit der Netzwerkverbindungen (passive Bauteile) **h** und **n** sowie $\mathbf{s}_{\mathrm{A}}$, $\mathbf{s}_{\mathrm{B}}$ und $\mathbf{s}_{\mathrm{C}}$ (Transitionen h, n, s_{A}, s_{B}, s_{C}).
- Funktionalität der Transaktions-Monitor- und HTTP-Server-Software auf **C** (Transition C_{GQ}).
- Funktionalität der Datenbank-Server bzw. Konsistenz der Datenbanken auf **A** und **B** (Transitionen A_{DB} und B_{DB} bzw. A_{CONS} und B_{CONS}).
- Funktionalität des Datenbank-Synchronisations und -Spiegelungs-Dienstes zwischen **A** und **B** (Transition AB_{UPD}).
- Verfügbarkeit und Funktionalität des Gesamtsystems aus Anwendersicht (Transition ABC_{TEST}).

Abhängigkeiten

- Voraussetzung für die Tests der passiven Netzwerke-Elemente ist die Funktionalität von Hardware sowie Betriebssystem von **T** (d.h. Testmodul T_i muß auf jeden Fall erfolgreich ausgeführt worden sein):

$$T_i \;\rightsquigarrow\; n \mid h \mid s_{\mathrm{A}} \mid s_{\mathrm{B}} \mid s_{\mathrm{C}}$$

- Bevor die Netzwerkschnittstellen von **A**, **B** oder **C** getestet werden können, müssen die entsprechenden Netzwerkverbindungen in Ordnung sein:

$$n \rightsquigarrow A_n \mid B_n \mid C_n, \quad h \rightsquigarrow A_h \mid B_h \mid C_h$$
$$s_{\mathrm{A}} \rightsquigarrow A_s, \quad s_{\mathrm{B}} \rightsquigarrow B_s, \quad s_{\mathrm{C}} \rightsquigarrow C_s$$

- Um die Funktionalität der Betriebssysteme von **A**, **B** oder **C** testen zu können, müssen mindestens eine Netzwerkverbindung sowie die daran angeschlossene Schnittstelle des jeweiligen zu testenden Knotens funktionieren:

$$A_h \vee A_n \vee A_s \rightsquigarrow A_{\mathrm{OS}}, \quad B_h \vee B_n \vee B_s \rightsquigarrow B_{\mathrm{OS}}, \quad C_h \vee C_n \vee C_s \rightsquigarrow C_{\mathrm{OS}}$$

- Um erweiterte Offline-Hardware-Tests auf **A**, **B** oder **C** durchführen zu können, müssen die Punkt-zu-Punkt-Verbindung zum zu testenden Knoten sowie die entsprechende Schnittstelle verfügbar sein:

$$A_s \rightsquigarrow A_{\mathrm{HW}}, \quad B_s \rightsquigarrow B_{\mathrm{HW}}, \quad C_s \rightsquigarrow C_{\mathrm{HW}}$$

- Bevor die Funktionalität der HTTP- und Transaktions-Monitor-Dienste auf **C** überprüft werden kann, müssen die Netzwerkschnittstelle zu Netz **n** sowie Hardware und Betriebssystem-Dienste dieses Knotens verfügbar sein:

$$C_n \wedge C_{\mathrm{HW}} \wedge C_{\mathrm{OS}} \rightsquigarrow C_{\mathrm{GQ}}$$

- Voraussetzung für den Test der Datenbank-Dienste auf **A** oder **B** ist die Verfügbarkeit von Hardware, Netzwerkschnittstelle zu **h** sowie Betriebssystem des zu testenden Knotens:

$$A_h \wedge A_{\mathrm{HW}} \wedge A_{\mathrm{HW}} \rightsquigarrow A_{\mathrm{DB}}, \quad B_h \wedge B_{\mathrm{HW}} \wedge B_{\mathrm{HW}} \rightsquigarrow B_{\mathrm{DB}}$$

- Falls die Datenbank-Dienste auf **A** bzw. **B** verfügbar sind, kann die Konsistenz der Datenbestands-Kopien auf dem jeweiligen Knoten überprüft werden:

$$A_{\mathrm{DB}} \rightsquigarrow A_{\mathrm{CONS}}, \quad B_{\mathrm{DB}} \rightsquigarrow B_{\mathrm{CONS}}$$

- Die Konsistenz der beiden lokalen Kopien der Datenbank auf **A** und **B** ist Voraussetzung für den Test des Spiegelungs-Dienstes zwischen den beiden Knoten:

$$A_{\mathrm{CONS}} \wedge B_{\mathrm{CONS}} \rightsquigarrow AB_{\mathrm{UPD}}$$

- Schließlich kann die Funktionalität des Gesamtsystems aus Anwendersicht überprüft werden, wenn sowohl die Transaktions-Monitor- und HTTP-Dienste auf **C** als auch die Synchronisation der Datenbank-Kopien zwischen **A** und **B** verfügbar sind:

$$C_{\mathrm{GQ}} \wedge AB_{\mathrm{UPD}} \rightsquigarrow ABC_{\mathrm{TEST}}$$

5.2.2 Modellierung von Abhängigkeiten

Der erste Schritt beim Aufbau eines RPN-Modells zur Testplan-Generierung unter der in 5.1.3 vorgestellten Diagnose-Strategie ist die Abbildung der Komponenten- bzw. Testmodul-Abhängigkeiten in ein Petrinetz. Da hier zunächst nur die qualitativen Vorbedingungen und Konsequenzen der Ausführung von Testmoduln modelliert werden sollen, wird ein zeitloses Petrinetz erstellt:

- Jedem Testmodul wird eine zeitlose Transition als ***Testmodul-Transition*** zugeordnet – das Feuern dieser Transition repräsentiert die Ausführung des Moduls im Abhängigkeitenmodell.

- Jede Testmodul-Transition wird mit zwei Ausgangsstellen verbunden – der Transport von Token in diese Stellen signalisiert den Erfolg bzw. das Scheitern der Ausführung des zugehörigen Testmoduls.

  Es soll an dieser Stelle betont werden, daß die eigentliche Fehlerdiagnose nicht der Zweck der hier vorgenommenen Modell-Bildung ist. Deshalb wird auch nicht versucht, durch den Aufbau des Petrinetzes das Ziehen von diagnostischen Schlußfolgerungen aus den Testergebnissen zu ermöglichen: Die die gescheiterte Testausführung modellierenden Ausgangsstellen aller Testmodul-Transitionen werden daher zu einer Stelle *Failed* gefaltet, um die Komplexität des Modells zu reduzieren.

- Die Eingangsstellen der einem Modul T zugeordneten Transition modellieren die Vorbedingungen für dessen Ausführung – dabei wird für jedes Element aus $\mathrm{D}^{\mathrm{T}}(T)$ eine solche Stelle zugewiesen.

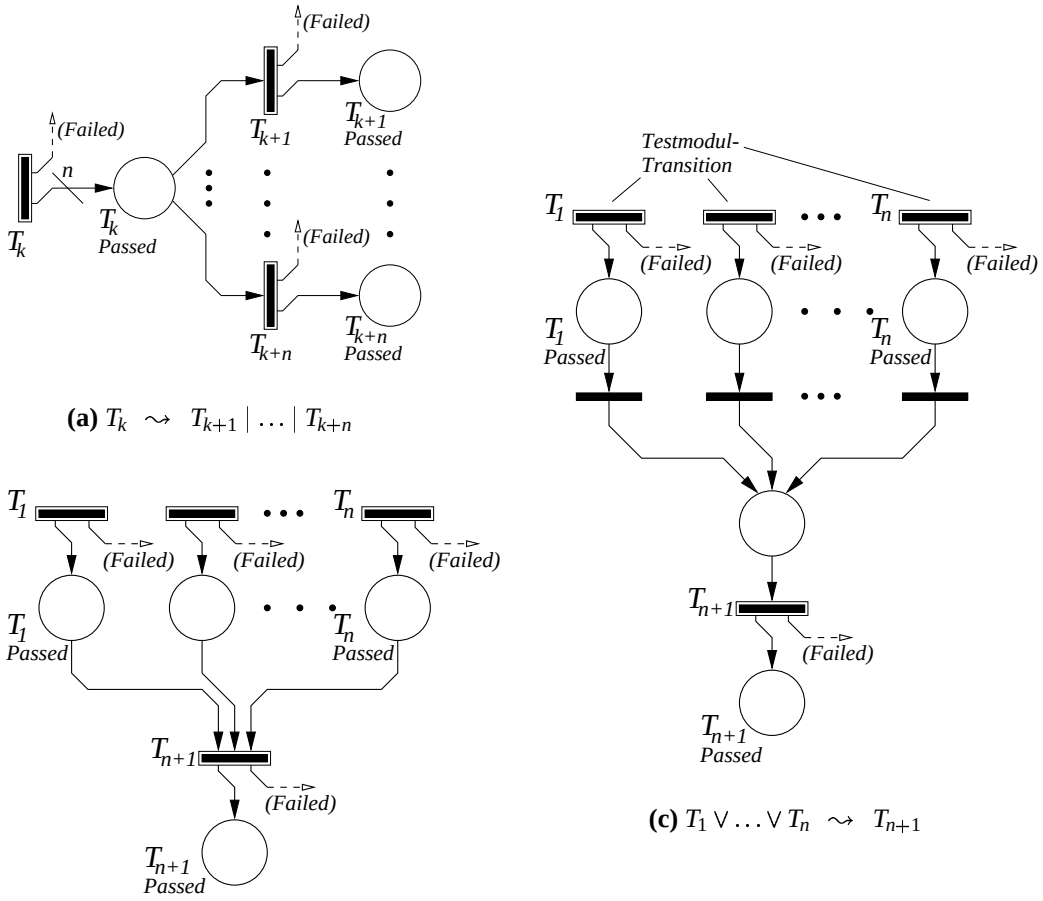
  Gemäß den Feuerregeln für Petrinetze kann damit die Ausführung des mit der Transition assoziierten Moduls (entsprechend dem Schalten der zugehörigen Transition) im Modell erst erfolgen, nachdem alle Moduln in $\mathrm{D}^{\mathrm{T}}(T)$ erfolgreich ausgeführt wurden (d.h. deren entsprechende Transitionen geschaltet haben).

- Durch die Verschaltung der die erfolgreiche Testausführung repräsentierenden Ausgangsstelle der Testmodul-Transition eines Moduls T_i mit einer der Eingangsstellen der Testmodul-Transition eines von T_i abhängigen Moduls T_j kann nun diese Abhängigkeit im Petrinetz umgesetzt werden (dabei können bei Bedarf auch noch weitere Transitionen und Stellen in das Modell eingefügt werden, um ggf. komplexere Abhängigkeiten zu modellieren).

Die Überführung der Abhängigkeiten von Testmoduln in die Petrinetz-Semantik ist für die verschiedenen möglichen Beziehungen von Testmoduln in Abb. 5.2 dargestellt[4], mit Bezug auf die in 5.1.4 eingeführte Notation.

[4] Die Stelle *Failed* ist hier aus Gründen der Übersichtlichkeit nicht abgebildet.

Das in Abb. 5.3 gezeigte zeitlose Petrinetz bildet schließlich (unter Anwendung der oben beschriebenen Vorgehensweise) als Abhängigkeitenmodell die in 5.2.1 aufgeführten Testmodul-Abhängigkeiten in ein Petrinetz ab.



(a) $T_k \rightsquigarrow T_{k+1} \mid \ldots \mid T_{k+n}$

(b) $T_1 \wedge \ldots \wedge T_n \rightsquigarrow T_{n+1}$

(c) $T_1 \vee \ldots \vee T_n \rightsquigarrow T_{n+1}$

Abbildung 5.2: Petrinetz-Ausschnitte für die Modellierung von Abhängigkeiten unter Testmoduln[4]

5.2.3 Modellierung der Testmodul-Ausführung

Im Abhängigkeitenmodell sind noch keinerlei Alternativen bei der Ausführung von Testmoduln modelliert – alle Testmodul-Transitionen feuern (wie für zeitlose Transitionen definiert) sofort, sobald die Bedingungen zur Feuerbereitschaft er-

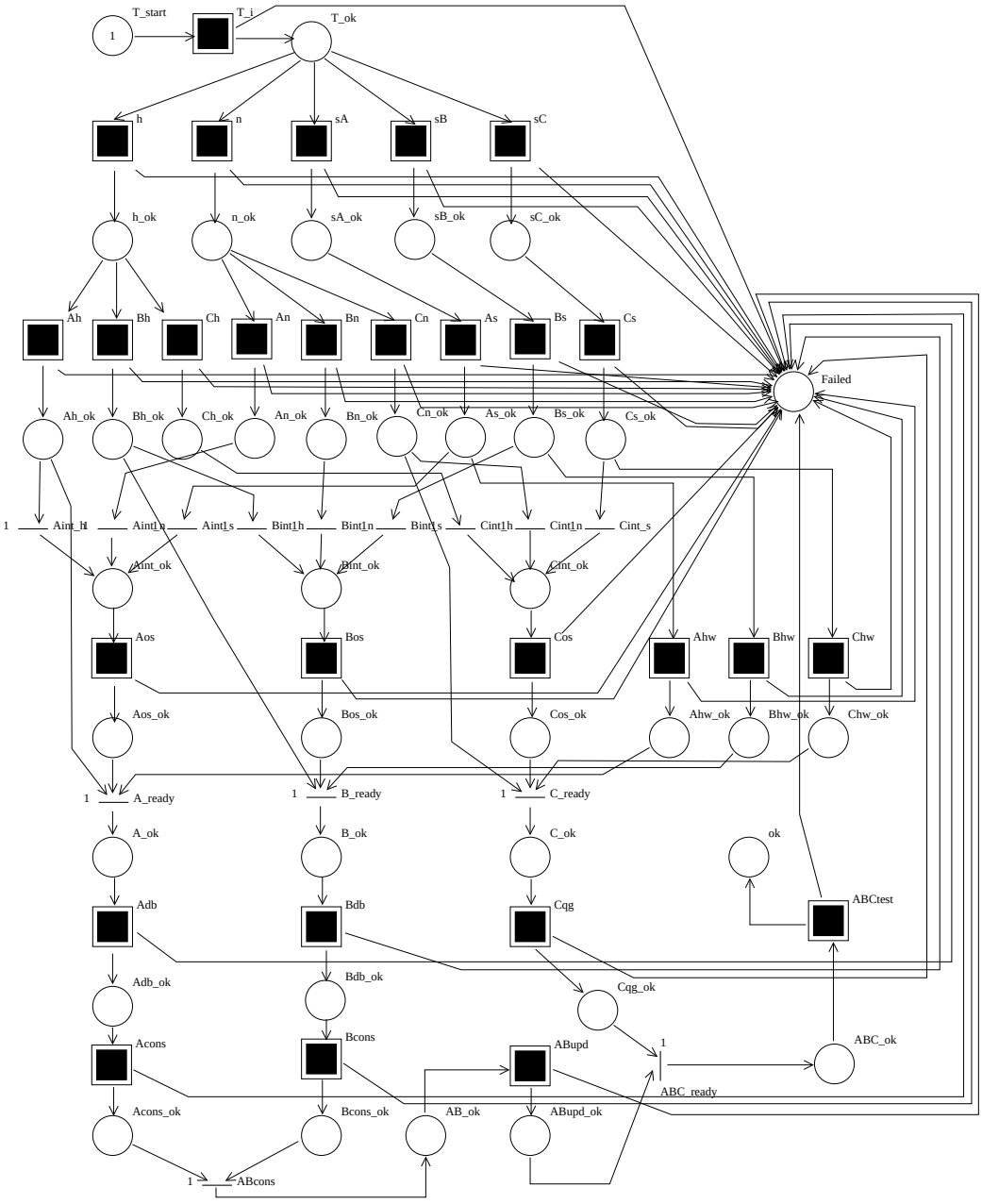


Abbildung 5.3: Abhängigkeitenmodell für Fallstudien-System aus 5.2.1 (Netzplan eines mit PANDA erstellten RPN)[5]

füllt sind. Das so aufgebaute Petrinetz berücksichtigt auch die zeitlichen Aspekte der Durchführung von Tests noch nicht und würde das wiederholte Feuern von

[5]Da es sich um den Netzplan eines vollständigen Modells zur Testplan-Generierung handelt, sind hier an Stelle der Testmodul-Transitionen aus 5.2.2 schon die graphischen Symbole für Subnetze (i.e. Testsubnetze,) eingezeichnet (Kantenmultiplizitäten sind nicht angegeben).

Testmodul-Transitionen erlauben, falls sie nach dem ersten Feuern nochmals feuerbereit wären.

Dabei entstehen immer dann Entscheidungs-Möglichkeiten – d.h. Optionen auf die Auswahl eines auszuführenden Testmoduls – wenn die Vorbedingungen für die Ausführung von mehreren Testmoduln gleichzeitig erfüllt sind. Im Abhängigkeitenmodell entspricht dem der Konflikt zwischen mehreren feuerbereiten Testmodul-Transitionen: Sobald mehrere dieser Transitionen gleichzeitig schalten könnten, kann eine Entscheidung getroffen werden welche Transition feuern (d.h. welches Testmodul ausgeführt werden) soll.

Das Abhängigkeitenmodell kann also auch als eine *Kombination von verschiedenen Modellvarianten* (vgl. S. 38) im Sinne der Entscheidungsmodellierung betrachtet werden: Jede der Varianten beschreibt genau einen möglichen Testplan, indem sie für alle Auswahlmöglichkeiten bei der Ausführung von Testmoduln eine spezifische Entscheidungsaktion festlegt. Diese Entscheidungen sollen mit Hilfe der RPN-Modellierung optimiert, also eine Folge von solchen Entscheidungen vom DSS als (im Mittel) optimaler Testplan errechnet werden.

Testsubnetze

Um die detailliertere Modellierung der Testmodul-Ausführung zu ermöglichen, werden im weiteren Aufbau der Modelle die Testmodul-Transitionen in kantenberandete Subnetze (wie in Kap. 4 beschrieben) verfeinert; diese Testsubnetze haben zwar bei den aufgebauten Modellen für alle Testmoduln eine einheitliche Petrinetz-Struktur, ihre Elemente werden aber natürlich modulabhängig unterschiedlich parametrisiert sowie mit unterschiedlichen Bewertungsmaßen versehen.

Die Einsprungtransitionen in die Testsubnetze werden als »Entscheidungs-Punkte« mit den in Kap. 3 für die Erweiterung von SRN zur Strategie-Modellierung beschriebenen Rekonfigurationstransitionen belegt: Auf diese Weise kann die Aktivierung eines solchen Subnetzes nur erfolgen, falls dadurch der Erwartungswert für das Bewertungsmaß des Gesamtmodells optimiert wird. Weiter wird die erfolgreiche sowie die gescheiterte Ausführung des modellierten Testmoduls mit je einem Zweig im Subnetz umgesetzt – dazu werden je eine zeitlose Transition (die in direkter Konkurrenz zu der im anderen Zweig steht und so die Wahrscheinlichkeit für den jeweiligen Ausgang des Tests repräsentiert) und eine Transition mit exponentiell verteilter Schaltverzögerung (die die für die Testausführung benötigte Zeitdauer repräsentiert) eingeführt. Dies resultiert für ein Testsubnetz T in der in Abb. 5.4 gezeigten Struktur[6]:

[6] Bezüglich des Erwartungswertes von Bewertungsfunktionen stochastisch äquivalente Modelle könnten natürlich auch mit Testsubnetzen aufgebaut werden, die weniger Stellen und Transitionen enthalten. Die hier gezeigte Struktur ist gewählt worden, da sie sich einfacher parametrisieren

- Die Schnittstelle von außen ist die Eingangskante zur Rekonfigurationstransition T_c (*»test control«*); Stelle T_S (*»test started«*) stellt zusammen mit der inhibitorischen Kante sicher, daß jedes Testsubnetz maximal einmal durchlaufen werden kann (entsprechend der einmaligen Ausführung der Moduln im Testplan). Die Existenz eines Token in Stelle T_S ist auch Indikator dafür, daß mit der Ausführung des entsprechenden Testmoduls begonnen wurde.

- Falls T_c gefeuert hat, befindet sich ein Token in Stelle T_B (*»test begin«*). Über die Feuerwahrscheinlichkeiten der nachgeschalteten zeitlosen Transitionen T_p (*»passed test«*) bzw. T_f (*»failed test«*) wird die Wahrscheinlichkeit modelliert, daß das Testmodul erfolgreich ausgeführt wird (T_p schaltet) bzw. daß durch die Ausführung ein Fehler diagnostiziert wird (T_f schaltet): Es wird ein Token entweder in die Stelle T_{P_r} (*»passed test running«*) oder in die Stelle T_{F_r} (*»failed test running«*) transportiert.

- Die mittlere Zeitdauer für die Testausführung wird mit den Schaltraten der exponentiell verzögerten Transitionen T_{p_o} (*»passed test over«*) für erfolgreiche und T_{f_o} (*»failed test over«*) für fehlgeschlagene Testausführung modelliert.

- Nach dem Schalten von T_{p_o} oder T_{f_o} wird ein Token (aus dem Subnetz heraus) entweder in die Stelle *Failed* oder in die erfolgreichen Testausgang signalisierende Ausgangsstelle T_{Passed} des Testsubnetzes transportiert – die Ausführung des Moduls ist abgeschlossen. Im Subnetz wird als Indikator für die beendete Ausführung ein Token in die Stelle T_D (*»test done«*) transportiert.

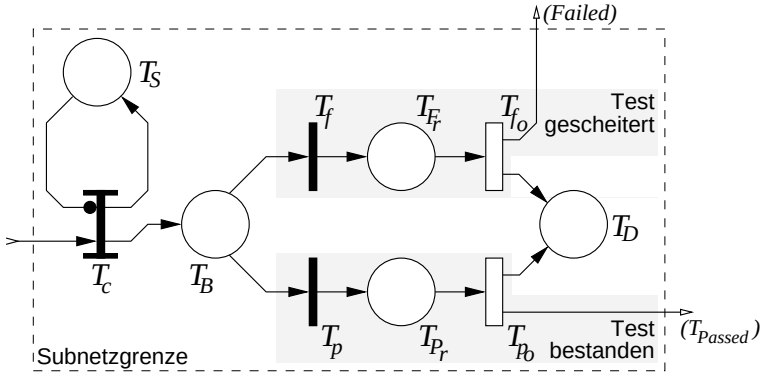


Abbildung 5.4: Testsubnetz mit Rekonfigurationstransition

läßt (d.h. Testmodul-Parameter wie z.B. die Wahrscheinlichkeit für die Testergebnisse lassen sich direkt als RPN-Parameter auf Testsubnetz-Elemente übertragen).

5.2.4 Zuordnung von Bewertungsmaßen

Um die bisher aufgebauten Modelle den Strategieoptimierungs-Algorithmen der RPN-Analyse zugänglich zu machen, müssen noch Bewertungsmaße eingeführt werden. Dazu werden zunächst die Testmoduln numerisch klassifiziert, um sie vergleichend bewerten zu können; ein entsprechendes Verfahren wird in Kap. 6 vorgestellt. An dieser Stelle wird die Umsetzung der Klassifizierung in das RPN-Modell beschrieben.

Wegen der Modellierung der Ausführung von Testmoduln durch die Testsubnetze im Petrinetz bedeutet dies die Bildung von RPN-Bewertungsmaßen aus Bewertungsparametern, die Aspekte der Testmoduln beschreiben. Auf RPN-Ebene wird daher für jedes Testsubnetz T ein Zustandsmaß-Term $\boldsymbol{\rho}_T$ sowie ein Impulsmaß-Term $\boldsymbol{r}_T$ definiert – diese repräsentieren den zeitabhängigen bzw. zeitunabhängigen Aspekt von Kosten und Nutzen der potentiellen Ausführung des modellierten Testmoduls im RPN (wie üblich werden hier Kosten durch negative und Nutzen durch positive Werte repräsentiert).

Mit den Bewertungsmaß-Termen werden drei wesentliche Aspekte der Ausführung von Testmoduln modelliert:

Zeit- und Ressourcenverbrauch

Die Ausführung von Tests ist mit ***Kosten*** verbunden – während des Ablaufs werden Systemressourcen (CPU-Zeit, Hauptspeicher- und Plattenplatz, Netzwerkbandbreite usw.) verbraucht um die diagnostische Ergebnisse zu erzielen.

Der so verursachte Ressourcenverbrauch kann als zusätzliche Last (zu den Nutzanwendungen) auf das getestete System betrachtet werden (entsprechend dem Ausmaß in dem der Test die Nutzanwendungen des Systems behindert) und ist zeitabhängig: Die Ressourcen werden gebunden solange die Testausführung eine erhöhte Last verursacht – dabei kann das Ausmaß der Kosten u.U. in hohem Grad vom Ausgang des Testlaufs abhängen (z.B. ist es möglich daß ein erfolgreicher Test einer Netzwerkverbindung fast keine zusätzliche Last verursacht, während ständig wiederholte Test-Pakete an einen defekten Knoten die für andere Knoten verfügbare Bandbreite der Verbindung erheblich einschränken).

Für die RPN-Modellierung werden solche zeitabhängigen Bewertungsmaße als Zustandsmaße formuliert: Zuständen des Erreichbarkeitsgraphen (die in Termen der Belegung von Petrinetz-Stellen mit Token formuliert werden) werden Bewertungsraten zugewiesen – für jede Zeitspanne, in der sich das Modell in dem bewerteten Zustand befindet (d.h. solange der Test läuft), fallen dann Gewinn oder Kosten proportional zur angegebenen Rate an.

Die zeitabhängigen Kosten der Testausführung werden so durch negative Bewertungsmaße an den Stellen T_{P_r} (bei erfolgreichem Testausgang) und T_{F_r} (bei gescheitertem Testausgang) modelliert.

Zeitunabhängige Kosten und Informationsgewinn

Der durch die Testausführung potentiell erzielbare Gewinn an Informationen über das getestete System sowie beim Ablauf von Testmoduln anfallende einmalige, zeitunabhängige Kosten werden auf Impulsmaße abgebildet:

- Wie in 5.1.2 beschrieben wird vorausgesetzt, daß jedes durchgeführte Testmodul in einem Zuwachs an diagnostischem Wissen über das getestete System resultiert; dabei kann die Menge des erworbenen Wissen stark vom Testausgang abhängen (z.B. kann ein gescheiterter Betriebssystem-Test mit einer präzisen Fehlermeldung einen hohen diagnostischen Wert haben, während derselbe Test nahezu keine zusätzlichen, verwertbaren Informationen liefert falls er erfolgreich endet).

  In den Testsubnetzen wird dieser Informationsgewinn durch (positive) Impulsmaße an den Transitionen umgesetzt, deren Schalten die erfolgte Ausführung des Testmoduls modelliert (Transitionen T_{p_o} und T_{f_o} in Abb. 5.4).

- Auch einmalig und zeitunabhängig bei der Testausführung anfallende Kosten (wie z.B. Verschlechterung von Dienst-Qualität oder Ausfall-Sicherheit durch Abschalten von Knoten für spezielle Hardware-Tests) könnten in den Testsubnetzen ergebnisabhängig als (negative) Impulsmaße an den Transitionen T_p bzw. T_f modelliert werden; da die negative Nebenwirkung der Tests jedoch sowohl bei erfolgreichen als auch bei gescheitertem Ausgang auftritt, wird hier für beide Fälle derselbe negative Parameter zugewiesen.

5.2.5 DSS-Modelle

Die in 5.2.2 und 5.2.4 eingeführte Zuordnung von Testmodul-Parametern an Petrinetz-Elemente der Testsubnetze ergibt für jedes Testmodul einen ***Testmodul-Parametersatz***, der aus einem 8-Tupel von Werten besteht. Die Komponenten des Testmodul-Parametersatzes, ihr Wertebereich und ihre Rolle bei der Modellierung sind in Tabelle 5.1 zusammengefaßt; für die Bewertungsmaß-Terme sind zusätzlich noch die charakterisierenden Funktionen angegeben, mit denen die zu bewertenden Elemente im Erreichbarkeitsgraphen des RPN ausgewählt werden (Syntax gemäß App. A).

Schließlich werden noch die Bewertungsmaß-Terme der Testsubnetze (sowohl für Zustands- wie für Impulsmaße) durch Addition zu einem Bewertungsmaß zusammengefaßt: Damit wird für das Gesamtmodell je ein Zustandsmaß $\boldsymbol{\rho} = \sum \boldsymbol{\rho}_T$ sowie ein Impulsmaß $\boldsymbol{r} = \sum \boldsymbol{r}_T$ definiert – diese werden von PANDA auf Maße für Zustände und Zustandsübergänge im aus dem RPN-Modell generierten EMRM umgerechnet und legen die Zielfunktion für die Entscheidungsoptimierungs-Algorithmen fest.

Testmodul-Parameter	Zuordnung zu Testsubnetz-Element	Werte in	charakterisierende Funktion
Wahrscheinlichkeit für Testausgang	Feuerwahrscheinlichkeiten für T_p und T_f	$]0,1[$	—
erwartete Dauer	mittlere Schaltraten für T_{p_o} bzw. T_{f_o}	$\mathbb{R}^+$	—
zeitabhängiger Ressourcenverbrauch	Zustands-Bewertungsterme ρ_T, bez. auf T_{P_r} bzw. T_{F_r}	$\mathbb{R}^-$	mark(T_{P_r}) bzw. mark(T_{F_r})
zeitunabhängige Kosten	Impuls-Bewertungsterm r_T, bez. auf T_p und T_f	$\mathbb{R}^-$	fire(T_p) bzw. fire(T_f)
Informationsgewinn	Impuls-Bewertungsterme r_T, bez. auf T_{p_o} bzw. T_{f_o}	$\mathbb{R}^+$	fire(T_{p_o}) bzw. fire(T_{f_o})

Tabelle 5.1: Abbildung von Werten aus Testmodul-Parametersatz auf Testsubnetz-Elemente

Mit dem Aufbau der Netztopologie aus der Diagnose-Strategie sowie den Zusammenhängen und Abhängigkeiten der Systemkomponenten und Testmoduln, der Parametrisierung des entstandenen Petrinetzes sowie der Angabe der Entscheidungsalternativen (d.h. der Einführung von Rekonfigurationstransitionen) und des Bewertungsmaßes ist der Aufbau der Petrinetz-Modelle für das DSS vollständig[7].

Die auf diese Weise entstandenen RPN bilden das Problem der Testplan-Erstellung auf Rekonfigurations-Petrinetze nach Kap. 3 ab und werden im weiteren als ***DSS-Modelle*** bezeichnet. Nach der (in 3.3.2 beschriebenen Umrechnung) des Erreichbarkeitsgraphen der DSS-Modelle in EMRM und deren Analyse können die Ergebnisse der Markov-Entscheidungs-Optimierung direkt in Testpläne transformiert werden.

Bei der Implementierung des DSS wurden die Angabe der Testmodul-Parametersätze sowie die Berechnung der daraus abgeleiteten (in Tab. 5.1 angegebenen) Parameter für RPN-Transitionen und der Bewertungsfunktionen aus den Parametersätzen in einer interpretierten Hochsprache (Perl, [WCS96]) realisiert; damit können die DSS-Modelle im PANDA-Format aus dem PANDA-Netzplan des Abhängigkeitenmodells und den Testmodul-Parametersätzen automatisch generiert werden. Die Vorgehensweise zum Aufbau von DSS-Modellen ist in Abb. 5.5 zusammenfassend graphisch dargestellt.

[7] Um den Ablauf der Modelle den auf S. 71 aufgeführten Einschränkungen anzupassen (d.h. den Abbruch des Testvorgangs nach dem ersten gescheiterten Test zu modellieren), werden zusätzlich alle Transitionen mit der Inhibitorfunktion »mark(Failed) = 0« belegt.

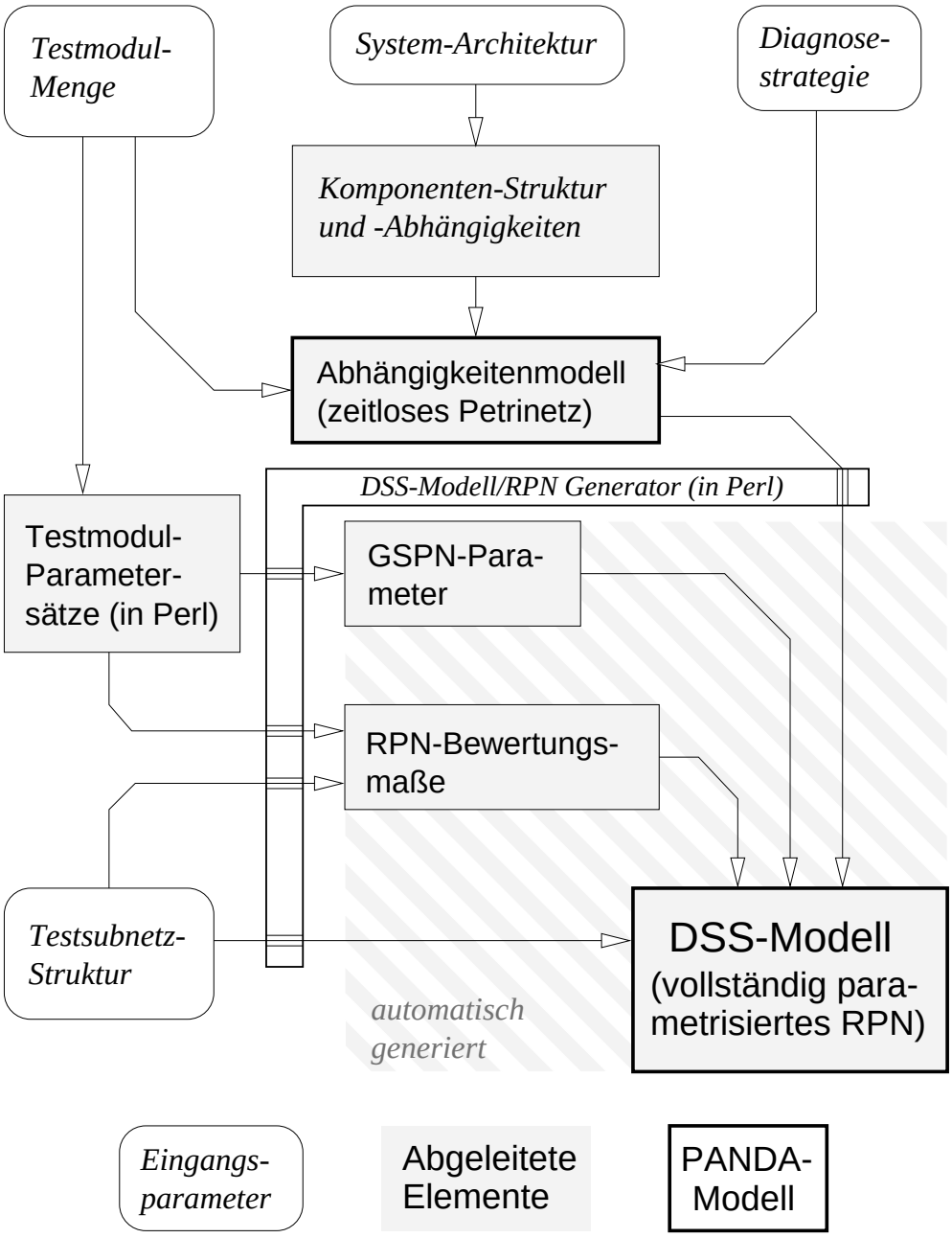


Abbildung 5.5: Übersicht zum Aufbau der DSS-Modelle

Kapitel 6

Klassifizierung von Testmoduln

In den bisherigen Kapiteln wurden die für das DSS eingesetzten modellierungstheoretischen Grundlagen und die Erstellung der Petrinetz-Modelle beschrieben. Ein wesentlicher Aspekt der Modellierung ist aber immer die Parametrisierung der Modelle; dies ist für die DSS-Modelle um so wichtiger als die Parametrisierung Grundlage einer Optimierung ist.

In diesem Kapitel wird daher eine Methodik vorgestellt, um aus technischen Daten, heuristisch ermittelten Werten über Zuverlässigkeitsgrößen und den Dienstgüte-Anforderungen an die zu diagnostizierenden Systeme eine Klassifizierung der vorhandenen Testmoduln zu erstellen. Mit Hilfe der klassifizierten Moduln können dann die Bewertungsmaße der Rekonfigurations-Petrinetze generiert werden, die die Zielgrößen der Entscheidungs-Optimierung für das DSS darstellen.

Dabei muß betont werden, daß die Parametrisierung stochastischer Modelle nicht unproblematisch ist:

- Da die Modellierung auf hohem Abstraktionsniveau stattfindet, haben die mathematischen Parameter oft auch eine abstrakte Semantik. In der Folge sind numerische Werte für diese Parameter oft nicht mehr direkt aus technischen Daten ableitbar – die Wahrscheinlichkeit, daß ein Fehler in einer Komponente auftritt, wird als Zahlenwert in keinem Handbuch zu finden sein, und wie hoch der Wert eines Testresultats als diagnostische Information ist, kann nicht aus Fehlerinjektions-Experimenten entnommen werden.

- Für viele gerade der Daten, die bei der Modellierung im Zuverlässigkeitsbereich benötigt werden, können konkrete Werte nur durch Abschätzung, heuristisch oder mit statistischer Unsicherheit behaftet ermittelt werden (Ausfallwahrscheinlichkeiten von Hardware-Komponenten, Aufwand für Software-Wartung etc.).

- Gleichzeitig liegt jedoch oft eine Vielzahl von Informationen aus verschiedenen Quellen über das Zielsystem und die Tests vor (technische Dokumentation von Hard- und Softwarekomponenten, Erfahrungswerte aus dem Betrieb, aus der System-Architektur resultierende Kenntnisse etc.). Diese Daten sind in der Regel nicht direkt vergleichbar, weil sie in unterschiedlichen Formaten vorliegen oder sich auf Komponenten völlig unterschiedlicher Natur (Hardware-Bauteile, Software-Pakete) beziehen, müssen aber für die Erstellung von Bewertungsmaßen in einen einheitlichen Bezugsrahmen zueinander gesetzt werden. Dabei müssen für die Belegung eines Modell-Parameters auch oft mehrere der vorhandenen Eingangsdaten zu einem Zahlenwert zusammengefaßt werden.

Das in [Kra98] entwickelte und hier vorgestellte Verfahren zur Klassifizierung und Bewertung von Testmoduln sowie zur Verrechnung der Werte zu RPN-gerechten Bewertungsmaßen stellt keinen algorithmisch festgelegten Weg zur Generierung von Parametersätzen für die DSS-Modelle dar – es soll vielmehr auf nachvollziehbare und flexible Weise ermöglichen, den Wert der durch die Ausführung der Testmoduln erzielbaren Informationen sowie deren Bedeutung für eine Diagnose des Zielsystems zu beurteilen und den Aufwand für die Gewinnung dieser Informationen abzuschätzen. Dabei soll es die Verwendung von möglichst vielen vorhandenen Kenntnissen aus allen Bereichen über Tests, Komponenten und Architektur des Zielsystems erlauben. Das Ziel der Klassifizierung ist weniger eine absolute Bewertung der Moduln an sich, sondern die Herstellung einer Vergleichbarkeit der Moduln relativ zueinander (immer bezogen auf ihren Einsatz in einem speziellen Zielsystem).

6.1 Definitionen

Für die Klassifizierung der Testmoduln wird das zu testende System als 3-Tupel $(\boldsymbol{C}, \boldsymbol{T}, \boldsymbol{F})$ definiert:

1. $\boldsymbol{C} = \{c_1, \ldots, c_{|\boldsymbol{C}|}\}$ bezeichnet die Menge der Komponenten aus denen das System aufgebaut ist; diese können sowohl in Hardware als auch in Betriebssystem- oder Anwendungs-Software vorliegen.

2. $\boldsymbol{T} = \{t_1, \ldots, t_{|\boldsymbol{T}|}\}$ ist die Menge der Testmoduln; jedes Testmodul kann die Funktionalität einer oder mehrerer[1] Komponenten aus $\boldsymbol{C}$ überprüfen (die

[1] Ein Testmodul kann entweder explizit als Entwurfseigenschaft (d.h. weil es zum Testen mehrerer Komponenten gleichzeitig konzipiert wurde) oder implizit (meistens wegen beschränkter Auflösung der Testresultate, d.h. die Ergebnisse können nicht präzise einer der getesteten Komponenten zugeordnet werden) mehrere Komponenten beaufschlagen.

Elemente von $\boldsymbol{T}$ entsprechen im DSS-Modell den in 5.2.3 beschriebenen Testsubnetzen). Die komplette Ausführung eines Testmoduls kann dabei auch aus mehreren einzelnen Tests bestehen, aber für den Zweck der Klassifizierung werden diese als eine Einheit zusammengefaßt.

3. $\boldsymbol{F} = \{f_1, \ldots, f_{|\boldsymbol{F}|}\}$ ist die Menge der Fehler, die in den Komponenten in $\boldsymbol{C}$ auftreten können und zu deren Diagnose die Tests in $\boldsymbol{T}$ angewendet werden können.

Die Parametrisierung muß zwei wesentliche Größen für jedes Testmodul quantisieren:

- Den potentiellen Gewinn an Information der durch die Ausführung des Testmoduls möglich wird.
- Den Kosten, d.h. den Verbrauch an Systemressourcen, der für die Ausführung des Testmoduls benötigt (d.h. dem getesteten System für Nutzleistung entzogen) wird.

Durch die Quantisierung muß ein gemeinsamer (numerischer) Bezugsrahmen zwischen dem Informationsgewinn und den Kosten der Ausführung aller Testmoduln hergestellt werden, der bewertende Vergleiche ermöglicht.

6.2 Kategorien zur Beschreibung des Informationsgewinns

Die Ausführung jeden Testmoduls resultiert in einem gewissen Gewinn an Information über den Zustand des getesteten Systems; dieser Informationsgewinn muß quantitativ bewertet werden. Da die Testmoduln einen weiten Bereich unterschiedlicher Funktionalitäten überprüfen, wurden zur Klassifizierung verschiedene ***Bewertungskategorien*** eingeführt, die im Folgenden näher beschrieben werden. Zur Bildung eines vollständigen Bewertungsmaßes für ein Testmodul werden dessen Werte in den einzelnen Kategorien miteinander verrechnet.

Da bedingt durch die unterschiedliche Natur der Testmoduln nicht alle Testmoduln bezüglich aller Kategorien bewertet werden können, wurde eine hohe Zahl von Kategorien eingeführt (dies ermöglicht auch eine detailliertere Bewertung in den Fällen, in denen durch ein Testmodul mehrere Systemkomponenten überprüft werden).

Im Folgenden werden zunächst die Bewertungskategorien eingeführt und motiviert; danach wird die Berechnung von Bewertungsmaßen aus den Wertungen in den einzelnen Kategorien beschrieben.

6.2.1 Überdeckung

Überdeckung ist ein klassisches Maß zur Beurteilung der Güte eines Tests; sie wird definiert als das Verhältnis der Zahl der durch den Test detektierbaren Fehler zur Gesamtzahl der Fehler als $i/|\boldsymbol{F}|$ (wo i die Zahl der verschiedenen Fehler $f_i \in \boldsymbol{F}$ ist, die mittels des Moduls diagnostizierbar sind). Die Bestimmung der Überdeckung eines Testmoduls kann durch ***Fehler-Matrizen*** (*fault matrix* oder *fault dictioniary*, s. [SS94] S. 39) über $\boldsymbol{T} \times \boldsymbol{F}$ erfolgen: Fehler werden simuliert oder in das zu diagnostizierende System injiziert, und die Ergebnisse des Testmoduls (d.h. ob der Fehler durch Ausführung des Moduls diagnostizierbar ist oder nicht) werden in die Fehlermatrix übertragen. Falls ein Testmodul mehrere Komponenten testen kann, muß die Überdeckung für jede getestete Komponente separat bestimmt und entsprechend bewertet werden. Die Überdeckung eines Testmoduls wird als unabhängig vom Testausgang betrachtet.

Je höher die Überdeckung eines Testmoduls für eine Komponenten ist, desto höher ist der potentielle Informationsgewinn durch die Ausführung (d.h. um so mehr Fehler können diagnostiziert bzw. ausgeschlossen werden), und in der Folge die numerische Bewertung in dieser Kategorie.

6.2.2 Genauigkeit

Für die Zwecke des DSS kann mit dieser Kategorie bewertet werden, wie präzise ein Fehler durch die Ausführung eines Testmodul zurückverfolgt und lokalisiert werden kann. Die Genauigkeit ist dann die durchschnittliche Zahl von möglichen Fehlern f_i, die das Testmodul zurückliefert wenn seine Ausführung scheitert – je geringer diese Zahl, desto genauer wird das Testmodul eingestuft (und in dieser Kategorie numerisch bewertet)[2]. Dabei ist die Genauigkeit bei fehlgeschlagenen Test meist höher als bei erfolgreichen, da im zweiten Fall in der Regel keine detaillierten Meldungen ausgegeben werden.

In der Praxis kann die Genauigkeit nur empirisch (durch Untersuchung der Ausgabe der Testmoduln im Fehlerfall) festgelegt werden und in einem weiten Bereich variieren (von präzisen Identifizierungen schadhafter Bauteile beim Offline-Test von IC-Boards oder detaillierten Meldungen über Fehlkonfigurationen von Betriebssystemsoftware bis zu sehr allgemein gehaltenen Meldungen beim Test von Netzwerkverbindungen).

Werden durch ein Testmodul mehrere Komponenten überprüft, muß auch die Genauigkeit des Tests separat für jede Komponente bewertet werden.

[2] Solche Einstufungen wurden in [CMM70] eingeführt und in sog. Diagnose-Graphen zur Ableitung diagnostischer Schlüsse eingesetzt.

6.2.3 Detailniveau

In [Kim86] werden Niveaus definiert, auf denen ein Test ein System untersuchen kann. Diese Niveaus sind auch in angepaßter Form für die Bewertung der DSS-Testmoduln herangezogen worden:

- System-Level (Level 7)
- LAN-Level (Level 6)
- Knoten-Level (Level 5)
- Prozessor-Level (Level 4)
- ALU-Level (Level 3)
- NOR-Gate-Level (Level 2)
- Verdrahtungs-Level (Level 1)

Dabei werden Tests in den Niveaus 1 – 3 üblicherweise nur mit spezialisierter Ausrüstung und Kenntnissen möglich sein (d.h. bei der Diagnose von Bauteilen durch den Hersteller oder Reparaturbetriebe), während Tests in den Niveaus 4 – 7 auch bei der Systemdiagnose im laufenden Betrieb stattfinden können.

Die Ausführung eines Testmoduls eines der höheren Niveaus wird normalerweise einen hohen Informationszuwachs erbringen wenn sie erfolgreich verläuft (da damit die Funktionalität von großen Teilen des Systems angenommen werden kann). Im Fehlerfall wird hingegen wenig diagnostische Information gewonnen, da die fehlerhafte Komponente durch das Testmodul nur sehr ungenau lokalisiert wird.

Andererseits erbringt die erfolgreiche Ausführung eines Testmoduls der niedrigeren Niveaus umgekehrt relativ wenig Information (nur kleine Teile des Systems werden überprüft), während fehlgeschlagene Tests der niedrigeren Niveaus ausgefallene Komponenten präzise lokalisieren. Bezogen auf das DSS wird daher die Kombination aus erfolgreichen Tests auf höheren und fehlgeschlagenen Test auf niedrigeren Niveaus als besonders wünschenswert betrachtet (und entsprechend numerisch bewertet).

6.2.4 Personelle Voraussetzungen

Mit dieser Kategorie kann eine Klassifizierung der Testmoduln nach den für die Ausführung benötigten Zugangsberechtigungen sowie den zur Auswertung der Ergebnisse erforderlichen Vorkenntnissen vorgenommen werden. Damit kann berücksichtigt werden, daß für den Einsatz mancher Testmoduln Fachpersonal benötigt wird, das u.U. nur eingeschränkt verfügbar oder lokal nicht vorhanden ist (also bei Bedarf unter hohen zusätzlichen Kosten von außerhalb hinzugezogen werden müßte). Für die Zwecke des DSS wurde folgende Einstufung der personellen Voraussetzungen vorgenommen:

- Normaler Benutzer (Level 1)
- Operateur (Level 2)
- System-Manager (Level 3)
- Externer Experte (Level 4)

Bei der Bewertung der Testmoduln wird die Verfügbarkeit des entsprechend qualifizierten Fachpersonals dahingehend einbezogen, daß Testmoduln mit niedrigeren Anforderungen bevorzugt ausgeführt werden (obwohl der diagnostische Informationsgewinn i.d.R. bei Tests mit höheren Anforderungen größer sein wird), da sie immer und ohne zusätzlichen finanziellen Aufwand benützt werden können. Dies reflektiert auch die in der Praxis übliche Vorgehensweise, mit den Tests zu beginnen die am einfachsten auszuführen sind, und die Tests für die spezielle Rechte und/oder Kenntnisse nötig sind erst dann durchzuführen, wenn anders kein Ergebnis erzielt werden kann. Bei der numerischen Bewertung in dieser Kategorie werden in der Folge die Moduln um so höher eingestuft, je niedriger die notwendigen personellen Voraussetzungen zu ihrer Ausführung sind.

Falls mit einem Testmodul mehrere Systemkomponenten mit unterschiedlichen personellen Voraussetzungen überprüft werden, wird das Modul auf dem Niveau der Komponente klassifiziert, die die höchsten Anforderungen bezüglich personeller Voraussetzungen stellt. Die Bewertung der Testmoduln erfolgt in dieser Kategorie unabhängig vom Testausgang.

6.2.5 Vorgeschichte

Hiermit soll es ermöglicht werden, Erkenntnisse aus bisherigen Ausführungen eines Testmoduls in dessen Bewertung einzubeziehen. Dies kann aus zwei Gründen nützlich sein:

- Bisher sind keine praktisch anwendbaren Standard-Metriken für die Stabilität bzw. »Ausfallwahrscheinlichkeit« (Abstürze, fehlerhaftes Verhalten) von Software bekannt; in der Kategorie Vorgeschichte können als Kompensation dafür Erfahrungen beim Betrieb einer individuellen Konfiguration bewertet werden. Beispielsweise kann die Ausführung von Testmoduln, die Software-Komponenten mit erfahrungsgemäß hoher Ausfallhäufigkeit überprüfen, als potentiell wünschenswerter eingestuft werden als die von anderen, deren Ausführung noch nie ein fehlerhaftes Resultat erbracht hat.

- Auch bei der Bewertung von Hardware-Tests können Betriebserfahrungen oder lokale Gegebenheiten eine Rolle spielen; z.B. können erschwerte Umweltbedingungen (klimatisch ungeeignete Stellplätze etc.) zu erhöhten Ausfallwahrscheinlichkeiten von Hardware-Komponenten führen. Die entsprechenden Testmoduln können dann so eingestuft werden, daß sie bevorzugt auszuführen sind. Allgemeine bauartbedingte Informationen bezüglich der Ausfallwahrscheinlichkeit von Hardware-Komponenten (wie etwa technische Daten des Komponentenherstellers) können mit der Kategorie Ausfallrate (s. 6.2.9) besser in die Bewertung einbezogen werden.

Die Bewertungen in dieser Kategorie sind weitgehend von den speziellen Komponenten einer Installation und ihrem Zusammenspiel abhängig, und müssen deshalb im laufenden Betrieb des DSS ständig und wiederholt erneut vorgenommen werden (dabei ist es auch denkbar, die Ergebnisse früherer Läufe nur innerhalb eines gewissen Zeitintervalls mit zu berücksichtigen).

Da die Kategorie Vorgeschichte eine Rückkopplung der Erfahrungen mit der Ausfallhäufigkeit von Komponenten bei der Anwendung des DSS darstellt, werden die Moduln in dieser Kategorie nur bei in einem Fehler resultierenden Testläufen bewertet; die Bewertung muß natürlich auch hier separat für jede vom Modul beanspruchte Komponente vorgenommen werden.

Die der Bewertung in dieser Kategorie zugrundeliegenden Daten können (neben der Anwendung für die Einstufung der Testmoduln) auch dazu herangezogen werden, die Einschätzungen für die Wahrscheinlichkeit von fehlgeschlagenen Testläufen bei der Ausführung eines Moduls zu präzisieren.

6.2.6 Zuverlässigkeit

Ein anderes Maß zur Bewertung der Testmoduln ist ihre Zuverlässigkeit, für diesen Zweck beschrieben als das Verhältnis von Fehlalarmen (*false-alarm rate* in [SS94]), d.h. das Testmodul meldet einen Fehler wo keiner vorhanden ist, bzw. falschen Funktionszusicherungen (*false-assurance rate*, ein vorhandener Fehler wird fälschlicherweise bei der Ausführung des Testmoduls nicht gemeldet) zur Gesamtanzahl der Ausführungen des Moduls.

Sheppard und Simpson zeigen in [SS92a], daß die Raten für Fehlalarm oder falsche Funktionszusicherungen nicht genau berechnet werden können; es müssen Abschätzungen verwendet werden. Trotzdem kann diese Kategorie nützlich sein, da es hiermit möglich wird, das Vertrauen in die Korrektheit des Ergebnisses eines Testmoduls in dessen Bewertung einfließen zu lassen. Zusätzlich kann damit berücksichtigt werden, daß bei verteilten Systeme die meisten der Tests, die während des normalen Anwendungsbetriebs eines Systems ausgeführt werden können, das System auf einem relativ hohen Niveau (im Sinne von 6.2.3) untersuchen. Da sie dabei eine Vielzahl von Komponenten zumeist nur oberflächlich (verglichen mit Tests auf niedrigem Niveau) überprüfen, sind ihre Ergebnisse mit einem gewissen Grad an Unsicherheit behaftet.

Offensichtlich ist der von der Ausführung eines Testmoduls erwartete Informationsgewinn um so höher, je zuverlässiger das Modul eingestuft wurde. Dabei können je nach Testmodul unterschiedliche Bewertungen für erfolgreiche (korreliert mit der Rate an falschen Funktionszusicherungen) und gescheiterte (korreliert mit der Fehlalarmrate) Ausführungen vorgenommen werden.

Mit den bisher vorgestellten Kategorien werden im Wesentlichen Eigenschaften von Testmoduln bewertet. Für eine umfassende Klassifizierung der Tests hat es sich jedoch als notwendig erwiesen, auch Kategorien zur Bewertung der Komponenten der untersuchten Systeme zu einzuführen. Diese Komponenten-Wertungen wurden verwendet, um zusätzlich zu den bisher eingeführten Kategorien und bezogen auf inhärente Eigenschaften des zu diagnostizierenden Systems die Bedeutung der Moduln für die Gewinnung von Informationen genauer zu kennzeichnen. Zur Umrechnung der in den folgenden vier Kategorien vergebenen Werte in Maße für Testmoduln wird über eine Testmodul-zu-Komponenten-Matrix (s. 6.3.1) für jedes Modul, das eine Komponente testet, die entsprechende Einstufung übernommen (unabhängig vom Ausgang des Tests, da die Komponenten-Wertungen Systemeigenschaften beschreiben, die durch das Testresultat nicht beeinflußt werden).

6.2.7 Komponenten-Redundanz

Mit dieser Kategorie kann berücksichtigt werden, daß in vielen verteilten Systemen Komponenten redundant vorhanden sind. Die Redundanz kann in Hardware vorliegen, etwa als Nebeneffekt des Einsatzes mehrerer gleichartiger Bauteile (Festplatten etc.) in den einzelnen System-Knoten, oder explizit als Maßnahme zur Erhöhung der Zuverlässigkeit des Gesamtsystems (z.B. Einbau mehrerer Netzwerkanschlüsse mit Failover-Verhalten bei Ausfall). Software-Komponenten können durch parallelen Ablauf mehrerer Instanzen desselben Programms ebenfalls redundant sein (konstruktionsbedingt durch die Notwendigkeit, die gleichen Dienste auf verschiedenen Knoten anbieten zu müssen, oder explizit dupliziert zur Absicherung gegen Software-Fehler).

Bezogen auf den Wissensstand über das Gesamtsystem kann es wünschenswert sein, die Komponenten mit geringerer Redundanz bevorzugt zu überprüfen (da ihr Ausfall schlechter kompensiert werden kann, und in der Folge ein Fehler in ihnen das System oft stärker beeinträchtigen wird). Hier kann über die Bewertung in der Kategorie Komponenten-Redundanz entsprechend in die Gewichtung der Testmoduln eingegriffen werden. Im Gegensatz zu den meisten bisherigen Kategorien ist die Zuordnung von numerischen Werten reziprok: Je höher die Redundanz einer Komponente, desto niedriger fällt die Bewertung aus (um konsistent mit den bisherigen Kategorien zu bleiben, bei denen höhere Werte auch höhere Präferenz wegen höherem potentiellen Informationsgewinn ausdrücken).

6.2.8 Missionsrelevanz

Ähnlich wie bei der letzten Kategorie geht es hier darum, die Wichtigkeit spezieller Komponenten für ein System so zu markieren, daß Tests dieser Komponenten

eine erhöhte Bedeutung zumessen wird. Während durch die letzte Kategorie die Möglichkeit geschaffen wird, Tests von Komponenten mit »Reserve« als weniger wichtig einzustufen, wird mit der Kategorie Missionsrelevanz direkt gemessen, in welchem Maß die jeweilige Komponente zum »Überleben« eines verteilten Systems (im Sinne der Erfüllung seiner Aufgaben) beiträgt.

Eine eigene Kategorie für diesen Zweck wurde als notwendig erachtet, da überlebenswichtige Komponenten von verteilten Systemen nicht immer redundant ausgelegt werden können oder u.U. auch alle Instanzen mehrfach vorhandener Komponenten für das Überleben eines Systems unerläßlich sein können. Die Bewertungen in dieser Kategorie sind hochgradig von der jeweiligen speziellen Systemkonfiguration abhängig und müssen angepaßt und verfeinert werden, falls das System rekonfiguriert werden sollte. Zur Bestimmung der numerischen Werte können die Abhängigkeiten der Komponenten untereinander herangezogen werden, z.B. aus dem Abhängigkeitenmodell oder Fehlerbaum-Modellierungen des Systems. Die Zuweisung der numerischen Werte erfolgt direkt nach der Bedeutung der Komponenten (je wichtiger, desto höher).

6.2.9 Ausfallrate

Eine andere Kategorie zur indirekten Beschreibung der Bedeutung eines Testmoduls und der durch seine Ausführung erhältlichen Informationen ist die Ausfallrate der getesteten Komponenten (im Sinne der pro Zeiteinheit durchschnittlich zu erwartenden Anzahl an Ausfällen einer gegebenen Komponente). Diese Ausfallrate wird häufig von Hardware-Herstellern zugesichert, kann den technischen Unterlagen der Bauteile entnommen werden und wird als durchschnittliches Zeitintervall zwischen dem Auftreten von Fehlern (***mean time between failures, MTBF***) angegeben. Je niedriger die MTBF einer Komponente, desto wahrscheinlicher ist ihr Ausfall, und um so »aussichtsreicher« (im Sinne der Fehlerdiagnose) wird es, sie einer Überprüfung zu unterziehen; Komponenten mit hoher MTBF wird deshalb in dieser Kategorie numerisch ein kleinerer Wert zugewiesen. Liegen solche Informationen für Hardware-Komponenten vor, können sie ebenso wie in der Kategorie Vorgeschichte (vgl. 6.2.5) auch verwendet werden, um die Wahrscheinlichkeit für fehlgeschlagene Läufe beim Testen der jeweiligen Komponente abzuschätzen.

Für Software-Komponenten ist bisher keine sinnvollen Definition einer Ausfallrate bekannt. Es wäre zwar denkbar, hier ein Maß für die Versagenswahrscheinlichkeit z.B. auf Grund der Komplexität (Code-Umfang und ähnliche Metriken) oder des Entwicklungsstandes der Hersteller-Organisation (etwa nach dem Capability Maturity Model [PCC93]) einzuführen, aber derartige Informationen sind i.A. nicht zugänglich. Daher wurde für Bewertungen im Hinblick auf die Ausfallrate von Software-Komponenten die Kategorie Vorgeschichte eingeführt.

6.2.10 Reparaturaufwand

Diese Kategorie beschreibt, wie schwerwiegend die Folgen eines Fehlers in einer gegebenen Komponente für das System sind. Im allgemeinen bedeutet dies eine Abschätzung des Aufwands, der für die Reparatur (d.h. in der Praxis den Austausch) von Systemkomponenten anfallen würde, falls sie defekt wären. Hintergrund dafür ist, daß es wünschenswert sein kann, Komponenten mit sehr hohem Reparaturaufwand bevorzugt zu überprüfen (da so mit aufwendigen Instandsetzungsarbeiten früher begonnen werden kann).

In der Praxis variiert der mögliche Reparaturaufwand in einem weiten Bereich: Von kleineren Arbeiten, die nahezu ohne Kosten und innerhalb von Minuten erledigt werden können (unterbrochene Steckverbindung wieder herstellen, Software-Konfigurationsfehler beheben) über Tätigkeiten, die zu Dienstunterbrechungen des betroffenen Systems im Bereich von Stunden führen können, erhebliche Kosten verursachen und für die qualifiziertes Fachpersonal benötigt wird (auf Vorrat gehaltenes Ersatzteil für defekte Hardware-Komponente einbauen, neue Versionen von Software-Moduln einspielen und konfigurieren) bis hin zu Maßnahmen, die auf die koordinierte Zusammenarbeit von Experten über Tage oder Wochen hin angewiesen sind (Ersatzteilbeschaffung von externen Lieferanten, Umbauarbeiten durch Service-Techniker von außerhalb, Umrüstung eines kompletten Betriebssystems auf neue Version, Korrektur von Software-Defekten) und während dieser Zeit keinen oder nur eingeschränkten nutzbringenden Betrieb zulassen.

Normalerweise wird für Reparaturarbeiten nur eine durchschnittlich zu erwartende Dauer (***mean time to repair***, *MTTR*) ermittelbar sein. Die Zuweisung numerischer Werte in dieser Kategorie erfolgt proportional zur MTTR.

6.3 Numerische Verarbeitung der Kategorien-Maße

Nachdem Kategorien zur Bewertung der Testmoduln und Systemkomponenten bez. potentiellem Informationsgewinn und Bedeutung eingeführt wurden, müssen die Bewertungen in den Kategorien zu einem Wert[3] verrechnet werden, der dann als Impulsmaß in die DSS-Modelle übertragen werden kann.

In Tabelle 6.1 ist nochmals zusammengefaßt, in welchen der beschriebenen Kategorien die Bewertungen direkt den Testmoduln ($f(\boldsymbol{T})$) und/oder den getesteten Komponenten ($f(\boldsymbol{C})$) zugewiesen werden bzw. in welchen Kategorien der Informationsgewinn für erfolgreiche und fehlgeschlagene Testausführungen gleich

[3] Die in 2.2.2 bzw. App. A vorgestellte und in PANDA implementierte Beschreibungssprache würde auch die direkte Spezifikation der Petrinetz-Bewertungsmaße aus den Bewertungen in den einzelnen Kategorien gestatten; für die DSS-Modellierung wurde aber aus Gründen der besseren Verständlichkeit der Modelle die Umrechnung der Kategorien-Maße in einen separaten Schritt ausgelagert.

f	Symbol	$f(\boldsymbol{T})$	$f(\boldsymbol{C})$	*pass = fail*
Überdeckung	$\mathcal{U}$	•	•	•
Genauigkeit	$\mathcal{G}$	•	•	
Detailniveau	$\mathcal{D}$	•		
Personelle Voraussetzungen	$\mathcal{P}$	•		•
Vorgeschichte	$\mathcal{V}$	•	•	
Zuverlässigkeit	$\mathcal{Z}$	•	•	
Komponenten-Redundanz	$\mathcal{K}$		•	•
Missionsrelevanz	$\mathcal{M}$		•	•
Ausfallrate	$\mathcal{A}$		•	•
Reparaturaufwand	$\mathcal{R}$		•	•

Tabelle 6.1: Übersicht der Klassifizierungskategorien

ausfällt (*pass = fail*); zusätzlich wird für die weiteren Ausführungen für jede Kategorie ein Symbol eingeführt.

6.3.1 Test-Komponenten-Matrizen

Aus Tabelle 6.1 bzw. der Beschreibung der Kategorien wird ersichtlich, daß die Bewertungen in den Kategorien sowohl von den Testmoduln, den getesteten Komponenten als auch von einer Kombination aus beiden abhängen können. Außerdem ist die Beziehung zwischen Tests und Komponenten multidimensional: Testmoduln können mehrere Komponenten überprüfen, und Komponenten können von mehreren Testmoduln überprüft werden. Das Resultat der Bewertungen muß jedoch letzlich den Testmoduln zugewiesen werden, und zwar abhängig von den möglichen Testausgängen.

Um dies zu erreichen, werden zwei ***Test-Komponenten-Matrizen*** auf $|\boldsymbol{T}| \times |\boldsymbol{C}|$ eingeführt: M_p für Bewertungen bei erfolgreichen und M_f für Bewertungen bei fehlgeschlagenen Testausführungen. Die Elemente $m_{f_{i,j}} \in M_f$ bzw. $m_{p_{i,j}} \in M_p$ werden durch Funktionen γ_f bzw. γ_p auf Vektoren mit den Bewertungen in den einzelnen Kategorien abgebildet:

$$\gamma_{s_{i,j}} : \left\{ \begin{array}{rcl} M_s & \rightarrow & \mathbb{R}_+^{10} \\ m_{s_{i,j}} & \mapsto & \{\mathcal{U}_{s_{i,j}}, \mathcal{G}_{s_{i,j}}, \mathcal{D}_{s_{i,j}}, \mathcal{P}_{s_{i,j}}, \mathcal{V}_{s_{i,j}}, \mathcal{Z}_{s_{i,j}}, \mathcal{K}_{s_{i,j}}, \mathcal{M}_{s_{i,j}}, \mathcal{A}_{s_{i,j}}, \mathcal{R}_{s_{i,j}}\} \end{array} \right. \tag{6.1}$$

$$\text{mit } i \in \{1, \ldots, |\boldsymbol{T}|\},\ j \in \{1, \ldots, |\boldsymbol{C}|\} \text{ und für } s \in \{f, p\}$$

Dabei werden die Werte der Elemente der durch γ_f bzw. γ_p erzeugten Vektoren auf $[0,1]$ normiert; höhere numerische Werte repräsentieren einen potentiell höheren Informationsgewinn. Die Zuweisung von 0 bedeutet, daß für das entsprechende Testmodul in dieser Kategorie keine Wertung vorgenommen wurde.

6.3.2 Bildung von Impulsmaßen

Die weitere Verarbeitung der Kategorien-Bewertungen geschieht über eine doppelte Mittelwertbildung:

- Im ersten Schritt wird für jedes Element von M_f bzw. M_p der Mittelwert über die Wertung in allen Kategorien gebildet; dies führt zu 2 Matrizen $\overline{M}_f$ und $\overline{M}_f$ über $|\boldsymbol{T}| \times |\boldsymbol{C}|$ mit:

$$\begin{aligned} \overline{m}_{s_{i,j}} &= \frac{1}{10}(\mathcal{U}_{s_{i,j}} + \mathcal{G}_{s_{i,j}} + \mathcal{M}_{s_{i,j}} + \mathcal{A}_{s_{i,j}} + \ldots + \mathcal{R}_{s_{i,j}}) \\ &\quad \text{mit } i \in \{1,\ldots,|\boldsymbol{T}|\},\ j \in \{1,\ldots,|\boldsymbol{C}|\} \text{ und } s \in \{f,p\} \end{aligned} \tag{6.2}$$

  Dabei könnten bei Bedarf die einzelnen Kategorien auch nochmals separat gewichtet in die Summenbildung eingehen bzw. statt des arithmetischen Mittels könnten auch andere Formen zur Zusammenfassung der Werte gewählt werden; bei den für das DSS betrachteten Beispielen hat sich die oben beschriebene Form als ausreichend flexibel erwiesen (es wird hier davon ausgegangen, daß in allen 10 der in 6.2 beschriebenen Kategorien Wertungen vergeben werden).

- Danach wird über die Spalten von $\overline{M}_f$ und $\overline{M}_f$ gemittelt, d.h. für jedes Testmodul wird das arithmetische Mittel der Bewertungen aller Komponenten gebildet, die durch das Modul getestet werden. Zu diesem Zweck wird mit 6.3 eine Restriktionsfunktion Υ definiert, die jedes Testmodul auf die Teilmenge der durch dieses überprüften Komponenten abbildet:

$$\Upsilon : \left\{ \begin{array}{lcl} \boldsymbol{T} & \rightarrow & \boldsymbol{C} \\ t_i & \mapsto & C_{t_i} \end{array} \right. \quad \text{mit } i \in \{1,\ldots,|\boldsymbol{T}|\} \text{ und } C_{t_i} \subseteq \boldsymbol{C} \tag{6.3}$$

  Damit können die Werte für die Impulsmaße in den in Kap. 5.2.3 definierten Testsubnetzen formuliert werden:

$$\begin{aligned} t_{s_i} &= \frac{\sum_{j=1}^{|\boldsymbol{C}|} \overline{M}_{s_{t_i,c_j}}}{|\Upsilon(t_i)|} \\ &\quad \text{mit } i \in \{1,\ldots,|\boldsymbol{T}|\} \text{ und } s \in \{f,p\} \end{aligned} \tag{6.4}$$

Die in 6.4 berechneten Werte beschreiben für die Zwecke des DSS den potentiellen Informationsgewinn (für beide möglichen Ergebnisse), der bei der Ausführung der Testmoduln erwartet wird. Diese Werte werden den Transitionen T_{P_r} bzw. T_{F_r} der in Abb. 5.4 gezeigten Testsubnetze als (positive) Impulsmaße zugewiesen.

6.4 Kosten der Testausführung

Einem potentiell erzielbaren Gewinn an diagnostischen Informationen über das getestete System stehen Kosten gegenüber, die durch die Ausführung von Testmoduln entstehen. Diese lassen sich in zwei Formen untergliedern:

- Systemressourcen, die durch die Ausführung der Tests belegt werden. Diese werden als eine zusätzliche Last auf das System aufgefaßt und sind verbunden mit dem Zeitaufwand, der zur Durchführung der Tests benötigt wird.

- Systemressourcen, die während der Ausführung von Tests für die Verrichtung nutzbringender Arbeit gesperrt werden. Der Einfluß solcher Testmoduln kann als eine Art »Fehler« interpretiert werden, der Teile des Systems aus Sicht der Nutzanwendung außer Betrieb nimmt.

Neben den eigentlichen getesteten Komponenten können durch die Ausführung eines Testmoduls auch andere Teile des Systems zusätzlich belastet werden (z.B. kann ein Test eines Software-Dienstes auf einem System-Knoten beträchtliche Mengen an Rechenleistung bzw. Netzwerkbandbreite zur Übertragung der Testergebnisse in Anspruch nehmen).

Auch hier soll die Parametrisierung keine absoluten Werte für die durch die Ausführung der Testmoduln verursachte zusätzliche Belastung liefern, sondern die Möglichkeit bieten, die Moduln so zu klassifizieren, daß der mögliche negative Einfluß verschiedener Moduln auf das Zielsystem vergleichbar wird.

6.4.1 Ressourcenverbrauch während der Ausführung

Diese Kosten fallen während der gesamten Zeit an, die für die Ausführung eines Testmoduls benötigt wird, und werden deshalb als Kosten pro Zeiteinheit parametrisiert (bzw. als Zustandsmaß modelliert). Für die Testmoduln des DSS wurde der Ressourcenverbrauch in folgenden Bereichen betrachtet:

Rechenzeit – die Anforderungen an CPU-Leistung, die von den Tests gestellt werden. Wird hier Leistung für Tests abgezweigt, wirkt sich dies negativ auf die Antwortzeiten von Knoten des verteilten Systems aus.

Hauptspeicher/Plattenplatz – der Bedarf an Speicherplatz, der durch die Tests entsteht. Vor allem hohe eine Hauptspeicher-Auslastung schlägt sich erfahrungsgemäß ebenfalls sehr negativ auf die Antwortzeiten von System-Knoten durch.

Netzwerkbandbreite – die Kommunikationsleistung, die durch die Ausführung der Tests dem Zielsystem entzogen wird. Auch hier kann eine Einschränkung der für die Nutzanwendung zur Verfügung stehenden Bandbreite zu einer Verringerung der Güte der durch das System erbrachten Dienste führen.

Die genauen Daten über den Ressourcenverbrauch sind in der Praxis schwer zu bestimmen – reale verteilte Systeme sind in der Regel nicht mit Schnittstellen zur Messung der durch einzelne Applikationen belegten Ressourcen ausgestattet. Zudem ist der Ressourcenverbrauch während der Ausführung der Tests auch oft von anderen, ebenfalls schwer meßbaren System-Parametern (wie z.B. der Last durch die Nutzanwendungen oder der Natur des Fehlers auf den getestet wird) abhängig und kann außerdem zeitlich stark variieren.

für die Zwecke der vorliegenden Arbeit werden die Ausführungen der Testmoduln als zusätzliche vom System zu bewältigender Aufträge betrachtet, für deren Bewältigung eine exponentiell verteilte Zeitspanne angenommen wird. Dieser Zeitbedarf wird in den DSS-Modellen durch zeitbehaftete Transitionen mit exponentiell verteilter Schalt-Verzögerung repräsentiert; die durchschnittlichen Ausführungsdauern der Testmoduln werden diesen Transitionen (T_{p_o} und T_{f_o} in Abb. 5.4) als Schaltraten zugeordnet. Die Raten für die zeitabhängigen Kosten wurden auf $[0,1]$ normiert und die normierten Werte als Zustandsmaß den Stellen T_{p_r} bzw. T_{f_r} (s. Abb. 5.4) der DSS-Testsubnetze zugewiesen. Dabei können sowohl die Kosten als auch die Zeitdauern unterschiedlich ausfallen für erfolgreichen bzw. gescheiterten Ausgang der Testausführung.

6.4.2 Ressourcenverbrauch als Vorbedingung der Ausführung

Zur Ausführung mancher Tests muß das Zielsystem oder Teile davon in spezielle Betriebszustände überführt werden, die es teilweise oder auch ganz (im Fall von Offline-Tests) für die Nutzanwendungen unbrauchbar machen. Beispiele hierfür sind erweiterte Hardware-Tests (die schnelle und gründliche Diagnosen ermöglichen, aber nur erfolgen können wenn auf den getesteten Knoten kein Betriebssystem-Kern läuft), oder Konsistenz-Überprüfungen von Software-Komponenten (z.B. Datenbanken und Dateisystemen), während deren Ausführung kein Zugriff auf die getesteten Objekte erfolgen darf.

Um den negativen Einfluß solcher Tests auf das Zielsystem (zusätzlich zu den zeitabhängigen Kosten) für das DSS zu modellieren, wurden den entsprechenden

Testmoduln ein weiterer Kosten-Parameter zugewiesen: Für die Bestimmung numerischer Werte wurde abgeschätzt, um welchen Anteil die Nutzleistung des Systems als Voraussetzung der Testausführung vermindert wurde; die Werte wurden in $[0,1]$ normiert und als negative Impulsmaße (jeweils mit denselben Werten für beide Ausgänge der Tests) mit den potentiellen Informationsgewinnen verrechnet.

6.5 Erfolgswahrscheinlichkeit der Tests

Der letzte Parameter, der für die Test-Subnetze zu bestimmen ist, ist die Wahrscheinlichkeit für erfolgreichen bzw. fehlgeschlagenen Testausgang. Für eine praxisgerechte Belegung dieses Parameters mit numerischen Werten können bei einem realen System die schon unter 6.2.5 bzw. 6.2.9 eingeführten Bewertungskategorien Vorgeschichte (für Software-Komponenten) bzw. Ausfallrate (für Hardware-Komponenten) herangezogen werden; dabei sollten die jeweiligen Werte im Verlauf des Betriebs des DSS ständig angepaßt werden. Die Erfolgswahrscheinlichkeiten werden als Feuerwahrscheinlichkeiten an den zeitlosen Transitionen (T_p und T_f in Abb. 5.4) in die Testsubnetze übertragen.

6.6 Festlegung der Optimierungs-Ziele

Die bisher beschriebene Parametrisierung dient dazu, Eigenschaften des modellierten Systems und der Tests auf das DSS-Modell abzubilden. Für die Steuerung der Entscheidungs-Optimierung, die eines der wesentlichen Elemente des DSS ist, muß es zusätzlich noch ermöglicht werden, bezogen auf den Systembetrieb und die Bedürfnisse bei der Diagnose des modellierten Systems die Zielrichtung der Optimierung zu bestimmen.

Da einerseits durch die numerische Analyse der für die DSS-Modelle verwendeten RPN eine Optimierung von Entscheidungen (d.h. Testplänen) mit dem Effekt der Optimierung von Bewertungsmaßen vorgenommen wird, und andererseits bei Entwurf und Parametrisierung der Testsubnetze klar zwischen Modellierung von Kosten- (Systembelastung durch Testausführung) und Nutzenaspekten (potentieller Informationsgewinn durch Testresultate) unterschieden wurde, kann die Modellierung der Optimierungsziele einfach durch die Einführung einer ***Optimierungs-Gewichtung*** $\kappa \in \mathbb{R}^+$ vorgenommen werden: Die Kosten- bzw. Nutzenaspekte der Bewertungsmaße können bei der Übertragung in die Testsubnetze variabel gegeneinander gewichtet werden, indem die den Informationsgewinn reflektierenden Maße mit κ multipliziert werden.

Dabei können aus der Sicht eines Systembetreibers verschiedene Kriterien zum Tragen kommen:

Informations-Maximierung – Testläufe allein mit dem Ziel möglichst hohen Informationsgewinns, d.h. es sollen vorrangig potentielle Fehlerquellen lokalisiert werden (dies kann der Fall sein, wenn das System für eine begrenzte Zeit zur Fehlersuche außer Betrieb genommen werden kann). Erreicht wird dies durch anteilsmäßige Verringerung der Kostenmaße im Verhältnis zu den potentiellen Informationsgewinnen: $\kappa \gg 1$

Belastungs-Minimierung – Testläufe mit möglichst niedriger Belastung des Systems durch die Tests; durch die Ausführung der Tests sollte die Güte der durch das System erbrachten Dienste so wenig wie möglich beeinträchtigt werden. In der Regel wird unter solchen Maßgaben getestet werden, wenn das System durch die Nutzanwendung an der Grenze seiner Belastungsfähigkeit betrieben wird und keine zusätzlichen Leistungsminderungen toleriert werden können. Erreicht wird dies durch eine entsprechende Verringerung des Anteils der Maße zur Beschreibung des potentiellen Informationsgewinns im Verhältnis zu den Kostenmaßen: $\kappa \ll 1$

Effizienz-Optimierung – Kombinationen der beiden oben angeführten Zielsetzungen, wobei mit der Aussicht auf maximalen Informationsgewinn bei gleichzeitig möglichst geringer Systembelastung getestet werden soll. Mit diesen Ansprüchen wird die Testplanung bei Systemen mit gewissen Leistungsreserven und dennoch hohen Ansprüchen an konstante Dienstgüte normalerweise durchgeführt werden. Hier gehen beide Maße gleich gewichtet als Randbedingung in die Optimierung ein: $\kappa \approx 1$

Zeit-Optimierung – Testläufe, die in möglichst kurzer Zeit wichtige Informationen erbringen sollen. Hierzu können die Maße für den Ressourcenverbrauch während der Testausführung durch Multiplikation mit einem Gewichtungsfaktor $\gg 1$ selektiv negativ betont werden, sodaß Tests bevorzugt ausgeführt werden, die innerhalb kurzer Zeit Ergebnisse erwarten lassen.

Abb. 6.1 faßt den Informationsfluß bei der Generierung der (als RPN-Maße in die DSS-Modelle eingehenden) Testmodul-Parametersätze nochmals graphisch zusammen.

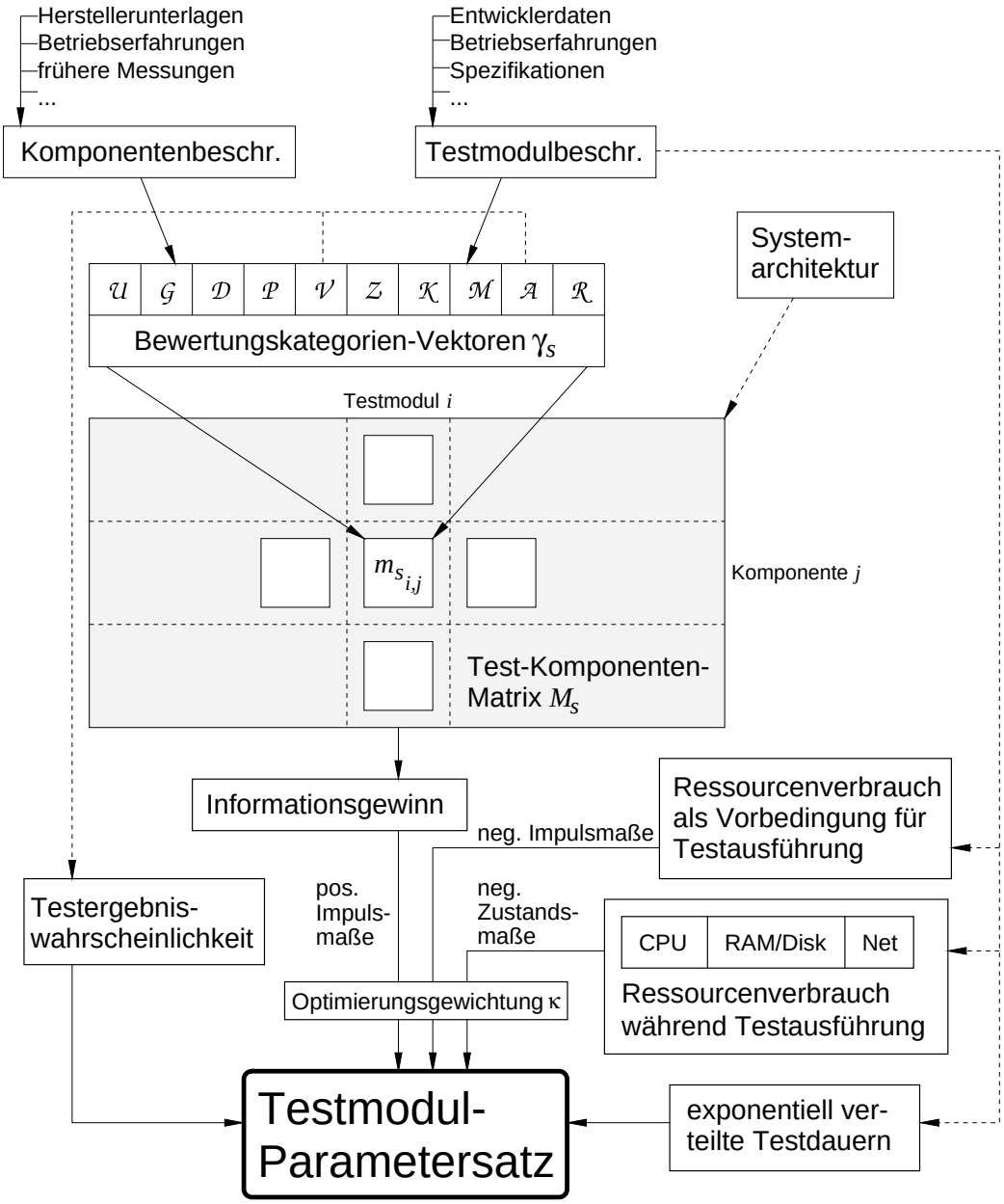


Abbildung 6.1: Übersicht Testmodul-Parametrisierung (mit $s \in \{f, p\}$)

Kapitel 7

Ergebnis-Auswertung

In den bisherigen Kapiteln wurden die Grundlagen für die DSS-Modellierung gelegt; hier werden zunächst die Algorithmen zur Ableitung von Testplänen aus den Ergebnissen der RPN-Analyse dargestellt. Danach werden die bisherigen Erfahrungen über die Anwendbarkeit der vorgestellten Verfahren zu Modellaufbau und -parametrisierung bei der Analyse des Fallstudien-Systems aus 5.2 beschrieben.

7.1 Berechnung von Testplänen

Die EMRM-Analyse der DSS-Modelle resultiert – für jede der im Modell enthaltenen Rekonfigurationskanten – in (im Mittel) optimalen Anweisungen zur Rekonfiguration des modellierten Systems bez. der jeweiligen Kante. Zur Erstellung von Testplänen muß nun diesen Anweisungen eine Semantik im Rahmen der DSS-Modelle zugeordnet werden, aus denen die EMRM entstanden sind.

7.1.1 Interpretation der EMRM-Analyseergebnisse

Die Ergebnisse der EMRM-Analyse sind definitionsgemäß *zustandsabhängig* (also auf Modellzustände der Markov-Ebene bezogen) und müssen auch so interpretiert werden. Es werden für jeden Markov-Zustand ω_i, der Anfangszustand einer Rekonfigurationskante e ist, Handlungsanweisungen der folgenden Form gegeben (die bei transienter Analyse zusätzlich vom Zeitpunkt abhängig sind, zu dem das System betrachtet wird):

Falls sich das System in Zustand ω_i befindet und in diesem die Rekonfiguration in den Zustand ω_j über die Rekonfigurationskante e möglich ist,

dann soll der Zustandsübergang über e zur Erzielung eines im Mittel optimalen Ertrags entweder durchgeführt werden (d.h. e ist ***aktiv***) oder unterbleiben (d.h. e ist ***inaktiv***).

Bei der Analyse von RPN sind die Zustände ω_i des EMRM als Markierungen (aus der Erreichbarkeitsmenge $\mathfrak{S}$ des RPN) gegeben, während die Übergänge $\mathfrak{e}$ zwischen den Zuständen (als Elemente der Kantenmenge $\mathfrak{E}$ des Erreichbarkeitsgraphen des RPN) mit dem Feuern von RPN-Transitionen assoziiert sind. Der Erreichbarkeitsgraph von DSS-Modellen wird aus den entsprechenden RPN gemäß den nach 3.3.2 erweiterten Petrinetz-Regeln gebildet; er beinhaltet daher die Aufzählung sämtlicher kombinatorisch möglichen Sequenzen von Testmoduln, die im Rahmen der Einschränkungen des Abhängigkeitenmodells erlaubt sind.

Die gewählte Struktur der Testsubnetze (vgl. 5.2.3 und Abb. 5.4) stellt sicher, daß die in den Testsubnetzen der DSS-Modelle gelegenen (d.h. die Ausführung des Testmoduls modellierenden) Transitionen nur nach dem Feuern einer vorgeschalteten Rekonfigurationstransition aktiviert werden können. Dies bedeutet, daß jeder Rekonfiguration entlang einer Kante $\mathfrak{e}$ im EMRM die Ausführung eines Testmoduls T_i entspricht – das Testmodul ist hier durch die $\mathfrak{e}$ zugeordnete Rekonfigurationstransition T_{ci} im RPN bestimmt. Dabei sind die genauen Bedingungen, unter denen die Rekonfiguration vorgenommen werden kann, über den Anfangszustand von $\mathfrak{e}$ mit einer Markierung des RPN-Modells verknüpft: Aus dieser Markierung kann abgelesen werden, in welchem Zustand sich das modellierte System des Testablaufs befindet – bezogen auf mögliche Sequenzen der Ausführung von Testmoduln kann so den RPN-Markierungen folgende Semantik zugeordnet werden:

- Ist in der Stelle T_S eines Testsubnetzes kein Token vorhanden, wurde mit der Ausführung des entsprechenden Testmoduls noch nicht begonnen.

- Über die Existenz von Token in den Stellen T_D der Testsubnetze kann festgestellt werden, welche anderen Testmoduln schon ausgeführt wurden.

- Diejenigen Testmoduln, bei denen sich in der Stelle T_S eines und in der Stelle T_D kein Token befindet, werden gerade ausgeführt.

In Begriffen der DSS-Modellierung ergibt die EMRM-Analyse also – für sämtliche möglichen Kombinationen der Ausführung von Testmoduln – im Mittel optimale Einzel-Entscheidungen (auf Markov-Zustands-Ebene), die Ausführung von Testmoduln *unter bestimmten Bedingungen* zu starten oder aber zu unterlassen; dabei stellen diese Bedingungen den Zusammenhang zu schon ausgeführten und noch laufenden Tests innerhalb einer Sequenz her.

Zur Berechnung eines kompletten Testplans müssen die zustandsbezogenen Einzel-Entscheidungen auf geeignete Weise zusammengefaßt werden, d.h. in der durch den Erreichbarkeitsgraphen des RPN gegebenen Ordnung (ausgehend von der Anfangsmarkierung). Die Ergebnisse der EMRM-Analyse werden hierbei als

zusätzliche Einschränkung wirksam: Von der Entscheidungs-Optimierung als inaktiv markierte Rekonfigurationskanten (und alle über diese Kante mit der Anfangsmarkierung verbundenen Teile des Graphen) werden ebenfalls als inaktiv betrachtet.

In Begriffen der Petrinetz-Datenstrukturen selektiert also die EMRM-Analyse aus dem Erreichbarkeitsgraphen eines RPN einen Subgraphen (genauer gesagt eine die Anfangsmarkierung enthaltende Zusammenhangs-Komponente), der durch inaktive Rekonfigurationskanten begrenzt wird; übertragen auf die Terminologie des DSS wird auf diese Weise aus der Menge der möglichen Testpläne, die durch den Erreichbarkeitsgraphen beschrieben wird, ein (im Mittel) optimaler Plan ausgewählt.

7.1.2 Transformation der Analyseergebnisse in Testpläne

Der verwendete EMRM-Löser Penelope [dMŠ97] ermöglicht (wie in 3.2.3 beschrieben) sowohl die Bestimmung von stationären als auch zeitabhängigen, transienten Strategien: Während für die DSS-Modelle im stationären Fall eine feste Strategie bis zur Absorption bestimmt wird, ergibt die transiente Analyse eine Strategie, bei der im Hinblick auf eine für den Testlauf maximal zur Verfügung stehende Zeitspanne optimiert wird (d.h. unter Berücksichtigung der Wahrscheinlichkeit, daß Ereignisse wie z.B. die Beendigung eines Tests noch oder nicht mehr vor dem Ende des Planungshorizont-Zeitintervalls eintreten).

Stationäre Analyse

Für diesen Fall resultiert die EMRM-Analyse in einfachen Anweisungen »*Rekonfiguration durchführen/unterlassen*«, die in Form einer Tabelle präsentiert werden: Jedem Anfangszustand einer Rekonfigurationskante wird ein optimaler Ziel-Zustand zugeordnet (dies kann der End-, aber auch der Anfangszustand der Kante selbst sein, falls die Rekonfiguration nicht vorgenommen werden sollte).

Transiente Analyse

Hier wird durch den EMRM-Löser jedem Anfangszustand einer Rekonfigurationskante eine Liste von ***Aktivierungsphasen*** zugeordnet: Jede dieser Phasen gibt einen optimalen Folge-Zustand sowie den Endpunkt eines Zeitintervalls an, innerhalb dessen das System in den Zustand konfiguriert werden sollte (dies kann wieder der Anfangszustand sein, falls eine Rekonfigurationskante in einer Phase inaktiv sein soll). Die Aktivierungsphasen starten zum Beginn des Planungshorizont-Zeitintervalls und sind in absteigender zeitlicher Folge aufgeführt (d.h. der

Endpunkt der letzten Phase ist der Zeitpunkt 0, an dem der Planungshorizont erreicht ist).

Auswertungs-Algorithmus

Mit der in 7.1.1 beschriebenen Interpretation der Analyseergebnisse ergibt sich das unten beschriebene Vorgehen zur Auswertung der DSS-Modelle. Dabei werden die Testpläne gebildet, indem Pfade durch den Erreichbarkeitsgraphen des RPN unter Berücksichtigung der EMRM-Analyseergebnisse ermittelt werden – Zustände im Erreichbarkeitsgraphen, die nur über inaktive Rekonfigurationskanten mit der Anfangsmarkierung verbunden sind, werden als »unerreichbar« betrachtet; auf diese Weise werden die durch die EMRM-Optimierung als ungünstig errechneten Test-Sequenzen ausgesondert.

Stationär:

1. Stationäre EMRM-Analyse des RPN zu einer gegebenen Menge von Testmodul-Parametersätzen.

2. Aufbau einer Datenstruktur, die jeder Rekonfigurationskante im EMRM die zugehörige Rekonfigurationstransition sowie die Anfangs- und End-Markierung der Kante im RPN zuordnet.

3. Traversieren des RPN-Erreichbarkeitsgraphen unter Berücksichtigung der Aktivierung von Rekonfigurationsübergängen – jede traversierte Rekonfigurationskante wird in eine Anweisung bezüglich der Ausführung von Testmoduln umgesetzt:

   - Ist die Kante aktiv, wird das Testsubnetz, in dem die entsprechende Rekonfigurations-Transition liegt, aktiviert (d.h. das zugehörige Testmodul gestartet); die Traversierung wird am Endzustand der Kante fortgesetzt.
   - Andernfalls wird das Testsubnetz, in dem die entsprechende Rekonfigurations-Transition liegt, nicht aktiviert (d.h. die Ausführung des zugehörigen Testmoduls unterbleibt); die Traversierung wird an dieser Kante abgeschnitten, d.h. diejenigen Teile des Graphen, die jenseits inaktiver Rekonfigurationskanten liegen, werden ausgelassen.

Transient: Hier muß zusätzlich vor Beginn der EMRM-Analyse ein Planungshorizont festgelegt werden; bei der Traversierung des Erreichbarkeitsgraphen sind die Aktivierungsphasen der Rekonfigurationskanten zu berücksichtigen:

1. Transiente EMRM-Analyse des RPN zu einer gegebenem Kombination aus Planungshorizont[1] und Menge von Testmodul-Parametersätzen.

2. Aufbau einer Datenstruktur, die jeder Rekonfigurationskante im EMRM die zugehörige Rekonfigurationstransition, die Anfangs- und End-Markierung der Kante im RPN sowie die vom EMRM-Löser berechnete Folge von Aktivierungsphasen zuordnet.

3. Traversieren des RPN-Erreichbarkeitsgraphen, mit folgenden Unterschieden zum stationären Fall:

   - Die Aktivierungsphasen der Rekonfigurationskanten übertragen sich auf die Zustände: Jeder Zustand ist nur innerhalb der Vereinigung der aktiven Phasen der Rekonfigurationskanten, über die er mit der Anfangsmarkierung des RPN verbunden ist, erreichbar.
   - Liegt das Ende der Aktivierungsphase(n) einer Rekonfigurationskante vor dem Beginn der Aktivierungsphase(n) des Anfangszustands der Kante, dann wird die Kante als inaktiv betrachtet und die Traversierung jenseits der Kante abgeschnitten.

Die oben beschriebenen Algorithmen zur Berechnung von Testplänen aus den Ergebnissen der transienten EMRM-Analyse wurden in einem Software-Modul *dss-eval* implementiert; die Testplan-Berechnung für die DSS-Modelle wird vom Werkzeug automatisch als der EMRM-Analyse nachgeschalteter Schritt ausgeführt.

7.2 Analyse der DSS-Modelle

Um die eingeführten Verfahren auf ihre Anwendbarkeit zu überprüfen, wurde das DSS-Modell für die in 5.2 beschriebene, verteilte Anlage mittels PANDA und dem Penelope-Löser analysiert.

Alle Analysen wurden auf einem am Rechenzentrum der Universität Erlangen betriebenen Sun Enterprise Server 4000 mit RISC CPUs vom Typ UltraSPARC II (250MHz Taktfrequenz) durchgeführt; von den in der Maschine insgesamt installierten 8GB Hauptspeicher waren durch Betriebssystem-interne Beschränkungen nur 2GB pro Analyse-Schritt (d.h. Erreichbarkeitsgraph-Erzeugung, EMRM-Optimierung und Testplan-Generierung) adressierbar.

[1] Bei der beschriebenen Fallstudie wurde als Planungshorizont die Summe aller mittleren für die Testmodul-Ausführung benötigten Zeiten verwendet (wobei je Testmodul das Maximum der durchschnittlichen Zeitspannen für gescheiterten/erfolgreichen Verlauf verwendet wurde).

Dabei traten die bei der Auswertung praktisch relevanter stochastischer Modelle üblichen Schwierigkeiten auf:

7.2.1 Problematik der Validierung

Eine umfassende quantitative Validierung – im Sinne einer fundierten Bewertung der errechneten Testpläne mit Bezug zum realen System sowie im direkten Vergleich zu manuell erstellten Testplänen – ist nur unter hohem zusätzlichen Aufwand durchführbar:

- Um praxisbezogen optimierte Testpläne zu erhalten, müßte zunächst eine Kalibrierung der Testmodul-Parameter vorgenommen werden; dazu wäre eine Vielzahl von statistischen Parametern am modellierten System zu ermitteln.

- Zur Ermittlung von Referenzwerten müßten Testpläne mit großer Varianz auf dem Zielsystem ausgeführt und Kenngrößen (Zeitbedarf, Lasteinfluß auf Nutzanwendung etc.) gemessen werden. Dabei stellt allein die dafür notwendige, reproduzierbare Emulation realistischer Betriebsbedingungen auf dem Zielsystem ein Problem dar.

- Für einen aussagekräftigen Vergleich mit den durch das DSS berechneten Testplänen müßten diese Messungen zusätzlich über einen sehr großen Parameterraum erfolgen.

Da die erforderlichen Daten bisher nicht vorliegen und der zusätzliche Aufwand den Rahmen der vorliegenden Arbeit sprengen würde, wurde hier nur die Analysierbarkeit der aufgebauten Modelle an Hand des in 5.2.1 vorgestellten Systems überprüft.

7.2.2 Numerische Problematik

Zustandsraum-Explosion

Bei der automatischen Generierung der Zustandsräume entstehen aus den RPN sehr große EMRM; das DSS-Modell für das in 5.2.1 vorgestellte Fallstudien-System (d.h. das daraus abgeleitete RPN, s. Abb. 5.3) hat einen Zustandsraum, der sich mittels der derzeit in PANDA implementierten Algorithmen (die ohne Faltung oder Kompression von Zustandsraum-Teilmengen arbeiten) und auf den zur Zeit verbreiteten Rechnern mit 32-Bit-Architektur schon nicht mehr aufspannen läßt, da hier in der Regel nicht mehr als 4GB Hauptspeicher adressiert werden können.

Um die Modelle mit der zur Verfügung stehenden Hardware dennoch analysieren zu können, wurden am in 5.2.1 dargestellten Modell folgende Änderungen vorgenommen:

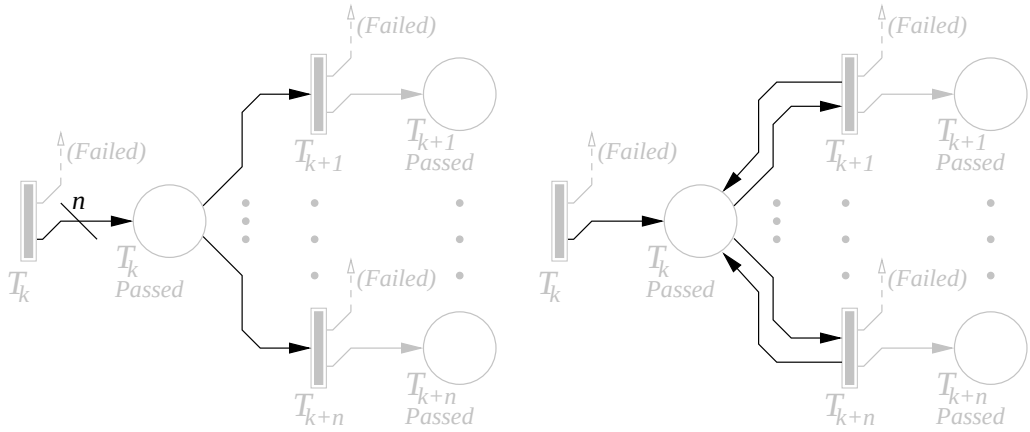
Verringerung des Modellierungs-Detailgrades: Die Tests der Funktionalität des Gesamtsystems (ABC_{TEST}), der Netzwerk-Schnittstellen der Knoten (A_n, B_n, C_n, A_h, B_h, C_h, A_s, B_s, C_s) sowie die Konsistenz-Überprüfungen der Datenbank-Kopien (A_{CONS}, B_{CONS}) wurden weggelassen – die entsprechenden Testsubnetze wurden durch zeitlose Transitionen ersetzt (d.h. der Verfeinerungs-Schritt beim Übergang vom Abhängigkeitenmodell zum RPN wurde für diese Testmoduln rückgängig gemacht).

Zustandsraum-Reduktion: Zusätzlich wurden die DSS-Modelle in 1-sichere Petrinetze transformiert. Dies ermöglicht im Erreichbarkeitsgraph-Generator von PANDA eine wesentliche effizientere Abspeicherung der RPN-Markierungen als Bit-Vektoren und ist ohne Änderung der Modell-Semantik möglich: Die als Testsubnetze verwendeten Teil-Strukturen sind ohnehin schon 1-sicher und müssen daher nicht modifiziert werden; die einzigen Stellen im RPN aus Abb. 5.3, in denen mehr als 1 Token auftreten kann, sind diejenigen, die Entscheidungsalternativen modellieren.

Soll die Möglichkeit zur gleichzeitigen Ausführung mehrerer Testmoduln modelliert werden, muß die Zahl der hier zur Verfügung stehenden Token der der im nächsten Schritt ausführbaren Testmoduln entsprechen (Abb. 7.1(a)). Dieselben möglichen Feuer-Sequenzen von Transitionen (d.h. ein äquivalentes RPN) können jedoch auch mit nur einem Token in den entsprechenden Stellen erzielt werden, falls die direkt nachfolgenden Transitionen ohne Zeitverzögerung feuern (d.h. zeitlose oder rekonfigurierend sind) und beim Feuern das »verbrauchte« Token über eine zusätzlichen Ausgangskante in die Ausgangsstelle zurück transportieren (Abb. 7.1(b)).

Mit der Erweiterung um die zusätzlich benötigen Kanten konnten daher die bisher verwendeten Kantenmultiplizitäten beseitigt werden; zur Herstellung der 1-Sicherheit des Gesamtnetzes war nur noch die Einführung einiger Inhibitorfunktionen notwendig (um mehrfaches Schalten von Transitionen mit der Folge von mehr als einem Token in einer der Stellen des Netzes zu unterbinden).

Das so veränderte DSS-Modell mit 16 Testsubnetzen ist in Abb. 7.2 dargestellt; sämtliche nachfolgenden Ausführungen zu Ergebnissen beziehen sich darauf. Das Modell war (mit einem Hauptspeicher-Bedarf von ca. 1,8GB) auf der oben erwähnten Anlage analysierbar – es enthält 106 Stellen und 94 Transitionen



(a) Modellierung Test-Sequenz ohne Tokenzahl-Beschränkung, wie in Abb. 5.2(a)

(b) 1-sichere Modellierung Test-Sequenz

Abbildung 7.1: Transformation der DSS-Modelle in 1-sichere Petrinetze

(46 zeitlose, 32 mit exponentiell verteilter Schaltverzögerung sowie 16 rekonfigurierende). Die Analyse ergab einen Erreichbarkeitsgraphen mit 1227603 Zuständen (714431 zeitbehaftete und 513172 zeitlose) und 1165965 Kanten (inkl. 468925 Rekonfigurationskanten).

Analyse-Zeitbedarf

Während auch die Generierung und Speicherung von Zustandsräumen in der Größenordnung von 10^6 Zuständen auf zur Zeit üblichen Rechenanlagen bei ausreichender Hauptspeicher-Ausstattung innerhalb relativ kurzer Zeit (d.h. für das oben beschriebene Modell in ca. 30 Minuten CPU-Rechenzeit) durchführbar ist, entfällt der Hauptanteil des für die Analyse benötigten Zeitbedarfs erwartungsgemäß auf die Entscheidungs-Optimierung des EMRM.

Der Zeitbedarf für die EMRM-Analyse ist hierbei zum Teil abhängig von den angegebenen Bewertungsmaßen – je günstiger bzw. wichtiger die Ausführung bzw. Ergebnisse der Tests bewertet (d.h. je mehr Rekonfigurationskanten als aktiv ermittelt) werden, desto länger dauert die Berechnung, da über mehr Entscheidungsalternativen optimiert werden muß.

Bei den durchgeführten Experimenten wurde die Optimierungs-Gewichtung κ in $\{1, 4, 7, 10, 20, 35, 50, 70, 90, 110, 130, 150\}$ variiert; dies resultierte bei einer Diskretisierungs-Schrittweite des Planungshorizont-Zeitintervalls von $0,1$ in einem Zeitbedarf zwischen 52 und 55 Stunden CPU-Rechenzeit für eine komplette Testplan-Generierung. Die Dauer der in 7.1.2 beschriebenen nachgeschalteten Berechnung von Testplänen aus den Ergebnissen der EMRM-Analyse lag dabei

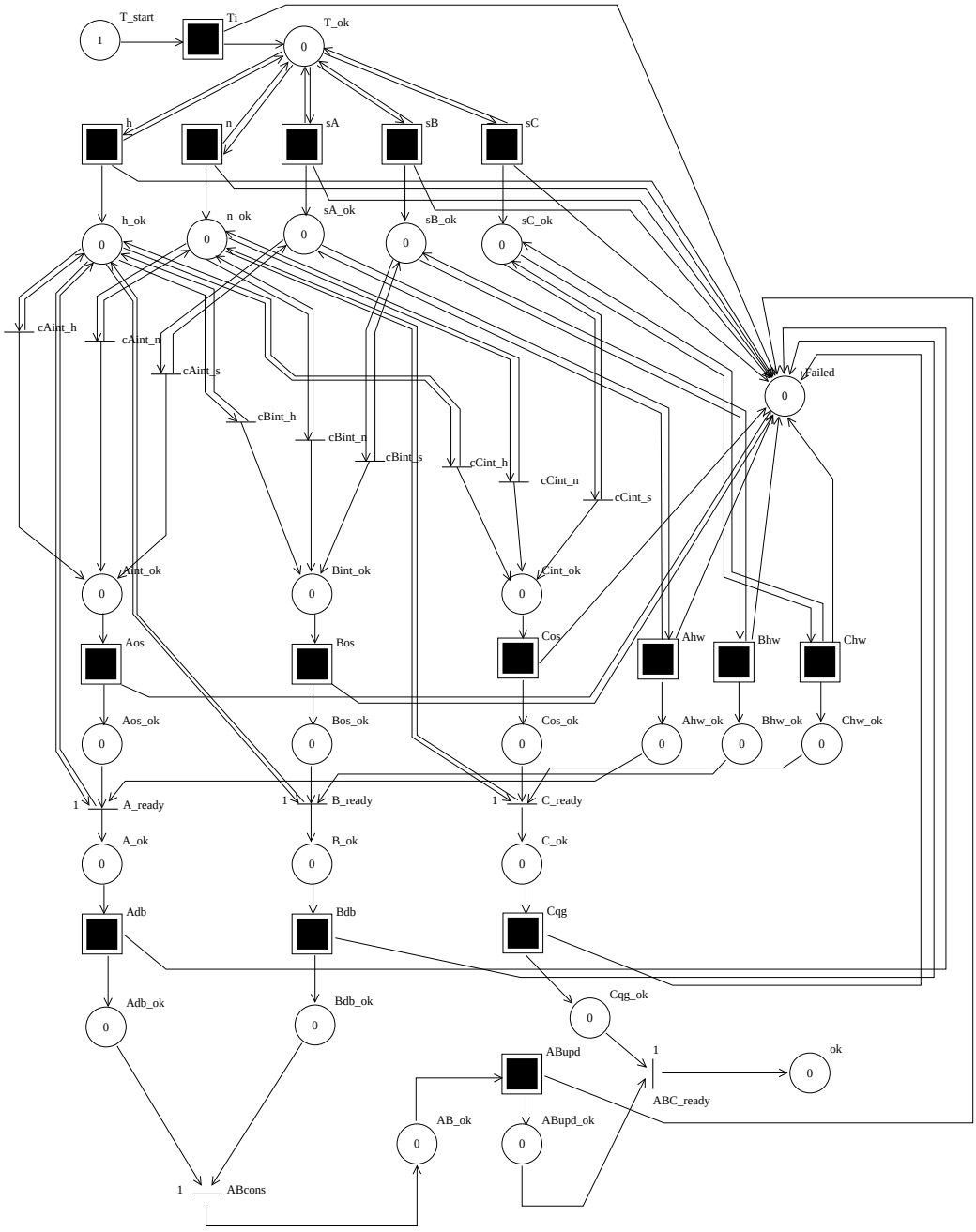


Abbildung 7.2: Reduziertes 1-sicheres RPN für Fallstudien-Cluster (mit PANDA erstellter Netzplan, vgl. Abb. 5.3)

mit ca. 30 Minuten in derselben Größenordnung wie der für die Erreichbarkeitsgraph-Generierung (d.h. war gegenüber der Dauer der EMRM-Analyse vernachlässigbar).

In Abb. 7.3 ist für die durchgeführten Experimente der prozentuale Anteil der von der Entscheidungs-Optimierung aktivierten Testmoduln und Rekonfigurationskanten in Abhängigkeit von der Optimierungsgewichtung dargestellt – an der Graphik wird ersichtlich, wie sich die Bewertung des durch das Testen erzielbaren Informationsgewinns in qualitativer Hinsicht (d.h. bezüglich der Frage, unter welchen Voraussetzungen ein Testmodul überhaupt ausgeführt werden sollte) in den generierten Testplänen niederschlägt: Je wichtiger (mit steigender Optimierungsgewichtung) die Ergebnisse der Tests eingestuft werden, desto mehr Tests werden für die Ausführung ausgewählt.

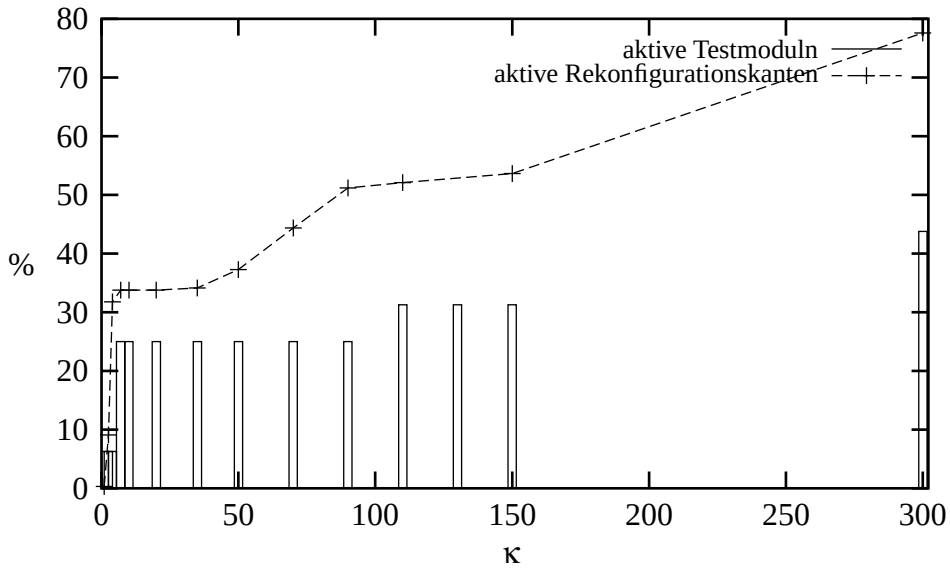


Abbildung 7.3: Einfluß der Optimierungsgewichtung κ auf den Anteil aktiver Rekonfigurationskanten/Testmoduln

Ein Beispiel für einen vom DSS generierten Testplan (für den Fall $\kappa = 70$) ist in Abb. 7.4 zu sehen: Es wird (auf den Zustand der Testmoduln bezogen) für die von Rekonfigurationstransitionen ausgelösten Zustandswechsel (d.h. den Start der Testmoduln) der erste Zeitpunkt angegeben, zu dem der Zustandswechsel zur Erzielung optimaler Erwartungswerte des Bewertungsmaßes vorgenommen werden sollte – in den durch einen Rahmen hervorgehobenen Zeilen wird z.B. der Start des Testmoduls $\mathbf{s}_B$ (d.h. das Schalten der Rekonfigurationstransition s_B) zum Zeitpunkt 289,6 vorgeschlagen, falls vorher die Testmoduln $\mathbf{T}$ und $\mathbf{s}_C$ erfolgreich ausgeführt wurden (Testmodul- und Transitionsbezeichner wie auf S. 75)[2].

[2]Durch die Struktur der Testsubnetze werden auch Zustandswechsel für den Fall gescheiterter Testausführungen vorgeschlagen, die vom Werkzeug angezeigt werden – bei der Ausführung des Testplans würden solche Zustandswechsel ignoriert, da damit der Fehler ja schon lokalisiert wäre.

```
dss-parm: Parametrizing DSS model 'without-int-1-safe':
    Loading test module parameter sets from 'without-int-1-safe.tmparms'...
    Parameter sets for 28 test-modules initialized.
    Parsing input Panda model 'without-int-1-safe.net'
        (output Panda model 'without-int-1-safe.dss.net')...
    Skipping 10 Panda subnets (no test-module parameter set defined):
        root cAint_h cAint_n cAint_s cBint_h cBint_n cBint_s cCint_h cCint_n cCint_s
    Generating Panda reward file 'without-int-1-safe.dss.reward'...
    Weighting factor 70 for information yield.
    Skipping 12 test-modules (not referenced in input net file):
        ABCtest Ah Bh Ch An Bn Cn Acons As Bcons Bs Cs
    Parametrized 16 impulse/15 rate function terms.
    Upper bound for MTTA: 500

    Analyzing model 'without-int-1-safe' by executing
        'panda -f without-int-1-safe.dss.net -rpn -threads 1 -1-safe'

PANDA version 2.56/Penelope Reconfiguration Strategy Optimization

Markov chain generation... done.
EMRM generation done... done.

Execution of the following command terminated with exit status 0:
  penelope <without-int-1-safe.dss.peninput >without-int-1-safe.dss.penoutput

dss-eval (v0.105): Evaluating DSS analysis results for model 'without-int-1-safe.dss':
    Parsing RG file (phase 1, RPN elements)... initialized 16 reconfiguration
        transitions and test module start/end indicator places.
    Parsing RG file (phase 2, generating RPN RG)... 1227603 states inserted.
    Parsing PTS file... parsed 97082 EMRM states with 4111153 activation phases
        (97082/121771 active/inactive reconfiguration edges, mission time starts at 499.9).
    Parsing RG file (phase 3, generating RPN markings)... 199139 markings initialized.
    RPN reachability analysis for DSS test plan:
        0. {}
         ->{Ti:started} reachable in: (482.7, 0.0]

        1. {Ti:finished}
         ->{Ti:finished sC:started} reachable in: (289.6, 0.0]

        2. {Ti:finished sC:started(pass)}
         ->{Ti:finished sB:started sC:started(pass)} reachable in: (44.2, 0.0]

        2. {Ti:finished sC:started(fail)}
         ->{Ti:finished sB:started sC:started(fail)} reachable in: (289.6, 0.0]

        2. {Ti:finished sC:finished}
         ->{Ti:finished sB:started sC:finished} reachable in: (289.6, 0.0]

        3. {Ti:finished sB:started(pass) sC:started(pass)}
         ->{Ti:finished sA:started sB:started(pass) sC:started(pass)} reachable in: (37.0, 0.0]

        3. {Ti:finished sB:started(fail) sC:started(pass)}
         ->{Ti:finished sA:started sB:started(fail) sC:started(pass)} reachable in: (44.2, 0.0]

        3. {Ti:finished sB:started(pass) sC:started(fail)}
         ->{Ti:finished sA:started sB:started(pass) sC:started(fail)} reachable in: (289.6, 0.0]

        3. {Ti:finished sB:started(fail) sC:started(fail)}
         ->{Ti:finished sA:started sB:started(fail) sC:started(fail)} reachable in: (289.6, 0.0]

        3. {Ti:finished sB:started(pass) sC:finished}
         ->{Ti:finished sA:started sB:started(pass) sC:finished} reachable in: (40.4, 0.0]

        3. {Ti:finished sB:started(fail) sC:finished}
         ->{Ti:finished sA:started sB:started(fail) sC:finished} reachable in: (289.6, 0.0]

        3. {Ti:finished sB:finished sC:started(pass)}
         ->{Ti:finished sA:started sB:finished sC:started(pass)} reachable in: (40.4, 0.0]

        3. {Ti:finished sB:finished sC:started(fail)}
         ->{Ti:finished sA:started sB:finished sC:started(fail)} reachable in: (289.6, 0.0]

        3. {Ti:finished sB:finished sC:finished}
         ->{Ti:finished sA:started sB:finished sC:finished} reachable in: (289.6, 0.0]

    126 states traversed.
```

Abbildung 7.4: Beispiel Ausgabe DSS für reduziertes Modell Fallstudien-Cluster (s. Abb. 7.2), Optimierungsgewichtung $\kappa = 70$

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Mit der vorliegenden Arbeit konnte gezeigt werden, daß sich die Erstellung von Testplänen für verteilte Systeme auf Markov-Entscheidungsmodelle abbilden läßt. Durch die Anwendung einer für die Entscheidungsmodellierung passend erweiterten Klasse von stochastischen, bewerteten Petrinetzen wird es dabei möglich, die für die Testplan-Generierung bei realen Systemen nötigen Markov-Modelle von erheblicher Größe und Komplexität aus anwendungsbezogenen Daten über die zur Verfügung stehenden Tests sowie der Kenntnis über die Struktur des zu testenden Systems zu erzeugen.

Die Repräsentation logischer Zusammenhänge zwischen einzelnen Modellteilen und die stochastische Parametrisierung wurden mittels eines hierarchisierten Aufbaus der Modelle separiert. Dadurch konnten die Berechnung von modellinhärenten Parametern und die Angabe der Optimierungskriterien, die Übernahme dieser Werte in das stochastische Modell sowie die Aufstellung, Analyse und Auswertung der DSS-Modelle automatisiert werden: Wenn die Struktur und die Abhängigkeiten innerhalb des Zielsystems als Petrinetz beschrieben und die Tests (relativ zueinander) bezüglich des Kosten-/Nutzen-Effekts klassifiziert sind, wird daraus von einem Werkzeug ein (im stochastischen Mittel) optimaler Testplan ohne die Notwendigkeit weiterer manueller Eingriffe berechnet.

Die Steuerung der Ziele der Markov-Entscheidungs-Optimierung kann hier sowohl feingranular (nämlich über die Klassifizierung der Tests) als auch auf einfache Weise (d.h. durch die Anpassung der globalen Gewichtung der Kosten in Bezug zum Nutzen des gesamten Testvorgangs) vorgenommen werden.

Sollen die aufgebauten Modelle der Analyse unterzogen werden, stößt man auf die für große Petrinetz-Modelle typischen Schwierigkeiten: Wird zu detailliert modelliert, entstehen Zustandsräume, die sich auf derzeit verfügbarer Hard-

ware nicht mehr komplett im Hauptspeicher fassen lassen. Dennoch ist am hier vorgestellten Modell zu sehen, daß sich auch mit heute existierenden Anlagen im üblichen Server-Bereich durchaus Testpläne von praktischer Relevanz mit ausreichendem Detailgrad berechnen lassen. Zusätzlich relativiert wird dieses Problem durch die schnell fortschreitende Entwicklung im Bereich der CPUs und Betriebssysteme (64-Bit-Architekturen) im Zusammenhang mit den immer größeren und billigeren Hauptspeicher Bausteinen – bei Kosten von derzeit ca. DM 3,– pro Megabyte finden auch Anlagen mit mehreren Gigabyte Hauptspeicher zunehmende Verbreitung.

Obwohl in der vorliegenden Arbeit – bedingt durch den hohen Zeitbedarf bei der Analyse von praxisrelevanten Modellen sowie auf Grund des für Kalibrierung und Validierung nötigen zusätzlichen Aufwandes – keine Überprüfung der generierten Testpläne am realen System erfolgen konnte, wurde die Lösbarkeit der DSS-Modelle an Hand eines realitätsbezogenen Beispiels nachgewiesen.

Der hohe Zeitbedarf dafür muß zwar ebenfalls als Einschränkung für den Einsatz des DSS gesehen werden (er führte z.B. dazu, daß mit dem Fallstudien-Modell keine umfassenden Experiment-Reihen durchgeführt werden konnten); allerdings bedeutet dies keine schwerwiegende Beeinträchtigung des potentiellen Nutzens des Systems: Die Testpläne können für verschiedene interessierende Rahmenbedingungen (Systemkonfigurationen, Optimierungsziele etc.) im Voraus und »Offline« berechnet werden – die zur Vorhaltung der Testpläne im Bedarfsfall nötigen Ressourcen sind gering.

8.2 Ausblick

Interessante Themen für an die vorliegende Arbeit anschließende Untersuchungen bieten sich zunächst konkret auf das DSS bezogen an:

- Bei einer Validierung der mit dem DSS berechneten Testpläne wäre insbesondere die Rechtfertigung der Annahme von exponentiell verteilten Werten für die Dauer der Tests zu überprüfen; in diesem Zusammenhang könnte auch die Approximation andersartiger Verteilungen durch Phasen-Verschaltung von Exponential-Verteilungen (wie z.B. der in PANDA für die SRN-Analyse schon implementierte Erlang-Verteilung) in die Modellierung der Testmoduln einbezogen werden.

  Durch die Anwendung von phasenverteilten Transitionen mit verschiedenen Gedächtnisstrategien könnte die Modellierung der Tests auch deutlich detaillierter erfolgen (z.B. könnte hierdurch auch der Abbruch von Testvorgängen abgebildet werden) – allerdings wäre dies sicherlich mit erheblich vergrößerten Zustandsräumen verbunden.

- Durch eine parallelisierte Reimplementierung des EMRM-Lösers könnten der Analyse-Zeitbedarf für große Modelle deutlich gesenkt werden; dabei könnte ein ähnliches Verfahren eingesetzt werden wie die schon zur Zeit in PANDA (zur transienten SRN-Analyse auf symmetrischen Multiprozessoren) implementierte parallele Uniformization.

- Im Sinne einer weiteren Formalisierung beim Aufbau der DSS-Modelle sowie einer Integration der DSS-Modellierung in den Systementwurfs-Prozess könnte sich die Erstellung der Abhängigkeitenmodelle aus anwendungsbezogenen Beschreibungen als günstig erweisen – hier kommen sowohl erweiterte Fehlerbäume [Buc00] als auch die Generierung der Petrinetze aus UML-Modellen des Zielsystems [DHK99b, DHK99a] als Ausgangspunkt in Frage.

- Im Hinblick auf einen möglichen Online-Betrieb des DSS wäre die Integration einer Rückkoppelungs-Komponente zu untersuchen: Nach der Ausführung der generierten Testpläne könnten die Ergebnisse dynamisch in Änderungen an den Modellen überführt und auf diese Weise der durch die Tests ermittelte System-Zustand bei der Erzeugung zukünftiger Testpläne berücksichtigt werden.

- Das DSS liefert die Grundlagen für eine wirtschaftliche Aspekte betonende Sichtweise auf den Prozess des Testens komplexer Systeme; es wäre zu überprüfen, ob die Voraussetzungen für die Anwendung allgemein genug gehalten sind (oder welche Modifikationen notwendig wären), um den Einsatz auch auf den Test anderer technischer Anlagen als den von Server-Clustern ausdehnen zu können.

Allgemeiner betrachtet wäre schließlich zu untersuchen, in wie weit die Verfügbarkeit eines Werkzeugs zur Modellierung auf Petrinetz-Ebene der Markov-Entscheidungstheorie weitere Anwendungsgebiete eröffnet.

Die hier gezeigte, strukturierte Vorgehensweise zu Aufbau, Parametrisierung und Auswertung von Entscheidungsmodellen ließe sich sicherlich ganz ähnlich auf andere Bereiche übertragen, in denen Entscheidungen aus diskreten Mengen von Alternativen unter durch zufällige Ereignisse geprägten Rahmenbedingungen zu treffen sind, bei denen (zumindest approximativ) die Markov-Eigenschaft angenommen werden kann – es stellt sich die Frage nach dem Potential der RPN-Modellierung als Kern-Komponente einer ganzen Klasse von »Decision Support Systems«.

Anhang

Anhang A

Spezifikation von Bewertungsmaßen für PANDA

Die in 2.1.1 angegebene Definition von Bewertungsmaßen muß für die Anwendung bei der Petrinetz-Modellierung in eine Beschreibungssprache umgesetzt werden, die von einem Werkzeug wie PANDA ausgewertet und interpretiert werden kann. Die Struktur der Bewertungsmaß-Spezifikation und die Integration in PANDA wurden in 2.2.2 beschrieben; hier wird eine kontextfreie Grammatik für die Beschreibung von SRN-Bewertungsmaßen sowie Beispiele für die Anwendung der Beschreibungssprache angegeben.

A.1 Grammatik

Die Syntax der Beschreibungssprache lehnt sich an die in [Ger97] beschriebene an; die Grammatik wird hier in erweiterter Backus-Naur-Form angegeben. Dabei sind terminale Symbole **fett** gedruckt, { Ausdruck } steht für null- oder mehrfaches Auftreten eines Symbols, [Ausdruck] für null- oder einfaches Auftreten, und Ausdruck1 | Ausdruck2 zeigt das alternative Auftreten von Symbolen an.

A.1.1 Bewertungsfunktionen

Die Grundlage für die Angabe der Bewertungsmaße bilden die Bewertungsfunktionen. Diese bestehen aus einem Bezeichner (der eindeutig gewählt werden muß) und Funktionsrumpf[1]. Der Funktionsrumpf kann Aufrufe von charakterisieren-

[1] Aus implementierungstechnischen Gründen müssen dem Parser-Generator für Zustands- bzw. Impulsmaße jeweils separate grammatikalische Konstruktionen angegeben werden (r_rew bzw. i_rew), weil auf unterschiedliche charakterisierende Funktionen zugegriffen wird. Da die beiden Grammatiken aber ansonsten identisch sind, wird hier nur die allgemeine Form aufgeführt.

den Funktionen, Verknüpfungen mittels arithmetischer Operatoren sowie bedingte Verzweigungen (if–then–else) enthalten; Ausdrücke im Funktionsrumpf können beliebig mittels Klammern gruppiert werden.

Bewertungsfunktionen liefern (wie alle Ausdrücke) als Datentyp Zahlenwerte zurück und können so miteinander verrechnet werden; die Definition beinhaltet auch die Möglichkeit, daß sich Bewertungsfunktionen untereinander aufrufen:

```
rew        ↪  rew_name rew_body

rew_name   ↪  string

rew_body   ↪  ( rew_body )
           |  arith_op1 rew_body
           |  rew_body arith_op2 rew_body
           |  arith_opn rew_list
           |  IF ( pred ) THEN ( rew_body ) [ ELSE ( rew_body ) ]
           |  number
           |  rew_basic
           |  rew_predef
           |  rew_name
rew_list   ↪  ( rew {, rew} )
```

Die arithmetischen Operatoren können ein- (arith_op1), zwei- (arith_op2) und n-stellig (arith_opn) sein. Es sind die Grundrechenarten, die Quadratwurzel (**sqrt**) sowie Maximum, Minimum und arithmetisches Mittel implementiert; für die Operatoren gelten die normalen Vorrangregeln:

```
arith_op1  ↪  – | SQRT
arith_op2  ↪  + | – | * | / | ^
arith_opn  ↪  min | max | mean
```

Mit den Verzweigungen können die von Bewertungsfunktionen zurückgelieferten Werte verglichen werden (d.h. mit ihnen und den charakterisierenden Funktionen kann jedes Element des Zustandsraums des Petrinetzes abgefragt werden); als Bedingungen dienen Prädikate, die aus booleschen und Vergleichs-Operatoren bestehen können:

```
pred       ↪  ( pred )
           |  NOT pred
           |  pred bool_arith pred
           |  rew_body compare rew_body
           |  TRUE | FALSE

bool_arith ↪  AND | OR | EOR | NOR | NAND
compare    ↪  = | <> | > | < | >= | <=
```

Zahlen werden als ganze und Fließkommazahlen in der üblichen Form akzeptiert. Bezeichner (string) können aus Buchstaben, Ziffern und dem Unterstrich (_) aufgebaut werden; für Bezeichner von Netzelementen (Stellen und Transitionen) ist zusätzlich der Doppelpunkt zulässig (da Bezeichner von Netzelementen in PANDA diesen zur Zuordnung in der Subnetz-Struktur enthalten können):

number	↪	real | integer
integer	↪	digit {digit}
real	↪	digit {digit} [**.** {digit}] [exponent]
	|	{digit} **.** digit {digit} [exponent]
exponent	↪	**e** [**+**|**–**] digit {digit}
string	↪	letter {letter | digit | _ }
net_element	↪	letter {letter | digit | _ | **:** }
letter	↪	**a** | ... | **z** | **A** | ... | **Z**
digit	↪	**0** | ... | **9**

A.1.2 Charakterisierende Funktionen

Für die Zugriffe auf Elemente des Erreichbarkeitsgraphen werden in den Bewertungsfunktionen die auf S. 31 beschriebenen charakterisierenden Funktionen sowie drei zusätzlich als Abkürzung definierte Funktionen eingesetzt; die Grammatik definiert:

r_rew_basic	↪	**mark(** place_name **)**
	|	**enabled(** place_name **)**
i_rew_basic	↪	**fire(** trans_name **)**
	|	**rate(** trans_name **)**
	|	**probfire(** trans_name **)**
r_rew_predef	↪	**maxmark**
	|	**minmark**
	|	**summark**
place_name	↪	net_element
trans_name	↪	net_element

Dabei sind **mark** und **enabled** bzw. **fire** und **rate** die in 2.2.2 definierten zustands- bzw. zustandsübergangsselektierenden Funktionen (als Argumente werden Namen von Stellen bzw. Transitionen angegeben). Mit **maxmark** (**minmark**) kann die Anzahl der Token in der Stelle abgefragt werden, die im jeweiligen Zustand des Petrinetzes die maximale (minimale) Zahl von Token enthält, **summark** liefert die Zahl aller Token, die im jeweiligen Zustand im Netz vorhanden sind.

Bewertungsfunktionen die Zustandsmaße modellieren, können nur aus den (zustandsselektierenden) charakterisierenden Funktionen vom Typ r_rew_basic (bzw. den vordefinierten Funktionen vom Typ r_rew_predef) gebildet werden, während zur Darstellung von Impulsmaßen sowohl zustands- als auch übergangsselektierende charakterisierende Funktionen (vom Typ i_rew_basic) verwendet werden können (da Impulsmaße auch von der Markierung abhängig sein können, in der sich das Netz vor dem jeweiligen Zustandsübergang befand).

A.1.3 Ergebnisfunktionen

Die Ergebnisfunktionen (vgl. S. 32) zur Abfrage der Resultate der stochastischen Analyse sind analog den Bewertungsfunktionen aufgebaut:

```
measure_func      ↪   meas_func_name meas_func_body
meas_func_name    ↪   string
meas_func_body    ↪   ( meas_func_body )
                    |  arith_op1 meas_func_body
                    |  meas_func_body arith_op2 meas_func_body
                    |  number
                    |  result_measure
                    |  meas_func_name
```

In diesem Fall können Ausdrücke aus arithmetischen Verknüpfungen von stochastischen Operatoren und anderen Ergebnisfunktionen (darunter einigen vordefinierten Funktionen) gebildet werden. Für die Grundbausteine wie Zahlen und Bezeichner sowie die hier implementierten arithmetischen Operatoren werden dieselben Ausdrücke verwendet wie in A.1.1.

Als stochastischer Operator ist in PANDA bis jetzt nur der Erwartungswert (**E**) implementiert; er wird angewendet auf die in A.1.1 definierten Bewertungsfunktionen. Gemäß der auf S. 24 eingeführten Definition der Bewertungsmaße hat der Operator zwei Argumente, wobei als erstes eine Zustands- und als zweites eine Impuls-Bewertungsfunktion anzugeben sind (für beide Argumente kann auch **NULL** eingesetzt werden, falls entsprechende Bewertungsfunktionen nicht berechnet werden sollen). Zusätzlich zum Erwartungswert können auch einige komplett vordefinierte Ergebnisfunktionen angewendet werden, denen nur noch die Bezeichner von Stellen bzw. Transitionen übergeben werden müssen:

```
result_measure      ↪   E[ [r_rew_name | NULL] , [i_rew_name | NULL ] ]
                      |  predef_res_measure

predef_res_measure  ↪   ENTOKEN( place_name )
                      |  ENOTEMPTY( place_name )
                      |  ETHROUGHPUT( trans_name )
                      |  EACTIVE( trans_name )
```

Dabei implementieren die vordefinierten Ergebnisfunktionen die Standardresultate der GSPN-Analyse: Durchschnittlich erwartete Zahl von Token pro Stelle (**ENOTOKEN**) und Wahrscheinlichkeit daß eine Stelle mindestens ein Token enthält (**ENOTEMPTY**), sowie Durchsatz (**ETHROUGHPUT**) und Wahrscheinlichkeit der Feuerbereitschaft (**EACTIVE**) von Transitionen.

A.2 Beispiele

Ein einfaches Beispiel für ein (zustandsabhängiges) Impulsmaß ist die folgende Bewertungsfunktion:

```
reward_func2(
        IF (enabled(trans1) AND (mark(place5) < mark(place8)) THEN (
                rate(trans2)
        ) ELSE (
                IF (minmark < 3) THEN (
                        probfire(trans3)
                ) ELSE (
                        probfire(trans4)
                )
        )
)
```

Hier wird die effektive Feuerwahrscheinlichkeit der Transition trans4 bzw. trans3 zurückgeliefert (abhängig davon, ob im aktuellen Zustand des Netzes mindestens 3 Token vorhanden sind), bzw. (unter der Bedingung daß Transition trans1 feuerbereit ist und in Stelle place8 mehr Token sind als in Stelle place5) die effektive Rate der Transition trans2.

Als Beispiel für eine simple Ergebnisfunktion liefert

```
covariance(
        E[NULL, product_impulse] - E[NULL, factor1] * E[NULL, factor2]
)
```

die Kovarianz von factor1 und factor2, wenn factor1 und factor2 als zwei Impuls-Bewertungsfunktionen und ihr Produkt als

```
product_impulse(
        factor1 * factor2
)
```

definiert ist.

Literaturverzeichnis

[AA86] ABRAHAM, J.A. und V.K. AGARWAL: *Test Generation for Digital Systems*. In: PRADHAN, D.K. (Herausgeber): *Fault-Tolerant Computing: Theory and Techniques*, Band 1, Kapitel 1, Seiten 1 – 94. Prentice-Hall, Eaglewood Cliffs, New Jersey (USA), 1986.

[ABF90] ABRAMOVICI, M., M.A. BREUER und A.D. FRIEDMAN: *Digital Systems Testing and Testable Design*. W. H. Freeman and Company, Oxford (UK), 1990.

[ACDK97] ALLMAIER, S., G. CSERTÁN, S. DALIBOR und D. KREISCHE: *Parallel Reachability Graph Generation in Stochastic Modeling on Shared and Distributed Memory Machines*. Technischer Bericht, IMMD III, Universität Erlangen-Nürnberg, Erlangen, 1997.

[AD97] ALLMAIER, S. und S. DALIBOR: *PANDA – Petri net ANalysis and Design Assistant*. In: *Tools Descriptions*, Seiten 58 – 60, St. Malo, Juni 1997. 9th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation.

[ADK97] ALLMAIER, S., S. DALIBOR und D. KREISCHE: *Parallel Graph Generation Algorithms for Shared and Distributed Memory Machines*. In: *Proc. Parallel Computing Conference (ParCo '97)*, Bonn, September 1997.

[AK99] ALLMAIER, S. und D. KREISCHE: *Parallel Approaches to the Numerical Transient Analysis of Stochastic Reward Nets*. In: *Proc. IEEE 20th Int. Conf. on Application and Theory of Petri Nets (ICATPN '99)*, Williamsburg, 1999. Springer Verlag (LNCS), to appear.

[AKH97] ALLMAIER, S., M. KOWARSCHIK und G. HORTON: *State-Space Construction and Steady-State Solution of GSPNs on a Shared-Memory Multiprocessor*. In: *Proc. IEEE Int. Workshop Petri Nets*

and Performance Models (PNPM '97), St. Malo, Juni 1997. IEEE Comp. Soc. Press.

[All98] ALLMAIER, S.: *A reward-based result measure concept for GSPNs*. Technischer Bericht, IMMD III, Universität Erlangen-Nürnberg, Erlangen, 1998.

[AMBC84] AJMONE MARSAN, M., G. BALBO und G. CONTE: *A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems*. ACM Trans. Comput. Systems, 2(2):Seiten 93 – 122, 1984.

[AMBC86] AJMONE MARSAN, M., G. BALBO, and G. CONTE: *Performance Models of Multiprocessor Systems*. The MIT Press, Cambridge / London, 1986.

[AMBC+95] AJMONE MARSAN, M., G. BALBO, G. CONTE, S. DONATELLI, and G. FRANCESCHINIS: *Modelling with Generalized Stochastic Petri Nets*. John Wiley, Chichester, 1995.

[Ave98a] AVERKAMP, P.: *Hochverfügbare Datenbanklösungen durch Clustering*. iX Magazin, 5/98: Seiten 126 – 131, Mai 1998.

[Ave98b] AVERKAMP, P.: *Kopplung von Highend-Servern*. iX Magazin, 2/98: Seiten 100 – 104, Februar 1998.

[Buc00] BUCHACKER, K.: *Definition und Auswertung erweiterter Fehlerbäume für die Zuverlässigkeitsanalyse technischer Systeme*. Doktorarbeit, Universität Erlangen-Nürnberg, Juli 2000.

[CBC+93] CIARDO, G., A. BLAKEMORE, P.F. CHIMENTO, J.K. MUPPALA und K.S. TRIVEDI: *Automated Generation and Analysis of Markov Reward Models using Stochastic Reward Nets*. In: MEYER, C.D. und R.J. PLEMMONS (Herausgeber): *Linear Algebra, Markov Chains and Queuing Models*, Band 48 der Reihe *The IMA Volumes in Mathematics and its Applications*, Seiten 145 – 193. Springer, New York, 1993.

[Cha65] CHANG, H.Y.: *An Algorithm for Selecting an Optimum Set of Diagnostic Tests*. IEEE Transactions on Electronic Computers, EC-14(5):Seiten 706 – 711, Oktober 1965.

[Cha68a] CHANG, H.Y.: *A Distinguishability Criterion for Selecting Efficient Diagnostic Tests*. In: *AFIPS Conference Proceedings*, Band 32

der Reihe *Spring Joint Computer Conference*, Seiten 529 – 534, Atlantic City, New Jersey (USA), April 1968. AFIPS.

[Cha68b] CHANG, H.Y.: *Figures of Merit for the Diagnostics of a Digital System*. IEEE Transactions on Reliability, R-17 (no. 3): Seiten 147 – 153, September 1968.

[CMM70] CHANG, H.Y., E. MANNING und G. METZE: *Fault Diagnosis of Digital Systems*. John Wiley, London, 1970.

[CMT89] CIARDO, G., J. MUPPALA und K.S. TRIVEDI: *SPNP: Stochastic Petri net package*. In: *Proc. IEEE 3rd Int. Workshop Petri Nets and Performance Models (PNPM '89)*, Seiten 142 – 151, Kyoto, 1989.

[DCP88] DAL CIN, M. und T. PHILIPP: *Expertensysteme für die Fehlerdiagnose*. it, 4/88, 1988.

[Des98] DESEIVE, S.: *Einführung von Reward-Maßen bei der Modellierung mit stochastischen Petri-Netzen*, 1998. Diplomarbeit im Fach Informatik, IMMD III, Universität Erlangen-Nürnberg.

[DHK99a] DAL CIN, M., G. HUSZERL und K. KOSMIDIS: *Quantitative evaluation of dependability critical systems based on guarded statechart models*. In: *Proc. HASE 99, Fourth IEEE Int. Symposium on High Assurance Systems Engineering*, Washington, November 1999.

[DHK99b] DAL CIN, M., G. HUSZERL und K. KOSMIDIS: *Transformation of guarded statecharts for quantitative evaluation of embedded systems*. In: *Proc. lOth European Work-shop on Dependable Computing*, Seiten 143 – 148, Wien, 1999. Österreichische Computer Gesellschaft.

[dM92] DE MEER, H.: *Transiente Leistungsbewertung und Optimierung rekonfigurierbarer fehlertoleranter Rechensysteme*. Doktorarbeit, Universität Erlangen-Nürnberg, Erlangen, Oktober 1992.

[dMD96] DE MEER, H. und O.-R. DÜSTERHÖFT: *Controlled Stochastic Petri-Nets*. Technischer Bericht FBI - HH - B - 193/96, Universität Hamburg, Hamburg, Dezember 1996.

[dMDF99] DE MEER, H., O.-R. DÜSTERHÖFT und S. FISCHER: *COSTPN for Modeling and Control of Telecommunication Systems*. In: DIAZ, M. (Herausgeber): *Applications of Petri Nets to Communication*

Networks. Springer Verlag, April 1999. Special Issue of Advances in Petri Nets (LNCS 1605).

[dMDT94] DE MEER, H., M. DAL CIN und K. TRIVEDI: *Guarded Repair of Dependable Systems*. Theoretical Computer Science, Band 128: Seiten 179 – 210, 1994.

[dMM91] DE MEER, H. und H. MAUSER: *Optimal Control of Responsive and Reconfigurable Systems*. Technischer Bericht, IMMD IV, Universität Erlangen-Nürnberg, Erlangen, 1991.

[dMŠ97] DE MEER, H. und H. ŠEVČÍKOVÁ: *PENELOPE – dependability evaluation and the optimization of performability*. In: MARIE, R., B. PLATEAU, M. CALZAROSSA und G. RUBINO (Herausgeber): *Computer Performance Evaluation, Modelling Techniques and Tools*, Seiten 19 – 31, St. Malo, Juni 1997. 9th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Springer Verlag (LNCS 1245).

[EP90] EICHLER, L. und H.W. POHL: *Diamond: Ein intelligentes Diagnosesystem für parallele Rechnerarchitekturen*. Fachgruppe: Fehlertolerierende Rechnersysteme, Dezember 1990.

[Ger97] GERMAN, R.: *SPNL: Processes as Language-Oriented Building Blocks of Stochastic Petri Nets*. In: MARIE, R., B. PLATEAU, M. CALZAROSSA und G. RUBINO (Herausgeber): *Computer Performance Evaluation, Modelling Techniques and Tools*, Seiten 123 – 134, St. Malo, Juni 1997. 9th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Springer Verlag (LNCS 1245).

[Har97] HARMS, U.: *LSF: Workstation-Cluster ersetzen Host bei der DASA Airbus*. iX Magazin, 5/97: Seiten 138 – 141, Mai 1997.

[HF96] HAVERKORT, B.R. und L.J.N. FRANKEN: *Reconfiguring Distributed Systems using Markov-Decision Models*. In: SPANIOL, O., C. LINNHOFF-POPIEN und B. MEYER (Herausgeber): *Short paper proceedings »Trends in Distributed Systems 1996«*, Band 17, Aachener Beiträge zur Informatik, Seiten 155 – 168, Aachen, 1. – 2. Oktober 1996. Verlag der Augustinus Buchhandlung.

[HK95] HEIN, A. und D. KREISCHE: *An Environment for the Simulation of Timed-Transition Petri Nets*. Technischer Bericht, IMMD III, Universität Erlangen-Nürnberg, Erlangen, August 1995.

[Hor95] HORTON, G.: *A Parallel Multi-Level Solution Method for Large Markov Chains*. In: KEYES, D.E., Y. SAAD und D.G. TRUHLAR (Herausgeber): *Domain-Based Parallelism and Problem-Decomposition Methods in Computational Science and Engineering*. SIAM, Philadelphia, 1995.

[HVMG82] HARTMANN, C.R.P., P.K. VARSHNEY, K.G. MEHROTRA und C.K. GERBERICH: *Application of Information Theory to the Construction of Efficient Decision Trees*. IEEE Transactions on Information Theory, IT-28(4): Seiten 565 – 577, Juli 1982.

[Kim86] KIME, C.R.: *System Diagnosis*. In: PRADHAN, D.K. (Herausgeber): *Fault-Tolerant Computing: Theory and Techniques*, Band 2, Kapitel 8, Seiten 577 – 632. Prentice-Hall, Eaglewood Cliffs, 1986.

[Kra98] KRAUSS, G.: *Classification Scheme for Test-Modules in a Diagnosis Support System*, 1998. Abschlußarbeit zum Euro Master of Science, IMMD III, Universität Erlangen-Nürnberg.

[Man64] MANDELBAUM, D.: *A Measure of Efficiency of Diagnostic Tests Upon Sequential Logic*. IEEE Transactions on Electronic Computers, EC-13(5), Oktober 1964.

[Mat96] MATTAUCH, G.: *Einführung graphisch hierarchischer Strukturen auf Petri-Netzen für ein Tool zur Modellierung mit GSPNs*, 1996. Studienarbeit im Fach Informatik, IMMD III, Universität Erlangen-Nürnberg.

[Mau90] MAUSER, H.: *Implementierung eines Optimierungsverfahrens für rekonfigurierbare Systeme*, 1990. Studienarbeit im Fach Informatik, IMMD IV, Universität Erlangen-Nürnberg.

[MB90] MASON, T. and D. BROWN: *lex & yacc*. O'Reilly & Associates, Sebastopol, 1990.

[McR95] MCREE, R.: *Testing and Diagnosing Managed Networks*. IEEE Design & Test of Computers, 12(4): Seiten 68 – 80, Dezember 1995.

[Met93] META SOFTWARE CORPORATION, Cambridge: *Design/CPN Tutorial for X-Windows*, version 2.0 edition, 1993.

[Mol81] MOLLOY, M.K.: *On the integration of delay and throughput measures in distributed processing models*. Doktorarbeit, University of California, Los Angeles, 1981.

[PCC93] PAULK, M.C., B. CURTIS, and M.B. CHRISIS: *Capability Maturity Model for Software, Version 1.1*. Technical Report CMU/SEI-93-TR, Software Engineering Institute, Februar 1993.

[Phi91] PHILIPP, T.: *An Expert System Shell for the Diagnosis of Parallel Computers*. In: *Proc. 5th Int. Conf. on Fault-Tolerant Comp.*, Band 283, Informatik Fachberichte. Springer Verlag, Nürnberg, 1991.

[Pup87] PUPPE, F.: *Diagnostisches Problemlösen mit Expertensystemen*. In: *Informatik Fachberichte*, Band 148. Springer Verlag, 1987.

[San95] SANDERS, W.H.: *UltraSan User's Manual*. University of Illinois, Version 3.0 edition, 1995.

[Sha48] SHANNON, C. E.: *A Mathematical Theory of Communication*. Bell Systems Technical Journal, 27:Seiten 379 – 423, Juli 1948.

[SS91a] SIMPSON, W.R. und J.W. SHEPPARD: *A Mathematical Model for Integrated Diagnostics*. IEEE Design & Test of Computers, 8(4):Seiten 25 – 38, Dezember 1991.

[SS91b] SIMPSON, W.R. und J.W. SHEPPARD: *System Complexity and Integrated Diagnostics*. IEEE Design & Test of Computers, 8(3):Seiten 16 – 30, September 1991.

[SS92a] SIMPSON, W.R. und J.W. SHEPPARD: *Analysis of False Alarms During System Design*. In: *Proceedings of the National Aerospace Electronics Conference*, Seiten 657 – 661, Piscataway, 1992. IEEE Press.

[SS92b] SIMPSON, W.R. und J.W. SHEPPARD: *Applying Testability Analysis for Integrated Diagnostics*. IEEE Design & Test of Computers, 9(3):Seiten 65 – 78, September 1992.

[SS92c] SIMPSON, W.R. und J.W. SHEPPARD: *System Testability Asessment for Integrated Diagnostics*. IEEE Design & Test of Computers, 9(1):Seiten 40 – 54, März 1992.

[SS93a] SIMPSON, W.R. und J.W. SHEPPARD: *Fault Isolation in an Integrated Diagnostic Environment*. IEEE Design & Test of Computers, 10(1):Seiten 16 – 30, März 1993.

[SS93b] SIMPSON, W.R. und J.W. SHEPPARD: *Performing Effective Fault Isolation in Integrated Diagnostics*. IEEE Design & Test of Computers, 10(2):Seiten 78 – 90, Juni 1993.

[SS94] SIMPSON, W.R. und J.W. SHEPPARD: *System Test and Diagnosis*. Kluwer Academic Publishers, Boston / Dordrecht / London, 1994.

[SS96] SIMPSON, W.R. und J.W. SHEPPARD: *Improving the Accuracy of Diagnostics Provided by Fault Dictionaries*. In: *Proceedings of the 14th VLSI Test Symposium*, 1996.

[SS99] SHOMBERT, L.A. und J.W. SHEPPARD: *A Behaviour Model for Next Generation Test Systems*. Journal of Electronic Testing: Theory and Applications, 1999.

[Tan95] TANENBAUM, A. S.: *Verteilte Betriebssysteme*. Prentice-Hall, München, 1995.

[Tij86] TIJMS, H.C.: *Stochastic Modeling and Analysis: A Computational Approach*. John Wiley, New York, 1986.

[WCS96] WALL, L., T. CHRISTIANSEN, and R.L. SCHWARTZ: *Programming Perl*. O'Reilly & Associates, Sebastopol, 1996.

[Weg96] WEGENER, J.: *Erweiterung Stochastischer Petri-Netze für die strukturelle Optimierung*, 1996. Diplomarbeit im Fach Informatik, IMMD III, Universität Erlangen-Nürnberg.

Schriftenverzeichnis

[ACDK97] ALLMAIER, S., G. CSERTÁN, S. DALIBOR und D. KREISCHE: *Parallel Reachability Graph Generation in Stochastic Modeling on Shared and Distributed Memory Machines*. Technischer Bericht, IMMD III, Universität Erlangen-Nürnberg, Erlangen, 1997.

[AD97] ALLMAIER, S. und S. DALIBOR: *PANDA – Petri net ANalysis and Design Assistant*. In: *Tools Descriptions*, Seiten 58 – 60, St. Malo, Juni 1997. 9th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation.

[ADK97] ALLMAIER, S., S. DALIBOR und D. KREISCHE: *Parallel Graph Generation Algorithms for Shared and Distributed Memory Machines*. In: *Proc. Parallel Computing Conference (ParCo '97)*, Bonn, September 1997.

[DHD+94] DAL CIN, M., W. HOHL, S. DALIBOR, T. ECKERT, A. GRYGIER, H. HESSENAUER, U. HILDEBRAND, J. HÖNIG, F. HOFMANN, C.-U. LINSTER, E. MICHEL, A. PATARICZA, V. SIEH, T. THIEL und S. TUROWSKI: *Architecture and Realization of the Modular Expandable Multiprocessor System MEMSY*. In: *Proc. First Intl. Conf. on Massively Parallel Computing Systems (MPCS'94)*, Seiten 7 – 15, Ischia, 1994. IEEE.

[DHH95] DALIBOR, S., A. HEIN und W. HOHL: *Simulation-Based Performability Evaluation of Fault-Tolerant Multiprocessors*. In: *Proc. European Simulation Multiconference 1995*, Seite 699, Prag, Juni 1995. SCS.

[DHH96] DALIBOR, S., A. HEIN und W. HOHL: *Application Dependent Performability Evaluation of Fault-Tolerant Multiprocessors*. In: *Proc. 4th Euromicro Workshop on Parallel and Distributed Processing*, Seite 310, Braga, Januar 1996. IEEE CS Press.

Index