Konfiguration verteilter Workflow-Management-Systeme mit Leistungsgarantien

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

von
Dipl.-Inform.

Michael H. Gillmann

Saarbrücken,
im August 2001

Hiermit erkläre ich, daß ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten sind unter Angabe der Quelle gekennzeichnet.
Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Saarbrücken, 27. August 2001

Michael H. Gillmann

Kurzfassung

Workflow-Management-Systeme (WFMS) werden immer wichtiger bei der Steuerung unternehmensweiter und unternehmensübergreifender Geschäftsprozesse. Ihr verteilter Einsatz auch über die Grenzen von Organisationseinheiten und Unternehmensstrukturen hinweg läßt sie zu komplexen Systemen werden. Sie setzen sich aus mehreren Workflow-Servern, Applikations-Servern und Kommunikations-Servern zusammen. Die geeignete Konfiguration eines verteilten WFMS für eine gegebene Anwendung zu finden, ist ein schweres Unterfangen.
Diese Dissertation präsentiert mathematisch fundierte Methoden für die Konfigurierung von verteilten WFMS mit dem Ziel, die Gesamtkosten der Systemkonfiguration unter Einhaltung anwendungsspezifischer Leistungs- und Verfügbarkeitsanforderungen zu minimieren. Der Freiheitsgrad, den sich die Methoden zu Nutzen machen, ist die Replikation von Software-Komponenten – Workflow-Server und Applikations-Server unterschiedlicher Typen sowie Kommunikations-Server – auf mehreren Rechnern, um die Verfügbarkeit zu erhöhen und anfallende Lasten skalierbar aufteilen zu können. Das mathematische Fundament der Methoden sind Markov-Modelle, die von den Workflow-Spezifikationen der Anwendung abgeleitet werden. Sie erlauben die Abschätzung des Verhaltens des WFMS bei geänderten Replikationsgraden bezüglich seiner Leistung, seiner Verfügbarkeit und auch seiner Performability bei transienten Ausfällen von Server-Replikaten. Durch Iteration über eine Menge in Frage kommender Systemkonfigurationen und durch analytische Abschätzung der von einer Konfiguration erzielten Leistung liefern die entwickelten Methoden eine Systemkonfiguration, die die geforderten Leistungs- und Verfügbarkeitsziele einhält und niedrige Kosten verursacht. Die Methoden sind in einem Konfigurationswerkzeug für verteilte WFMS implementiert.

Abstract

Workflow management systems (WFMS) that are geared for the orchestration of enterprise-wide or even "virtual-enterprise"-style business processes across multiple organizations are complex distributed systems. They consist of multiple workflow engines, application servers, and communication servers. Thus, deriving a suitable configuration of an entire distributed WFMS for a given application workload is a difficult task.
This thesis presents mathematically based methods for configuring a distributed WFMS such that the application's demands regarding performance and availability can be met while aiming to minimize the total system costs. The degree of freedom that the configuration method considers is the replication of the underlying software components, workflow engines and application servers of different types as well as communication servers, on multiple computers for load partitioning and enhanced availability. The mathematical core of the methods consists of Markov-chain models, derived from the application's workflow specifications, that allow assessing a system configuration's performance, availability, and also its performability in the degraded mode when some server replicas are down. By iterating over the space of feasible system configurations and assessing the quality of candidate configurations, the developed methods determine a configuration with near-minimum costs. The methods are implemented in an auto-configuration tool for distributed WFMS.

Inhaltsverzeichnis

Kurzfassung **4**

Abstract **4**

I Einleitung 11

1 Motivation und Einordnung der Arbeit **13**
1.1 Verwandte Arbeiten 14
1.2 Beitrag der Arbeit 15

2 Überblick über die Arbeit **17**

3 Grundlagen und Begriffe **19**
3.1 Spezifikation von Workflows mit State- und Activitycharts 19
3.2 Benchmark-Workflow 21
3.2.1 Bausteine 22
3.2.2 Szenario: Versandhaus-Bestellung 22
3.2.3 Kontrollfluß-Spezifikation als Statechart 23
3.2.4 Datenfluß-Spezifikation als Activitychart 25
3.2.5 Portierbarkeit 26
3.3 Stochastische Methoden 26
3.3.1 Stochastische Prozesse und Markov-Ketten 26
3.3.2 Zeitkontinuierliche Markov-Ketten mit allgemein verteilten Zustandsverweilzeiten 30
3.3.3 Markov-Reward-Modelle 31
3.3.4 M/G/1-Warteschlangenmodelle 32

II Modellbildung 35

4 Systemmodell **37**
4.1 Architektur-Ansätze 37
4.1.1 Vorgaben der Workflow Management Coalition (WfMC) 37
4.1.2 ADEPT 38
4.1.3 $WASA_2$ 38
4.1.4 $Meteor_2$ 39
4.1.5 Mentor-lite 39
4.1.6 Staffware97 40

4.1.7 MQSeries Workflow . . . 41
4.2 Verteiltes Server-Modell . . . 41
4.3 Systemkonfiguration und Systemzustand . . . 45
4.4 Zusammenfassung und Diskussion . . . 45

5 Stochastische Modellierung des Kontrollflusses 47
5.1 Zusammensetzung von Workflows . . . 48
5.2 Aufbau des Flußprozesses . . . 48
5.3 Handhabung von Schleifen . . . 50
5.3.1 Naive Methode mittels des Erwartungswertes für die Anzahl der Schleifendurchläufe . . . 51
5.3.2 Gleichmäßig verteilte Anzahl an Schleifendurchläufen . . . 52
5.3.3 Konstant verteilte Anzahl an Schleifendurchläufen . . . 54
5.4 Normalisierung des Flußprozesses . . . 55
5.5 Zusammenfassung und Diskussion . . . 56

6 Leistungsmodell 59
6.1 Metrik . . . 60
6.2 Durchlaufzeit von Workflows . . . 60
6.3 Lastmodell . . . 61
6.3.1 Aktivitäten-spezifische Last . . . 61
6.3.2 Last pro Workflow . . . 62
6.3.3 Gesamtlast an Servern und erreichbarer Durchsatz . . . 64
6.4 Berücksichtigung von Subworkflows . . . 65
6.5 Abschätzung der Wartezeit von Service-Aufträgen . . . 66
6.5.1 Modellierung der Server als M/G/1-Bedienstationen . . . 66
6.5.2 Mittlere Wartezeit von Service-Aufträgen . . . 66
6.6 Zusammenfassung und Diskussion . . . 67

7 Verfügbarkeitsmodell 71
7.1 Metrik . . . 71
7.2 Fehlermodell . . . 72
7.3 Stochastische Modellierung der Verfügbarkeit von Servern . . . 73
7.4 Stationäre Analyse der Verfügbarkeit von Servern . . . 75
7.5 Zusammenfassung und Diskussion . . . 76

8 Performability-Modell 79
8.1 Metrik . . . 79
8.2 Stationärer Erwartungswert für Wartezeiten . . . 80
8.3 Stationärer Erwartungswert für maximalen Durchsatz . . . 81
8.4 Zusammenfassung und Diskussion . . . 82

III Anwendung 83

9 Entwurf eines Konfigurationswerkzeuges 85
9.1 Funktionalität und Architektur . . . 85
9.1.1 Modellierung . . . 86
9.1.2 Kalibrierung . . . 86

9.1.3 Evaluation . . . 87
9.1.4 Optimierung . . . 87
9.2 Ermittlung der kostengünstigsten Konfiguration . . . 88
9.3 Zusammenfassung und Diskussion . . . 90

10 Implementierung eines Konfigurationswerkzeuges für Mentor-lite 93
10.1 Implementierung . . . 93
10.2 Benutzerschnittstellen . . . 94

IV Evaluation 99

11 Meßaufbau 101
11.1 Abbildung von Service-Aufträgen . . . 102
11.2 Abbildung von Server-Ausfällen . . . 103

12 Meßreihen und Parametrisierung 105
12.1 Meßreihen . . . 105
12.2 Feste Parameter . . . 106
12.2.1 Parametrisierung des Benchmark-Workflows . . . 106
12.2.2 Bedienzeiten und Ausfälle der Server-Typen . . . 107

13 Diskussion der Meßergebnisse 111
13.1 Last . . . 111
13.2 Leistung . . . 112
13.3 Performability . . . 114
13.4 Kostengünstigste Konfiguration . . . 115
13.5 Zusammenfassung und Diskussion . . . 117

V Fazit 119

14 Zusammenfassung der Arbeit 121

15 Ausblick 123

16 Summary of the Thesis 125

VI Anhang 127

A Parameterdatei von Goliat 129

B Glossar 131

Literaturverzeichnis 135

Abbildungsverzeichnis

2.1 Aufbau und Zusammenhänge der Modellbildung 18

3.1 Beispiel eines Activitycharts 20
3.2 Beispiel eines Statecharts 21
3.3 Top-level Statechart des Beispiel-Workflow-Typs *Versandhaus* 24
3.4 Statechart des (Sub-)Workflow-Typs *Bestätigung* 25
3.5 Statechart des (Sub-)Workflow-Typs *Versand* 25
3.6 Activitychart des Workflow-Typs *Versandhaus* 26

4.1 Referenzmodell der Workflow Management Coalition [WFMC] 38
4.2 Architektur von Mentor-lite 40
4.3 Sequence Diagram für die Ausführung zweier Aktivitäten 43
4.4 Server-Modell eines verteilten WFMS 44

5.1 CTMC des Workflow-Typs *Versandhaus* 51
5.2 Naive CTMC des Workflow-Typs *Versand* 52
5.3 Expandierte CTMC des Workflow-Typs *Versand* bei Gleichverteilung der Schleifendurchläufe 55
5.4 Expandierte CTMC des Workflow-Typs *Versand* bei konstanter Verteilung der Schleifendurchläufe 56

6.1 CTMC des Workflow-Typs *Bestätigung* mit Lastvektoren 63
6.2 Bottom-up Berechnung bei Subworkflows 66

7.1 Beispiel für eine CTMC des Verfügbarkeitsmodelles 74

9.1 Vorhersagen für Testkonfigurationen 86
9.2 Vorschläge für kostengünstigste Konfigurationen 88
9.3 Pseudocode des Greedy-Algorithmus 89

10.1 Kontrollfenster von Goliat 95
10.2 Workflow-Editor von Goliat 96
10.3 System-Editor von Goliat 97
10.4 Ergebnisse der Was-Wäre-Wenn-Analyse 98

13.1 Service-Aufträge pro Workflow-Instanz 112
13.2 Leistung des stabilen WFMS bei zentraler Workflow-Ausführung 113
13.3 Leistung des stabilen WFMS bei verteilter Workflow-Ausführung 113
13.4 Performability bei zentraler Workflow-Ausführung 114
13.5 Performability bei verteilter Workflow-Ausführung 115

13.6 Performability unter der von Goliat vorgeschlagenen Systemkonfiguration Y=(3,5,3,3) 116
13.7 Performability unter der optimalen Systemkonfiguration Y=(3,4,3,3) 117

Teil I

Einleitung

Kapitel 1

Motivation und Einordnung der Arbeit

Die Workflow-Technologie steht auf dem Sprung, zu einem wichtigen Pfeiler der informationstechnischen Infrastruktur moderner Unternehmen zu werden [BRD01a, CHR+98, Här97, JBS97, LR99, VB96, WFMC]. Ein Workflow-Management-System, kurz: WFMS, ist in der Lage, verschiedenste, zu einem Geschäftsprozeß gehörende Aktivitäten zu koordinieren, indem es sie gemäß einer Kontroll- und Datenfluß-Spezifikation für den vorliegenden Prozeßtyp startet. Das Spektrum der so unterstützten Prozesse reicht von sogenannten Produktions-Workflows mit ausschließlich vollautomatischen Aktivitäten bis zu komplexen, kooperativen Workflows mit einem signifikanten Anteil an intellektuell-interaktiv zu bearbeitenden Aktivitäten. Aktivitäten können, ggf. über Dienste der zugrundeliegenden Middleware (zum Beispiel einem Object Request Broker, kurz: ORB), den unmittelbaren Start eines Applikations-Programmes nach sich ziehen, wie es üblicherweise bei automatischen Aktivitäten der Fall ist, oder zunächst die Zuordnung eines Arbeitsauftrages an einen Benutzer oder eine Organisationseinheit erfordern, was allgemein als *Worklist-Management* bekannt ist.

Die Unternehmen sind darauf angewiesen, daß das WFMS in der Lage ist, eine große Zahl von Workflows verschiedenster Typen verläßlich und ohne spürbare Verzögerungen des Geschäftsablaufes auszuführen. Das WFMS selbst ist jedoch ein äußerst kompexes System, das die Geschäftsprozesse unternehmensweit oder gar unternehmensübergreifend orchestriert. Es ist zusammengesetzt aus mehreren untereinander kooperierenden Workflow-Servern und Applikations-Servern und bedient sich einer Kommunikationsinfrastruktur (zum Beispiel einem TP-Monitor oder einem ORB). Eine grundlegende Charakteristik unseres Systemmodelles ist die Replikation der zu Grunde liegenden Software-Komponenten auf verschiedenen Rechnern aus Verfügbarkeits- und Skalierbarkeitsgründen. Für manche, insbesondere die langlebigen, Workflow-Typen ist eine hohe Verfügbarkeit des verteilten WFMS unerläßlich. Andere Workflow-Typen, zum Beispiel mit oft auftretenden und kurzlebigen Instanzen, erfordern hingegen kurze Antwortzeiten und einen hohen Durchsatz.

Für die Administratoren eines WFMS ist es eine schwierige Aufgabe, eine geeignete, kostengünstige Konfiguration für das WFMS zu finden. Die geeignete Anzahl und die Anordnung der (replizierten) Komponenten des WFMS, die die Konfiguration im wesentlichen bestimmen, hängen von einer Vielzahl von Faktoren ab und sollen verschiedene Anforderungen sowohl im Hinblick auf die Leistung der Workflow-Ausführung als auch

auf deren Kosten erfüllen. Darüber hinaus wird es oft notwendig sein, nicht nur eine initiale Konfiguration zu wählen, sondern das WFMS mit der Zeit auch an wechselnde Lastanforderungen, wie zum Beispiel durch neu hinzukommende Geschäftsfelder oder -schwerpunkte, dynamisch anzupassen und zu rekonfigurieren.

1.1 Verwandte Arbeiten

Obwohl in der Literatur viel über Architekturen skalierbarer WFMS veröffentlicht wurde [AAA+97, CGS97, DKO+98, GHS95, GT98, JB96, Moh99, SH97, VW98], gibt es bisher nur wenig Forschung auf dem Gebiet der quantitativen Erfassung und Einschätzung der Qualität von Konfigurationen verteilter WFMS unter Berücksichtigung von Leistung und Verfügbarkeit.

[BD99a] stellt mehrere Klassen von Architekturen für WFMS vor und diskutiert die Auswirkungen verschiedener Verteilungsstrategien während der Workflow-Ausführung auf die Last an Servern und Teilnetzen. Anhand von Simulationen für zwei Beispielszenarien werden die Unterschiede der einzelnen Klassen verdeutlicht. Aus den gewonnenen Erkenntnissen werden in [BD99b] Entscheidungshilfen für die Auswahl geeigneter Architekturen für verschiedene Anwendungstypen abgeleitet. In [BD97] wird der erreichbare Durchsatz eines WFMS durch Minimierung der Kommunikationskosten erhöht. Dazu werden Subworkflows auf geeignete Server migriert. Das Ziel einer solchen Migration wird in Abhängigkeit von der Netzwerkerkpartitionierung, der Last im Netzwerk und dem erwarteten Gewinn bezüglich der Kommunikationskosten bestimmt. [BD99c, BD00] erweitern diesen Ansatz zu einem Online-Verfahren, das Migrationsentscheidungen ohne gewichtigen Zusatzaufwand auch zur Ausführungszeit einer Workflow-Instanz treffen kann. Alle diese Arbeiten konzentrieren sich auf die Last durch die Kommunikation zwischen den Servern, lassen die Last auf den Servern selbst jedoch außen vor. Darüber hinaus werden Ausfälle einzelner Server oder von Teilnetzen bei den Analysen nicht in Betracht gezogen.

[SNS99] präsentiert einfache Heuristiken für die Zuweisung von Workflow-Instanzdaten an die Komponenten eines WFMS unter Berücksichtigung der aktuellen Systemkonfiguration. Die Trennung von Komponenten des WFMS und der logischen Struktur von Workflows spielt dabei die wesentliche Rolle. Es werden jedoch keinerlei Aussagen darüber gemacht, wie eine geeignete Systemkonfiguration ermittelt werden kann.

In [KSW+01a, KSW+01b] wird ein Verfahren zur Einhaltung von Antwortzeitzielen vorgestellt, das Aufträgen verschiedener Anwenderklassen automatisch Prioritäten an einem Server zuweist. Grundlage des Verfahrens ist die Modellierung des Leistungsverhaltens der Server bei gegebener Last durch ein stochastisches Warteschlangenmodell. Die Ergebnisse des analytischen Modelles wurden anhand des kommerziellen Systems MQSeries von IBM evaluiert. Das vorgestellte Verfahren ist jedoch für nur genau einen Server konstruiert. Die Replikation von Servern sowie das Zusammenspiel mehrerer Server-Typen und das Auftreten von Auftragssequenzen in Abhängigkeit einer Workflow-Spezifikation werden nicht betrachtet.

Arbeiten über die Verfügbarkeit von WFMS sind [HA98, HA99, KAG+96]. Dort werden

Methoden diskutiert, mit denen die Verfügbarkeit von WFMS effizient gesteigert werden kann, indem mehrere Backup-Server bereitgestellt werden, die im Falle eines Ausfalls eines Servers dessen Arbeit übernehmen können. Analytische Vorhersagen für die (zusätzliche) Last an den Servern werden aber weder für den laufenden Betrieb noch für den Fall eines Server-Ausfalles erarbeitet.

1.2 Beitrag der Arbeit

Diese Arbeit befaßt sich erstmalig in der Forschung auf dem Gebiet des Workflow-Management mit der Entwicklung eines Werkzeuges, das einen Administrator bei der (dynamischen) Konfigurierung eines verteilten WFMS analytisch unterstützt. Das Werkzeug analysiert die Leistungsfähigkeit der Komponenten des WFMS und setzt sie in Relation zu den Lastcharakteristika der Arbeitsumgebung, in der es eingesetzt wird. Darauf basierend kann es Vorhersagen über Leistung und Verfügbarkeit des WFMS unter einer neuen Konfiguration machen und die kostengünstigste Konfiguration vorschlagen, die zuvor spezifizierten Anforderungen an die Leistung, an die Verfügbarkeit oder an die Kombination aus beiden, die sogenannte Performability, gerecht wird. So wird ein kritischer Teil der Administration verteilter WFMS analytisch erfaßbar und bleibt nicht länger abhängig von zeit- und kostenintensiven empirischen Versuchsreihen oder von der individuellen Erfahrung und Intuition des Administrationsstabs.

Die Analysetechniken des Werkzeuges basieren auf bekannten stochastischen Methoden, insbesondere auf zeitkontinuierlichen Markov-Ketten erster Ordnung [All90, Nel95, STP96, Tij94]. Die vorliegende Arbeit zeigt die Anwendbarkeit dieser mathematischen Methoden für die Leistungsvorhersage und Konfiguration des Workflow-Management und rechtfertigt deren Einsatz bei der Modellierung des Verhaltens einzelner Instanzen gegebener Workflow-Typen. Dazu präsentieren wir ausführliche Ergebnisse, die durch Messungen in einer eigens entwickelten, realitätsnahen Simulationsumgebung ermittelt wurden. Die Resultate der Arbeit sind in einem lauffähigen Prototypen eines Konfigurationswerkzeuges implementiert.

Kapitel 2

Überblick über die Arbeit

Die Arbeit gliedert sich in fünf Teile.

- Die Einleitung endet mit Kapitel 3, in dem wir Grundlagen aus den Bereichen des Workflow-Management und der Stochastik einführen.

- Der zweite Teil der Arbeit befaßt sich mit der Entwicklung von Modellen, mit denen wir das Verhalten von WFMS abschätzen werden. Abbildung 2.1 zeigt einen Überblick über die Zusammenhänge dieses Teiles. Dabei stellen die Pfeile Abhängigkeiten zwischen den einzelnen Kapiteln dar. In einem Kapitel am Anfang eines Pfeiles werden Grundlagen gebildet, auf die das Kapitel am Ende des Pfeiles aufbaut.
  In Kapitel 4 beschreiben wir das Systemmodell für ein verteiltes WFMS. In Kapitel 5 bilden wir Kontrollfluß-Spezifikationen von Workflow-Typen auf ein stochastisches Modell ab. In Kapitel 6 präsentieren wir das Leistungsmodell. Dabei ergänzen wir das stochastische Kontrollflußmodell aus Kapitel 5 um Lastanteile für die Aktivitäten der Workflow-Typen. Diese Lastanteile werden basierend auf dem Systemmodell aus Kapitel 4 auf die verschiedenen Komponenten des WFMS verteilt. In Kapitel 7 leiten wir aus dem Systemmodell aus Kapitel 4 das Verfügbarkeitsmodell ab. In Kapitel 8 kombinieren wir das Leistungsmodell und das Verfügbarkeitsmodell zu einem Performability-Modell. Als Performability bezeichnet man die Leistung eines Systems unter Berücksichtigung transienter Komponentenausfälle.

- Im dritten Teil der Arbeit zeigen wir, wie sich die Modelle in der Praxis zur Konfiguration von verteilten WFMS einsetzen lassen. Dazu entwerfen wir in Kapitel 9 das Konzept eines automatischen Konfigurationswerkzeuges für verteilte WFMS und beschreiben in Kapitel 10 eine erste Implementierung des Werkzeuges.

- Im vierten Teil der Arbeit liefern wir die Evaluation der Modelle anhand von simulierten Workflow-Ausführungen. Der Meßaufbau orientiert sich an einer realen Einsatzumgebung von WFMS und verwendet Teile eines lauffähigen WFMS. Die Beschreibung des Messaufbaus findet sich in Kapitel 11. Kapitel 12 beschreibt die ausgeführten Meßreihen und deren Parametrisierung. In Kapitel 13 diskutieren wir die Ergebnisse der Meßläufe.

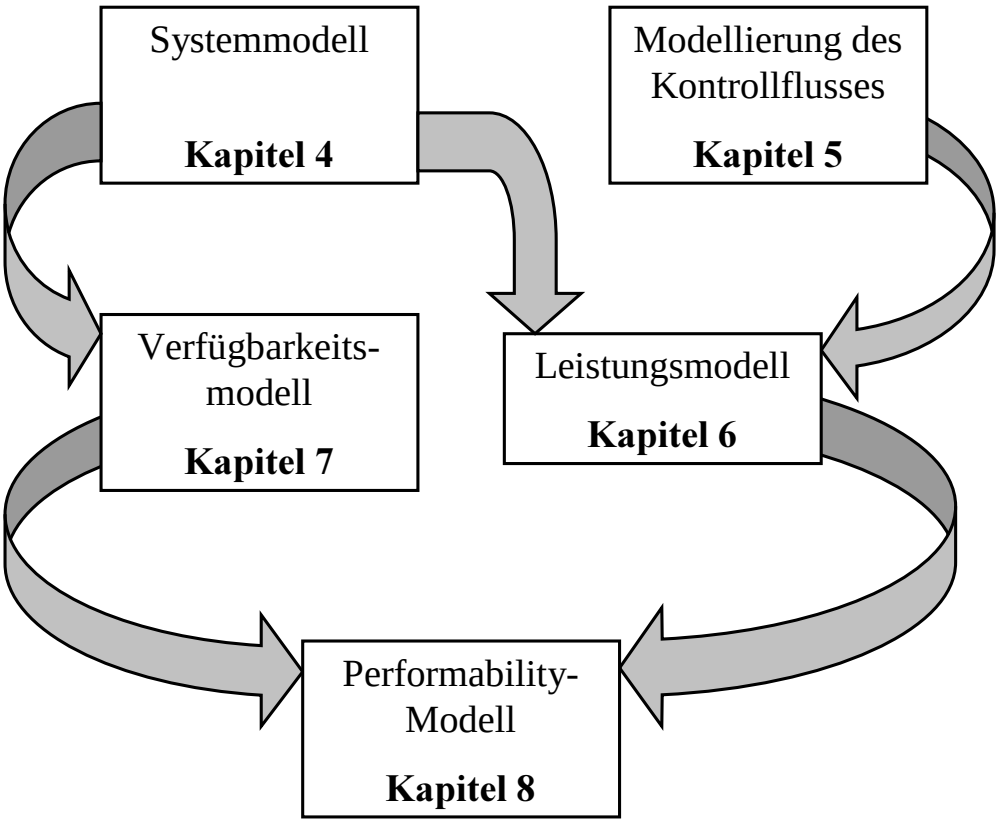


Abbildung 2.1: Aufbau und Zusammenhänge der Modellbildung

- Der fünfte Teil zieht ein Fazit über die erörterten Probleme und aufgezeigten Lösungen. Kapitel 15 gibt einen Ausblick auf offene Probleme und mögliche Erweiterungen. Die Kapitel 14 und 16 liefern eine Zusammenfassung der Arbeit.

Kapitel 3

Grundlagen und Begriffe

In diesem Kapitel führen wir Grundlagen aus den Bereichen des Workflow-Management und der Stochastik ein. In Abschnitt 3.1 beschreiben wir den Formalismus, den wir zur Spezifikation von Workflows verwenden. Abschnitt 3.2 stellt einen Benchmark-Workflow vor, den wir entwickelt haben um in späteren Kapiteln entwickelte Modelle zu illustrieren und zu evaluieren. In Abschnitt 3.3 führen wir Begriffe und Methoden aus der Stochastik ein, die im weiteren Verlauf dieser Arbeit als Basis für unsere analytischen Modelle dienen.

3.1 Spezifikation von Workflows mit State- und Activitycharts

In diesem Abschnitt beschreiben wir den Formalismus, den wir in dieser Arbeit zur Spezifikation von Workflow-Typen benutzen. Die Spezifikations-Methode basiert auf dem State- und Activitychart-Formalismus, der in den Arbeiten von David Harel [Har87, HG97] entwickelt und inzwischen in den Industrie-Standard UML [UML] übernommen wurde. State- und Activitycharts stellen duale Sichtweisen auf eine Spezifikation dar.

Activitycharts spezifizieren den Datenfluß zwischen den Aktivitäten in Form von gerichteten Graphen mit Aktivitäten als Knoten und Datenelementen als Kantenbeschriftung.

Statecharts beschreiben das Verhalten eines Workflows, indem sie den Kontrollfluß zwischen Aktivitäten als Transitionen zwischen Zuständen spezifizieren. Ein Statechart ist im wesentlichen ein endlicher Automat mit einem ausgewiesenen Anfangszustand und Transitionen, die mit Event-Condition-Action Regeln (ECA-Regeln) beschriftet sind (vergleiche Abbildung 3.2). Eine Transition von Zustand X nach Zustand Y, die mit der ECA-Regel $E[C]/A$ beschriftet ist, feuert genau dann, wenn sich der Automat in Zustand X befindet, das Ereignis E eintritt und die Bedingung C erfüllt ist. Der Automat wechselt dann in den Zustand Y und die Aktion A wird ausgeführt. Bedingungen und Aktionen werden durch Variablennamen oder Ausdrücke identifiziert, wobei insbesondere diejenigen Datenbezeichner, die zur Spezifikation des Datenflußes im entsprechenden Activitychart benutzt wurden, verwendet werden können. Darüber hinaus kann eine Aktion sowohl explizit eine Aktivität starten (durch den Ausdruck *st!(aktivität)*), als auch Ereignisse erzeugen oder Bedingungen setzen (z.B. *fs!(C)* setzt die Bedingung C auf den Wert falsch). Jeder der drei Teile des E[C]/A-Tripels kann auch leer sein.
Zwei weitere Besonderheiten unterscheiden Statecharts von herkömmlichen Zustandsauto-

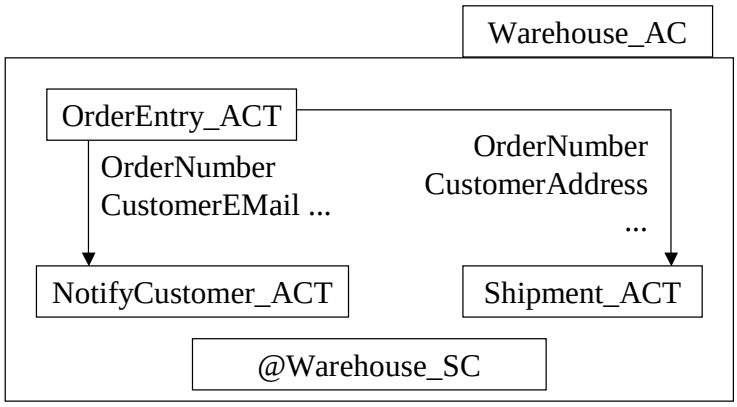


Abbildung 3.1: Beispiel eines Activitycharts

maten. Zum einen ist dies die im Hinblick auf schrittweise Verfeinerung und Abstraktion wichtige Möglichkeit zur Schachtelung von Zuständen. Ein Zustand kann ein komplettes Statechart beinhalten, wobei beim Betreten eines übergeordneten Zustands implizit der jeweilige initiale Unterzustand betreten wird und beim Verlassen eines Elternzustandes alle betretenen Kinderzustände verlassen werden. Zum anderen erlauben Statecharts orthogonale Zustandskomponenten, mit denen Parallelität ausgedrückt wird, so daß Transitionen in verschiedenen orthogonalen Komponenten gleichzeitig feuern können.

Im Rahmen des Mentor-Projektes wurde für die Spezifikation von Workflows ein State- und Activitychart-Dialekt ("WFCharts" [Wod97]) entwickelt. Dabei wurde die Syntax der State- und Activitycharts mit Hilfe von Regeln für das Workflow-Design auf die speziellen Belange von Workflow-Management eingeschränkt. Dadurch ergab sich eine einfachere, intuitiv leichter verstehbare Semantik, die alle Belange der Workflow-Spezifikation abdeckt. Wesentliche Eigenschaften von WFCharts liegen in der formalen Verifizierbarkeit und der beweisbaren Äquivalenz von Workflow-Spezifikationen, insbesondere von Spezifikationen bei partitioniert-verteilter gegenüber zentraler Workflow-Ausführung [Wod97, WW97].

Beispiel:

Abbildung 3.1 zeigt das Activitychart eines einfachen Versandhaus-Workflows. Das Activitychart enthält die drei am Workflow beteiligten Aktivitäten *OrderEntry_ACT, Shipment_ACT und NotifyCustomer_ACT*, sowie einen Verweis auf das zugehörige Statechart *@Warehouse_SC*. Innerhalb der Aktivität *OrderEntry_ACT* werden die Bestelldaten erfaßt und bedarfsweis an die beiden anderen Aktivitäten verteilt. Die Aktivität *NotifyCustomer_ACT* erstellt eine Benachrichtigung an den Kunden und die Aktivität *Shipment_ACT* unterstützt den Versand der bestellten Waren.

Das zugehörige Statechart ist in Abbildung 3.2 abgebildet. Zu Beginn des Workflows wird die Aktivität *OrderEntry_ACT* gestartet. Dabei betritt der Workflow den Zustand *OrderEntry_S*. Wenn die Aktivität erfolgreich beendet wurde, wird die Bedingung *OrderEntry_DONE* auf wahr gesetzt. Durch das Setzen der Bedingung *OrderEntry_DONE* auf wahr wechselt der Workflow in den geschachtelten Zustand *OrderProcessing_S*. Dieser enthält zwei orthogonale Komponenten, die als parallele Subworkflows abgearbeitet werden. Der

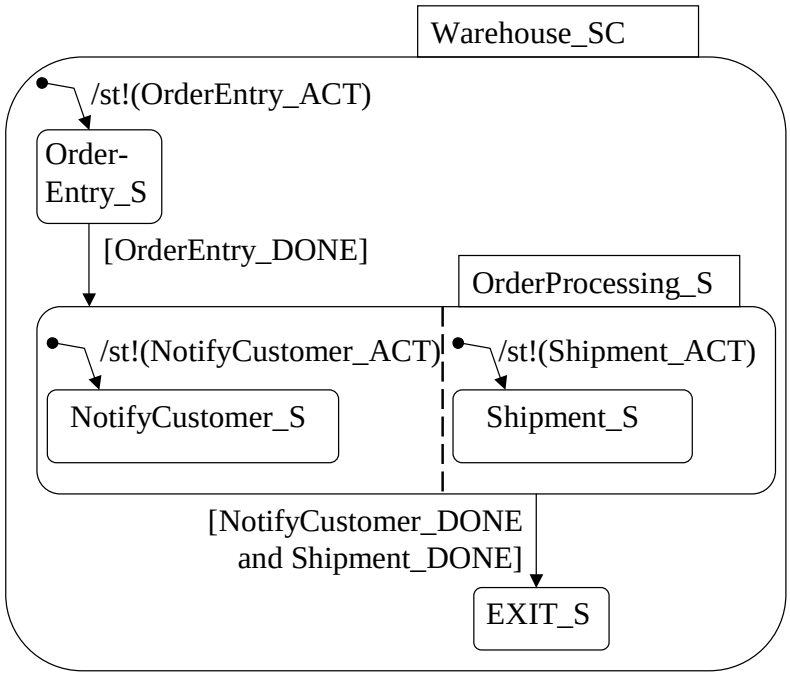


Abbildung 3.2: Beispiel eines Statecharts

Einfachheit halber bestehen die beiden Subworkflows in unserem Beispiel jeweils aus nur einer Aktivität. So werden die Aktivität *NotifyCustomer_ACT* sowie die Aktivität *Shipment_ACT* gestartet. Wurden beide Aktivitäten erfolgreich beendet, endet der Workflow im Zustand *EXIT_S*.

3.2 Benchmark-Workflow

Das in diesem Abschnitt vorgestellte Beispiel basiert auf einem Workflow-Typ, den wir in einer ausführlicheren Variante zum Benchmarking von WFMS entwickelt haben [GMW+99, GMW00, Min00]. Anhand des Beispieles werden wir in späteren Kapiteln gezeigte Modelle illustrieren und evaluieren. Dabei soll die Evaluation der theoretischen Modelle, also die Analyse der Aussagekraft und der Realitätsnähe der Modelle, durch den Einsatz des Benchmarks als durchgehender Beispiel-Workflow auf eine systematische, objektive Basis gestellt werden. Der Workflow beschreibt - ausgehend von den Basisaktivitäten des TPC-C-Benchmarks [TPC] - ein realistisches Szenario aus dem Bereich des Electronic Commerce.

Bisher sind in der Literatur aufgeführte Beispiele von Workflows oft wenig repräsentativ und nicht genügend umfassend. Auch das Benchmarking von WFMS steckt noch in den Kinderschuhen. Die einzigen uns bekannten Arbeiten, die sich mit dem Thema intensiver befaßt haben, sind sehr spezieller Natur: in [MP96] wird ein einziges System (Lotus Notes) betrachtet, das eher als Dokumentenverwaltungs- und -kommunikationssystem denn als WFMS zu bezeichnen ist, und es werden nur Leistungsaspekte der E-Mail-Komponente untersucht; in [BSR96] steht die spezielle Anwendung von DNS-Laboruntersuchungen für die Genomforschung im Blickpunkt, wobei ein funktional spezialisierter WFMS-Prototyp

zugrundeliegt, und der vorgeschlagene Benchmark nur extrem spezialisierte Leistungsaspekte abdeckt; in [DHK+97] wird mit Hilfe des TPC-D Benchmarks ein komplettes Anwendungssystem (SAP R/3) statt eines isolierten Datenbanksystems untersucht; in [GBL+98] schließlich wird ein Benchmark für aktive Datenbanksysteme vorgeschlagen, der aber keine spezifischen Elemente hinsichtlich Workflow-Anwendungen enthält. [GMW+99] und [GMW00] sind somit mehr oder weniger die ersten Arbeiten, die versuchen, einen allgemeinen, systematischen Benchmark für WFMS zu definieren.

3.2.1 Bausteine

In diesem Abschnitt diskutieren wir, aus welchen Bausteinen sich ein repräsentativer Beispiel-Workflow zusammensetzen sollte und welche Eigenschaften eines WFMS damit im Rahmen der Evaluation der Modelle ausgelotet werden können.

Kontrollflußkonstrukte. Die Hauptaufgabe eines WFMS besteht in der Koordinierung des Kontrollflusses zwischen einzelnen Aktivitäten. Der Kontrollfluß ist typischerweise nicht linear, sondern enthält Konstrukte wie Verzweigungen, Joins, Schleifen und Parallelität. Um ihre Auswirkungen auf das Systemverhalten bestimmen zu können, sollte das volle Spektrum der Kontrollflußkonstrukte enthalten sein.

Interaktive und vollautomatische Aktivitäten. Eine weitere Aufgabe von WFMS ist die Integration von Anwendungen, die die einzelnen Aktivitäten eines Workflows implementieren. Wir unterscheiden zwischen interaktiven und vollautomatischen, das heißt ohne Benutzerinteraktion ablaufenden Aktivitäten, da beide Arten in der Regel unterschiedlich vom WFMS gehandhabt werden und ein unterschiedliches Laufzeitverhalten aufweisen. Darüber hinaus sind viele WFMS auf einen Aktivitäten-Typ (vollautomatisch oder interaktiv) optimiert. Während vollautomatische Aktivitäten häufig kurze Laufzeiten aufweisen, wird die Laufzeit der interaktiven Aktivitäten hauptsächlich durch den menschlichen Bearbeiter bestimmt. Durch die "Denkzeiten" der Bearbeiter belastet eine interaktive Aktivität relativ zu ihrer Gesamtdauer das WFMS weniger als eine vollautomatische Aktivität. So werden Effekte des WFMS auf das Durchlaufzeitverhalten der Workflows weniger relevant. Andererseits können unter Umständen Ressourcen für Verwaltungsdaten lange belegt werden, so daß sie für die Verwaltung anderer Aktivitäten nicht mehr zur Verfügung stehen und der Systemdurchsatz vermindert wird. Die Auswirkungen solcher Aspekte sollen in die Ergebnisse der Betrachtungen eingehen. Das heißt sowohl vollautomatische als auch interaktive Aktivitäten bilden weitere Bausteine.

Schnittstelle zu externen Anwendungen. Um auch die Schnittstellen zu externen Anwendungen in die Last miteinbeziehen zu können, sind die Aufrufe von Anwendungs-Programmen durch das WFMS und der Austausch von (kontrollflußrelevanten) Daten weitere Bestandteile.

3.2.2 Szenario: Versandhaus-Bestellung

Bei der Spezifikation des Beispiel-Workflows orientieren wir uns, ähnlich dem TPC-C Benchmark für Transaktionssysteme [TPC], an der Gesamtverarbeitung einer Bestellung wie sie in einem großen Versandhaus vorkommt. Der Unterschied zum TPC-C Benchmark ist jedoch, daß Kontroll- und Datenfluß zwischen den Aktivitäten *Bestellung, Lieferung*

und *Bezahlung* sowie weitere Aktivitäten hinzugefügt wurden.

Der Workflow beginnt mit einer Aktivität, die eine neue Bestellung in einer Datenbank erfaßt. Das Versandhaus stellt zwei Arten von Zahlungsmodalitäten zur Verfügung. Der Kunde kann entweder bei der Bestellung seine Kreditkartennummer angeben oder eine Rechnung anfordern, nach deren Erhalt er die angefallene Rechnungssumme überweisen muß. Der Kontrollfluß muß an dieser Stelle aufgespaltet werden. Wurde eine Kreditkartenzahlung gewünscht, wird zunächst eine Aktivität zur Bonitätsprüfung der Kreditkarte gestartet. Nach der Prüfung der Kreditkarte werden die beiden Kontrollfluß-Pfade wieder zusammengeführt.
Nach der Registrierung der Bestellung werden der Versand sowie das Senden einer Bestätigung an den Kunden initiiert. Dazu werden parallel zwei Subworkflows gestartet. Der eine der beiden Subworkflows startet eine Aktivität, die eine elektronische Mail an den Kunden versendet. Das Versenden von Waren erfolgt extern aus einem oder mehreren Lagern, die unter Umständen sogar von separaten Großhändlern autonom verwaltet werden. Daher wird im zweiten Subworkflow zunächst eine Aktivität gestartet, die für jedes Lager die auszuliefernden Artikel bestimmt. Die Zuordnung wird in Artikellisten festgehalten. Innerhalb eines jeden Lagers, aus dem Artikel für die aktuelle Bestellung ausgeliefert werden müssen, wird eine Anwendung gestartet, die als Parameter die Artikelliste erhält und eine Versendegarantie zurückliefert. In einer Schleife werden so oft einzelne Lager angesprochen, bis alle Artikel auf der Bestellung bearbeitet wurden und die Versendung garantiert wurde. Das WFMS muß das Ende der Schleife registrieren. Wenn beide Subworkflows beendet sind, wird der Kontrollfluß erneut abhängig davon aufgespaltet, welche Zahlungsmodalität gewünscht wurde.
Wurde Zahlung per Kreditkarte gewählt, wird die Kreditkarte nun belastet und der Workflow beendet. Soll eine Rechnung gestellt werden, wird eine Aktivität gestartet, die auf den Eingang der Zahlung wartet. Ist die Zahlung erfolgt, ist der Workflow beendet.

Der Datenfluß zwischen den Aktivitäten geschieht wie folgt. Alle Eingabedaten werden in der Aktivität gesammelt, die die Bestellung erfaßt, und von ihr (bedarfsweise) an die übrigen Aktivitäten verteilt. Es ergibt sich ein sternförmiger Datenfluß, ausgehend von dieser Aktivität. Lediglich innerhalb der Schleife, die die Artikel auf die Lager verteilt, gibt es zusätzlichen Datenfluß, nämlich die jeweiligen Artikellisten in die eine und die Versandgarantien in die andere Richtung.

3.2.3 Kontrollfluß-Spezifikation als Statechart

Im folgenden präsentieren wir die Statecharts der vollständigen Spezifikation des Beispiel-Workflows. Der Übersichtlichkeit halber haben wir generell auf die Darstellung von Ausnahmebehandlungen verzichtet (z.B. Nichtlieferbarkeit bestellter Artikel).
Für den Rest dieser Arbeit gehen wir davon aus, daß jeder Statechart, der einen (Sub-) Workflow-Typ spezifiziert, genau einen Endzustand, also einen Zustand ohne ausgehende Transitionen, besitzt. Sollte die Spezifikation eines Workflows ursprünglich mehrere Endzustände aufweisen, kann sie leicht durch jeweils eine zusätzliche Transition von den Endzuständen in einen zusätzlichen Terminierungszustand ergänzt werden.
In Anlehnung an die Entwurfsregeln für WFCharts [Wod97] korrespondiert jeder Zustand des Statecharts zu einer Aktivität oder einem oder mehreren parallelen Subworkflows. Das heißt, daß die Aktivitäten oder die Subworkflows genau dann ausgeführt werden, wenn

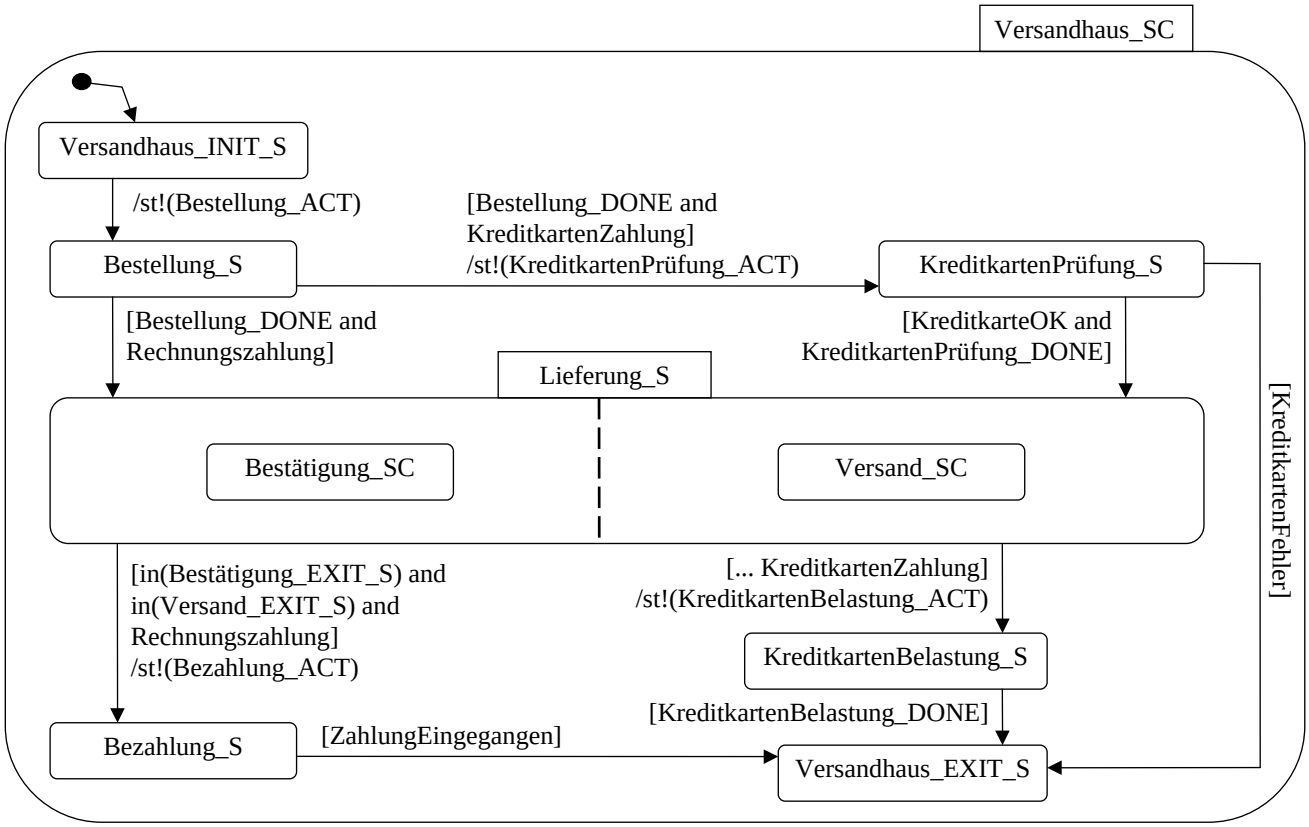


Abbildung 3.3: Top-level Statechart des Beispiel-Workflow-Typs *Versandhaus*

ihr korrespondierender Zustand betreten ist. Darüber hinaus nehmen wir an, daß für jede Aktivität *act* die Bedingiung *act*_DONE auf wahr gesetzt wird, wenn *act* beendet ist. Dies ermöglicht uns die Synchronisation des Kontrollflusses, so daß ein Zustand genau dann verlassen wird, wenn seine korrespondierende Aktivität beendet ist. Zur Synchronisation eines Joins des Kontrollflusses nach der Abarbeitung paralleler Subworkflows benutzen wir die Information, ob die Endzustände der entsprechenden orthogonalen Komponenten betreten sind.

Abbildung 3.3 zeigt das top-level Statechart des Workflow-Typs. Initial wird die Aktivität *Bestellung_ACT* gestartet und der Workflow geht in den Zustand *Bestellung_S* über. Die Bedingungen *KreditkartenZahlung* und *Rechnungszahlung* spezifizieren die gewünschte Zahlungsmodalität und werden von der Aktivität *Bestellung_ACT* gesetzt. Ist die Bedingung *KreditkartenZahlung* wahr, wird die Aktivität *KreditkartenPrüfung_ACT* gestartet, die bei ihrer Terminierung die Bedingung *KreditkarteOK* auf wahr oder falsch setzt. Schlägt der Test fehl, wird die Bedingung *KreditkartenFehler* wahr, und der Workflow wird beendet. Ansonsten geht er in den geschachtelten Zustand *Lieferung_S* über. Innerhalb dieses geschachtelten Zustandes werden zwei Subworkflows parallel ausgeführt. Die beiden Subworkflows sind in den beiden Statecharts *Bestätigung_SC* (Abbildung 3.4) und *Versand_SC* (Abbildung 3.5) spezifiziert, die sich als orthogonale Komponenten in dem geschachtelten top-level Zustand *Lieferung_S* wiederfinden.
In den Subworkflows werden die Aktivitäten *E-Mail_ACT*, *LagerErmittlung_ACT* und *LagerInstruierung_ACT* gestartet. Sie setzen jeweils die Bedingungen *E-Mail_DONE*, *LagerErmittelt* und *VersandGarantiert* auf wahr. Dabei durchlaufen die Subworkflows die Zustände *E-Mail_S*, *LagerErmittlung_S* und *LagerInstruierung_S* und terminieren in den

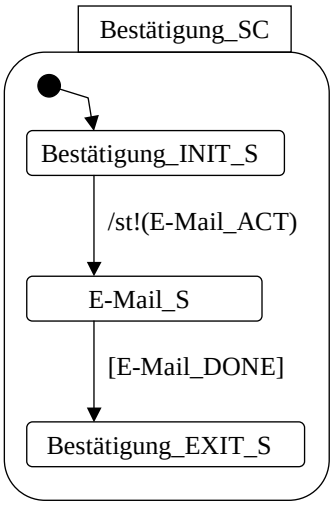


Abbildung 3.4: Statechart des (Sub-)Workflow-Typs *Bestätigung*

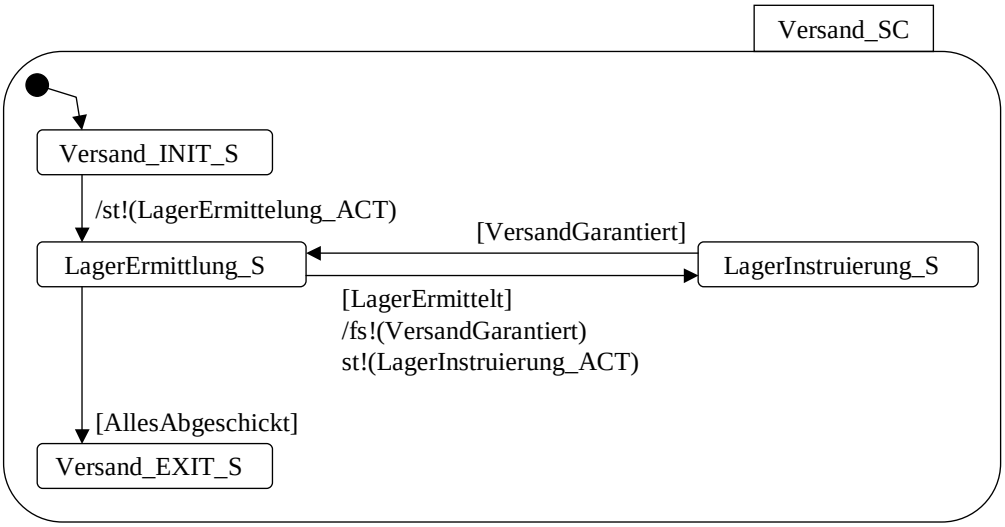


Abbildung 3.5: Statechart des (Sub-)Workflow-Typs *Versand*

Zuständen *Bestätigung_EXIT_S* beziehungsweise *Verand_EXIT_S*.
Haben beide Subworkflows ihren Endzustand erreicht, angezeigt durch die Bedingungen *in(Bestätigung_EXIT_S)* und *in(Versand_EXIT_S)*, wird der geschachtelte Zustand *Lieferung_S* verlassen. Abhängig vom Wert der Bedingungen *KreditkartenZahlung* und *Rechnungszahlung* wird die Kreditkarte durch Starten der Aktivität *KreditkartenBelastung_ACT* belastet und der Workflow endet im Zustand *Versandhaus_EXIT_S*, oder die Aktivität *Bezahlung_ACT* wird gestartet und der Workflow geht in den Zustand *Bezahlung_S* über. Bei der Aktivität *Bezahlung_ACT* wird die Bedingung *ZahlungEingegangen* auf wahr gesetzt, sobald die Rechnung beglichen wurde. Dadurch geht der Workflow in den Zustand *Versandhaus_EXIT_S* über.

3.2.4 Datenfluß-Spezifikation als Activitychart

Abbildung 3.6 skizziert das Activitychart des Workflow-Typs. Alle Eingabedaten werden in der Aktivität *Bestellung_ACT* gesammelt und von ihr bedarfsweise an die übrigen Aktivitäten verteilt. Zusätzlich übermittelt die Aktivität *LagerErmittlung_ACT* der Akti-

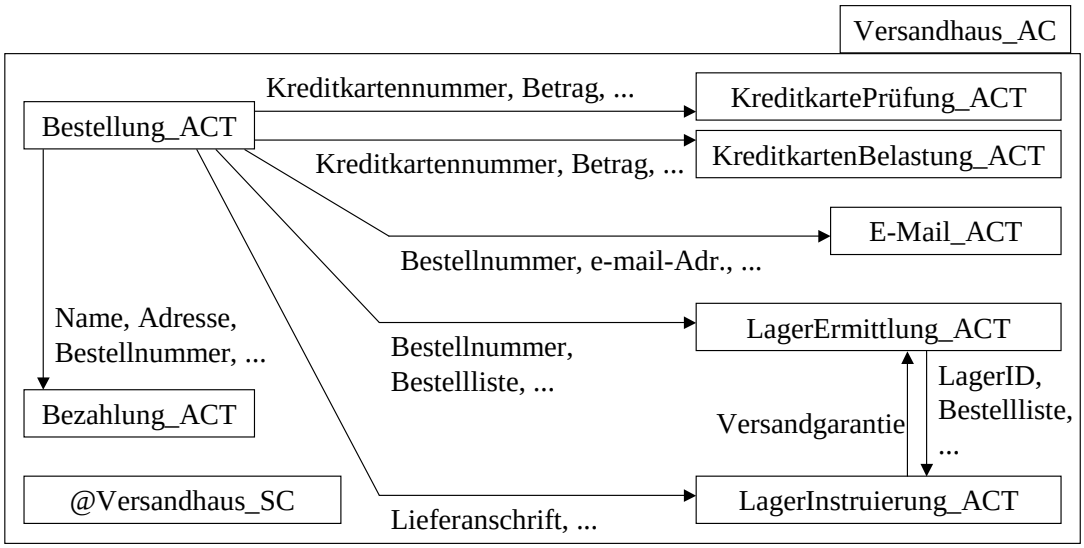


Abbildung 3.6: Activitychart des Workflow-Typs *Versandhaus*

vität *LagerInstruierung_ACT* eine *LagerID* mit der zugehörigen Artikelliste. Die Aktivität *LagerInstruierung_ACT* schickt an ihrem Ende der Aktivität *LagerErmittlung_ACT* die erhaltenen Versandgarantien zurück.

3.2.5 Portierbarkeit

Die Portierung der Workflow-Spezifikation auf Spezifikationsformalismen anderer WFMS ist immer dann einfach, wenn das WFMS ebenfalls einen wohlfundierten "High-Level"-Formalismus für Workflow-Spezifikationen verwendet, beispielsweise Petrinetzvarianten. Die in kommerziellen WFMS verwendeten Spezifikationssprachen sind freilich häufig hybrider Natur, die Elemente verschiedener Formalismen kombinieren - im schlimmsten Fall unter Hinzunahme ausgesprochener Ad-hoc-Konstrukte. Aufgrund unserer Erfahrungen mit verschiedenen Produkten (FlowMark, Staffware, Aris Toolset) stellt sich die Situation allerdings wesentlich besser dar als man pessimistischerweise erwarten könnte. Beispielsweise lassen sich Statechart-Spezifikationen und Ereignis-Prozeß-Ketten, wie sie z.B. dem BPR-Tool Aris zugrundeliegen, weitgehend automatisch ineinander konvertieren [CKS+01, NFZ98, RDR97, WWK+97]. Dadurch erschließt sich uns die Möglichkeit, die später vorgestellten theoretischen Modelle mit Hilfe dieses Beispiel-Workflows auch auf anderen Plattformen zu evaluieren.

3.3 Stochastische Methoden

In diesem Abschnitt führen wir Begriffe und Methoden aus der Stochastik ein, die im weiteren Verlauf dieser Arbeit als wesentliche Grundlagen für Modelle dienen, mit denen wir WFMS beschreiben und damit analytisch erfaßbar machen.

3.3.1 Stochastische Prozesse und Markov-Ketten

In der Literatur werden Prozesse immer dann stochastisch beschrieben, wenn man (meist in die Zukunft gerichtete) Aussagen über sie herleiten möchte, sie jedoch nicht deterministisch erfaßbar sind, sondern lediglich über mehrere Instanzen ein "typisches" Verhalten

erkennen lassen. Das typische Verhalten der Prozesse wird durch Zufallsvariablen beschrieben, deren Verhalten man aus statistischen Informationen ableitet [All90, Bol89, Hav98, Nel95].

Definition 3.1. *(Stochastische Prozesse)*

Ein **stochastischer Prozeß** $\{\mathcal{X}(\tau) | \tau \in T\}$ mit $\mathcal{X}(\tau) \in Z$ ist eine Familie von Zufallsvariablen über dem Zustandsraum Z und dem Parameterraum T. Jeder einzelne Wert $\mathcal{X}(\tau)$ wird als **Zustand** des stochastischen Prozesses bezeichnet.
Abhängig von der Beschaffenheit von Z und T unterscheidet man:

- **zustandsdiskrete Prozesse**: Z ist endlich oder abzählbar.
- **zustandskontinuierliche Prozesse**: Z besteht aus einem oder mehreren Intervallen der reellen Zahlen.
- **zeitdiskrete Prozesse**: T ist endlich oder abzählbar (Oft entspricht bei zeitdiskreten Prozessen T der Menge der natürlichen Zahlen, weswegen sie dann auch $\{\mathcal{X}_n\} = \{\mathcal{X}(n) | n \in \mathbb{N}\}$ geschrieben werden).
- **zeitkontinuierliche Prozesse**: $T = \{[0, s] | s \in \mathbb{R}, s > 0\}$ ist eine Menge von Zeitintervallen (Zeitkontinuierliche Prozesse werden oft vereinfacht $\{\mathcal{X}(\tau)\}$ geschrieben).

Im Folgenden werden wir uns auf Prozesse beschränken, deren Zustandswechsel lediglich von dem aktuellen Zustand und nicht von der Historie des zeitlichen Verlaufes abhängen. Diese Charakteristik heißt *Markov-Eigenschaft*, die resultierenden Prozesse heißen *Markov-Prozesse*:

Definition 3.2. *(Markov-Prozeß)*

Ein stochastischer Prozeß $\{\mathcal{X}(\tau) | \tau \in T\}$ mit $\mathcal{X}(\tau) \in Z$ heißt **Markov-Prozeß** (erster Ordnung), wenn für alle $n \in \mathbb{N}$, $x_1, x_2, ..., x_{n+1} \in Z$ und $\tau_1, \tau_2, ..., \tau_{n+1} \in T$ mit $\tau_1 < \tau_2 < ... < \tau_{n+1}$ gilt:

$$P[\mathcal{X}(\tau_{n+1}) = x_{n+1} | \mathcal{X}(\tau_1) = x_1, \mathcal{X}(\tau_2) = x_2, ..., \mathcal{X}(\tau_n) = x_n] = P[\mathcal{X}(\tau_{n+1}) = x_{n+1} | \mathcal{X}(\tau_n) = x_n]$$

Dabei entspricht $P[A|B]$ der bedingten Wahrscheinlichkeit für das Eintreten des Ereignisses A unter der Bedingung B.

Im Hinblick auf die spätere Anwendung interessieren wir uns im weiteren ausschließlich für Markov-Prozesse mit diskretem Zustandsraum:

Definition 3.3. *(Markov-Ketten)*

Ein Markov-Prozeß $\{\mathcal{X}(\tau) | \tau \in T\}$ mit diskretem Zustandsraum Z heißt **Markov-Kette** (englisch: *Markov chain*).
Markov-Ketten werden abhängig von ihrem Parameterraum weiter klassifiziert:

- Eine **zeitdiskrete Markov-Kette** (englisch: *discrete time Markov chain*, abgekürzt DTMC) $\{\mathcal{X}_n\}$ ist eine Markov-Kette mit diskretem Parameterraum T.
- Eine **zeitkontinuierliche Markov-Kette** (englisch: *continuous time Markov chain*, abgekürzt CTMC) $\{\mathcal{X}(\tau)\}$ ist eine Markov-Kette mit kontinuierlichem Parameterraum T.

Die bedingte Wahrscheinlichkeit $P[\mathcal{X}(\tau_{n+1}) = x_{n+1} | \mathcal{X}(\tau_n) = x_n]$ aus Definition 3.2 entspricht der Wahrscheinlichkeit, daß die Markov-Kette zum Zeitpunkt τ_{n+1} im Zustand x_{n+1} ist, wenn sie zum Zeitpunkt τ_n im Zustand x_n ist. Insbesondere für zeitkontinuierliche Markov-Ketten spielt es jedoch keine Rolle, wie lange der Prozeß bereits im Zustand x_n verweilt, was als *Gedächtnislosigkeit* der Zuvallsverteilung bezeichnet wird. Aus diesen Beobachtungen folgt, daß die Verweilzeiten auf den Zuständen einer zeitkontinuierlichen Markov-Kette exponentialverteilt sein müssen, da die Exponentialverteilung die einzige kontinuierliche Zufallsverteilung mit Gedächtnislosigkeit ist [Bol89, Hav98, Nel95, Tij94].

Hängt die Wahrscheinlichkeit $P[\mathcal{X}(\tau_{n+1}) = x_{n+1} | \mathcal{X}(\tau_n) = x_n]$ einer Markov-Kette ausschließlich von der Zeitdifferenz $(\tau_{n+1} - \tau_n)$ ab, nicht aber vom Wert des Zeitpunktes τ_n, spricht man von einer *homogenen Markov-Kette.*

Definition 3.4. *(Homogene Markov-Kette)*

Eine Markov-Kette ist eine **homogene Markov-Kette**, wenn für alle $n \in \mathbb{N}$, $x_n, x_{n+1} \in Z$ und $\tau_n, \tau_{n+1} \in T$ mit $\tau_n < \tau_{n+1}$ gilt:

$$P[\mathcal{X}(\tau_{n+1}) = x_{n+1} | \mathcal{X}(\tau_n) = x_n] = P[\mathcal{X}(\tau_{n+1} - \tau_n) = x_{n+1} | \mathcal{X}(0) = x_n]$$

Basierend auf der rechten Seite der Gleichung aus Definition 3.2 können wir die *Übergangswahrscheinlichkeit* von einem Zustand zu einem anderen Zustand innerhalb eines vorgegebenen Intervalls bestimmen:

Definition 3.5. *(Übergangswahrscheinlichkeit $p_{ij}^{(\tau)}$)*

Die **Übergangswahrscheinlichkeit**

$$p_{ij}^{(\tau)} = P[\mathcal{X}(\tau) = j | \mathcal{X}(0) = i]$$

einer homogenen Markov-Kette $\{\mathcal{X}(\tau) | \tau \in T\}$ entspricht der Wahrscheinlichkeit dafür, daß der Prozeß in τ Zeiteinheiten in Zustand j ist, wenn er zum aktuellen Zeitpunkt in Zustand i ist.

Für zeitdiskrete Markov-Ketten wird $p_{ij}^{(n)}$ für $n = 0, 1, 2, ...$ als *n-Schritt-Übergangswahrscheinlichkeit* bezeichnet. Für die Beschreibung von Markov-Ketten besonders interessant ist die 1-Schritt-Übergangswahrscheinlichkeit.

Definition 3.6. *(Schrittwahrscheinlichkeit p_{ij})*

Für zeitdiskrete Markov-Ketten bezeichnen wir die 1-Schritt-Übergangswahrscheinlichkeit $p_{ij}^{(1)}$ vereinfachend als **Schrittwahrscheinlichkeit** p_{ij}.

Um konkrete Vorhersagen über das Verhalten einer Markov-Kette innerhalb eines Beobachtungsintervalles vornehmen zu können, muß für jeden Zustand die Wahrscheinlichkeit festgelegt sein, daß sich der Prozeß zu Beginn der Beobachtung, also zum Zeitpunkt Null in dem Zustand befindet.

Definition 3.7. *(Anfangswahrscheinlichkeit $p_j^{(0)}$)*

Die **Anfangswahrscheinlichkeit** $p_j^{(0)}$ eines Zustandes $j \in Z$ einer Markov-Kette ist die Wahrscheinlichkeit, mit der Zustand j der aktuelle Zustand zum Zeitpunkt Null ist:

$$p_j^{(0)} = P\left[\mathcal{X}\left(0\right) = j\right]$$

Für die Berechnung der Übergangswahrscheinlichkeiten $p_{ij}^{(\tau)}$ für beliebige $i, j \in Z$ und $\varsigma, \tau \in T$ gilt die *Chapman-Kolmogorov-Gleichung* [Bol89, Hav98, Nel95, Tij94]:

$$p_{ij}^{(\tau)} = \sum_{k \in Z} p_{ik}^{(\tau-\varsigma)} p_{kj}^{(\varsigma)} \tag{3.1}$$

Für zeitdiskrete Markov-Ketten gilt damit insbesondere:

$$p_{ij}^{(n)} = \sum_{k \in Z} p_{ik}^{(n-1)} p_{kj} \qquad , p_{ij}^{(0)} = \begin{cases} 1 & , \text{falls } \; i = j \\ 0 & , \text{falls } \; i \neq j \end{cases} \tag{3.2}$$

Aus den Übergangswahrscheinlichkeiten lassen sich weitere Eigenschaften der Markov-Kette und ihrer Zustände ableiten:

Definition 3.8. *(irreduzibel, rekurrent, ergodisch)*

Eine Markov-Kette heißt **irreduzibel**, wenn jeder Zustand ausgehend von jedem anderen Zustand erreicht werden kann, das heißt

$$p_{ij}^{(\tau)} > 0$$

für alle $i, j \in Z$.
Ein Zustand heißt **rekurrent**, wenn der Prozeß nach Verlassen des Zustandes mit Wahrscheinlichkeit 1 wieder in diesen Zustand zurückkehrt. Ist dabei die Zeit bis zur ersten Rückkehr endlich, so heißt der Zustand **rekurrent nicht-null**, ansonsten heißt der Zustand **rekurrent null**.
Eine homogene, irreduzible Markov-Kette heißt **ergodisch**, wenn alle ihre Zustände rekkurent nicht-null sind.

Die *Ergodizität* einer Markov-Kette ist Voraussetzung für die *stationäre Analyse*, aus der die von den *Anfangswahrscheinlichkeiten* unabhängigen *stationären Zustandswahrscheinlichkeiten* der "eingeschwungenen" Markov-Kette resultieren [Bol89, Tij94].

Definition 3.9. *(stationäre Zustandswahrscheinlichkeit π_i)*

Der von den Anfangswahrscheinlichkeiten unabhängige Grenzwert

$$\pi_i = \lim_{\tau \to \infty} p_{ji}^{(\tau)}$$

mit beliebigem $j \in Z$ heißt *stationäre Zustandswahrscheinlichkeit* des Zustandes i [BGM+98].

Im Gegensatz zur stationären Analyse steht die *transiente Analyse* von Markov-Ketten, die Aussagen über das Verhalten des Prozesses in einem gegebenen zukünftigen Zeitintervall erlaubt [Tij94].

Die Anwendung von Markov-Ketten, die in der Literatur am häufigsten untersucht und als Beispiel verwendet wird, sind Warteschlangenmodelle [All90, Bol89, Hav98, Nel95, Tij94]. In [Kra98, KW97, KW98] werden Markov-Ketten dazu verwendet, das Zugriffsverhalten von Benutzern auf Dokumente eines Informationssystemes zu modellieren.
Wir werden in dieser Arbeit mit Hilfe der Methode der zeitkontinuierlichen Markov-Ketten zum einen das Kontrollflußverhalten von Workflows (Kapitel 5) und zum anderen die Verfügbarkeit von Komponenten eines verteilten WFMS (Kapitel 7) modellieren.

3.3.2 Zeitkontinuierliche Markov-Ketten mit allgemein verteilten Zustandsverweilzeiten

Aus Definition 3.2 haben wir abgeleitet, daß die Verweilzeiten in den einzelnen Zuständen einer zeitkontinuierlichen Markov-Kette exponentialverteilt sein müssen. Diese wesentliche Einschränkung kann jedoch bei der Modellierung realer Prozesse eine unakzeptable Voraussetzung sein. Im folgenden beschreiben wir, wie wir eine CTMC erweitern können, so daß auch allgemeine Verteilungen der Zustandsverweilzeiten berücksichtigt werden können.
Die Basis für diese Erweiterung bildet das folgende Theorem ([Tij94] Theorem 2.9.1):

Theorem 3.1. *Approximation einer allgemeinen Zufallsverteilung*

Sei $F(\tau)$ die Verteilungsfunktion einer beliebigen positiven Zufallsvariablen.
Für ein festes $\Delta > 0$ sei die Verteilungsfunktion $F_\Delta(\tau)$ für $\tau \geq 0$ durch

$$F_\Delta(\tau) = \sum_{j=1}^{\infty} p_j(\Delta) \left(1 - \sum_{i=0}^{j-1} e^{-\tau/\Delta} \frac{(\tau/\Delta)^i}{i!} \right)$$

mit $p_j(\Delta) = F(j\Delta) - F((j-1)\Delta)$, $j = 1, 2, \ldots$ definiert.
Dann gilt für jedes $\tau \geq 0$, an dessen Stelle $F(\tau)$ stetig ist, daß

$$\lim_{\Delta \to \infty} F_\Delta(\tau) = F(\tau).$$

Dabei entspricht die Funktion $F_{E(a,k)}(x) = 1 - \sum_{i=0}^{k-1} e^{-x/a} \frac{(x/a)^i}{i!}$ der Verteilungsfunktion der *Erlang-k-Verteilung* $E(a,k)$ mit Parameter a. Der Beweis des Theorems ist in [Tij94] beschrieben.
Das Theorem sagt aus, daß sich die Wahrscheinlichkeitsverteilung jeder positiven Zufallsvariablen beliebig nahe durch eine *verallgemeinerte Erlangverteilung* approximieren läßt. Die Verteilungsfunktion einer verallgemeinerten Erlangverteilung entspricht der Funktion $F_\Delta(\tau)$. Bei einer verallgemeinerten Erlangverteilung werden mehrere Erlang-k-Verteilungen über eine Hyperexponentialverteilung "parallel geschaltet".
Jede dieser Erlang-k-Verteilungen läßt sich wiederum durch die Summe von k Exponentialverteilungen mit identischen Mittelwerten beschreiben [Bol89, Tij94].

Angewendet auf unser Problem der allgemeinen Verteilung der Zustandsverweilzeiten einer CTMC heißt das, daß wir jeden nicht-exponentiellen Zustand der CTMC durch ein System von Zuständen mit exponentialverteilten Zustandsverweilzeiten substituieren können. Ein solches System von Zuständen "simuliert" dann eine die allgemeine Verteilung approximierende verallgemeinerte Erlang-Verteilung für die Verweilzeit in dem substituierten Zustand.

3.3.3 Markov-Reward-Modelle

Die Basis eines Markov-Reward-Modelles (MRM) ist eine CTMC $\{\mathcal{X}(\tau) | \tau \geq 0\}$ mit diskretem Zustandsraum Z. Auf dem Zustandsraum Z der CTMC definieren wir zwei Reward-Funktionen.

Definition 3.10. *(Reward-Rate r_i)*

Für jeden Zustand $i \in Z$ der einem MRM zugrundeliegenden CTMC beschreibt die Funktion

$$\begin{aligned} r : Z &\rightarrow \mathbb{R} \\ i &\mapsto r(i) = r_i \end{aligned}$$

eine "Belohnung", die die CTMC für jede Zeiteinheit erhält, die sie im Zustand i verbringt. $r(i)$ heißt die **Reward-Rate** des Zustandes i und wird verkürzt auch als r_i geschrieben.

Definition 3.11. *(Pauschal-Reward L_i)*

Für jeden Zustand $i \in Z$ der einem MRM zugrundeliegenden CTMC beschreibt die Funktion

$$\begin{aligned} L : Z &\rightarrow \mathbb{R} \\ i &\mapsto L(i) = L_i \end{aligned}$$

die "Belohnung", die die CTMC für jedes Eintreten in den Zustand i erhält. $L(i)$ heißt der **Pauschal-Reward** des Zustandes i und wird verkürzt auch als L_i geschrieben.

Beim Pauschal-Reward eines Zustandes spielt es somit keine Rolle, wie lange der Prozeß nach dem Betreten in dem Zustand verweilt. Die Reward-Rate hingegen berücksichtigt genau dieses Verhalten.

Ein MRM kombiniert diese beiden Funktionen mit der Analyse der zugrunde liegenden CTMC. Daraus können Aussagen über

- den akkumulierten Reward, sowie
- den Erwartungswert der Reward-Rate

resultieren. Dabei können diese Aussagen sowohl transienter als auch (falls die Anforderungen an die CTMC gegeben sind) stationärer Natur sein.
Wir werden uns in dieser Arbeit

- des transienten akkumulierten Rewards eines MRM zur Abschätzung von Service-Aufträgen eines Workflows (Kapitel 6) und
- des stationären Erwartungswertes der Reward-Rate eines MRM zur Abschätzung der Leistung unter Berücksichtigung von transienten Komponentenausfällen im WFMS (Kapitel 8)

bedienen.

3.3.4 M/G/1-Warteschlangenmodelle

In der Warteschlangentheorie wird das Verhalten des Modelles eines Rechensystems oft durch stochastische Prozesse modelliert. Die Modellparameter werden dabei durch Zufallsvariable wie Zwischenankunfts- und Bedienzeit von Service-Aufträgen beschrieben. Daraus lassen sich statistisch verteilte Leistungsmaße wie Warte- und Antwortzeiten der Service-Aufträge ableiten.

Definition 3.12. *(Wartezeit)*

> Die Zeit, die ein Service-Auftrag an einer Bedienstation auf seine Abarbeitung warten muß, heißt **Wartezeit** des Service-Auftrages.

Definition 3.13. *(Antwortzeit)*

> Die Zeit, die von der Initiierung eines Service-Auftrages bis zur Beendigung der Abarbeitung des Service-Auftrages vergeht, heißt **Antwortzeit** des Service-Auftrages.

Hinter der Abkürzung M/G/1 verbirgt sich das Modell eines Rechensystems mit einer Bedienstation ("1") mit exponentialverteilter Zwischenankunftszeit ("M") und beliebig verteilter Bedienzeit ("G"). Zur Analyse eines solchen Modelles wird eine eingebettete zeitdiskrete Markov-Kette verwendet, bei der die Zustandsübergänge auf die Zeitpunkte fallen, zu denen ein Auftrag die Bedienstation verläßt. Die Zustände selbst entsprechen dabei einer bestimmten Anzahl von Service-Aufträgen im System. Schließlich liefert die Analyse eines M/G/1-Warteschlangenmodelles Aussagen über die Verteilung der

Warteschlangenlänge, sowie den Erwartungswert für die Wartezeit der Service-Aufträge [All90, Hav98, Kle76, Lan92, Tak91].

In dieser Arbeit werden wir M/G/1-Warteschlangenmodelle zur Modellierung von Workflow-Servern benutzen, um die mittlere Wartezeit von Service-Aufträgen abzuschätzen, die bei der Abarbeitung von Workflow-Instanzen anfallen (Kapitel 6).

Teil II

Modellbildung

Kapitel 4

Systemmodell

In diesem Kapitel entwickeln wir ein Systemmodell für verteilte WFMS. Dazu untersuchen wir in Abschnitt 4.1 eine Auswahl von Architekturen von WFMS-Produkten und -Prototypen sowie die Vorgaben der Workflow-Management-Coalition [WFMC]. Danach stellen wir in Abschnitt 4.2 unser Systemmodell vor, auf das trotz seiner Einfachheit alle vorgestellten Architekturen abbildbar sind. Basierend auf diesem Modell präsentieren wir in Abschnitt 4.3 die Definitionen der zentralen Begriffe *Systemkonfiguration* und *Systemzustand* von verteilten WFMS. Das Kapitel endet mit einer Zusammenfassung und Diskussion des vorgestellten Modelles.

4.1 Architektur-Ansätze

4.1.1 Vorgaben der Workflow Management Coalition (WfMC)

Die Workflow Management Coalition (WfMC) ist ein Industriekonsortium, das sich zum Ziel gesetzt hat, Schnittstellen zwischen Schlüsselkomponenten von WFMS zu standardisieren. Dazu wurden eine einheitliche Terminologie für das Workflow-Umfeld erarbeitet und ein Referenzmodell entwickelt, das die Architektur von WFMS abstrakt beschreibt [WFMC]. Das Referenzmodell ist in Abbildung 4.1 dargestellt.

Das Referenzmodell umfaßt alle Komponenten einer Arbeitsumgebung, die in den Entwurf, die Ausführung und die Administration von Workflows involviert sind. Im Rahmen dieser Arbeit konzentrieren wir uns auf die Komponenten, die wesentlich an der Ausführung von Workflow-Instanzen mitwirken. Komponenten zur Administration und zum Entwurf kommen oft im Offline-Betrieb zum Einsatz oder stellen keine kritischen Faktoren im Bezug auf die Konfiguration von WFMS dar.

Die Ausführung und Steuerung von Workflow-Instanzen obliegt dem *Workflow Enactment Service*. Der Workflow Enactment Service koordiniert der Workflow-Spezifikation folgend unter anderem die Zuweisung von Arbeitsaufträgen und den Start von Aktivitäten. Seine Aufgaben können von einer oder mehreren Workflow-Engines ausgeführt werden, so daß auch eine verteilte Ausführung vorgesehen ist.

Weitere an der Ausführung von Workflows beteiligte Komponenten sind *Workflow Client Applications* und *Invoked Applications*. Die Schnittstellen zwischen den Anwendungsprogrammen und dem Workflow Enactment Service ("Interface 2" und "Interface 3" in

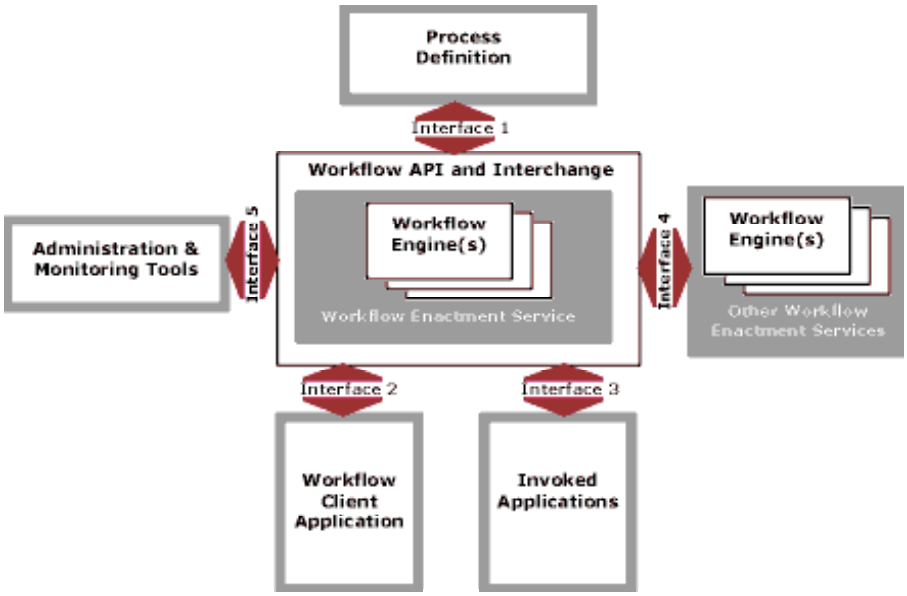


Abbildung 4.1: Referenzmodell der Workflow Management Coalition [WFMC]

Abbildung 4.1) werden zusammengefaßt als "Workflow Application Programming Interface" (WAPI) bezeichnet. Das WAPI definiert Aufruf- und Steuerungsfunktionen, die sowohl von den Workflow-Engines als auch von den Applikationen unterstützt werden sollen. Außer der Möglichkeit, diese Funktionen verfügbar zu machen, stellt das WAPI keine einschneidenden Anforderungen an die verwendete Middleware. Daraus resultiert der Freiheitsgrad der Ortsunabhängigkeit der Ausführung der Aktivitäten, zum Beispiel eine Verteilung auf dedizierte Applikations-Server.

4.1.2 ADEPT

Der Prototyp des ADEPT-Projektes [ADEPT, DR98, RBD00], das sich besonders mit der Möglichkeit dynamischer Anpassungen von Workflow-Instanzen zur Laufzeit befaßt [RD97, RD98, RHD98, RBD99, BRD01a], bedient sich einer Multi-Server-Architektur. Während der Ausführung einer Workflow-Instanz kann die Kontrolle über diese Instanz von einem Workflow-Server zu einem anderen migrieren. Dabei werden alle Laufzeitdaten der Instanz zu dem neuen Server überführt, damit sie in zukünftigen Arbeitsschritten "näher beim Bearbeiter" liegen [BD97, BRD01b].

Der Prototyp ist in Java implementiert. Für die Kommunikation zwischen den Servern und zu den Clients wurde Java-RMI benutzt [HRB+00].

4.1.3 $WASA_2$

Das $WASA_2$ Projekt [VW99, WV98] verfolgt einen objektorientierten Ansatz zur Realisierung von verteilten WFMS. Workflows werden als CORBA-Objekte implementiert, in die auch Funktionen zur Zuordnung von Arbeitsaufträgen zu Personen integriert sind. Die Workflow-Objekte werden durch CORBA Common Object Services (COSS) ergänzt. Dadurch werden unter anderem Fehlertoleranz sichergestellt und eine einfache Erzeugung neuer Workflow-Objekte ermöglicht.

Die einzelnen Workflow-Objekte kommunizieren über einen Object Request Broker (ORB) miteinander. Die Aktivitäten werden allgemein als Business-Objekte angesehen und ebenfalls über den ORB angesprochen. Für die Schnittstelle zu den Benutzern wurde Java verwendet.

4.1.4 Meteor$_2$

Im Rahmen des Meteor$_2$ Projektes [Meteor] wurden drei Architekturansätze entwickelt und implementiert, die sich hinsichtlich ihrer Verteilungsstrategien unterscheiden. Den Kern der Architektur bilden ein Task-Scheduler und mehrere Task-Manager. Tasks sind die Aktivitäten einer Workflow-Instanz. Sie werden von den Task-Managern über eine IDL (Interface Definition Language) Schnittstelle koordiniert gestartet und mit Daten versorgt.

- OrbWork [DKM+97, SK98] besitzt einen auf CORBA basierenden, komplett verteilten Enactment-Service. Der Task-Scheduler ist auf die Task-Manager verteilt, die ihrerseits die automatischen Tasks über Wrapper steuern. Die Task-Manager selbst sind ebenfalls CORBA-Objekte. Für Aktivitäten mit Benutzerinteraktion ("user tasks") werden vom Task-Manager HTML-Formulare generiert und in der Arbeitsliste des Benutzers hinterlegt. Die Bearbeitung der Aktivität findet im Browser statt.

- NeoWork ist ein Enactment-Service mit einem zentralen Scheduler. Die Kommunikation mit den Tasks findet auch hier mittels CORBA statt.

- Den neuesten Architekturansatz aus dem Meteor$_2$ Projekt bildet der vollständig verteilte Enactment-Service von WebWork, der ausschließlich auf Web-Technologien zurückgreift [MPS+98].

4.1.5 Mentor-lite

Das besondere Ziel von Mentor-lite [Mentor] war die Entwicklung eines leichtgewichtigen, erweiterbaren und anpassbaren WFMS mit geringen Anforderungen an Hardware-Ressourcen und benutzerfreundlicher Administrierbarkeit. Der architektonische Ansatz besteht darin, in der Workflow-Engine ausschließlich Kernel-Funktionalität bereitzustellen, während weitere Systemkomponenten wie History-Management und Worklist-Management als darüberliegende Zusatzschicht realisiert werden. Dabei werden solche Erweiterungen das Ziel der Leichtgewichtigkeit anstrebend selbst als (Sub-)Workflows implementiert [GWS+00b, MWG+98, MWG+99a, MWG+99b, Wei00b, WGR+00].

Abbildung 4.2 zeigt den Aufbau von Mentor-lite. Die zentrale Komponente bildet ein Interpreter für Workflow-Spezifikationen. An den Interpreter eng gekoppelt sind der Kommunikationsmanager (ComMgr) und der Logmanager (LogMgr). Der Kommunikationsmanager ist verantwortlich für den Austausch von Synchronisationsnachrichten zwischen verschiedenen Workflow-Engines bei partitioniert-verteilter Workflow-Ausführung. Um auch im Falle von Netzwerk- oder Rechnerausfällen die Konsistenz der verteilten Workflow-Ausführung erhalten zu können, stehen den Kommunikationsmanagern Reliable Message Queues zur Verfügung, die transaktionsgeschützt über die Object Transaction Ser-

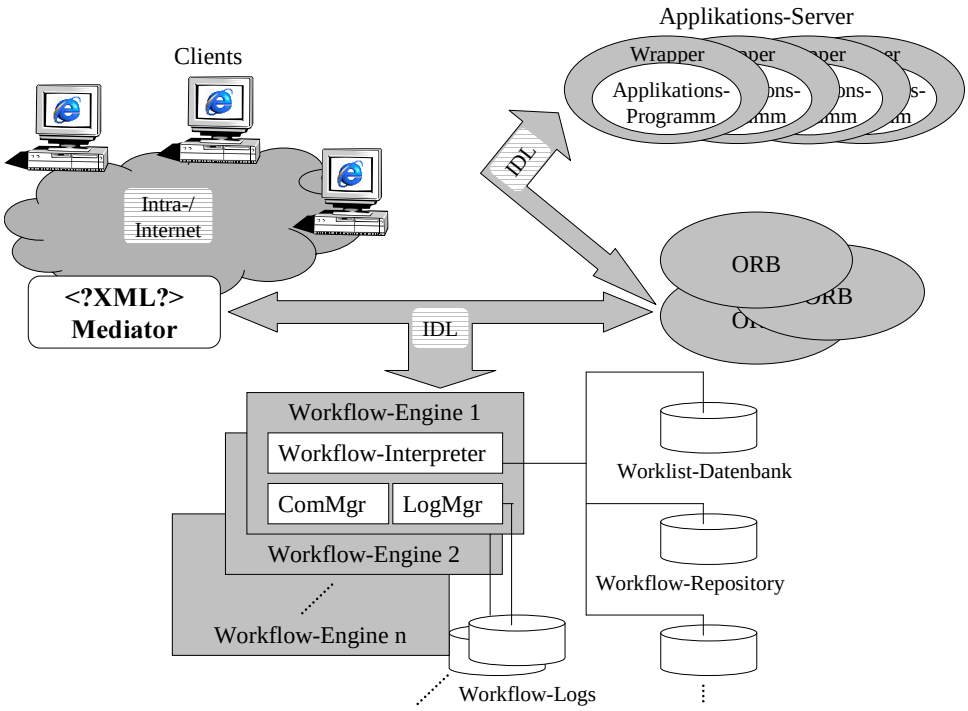


Abbildung 4.2: Architektur von Mentor-lite

vices (OTS) von CORBA angesprochen werden. Der Logmanager übernimmt das Logging der Workflow-Ausführung und die Recovery bei transienten Fehlern. Dazu steht jeder Workflow-Engine ein eigener Workflow-Log zur Verfügung. Weitere Datenbanken, wie das Workflow-Repository (eine Ansammlung aller Workflow-Spezifikationen) oder die Worklist-Datenbank, können von mehreren Workflow-Engines geteilt werden.

Als Schnittstelle zu Applikations-Programmen dienen ein generisches IDL-Interface auf Seite der Workflow-Engine sowie zugeschnittene Wrapper auf der Seite der Programme. Sowohl die Kommunikation der Workflow-Engines untereinander als auch die Kommunikation zwischen Workflow-Engine und Applikations-Programmen sind mit der CORBA-Implementation Orbix [IONA] realisiert.

Die neueste Version von Mentor-lite erlaubt den Benutzern, Arbeitsaufträge über das Internet abzurufen und zu bearbeiten. Als Front-End genügt ein handelsüblicher Web-Browser, der über einen XML-Mediator Daten mit der Workflow-Engine und den Datenbanken austauscht. Der Mediator bedient sich dabei der IDL-Schnittstelle der Workflow-Engine [SGW00, SGW01].

4.1.6 Staffware97

Das kommerzielle Produkt Staffware97 [Staff] basiert auf einer Client-Server-Architektur mit monolitischem Server. Alle Komponenten des WFMS wie zum Beispiel ein Log-Manager, der Worklist-Handler u.s.w. sind in die Workflow-Engine integriert. Auf den Client-Rechner dient eine graphische Benutzeroberfläche zum Starten und Beobachten von Workflow-Instanzen, sowie der Entgegennahme von Arbeitsaufträgen.

Staffware97 unterstützt keine partitioniert-verteilte Ausführung von Workflows. Zwar

können Replikate der Workflow-Engine auf mehreren Rechnern installiert werden, aber eine Workflow-Instanz ist während der Ausführung mit allen ihren Subworkflows an genau eine Instanz der Workflow-Engine gebunden.

Die Schnittstelle zu Applikations-Programmen basiert auf dem Dynamic Data Exchange (DDE) Standard. Externe Applikations-Programme werden über Skripte, in die DDE-Kommandos eingebettet sind, gestartet. Automatische, nicht-interaktive Aktivitäten, also Aktivitäten, bei denen keine Benutzerinteraktion stattfindet, können auch ohne Verwendung solcher Skripte aufgerufen werden. Die Anwendungsprogramme, die diese Aktivitäten implementieren, müssen dann jedoch auf demselben Rechner installiert sein, auf dem die Workflow-Engine läuft. Der Einsatz von dedizierten Applikations-Servern ist in diesem Fall nicht möglich. Der Datenaustausch zwischen Workflow-Engine und Applikations-Programmen erfolgt über das Dateisystem. Sowohl Input- als auch Output-Daten der Applikations-Programme werden in temporären Dateien hinterlegt. Bei asynchroner Ausführung der Applikations-Programme liegt es in der Verantwortung des aufrufenden Skripts, ein Event zu generieren, wenn das Applikations-Programm beendet ist und die Workflow-Engine die Rückgabedaten lesen darf. Automatische, nicht-interaktive Aktivitäten, die auf demselben Rechner wie die Workflow-Engine laufen, können über Pipes direkt mit der Workflow-Engine kommunizieren.

4.1.7 MQSeries Workflow

Das kommerzielle Produkt MQSeries Workflow basiert auf einer Drei-Schichten-Architektur mit einer zentralen Datenbank als Basisschicht [IBM]. In der Datenbank sind alle zur Workflow-Ausführung relevanten Daten hinterlegt. Die mittlere Schicht der Architektur bilden MQSeries Workflow-Server. Die Workflow-Server können über mehrere Rechner verteilt sein. Für diesen Fall wird ein Mechanismus zur Lastbalancierung bereitgestellt. Die dritte Schicht bilden die MQSeries Workflow-Clients. Auf ihnen werden die Aktivitäten mit Benutzerinteraktion und die Arbeitslisten ausgeführt. Zur Ausführung von automatischen Aktivitäten ohne Benutzerinteraktion gibt es einen Program Execution Server, der zur mittleren der drei Schichten gezählt wird.

Bei MQSeries handelt es sich um eine Produktfamilie von IBM. Zur Kommunikation innerhalb einer auf MQSeries basierenden Arbeitsumgebung wird ein System von Message Queues benutzt. Um auf Client-Seite Freiraum für anwendungsspezifische Erweiterungen zu schaffen, unterstützen die Workflow-Server auch die Kommunikation mittels CORBA IIOP (Internet InterORB Protocol).

4.2 Verteiltes Server-Modell

Im letzten Abschnitt haben wir eine Reihe von WFMS-Produnkten und -Prototypen untersucht. Basierend auf diesen Beobachtungen werden wir in diesem Abschnitt die Architekturmerkmale der Systeme abstrahieren und ein geeignetes, allgemeingültiges Model für verteilte WFMS ableiten.

Ein verteiltes WFMS führt Workflows auf eine dezentralisierte Art und Weise aus. Jeder Workflow ist in mehrere Subworkflows unterteilt, die potentiell von verschiedenen

Workflow-Servern ausgeführt werden. Zum Beispiel sind die Subworkflows entsprechend der an ihnen beteiligten Organisationseinheiten der Firma auf die jeweiligen Workflow-Server dieser Einheiten verteilt [BD99a]. Applikations-Programme unterschiedlicher Arten laufen auf spezifischen *Applikations-Servern*. Schließlich wird die Koordination der Kommunikation innerhalb der oftmals hochgradig verteilten und heterogenen Systemumgebung einer speziellen Art von *Kommunikations-Servern* überlassen, zum Beispiel Object Request Brokern (ORBs) [DGA+98, DKM+97, GCS+98, HHJ+00, MWG+99a, VW99], TP-Monitoren, Web-Servern [EGL98, MPS+98] oder ähnlichen Ausprägungen von Middleware [HRB+00].
Diese drei Arten von Bedienstationen in WFMS – Workflow-Server, Applikations-Server und Kommunikations-Server - betrachten wir in unserem Systemmodell als Server spezifischer Typen.

Definition 4.1. *(Server-Typ und Server)*

> Ein **Server-Typ** eines Workflow-Management-Systems ist ein spezieller Typ von Workflow-Servern, Applikations-Servern oder Kommunikations-Servern.
> Ein **Server** eines Workflow-Management-Systems ist eine Instanz eines Server-Typs.

In dieser Arbeit beschränken wir uns auf die Annahme, daß an der Ausführung einer Aktivität genau ein Typ von Workflow-Servern, ein Typ von Applikations-Servern und ein Typ von Kommunikations-Servern – also genau drei verschiedene Server-Typen – beteiligt sind. Jede Aktivität erzeugt jeweils eine für sie spezifische Last auf jedem der drei für sie zuständigen Server-Typen.

Das Zusammenwirken von Servern während der Ausführung eines Workflows ist in dem Sequence Diagram von Abbildung 4.3 illustriert. Bei der Erstellung von Abbildung 4.3 haben wir uns an Sequence Diagrams des UML-Industriestandards [UML] orientiert. Wir unterscheiden zwei Arten von Operationsaufrufen: *synchron* und *assynchron*. Bei einem asynchronen Operationsaufruf fährt der aufrufende Prozeß mit seiner Arbeit fort, nachdem er den Aufruf abgesetzt hat. Bei einem synchronen Operationsaufruf wartet er hingegen, bis die aufgerufene Operation beendet ist und eine Nachricht zurückgegeben hat. Der Ausschnitt des Sequence Diagrams zeigt drei an der Ausführung beteiligte Server-Typen sowie einen Client. Ein Client ist eine Arbeitsstation eines am Workflow beteiligten Benutzers. Clients können sowohl zum Starten von Workflows als auch – wie im Falle von Abbildung 4.3 – zum Bearbeiten interaktiver Aktivitäten dienen.

Der erste Teil des Ausschnittes skizziert den Austausch von Aufträgen und Nachrichten zwischen den Server-Typen, wie er während der asynchronen Ausführung einer automatischen Aktivität anfällt. Der zweite Teil beschreibt das Zusammenwirken der Komponenten bei Ausführung einer interaktiven Aktivität. Da das zur Aktivität gehörende Applikations-Programm auf dem Client läuft, ist der Applikations-Server an der Ausführung dieser Aktivität nicht beteiligt. Stattdessen wird hier das Worklist-Management auf der Seite des Workflow-Servers benötigt (*"Thread 2"* in Abbildung 4.3).

Detailliertere Informationen über das Zusammenwirken der Server-Typen, zum Beispiel zu welchen Zeitpunkten genau wieviele Aufträge zwischen den Server-Typen verschickt

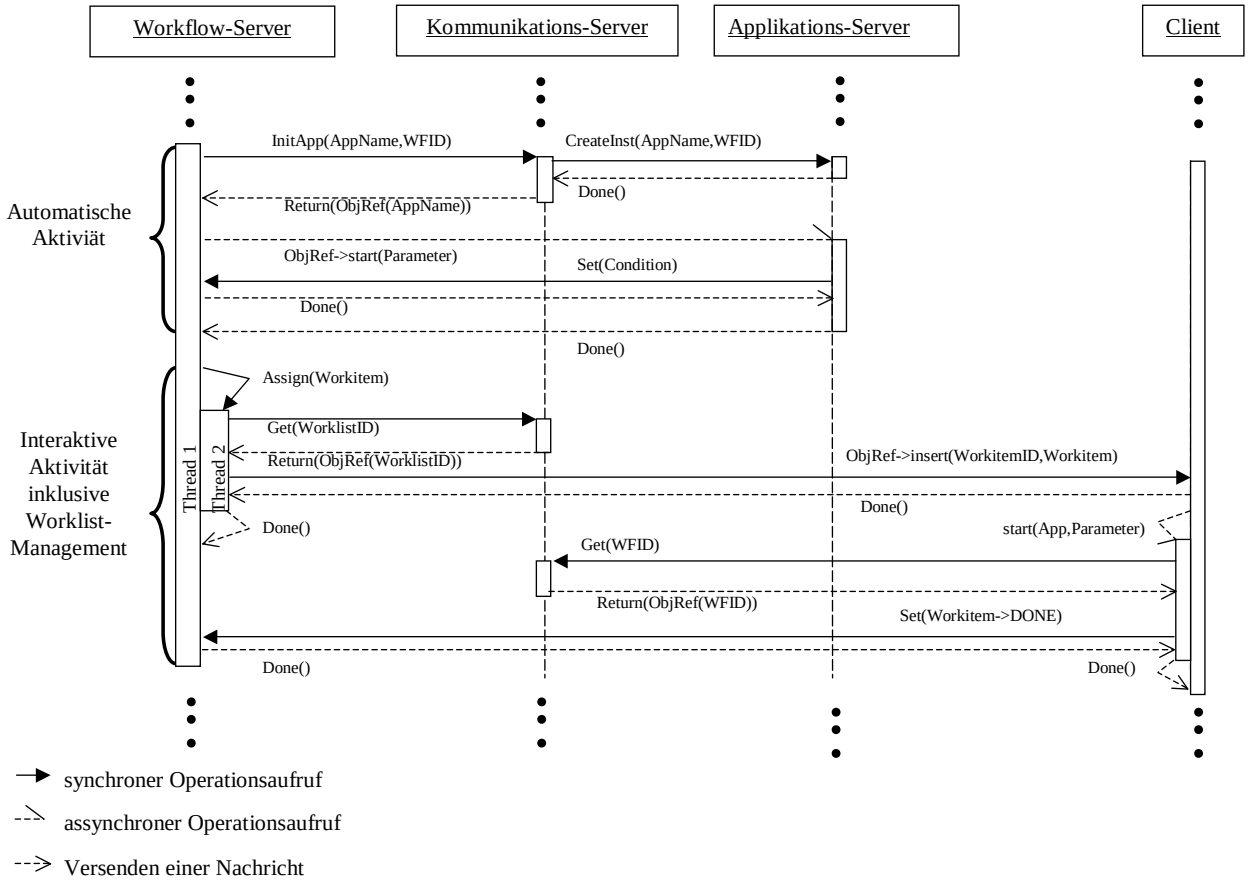


Abbildung 4.3: Sequence Diagram für die Ausführung zweier Aktivitäten

werden, benötigen wir im Zusammenhang mit der Bewertung und Konfiguration von verteilten WFMS nicht. Uns genügt die Abschätzung der Gesamtlast, die eine Aktivität bei den verschiedenen an ihrer Ausführung beteiligten Server-Typen erzeugt.

Definition 4.2. *(Last)*

> Die Anzahl von Aufträgen, die während eines Beobachtungszeitraumes an einem Server-Typ anfallen, heißt die **Last** an dem Server-Typ für den Beobachtungszeitraum.

In unserem Beispiel von Abbildung 4.3 können wir feststellen, daß die Ausführung der automatischen Aktivität eine Last von drei Aufträgen an dem Workflow-Server, zwei Aufträgen an dem Kommunikations-Server und drei Aufträgen an dem Applikations-Server erzeugt.

Um Skalierbarkeit zu unterstützen und eine höhere Verfügbarkeit zu gewährleisten bedienen sich die meisten WFMS der Replikation von Server-Typen. So kann zum Beispiel ein Workflow-Server, der für die Ausführung eines bestimmten Typs von Subworkflows zuständig ist, auf mehreren Rechnern installiert sein. Die auftretende Gesamtlast für diesen Server-Typ kann dann über alle Instanzen verteilt werden, zum Beispiel durch ein hash- oder round-robin-basiertes Zuweisungsverfahren. Darüber hinaus erfüllen die einzelnen Server eines Typs die Aufgabe sogenannter Backup-Server füreinander. Dabei werden

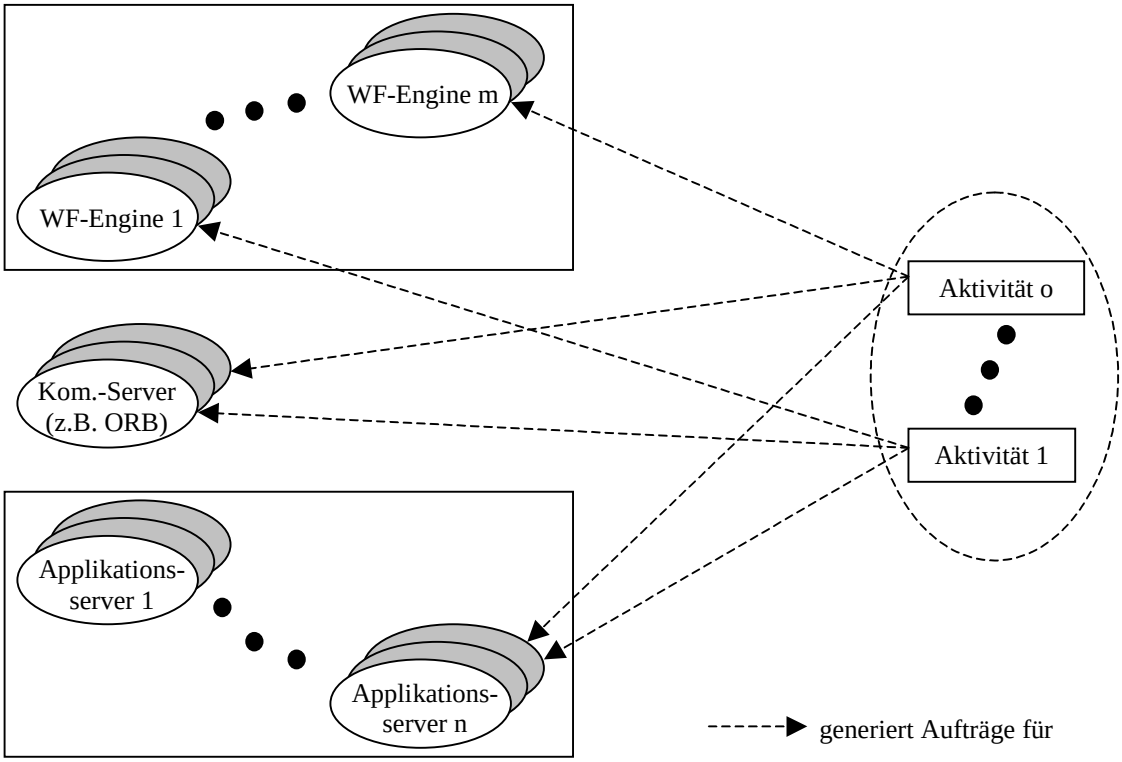


Abbildung 4.4: Server-Modell eines verteilten WFMS

zur Laufzeit Zustandsdaten zwischen den einzelnen Servern ausgetauscht, so daß sie die Arbeit bei einem Ausfall oder bei Wartungsarbeiten gegenseitig übernehmen können. Die Gesamtlast wird dann unter Leistungseinbußen auf die restlichen Server verteilt, bis die Wartung beziehungsweise Reparatur abgeschlossen ist [KAG+96, MAG+95].

Aus diesen Beobachtungen leiten wir das in Abbildung 4.4 dargestellte *Server-Modell* ab:

1. Ein WFMS besteht aus einem Typ von Kommunikations-Servern, m verschiedenen Typen von Workflow-Servern und n verschiedenen Typen von Applikations-Servern. Die Ausführung einer Aktivität erzeugt für die Aktivität spezifische Last auf genau drei verschiedenen Server-Typen, nämlich auf dem Kommunikations-Server-Typ, auf einem Applikations-Server-Typ und auf einem Workflow-Server-Typ. (Im Beispiel von Abbildung 4.4 belastet die Ausführung von *Aktivität 1* einen Kommunikations-Server, einen Workflow-Server vom Typ 1 und einen Applikations-Server vom Typ n.)

2. Jedem Server-Typ x ist eine *Ausfallrate* λ_x und eine *Reperaturrate* μ_x zugewiesen. Darin inbegriffen sind sowohl Ausfälle aufgrund von Wartungsarbeiten als auch unvorhersehbare Hardware- und Softwarefehler ("Heisenbugs" [GR93]).

Definition 4.3. *(Ausfallrate λ_x)*

Der Kehrwert der Zeit, die im Mittel zwischen der Inbetriebnahme und dem nächsten Ausfall eines Servers des Server-Typs x vergeht (englisch: *mean time to failure* (MTTF)), heißt **Ausfallrate** λ_x des Server-Typs x.

Definition 4.4. *(Reparaturrate μ_x)*

Der Kehrwert der Zeit, die im Mittel zur Reperatur oder zur Wartung eines Servers einschließlich eventuell notwendiger Recovery-Maßnahmen benötigt wird (englisch: *mean time to repair* (MTTR)), heißt **Reparaturrate** μ_x des Server-Typs x.

4.3 Systemkonfiguration und Systemzustand

Basierend auf dem in Abschnitt 4.2 vorgestellten Server-Modell definieren wir den zentralen Begriff der *Systemkonfiguration* eines verteilten WFMS wie folgt:

Definition 4.5. *(Systemkonfiguration)*

Die **Systemkonfiguration** eines Workflow-Management-Systems mit $k = m+n+1$ verschiedenen Server-Typen (m verschiedenen Typen von Workflow-Server, n verschiedenen Typen von Applikations-Servern und einem Kommunikations-Server-Typ) ist das Tupel $(Y_0, ..., Y_{k-1})$ aus der Anzahl der Server eines jeden Server-Typs x, $x = 0..k-1$.

Y_x ist also die Anzahl der Server vom Server-Typ x, mit der das WFMS konfiguriert wurde. Aufgrund von Ausfällen und Reparaturen von Servern variiert jedoch die Anzahl der verfügbaren Server über die Zeit. Das heißt, der *Systemzustand* des WFMS ist zeitabhängig.

Definition 4.6. *(Systemzustand)*

Der aktuelle **Systemzustand** eines Workflow-Management-Systems mit der Systemkonfiguration $(Y_0, ..., Y_{k-1})$ ist das Tupel $(X_0, ..., X_{k-1})$ aus der Anzahl der aktuell verfügbaren Server eines jeden Server-Typs x, $x = 0..k-1$. Dabei gilt für alle $x \in \{0, ..., k-1\}$, daß $X_x \leq Y_x$.

4.4 Zusammenfassung und Diskussion

In diesem Kapitel haben wir ein Systemmodell für verteilte WFMS entwickelt, auf dem basierend wir die zentralen Begriffe Systemkonfiguration und Systemzustand definiert haben. Das Systemmodell resultierte aus Beobachtungen, die wir bei der Analyse verschiedener Architekturen von WFMS-Prototypen und -Produkten angestellt haben.

Das vorgestellte Systemmodell ist problemlos erweiterbar. So können weitere Server-Typen (zum Beispiel externe Datenbankserver, separate Server für einen Directory Service [Wei00b] oder das Worklist-Management) integriert werden. Durch die Architekturanalysen von WFMS-Produkten und -Prototypen konnten wir die drei Server-Typen isolieren, die wir in diesem Kapitel angeführt haben. Es sind dies die Workflow-Server, die

den Workflow Enactment Service bilden, die Kommunikations-Server, die die Kommunikation zwischen den einzelnen aktiven Komponenten des WFMS unterstützen, und die Applikations-Server, auf denen die automatischen Aktivitäten der Workflow-Instanzen ausgeführt werden. Sie erscheinen uns als die relevantesten im Hinblick auf die Analyse von Leistung und Verfügbarkeit von WFMS und lassen sich aus allen untersuchten WFMS abstrahieren.

Explizit nicht in das Systemmodell mitaufgenommen wurden die Clients. Wir betrachten Client-Rechner als nicht leistungs-kritisch. Typischerweise sind Client-Rechner PCs oder Arbeitsstationen, die nie von mehr als einem Benutzer gleichzeitig genutzt werden. Somit werden die Laufzeiten der auf ihnen ausgeführten Programme von den Denkzeiten und den Eingaben der Benutzer dominiert. Die Flaschenhälse von verteilten Mehrbenutzersystemen sind daher vielmehr bei den hochgradig belasteten Ressourcen der Server zu suchen. Ausfälle von Client-Rechnern haben keinerlei Auswirkungen auf die Server eines WFMS.

Kapitel 5

Stochastische Modellierung des Kontrollflusses

Um die Last abzuschätzen, die während der Ausführung eines Workflows an den verschiedenen Server-Typen eines WFMS anfällt, müssen wir in der Lage sein, das Kontrollflußverhalten der Workflows vorhersagen zu können. Workflows enthalten jedoch Kontrollflußkonstrukte wie Schleifen und Verzweigungen, durch die das Verhalten des Kontrollflußes erst zur Laufzeit abhängig von instanzspezifischen Bedingungen eindeutig bestimmt wird. Was uns daher bleibt, ist, das "typische" Kontrollflußverhalten der Workflows stochastisch zu beschreiben.

In diesem Kapitel entwickeln wir ein stochastisches Modell für das Kontrollflußverhalten von Workflows. In Abschnitt 5.1 präsentieren wir, wie wir Workflows in sequentielle *Ausführungszustände* gliedern. In Abschnitt 5.2 entwickeln wir ein CTMC-Modell, mit dem wir das Kontrollflußverhalten von Workflows beschreiben. In Abschnitt 5.3 diskutieren wir Möglichkeiten, das Verhalten des Kontrollflusses bei Schleifendurchläufen realistischer abzuschätzen. Schließlich formen wir die so entwickelte CTMC in Abschnitt 5.4 mit Hilfe der Methode der *Normalisierung* in eine äquivalente aber handhabbarere, normalisierte CTMC um.
Das Ziel, auf das wir dabei hinarbeiten, ist, die Wahrscheinlichkeitsverteilung der Anzahl von Aufrufen der verschiedenen Aktivitäten vorauszusagen, die bei der Ausführung von Workflows der verschiedenen Workflow-Typen anfallen. Daraus können wir die Last ableiten, die ein Workflow auf den verschiedenen Server-Typen des WFMS erzeugt.

Zur Beschreibung des Kontrollflußverhaltens eines Workflow-Typs benutzen wir die Methode der zeitkontinuierlichen Markov-Ketten erster Ordnung (CTMC) [Nor97, STP96, Tij94].
In der Literatur wurden CTMC zur Beschreibung von Workflows bereits in [KWA99] benutzt. Dort wird die stationäre Analyse von CTMC dazu verwendet, die Effizienz verschiedener Outsourcing-Strategien bei unternehmensübergreifenden Workflows im Virtual-Enterprise-Kontext zu analysieren. Unser Ansatz ist jedoch weitreichender, da er uns durch die Methode der transienten Analyse von Markov-Ketten Einblicke in das dynamische Verhalten und das Lastaufkommen einzelner Workflow-Instanzen ermöglicht.

5.1 Zusammensetzung von Workflows

Bei der Ausführung eines Workflows werden Aktivitäten abgearbeitet oder Subworkflows gestartet. Ohne Beschränkung der Allgemeinheit nehmen wir an, daß solche Aufrufe in einer wohldefinierten Ordnung ablaufen. Das heißt, daß eine Aktivität oder eine Menge paralleler Subworkflows erst dann gestartet werden, wenn die vorangegangene(n) beendet sind. Parallele oder überlappende Aktivitäten können leicht als einzelne Subworkflows spezifiziert werden. Ebenso können Subworkflows, die mit anderen Subworkflows zum Beispiel durch ein Ereignis synchronisiert werden, in zwei Subworkflows gesplittet werden. Damit ist die Annahme für alle uns bekannten Fälle von Workflow-Spezifikationen erfüllbar.

Wir kapseln die Aktivitäten und Subworkflows zu sequentiellen Einheiten und assoziieren mit jeder einzelnen dieser Einheiten einen (potentiellen) *Ausführungszustand* der Workflows.

Definition 5.1. *(Ausführungszustand)*

> Der aktuelle **Ausführungszustand** eines Workflows ist gegeben durch die gerade in Ausführung befindliche Aktivität oder die nicht-leere Menge gerade in Ausführung befindlicher paralleler Subworkflows.

Im Folgenden bezeichnen wir die Menge aller möglichen Ausführungszustände des Workflow-Typs t mit Z^t.

5.2 Aufbau des Flußprozesses

Bei der Abbildung der Kontrollfluß-Spezifikation auf unser CTMC-Modell machen wir uns die Vorteile der Wohldefiniertheit von Statecharts zu Nutze. Allerdings können wir unseren Ansatz auch leicht auf andere ähnliche Spezifikationssprachen anwenden, nicht zuletzt, da sich viele Workflow-Spezifikationen anderer Sprachen in Statecharts übertragen lassen.

Bei der Verwendung von Statecharts als Spezifikationssprache für Workflows entspricht jeder Zustand des (top-level) Statecharts für einen Workflow einem potentiellen Ausführungszustand des Workflows. Sei $Z^t = \{s_i^t | i = 0..n-1\}$ die Menge der n möglichen Ausführungszustände s_i^t des Workflow-Typs t. Wir modellieren das stochastische Kontrollflußverhalten der Workflows des Workflow-Typs t durch eine CTMC, deren Zustände jeweils einen Ausführungszustand s_i^t repräsentieren. Jede Schrittwahrscheinlichkeit p_{ij}^t der CTMC reflektiert die Wahrscheinlichkeit, daß ein Workflow des Typs t unmittelbar nach Verlassen des Ausführungszustandes s_i^t den Ausführungszustand s_j^t betritt. Jede mittlere Verweilzeit H_i^t der Zustände der CTMC entspricht dem Erwartungswert der Dauer, die ein Workflow des Typs t in dem Ausführungszustand s_i^t verweilt. Dieser Wert entspricht der mittleren Laufzeit der korrespondierenden Aktivität oder der mittleren Durchlaufzeit (siehe Definition 6.1) der geschachtelten Subworkflows.

In Übereinstimmung mit den Design-Regeln für die Spezifikation von Workflows [Wod97] hat auch die CTMC eines Workflow-Typs genau einen Anfangszustand s_0^t. Daher stehen

in dem Vektor der Anfangswahrscheinlichkeiten der CTMC eine 1 für den Zustand s_0^t, der dem initialen Ausführungszustand des Workflows entspricht, und 0 für alle anderen Zustände.

Schließlich fügen wir der so entstandenen CTMC einen künstlichen *absorbierenden Zustand* s_A^t hinzu. s_A^t repräsentiert in dem Modell die Zeit nach der Terminierung des Workflows. Vom letzten Ausführungszustand des Workflows verläuft daher eine Transition in den absorbierenden Zustand mit Schrittwahrscheinlichkeit 1.

Definition 5.2. *(Absorbierender Zustand s_A^t)*

Der Zustand der den Workflow-Typ t modellierenden CTMC, der die Zeit nach der Terminierung eines Workflows repräsentiert, heißt **absorbierender Zustand** s_A^t des Workflow-Typs t.

Mit dieser Menge von Zuständen können wir den stochastischen Prozeß, der das Kontrollfluß-Verhalten eines Workflow-Typs modelliert, bilden:

Definition 5.3. *(Flußprozeß, mittlere Verweildauer, Abgangsrate)*

Der **Flußprozeß** $\{\mathcal{F}^t(\tau)\}$ des Workflow-Typs t ist ein stochastischer Prozess über dem endlichen Zustandsraum $Z^t \cup \{s_A^t\}$, dessen Zustandswechsel gemäß der folgenden Regeln erfolgen:

1. *Regel für die Übergangszeitpunkte*
   Geht der Flußprozeß in einen Zustand $s_i^t \in Z^t$ über, so folgt die Zeit bis zum Übergang in einen Folgezustand einer Zufallsverteilung mit dem Mittelwert H_i^t. Dabei heißen H_i^t die **mittlere Verweildauer** des Prozesses in Zustand s_i^t und

   $$\nu_i^t = \frac{1}{H_i^t}$$

   die **Abgangsrate** des Prozesses von Zustand s_i^t in einen Nachfolgezustand, und es muß für alle $s_i^t \in Z^t$ gelten: $0 < \nu_i^t < \infty$.

2. *Regel für die Übergangsziele*
   Beim Verlassen des Zustandes $s_i^t \in Z^t$ geht der Flußprozeß mit Wahrscheinlichkeit p_{ij}^t in den Nachfolgezustand $s_j^t \in Z^t \cup \{s_A^t\}$ über, wobei gilt:

   $$\begin{aligned} p_{ii}^t &= 0 \\ \sum_{s_j^t \in Z^t \cup \{s_A^t\}} p_{ij}^t &= 1 \end{aligned}$$

   Die Zeit, die der Flußprozeß auf Zustand s_i^t vor dem Übergang verweilt, spielt dabei keine Rolle. Die Zustandsübergänge können somit mittels einer zeitdiskreten Markov-Kette $\{\mathcal{F}_n^t\}$ mit den Schrittwahrscheinlichkeiten $\left(p_{ij}^t\right)$ beschrieben werden.

3. *Regel für die Anfangswahrscheinlichkeiten*
   Für die Anfangswahrscheinlichkeiten des Flußprozesses p_i^t gilt:

$$\begin{aligned} p_0^t &= 1 \\ p_i^t &= 0 \quad , i = 1, ..., n-1, A \end{aligned}$$

Unter der Annahme, daß für alle $s_i^t \in Z^t$ die Verweildauer exponentialverteilt ist, ist der so definierte Flußprozeß $\{\mathcal{F}^t(\tau)\}$ eine zeitkontinuierliche Markov-Kette (CTMC).

Die mittlere Verweildauer H_i^t in einem Ausführungszustand hängt in starkem Maße von der Aktivität oder den Subworkflows ab, die durch den Zustand repräsentiert werden. Bei interaktiven Aktivitäten kommen auch benutzerspezifische Aspekte hinzu, wie zum Beispiel Denkzeiten und die Zeit, die die Aktivität in der Arbeitsliste des Benutzers steht, bevor sie bearbeitet wird. Die Dauer, die ein Ausführungszustand eines Workflow-Typs im Mittel betreten ist, kann durch Beobachtungen bereits beendeter Workflow-Instanzen, zum Beispiel durch Auswerten von Logs, ermittelt werden. Liegen solche Daten noch nicht vor, so muß sie von einem Experten abgeschätzt werden. Ebenso verhält es sich mit den Schrittwahrscheinlichkeiten p_{ij}^t bei einem Split des Kontrollflusses. Das typische Kontrollflußverhalten beim Verlassen eines Ausführungszustandes ist empirisch aus früheren Workflow-Instanzen des Workflow-Typs herleitbar.

Beispiel:

Abbildung 5.1 zeigt als Beispiel die CTMC, die aus unserem Beispiel-Workflow aus Abbildung 3.3 resultiert. Die CTMC hat insgesamt acht Zustände: den absorbierenden Zustand $s_A^{Versandhaus}$ sowie sieben weitere Zustände, die jeweils einen potentiellen Ausführungszustand des Workflow-Typs repräsentieren. Zur Veranschaulichung ist in Abbildung 5.1 hinter jedem Namen eines Zustandes der CTMC der Name des korrespondierenden Ausführungszustandes des Beispiel-Workflows in Klammern aufgeführt.

Hat ein Ausführungszustand in der Spezifikation nur einen potentiellen Nachfolger, so hat der korrespondierende Zustand der CTMC einen ausgehenden Übergang, der mit Wahrscheinlichkeit 1 beschritten wird. Bei einem Split des Kontrollflusses in der Spezifikation gibt es mehrere ausgehende Übergänge aus einem Zustand der CTMC. Die Schrittwahrscheinlichkeiten an den beiden Splits des Beispiel-Workflows sind fiktiv. In der Praxis könnten zum Beispiel die Werte bei dem Split hinter dem Zustand $s_2^{Versandhaus}$ bedeuten, daß 95% der Kreditkartenprüfungen zu einem positiven Ergebnis führen und 5% scheitern. In Abbildung 5.1 haben wir die Zustandsübergänge mit einer Schrittwahrscheinlichkeit von Null der Übersichtlichkeit halber weggelassen.

5.3 Handhabung von Schleifen

Erscheint in der Spezifikation eines Workflow-Typs eine Schleife im Kontrollfluß, so ist die Anzahl der Schleifendurchläufe, die eine bestimmte Workflow-Instanz dieses Workflow-Typs ausführt, eine Zufallsvariable mit einer diskreten Wahrscheinlichkeitsverteilung. In

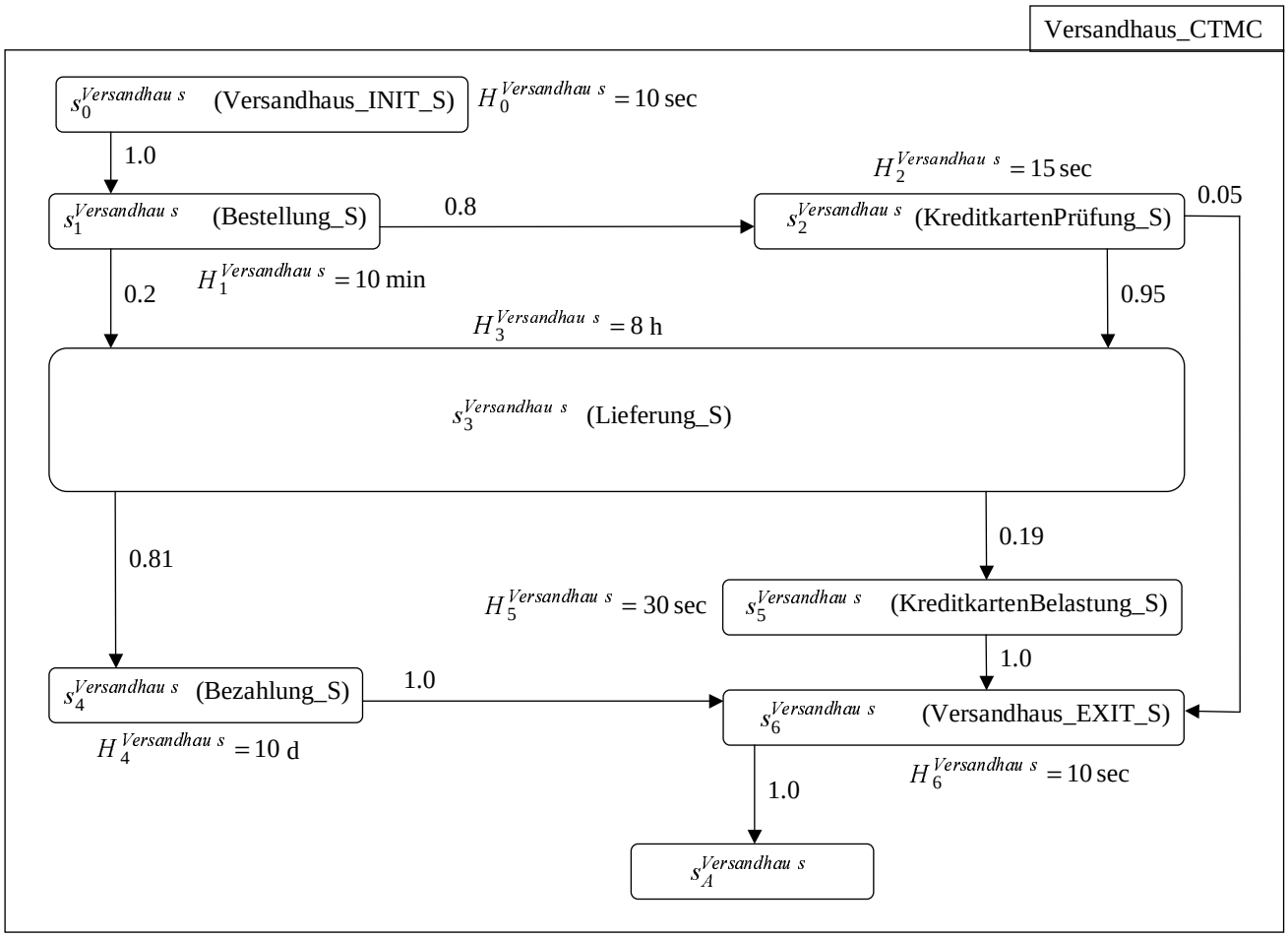


Abbildung 5.1: CTMC des Workflow-Typs *Versandhaus*

diesem Abschnitt beschreiben wir, durch welche Methoden man Schleifen in der CTMC für das Kontrollflußverhalten unter Zuhilfenahme verschiedener statistischer Charakteristiken dieser Wahrscheinlichkeitsverteilung so repräsentieren kann, daß das stochastische Verhalten der resultierenden CTMC die Realität möglichst nah approximiert.

5.3.1 Naive Methode mittels des Erwartungswertes für die Anzahl der Schleifendurchläufe

Die einfachste Art und Weise der Modellierung einer Schleife ist, die Transitionen innerhalb der Schleife genauso zu behandeln, wie wir es bei den anderen Transitionen handhaben: Für jede Transition schätzen wir mit Hilfe von Expertenwissen oder Beobachtungen beendeter Workflow-Instanzen dieses Workflow-Typs die Wahrscheinlichkeit, daß diese Transition beschritten wird, wenn ihr Quellzustand verlassen wird. Diese Methode ist manchmal sogar die bestmögliche, insbesondere dann, wenn es mehrere Ausführungszustände gibt, bei denen die Schleife betreten oder verlassen werden kann.

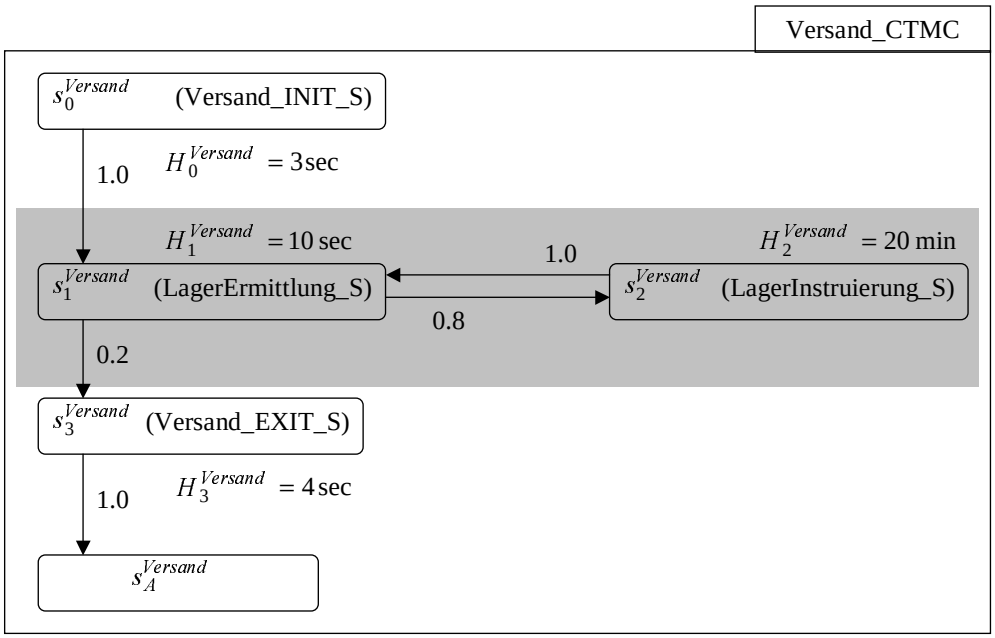


Abbildung 5.2: Naive CTMC des Workflow-Typs *Versand*

Beispiel:

Abbildung 5.2 zeigt die resultierende CTMC unseres Beispiel-Workflow-Typs *Versand* unter Anwendung dieser naiven Methode. Die Anzahl der Zustände der CTMC bleibt gleich zu der Anzahl der potentiellen Ausführungszustände des Workflow-Typs zuzüglich eines absorbierenden Zustandes. Wir nehmen an, daß durchschnittlich vier Artikel pro Bestellung geordert werden. Somit wird die Schleife im Schnitt viermal durchlaufen. Die Transition vom Zustand $s_1^{Versand}$ in den Zustand $s_2^{Versand}$ feuert damit durchschnittlich in vier von fünf Fällen (im fünften Fall wird die Schleife verlassen), also mit einer Wahrscheinlichkeit von 4/5.

Bei der naiven Methode für die Modellierung von Schleifen wird lediglich der Erwartungswert für die Verteilung der Anzahl der Schleifendurchläufe berücksichtigt. Weitere Charakteristiken der Verteilung werden vernachlässigt, wodurch das Kontrollflußverhalten verzerrt wird. Durch die Berücksichtigung von anwendungsspezifischen Nebenbedingungen, die die Verteilung der Anzahl der Schleifendurchläufe prägen (im Beispiel muß immer mindestens ein Artikel bestellt worden sein und so gibt es immer mindestens einen Schleifendurchlauf), kann das tatsächliche Kontrollflußverhalten besser approximiert werden.

5.3.2 Gleichmäßig verteilte Anzahl an Schleifendurchläufen

Beobachtungen der Realität zeigen, daß die meisten Schleifen im Kontrollfluß von Workflows einer Reihe von vereinfachenden Charakteristiken genügen:

- Es gibt genau einen Ausführungszustand, über den die Schleife betreten werden kann.

- Es gibt genau einen Ausführungszustand, nach dem die Schleife verlassen werden kann. Dieser hat beliebig viele Transitionen, die aus der Schleife hinausführen, und genau eine Transition, die in einen Ausführungszustand innerhalb der Schleife führt.
- Es gibt eine positive untere Schranke $m > 0$ für die Anzahl der Schleifendurchläufe (z.B. enthält jede Bestellung mindestens einen Artikel).
- Es gibt eine endliche obere Schranke $n < \infty$ für die Anzahl der Schleifendurchläufe (z.B. kann man maximal das komplette Warensortiment bestellen).
- Die Anzahl der Schleifendurchläufe läßt sich durch eine diskrete Wahrscheinlichkeitsverteilung über die Werte von m bis n beschreiben.

Eine Wahrscheinlichkeitsverteilung, die sehr häufig in der Realität vorkommt oder zumindest sehr viele tatsächlich auftretende Verteilungen der Anzahl von Schleifendurchläufen gut approximiert, ist die diskrete Gleichverteilung (englisch: *discrete uniform distribution*) $UD(m, n)$ über die Werte von m bis n [GT96, Jai91].

Wenn wir annehmen, daß die Anzahl der Schleifendurchläufe eines Workflow-Typs t einer diskreten Gleichverteilung $UD(m^t, n^t)$ über die Werte von m^t bis n^t folgt oder gut durch $UD(m^t, n^t)$ angenähert werden kann, können wir die naive CTMC, die das Kontrollflußverhalten von t beschreibt, im Bereich der Schleife expandieren, um so eine genauere Beschreibung des Kontrollflußverhaltens zu erlangen. Diese Expansion wird folgendermaßen vorgenommen:

Sei $\{s_i^t, ..., s_{i+j}^t\}$ die Menge aller Ausführungszustände, die an der Schleife beteiligt sind. Dabei seien s_{i+j}^t der einzige Zustand, nach dem die Schleife verlassen werden kann, und s_i^t dessen Nachfolger innerhalb der Schleife, also der Ausführungszustand, mit dem ein neuer Schleifendurchlauf beginnt.

1. *(Zustandsexpansion)*
   Wir substituieren jeden Zustand s_x^t der CTMC, der einen der Ausführungszustände aus $\{s_i^t, ..., s_{i+j}^t\}$ repräsentiert, durch n neue Zustände $s_{x,1}^t, ..., s_{x,n^t}^t$. Wir "klonen" praktisch die an der Schleife beteiligten Zustände, damit die maximale Anzahl der Schleifendurchläufe mit genau einem Besuch eines jeweilgen Klons pro Schleifendurchlauf erreichbar wird.

2. *(Betreten der Schleife)*
   Für jeden Zustand s_x^t mit $x \notin \{i, ..., i+j\}$ erzeugen wir eine Transition von s_x^t nach $s_{y,1}^t$ (wobei $y \in \{i, ..., i+j\}$) mit Schrittwahrscheinlichkeit $p_{x,y}^t$, wobei $p_{x,y}^t$ die Schrittwahrscheinlichkeit des Zustandes s_x^t in den Zustand s_y^t in der naiven CTMC ist.

3. *(Verlassen der Schleife oder neuer Durchlauf)*
   Für jede mögliche Anzahl an Schleifendurchläufen $o \in \{m^t, ..., n^t\}$ erzeugen wir

   - eine Transition, die den letzten Zustand der Schleife nach dem o-ten Schleifendurchlauf $s_{i+j,o}^t$ mit Wahrscheinlichkeit

$$p_{\{i+j,o\},x}^t = \frac{1}{(n^t - o + 1)} p_{i+j,x}^t \tag{5.1}$$

in einen der Schleife folgenden Zustand s_x^t mit $x \notin \{i, ..., i+j\}$ verläßt, wobei $p_{i+j,x}^t$ die Schrittwahrscheinlichkeit des Zustandes s_{i+j}^t in den Zustand s_x^t in der naiven CTMC ist, und

- für jeden Zustand $s_{i,o+1}^t$ mit $o < n^t$ eine Transition, die den Zustand $s_{i+j,o}^t$ mit Wahrscheinlichkeit

$$p_{\{i+j,o\},\{i,o+1\}}^t = \left(1 - \frac{1}{(n^t - o + 1)}\right) \left(p_{i+j,i}^t + \sum_{z \notin \{i,...,i+j\}} p_{i+j,z}^t\right) \tag{5.2}$$

in den Zustand $s_{i,o+1}^t$ verläßt, also einen neuen Schleifendurchlauf beginnt. Dabei ist $\sum_{z \notin \{i,...,i+j\}} p_{i+j,z}^t$ die Summe der Schrittwahrscheinlichkeiten von s_{i+j}^t in die Zustände s_z^t außerhalb der Schleife in der naiven CTMC.

In den Gleichungen 5.1 und 5.2 ist $\frac{1}{(n^t - o + 1)}$ die Wahrscheinlickeit, daß die Schleife nach o Durchläufen verlassen wird.
Für alle $o < m^t$ erzeugen wir eine Transition, die den Zustand $s_{i+j,o}^t$ mit Wahrscheinlichkeit 1 in den Zustand $s_{i,o+1}^t$ verläßt, also einen neuen Schleifendurchlauf beginnt.

4. *(Innerhalb der Schleife)*
Für alle $o \in \{1, ..., n^t\}$, alle $x \in \{i, ..., i+j-1\}$ und alle $y \in \{i, ..., i+j\}$ erzeugen wir eine Transition von Zustand $s_{x,o}^t$ nach Zustand $s_{y,o}^t$ mit Schrittwahrscheinlichkeit $p_{x,y}^t$, wenn es in der naiven CTMC eine Transition von s_x^t nach s_y^t mit Schrittwahrscheinlichkeit $p_{x,y}^t$ gibt.

Beispiel:

Abbildung 5.3 zeigt die expandierte CTMC unseres Beispiel-Workflow-Typs *Versand* unter der Annahme, daß die Anzahl der Schleifendurchläufe über die Werte von 3 bis 5 gleichverteilt ist.
Der erste Besuch des Ausführungszustand $s_1^{Versand}$ ist noch nicht Teil der Schleife und muß gesondert behandelt werden. Ein neuer (und implizit auch der erste) Schleifendurchlauf beginnt, wenn der Ausführungszustand $s_1^{Versand}$ verlassen ist.

5.3.3 Konstant verteilte Anzahl an Schleifendurchläufen

Eine weitere Möglichkeit für die Art der Verteilung der Anzahl der Schleifendurchläufe ist die konstante Verteilung. Ist die Anzahl der Schleifendurchläufe für alle Instanzen des Workflow-Typs dieselbe, so bildet dies einen Sonderfall zu den Ausführungen des vorangegangenen Abschnittes.

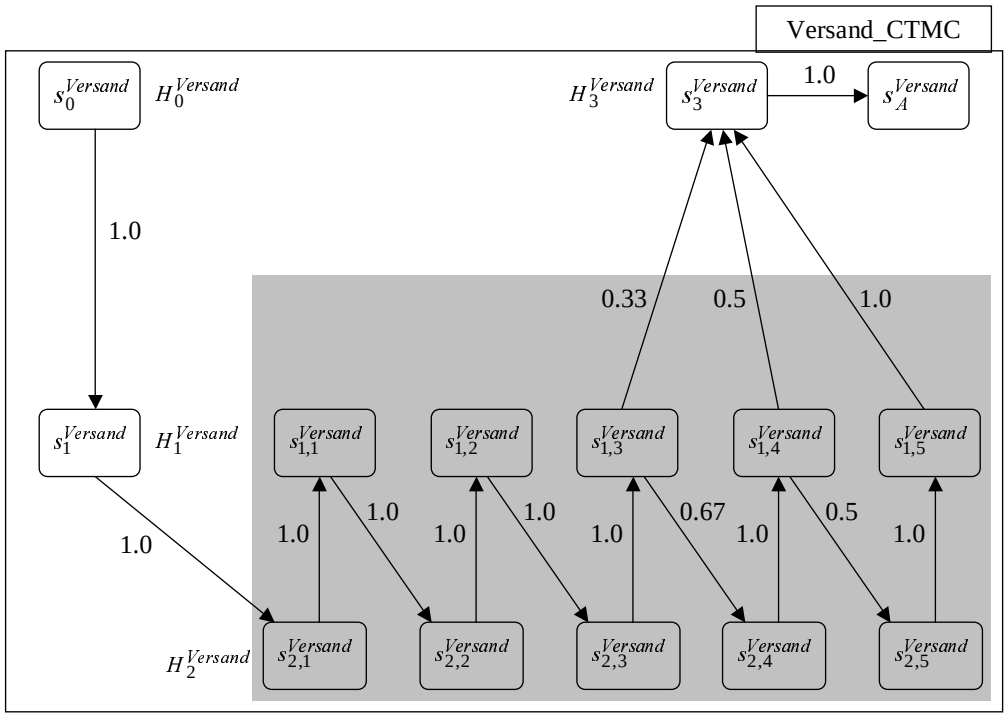


Abbildung 5.3: Expandierte CTMC des Workflow-Typs *Versand* bei Gleichverteilung der Schleifendurchläufe

Beispiel:

Abbildung 5.4 zeigt die resultierende expandierte CTMC für den Workflow-Typ $Versand$ unter der Annahme, daß bei jeder Bestellung immer genau vier verschiedene Artikel bestellt werden. Die CTMC wird so expandiert, daß es genau vier Zustände gibt, die den Ausführungszustand $s_2^{Versand}$ repräsentieren, nämlich $s_{2,1}^{Versand}, ..., s_{2,4}^{Versand}$, und fünf Zustände, die den Ausführungszustand $s_1^{Versand}$ repräsentieren, nämlich $s_{1,1}^{Versand}, ..., s_{1,5}^{Versand}$. Um die genaue Anzahl an Schleifendurchläufen von vier zu garantieren, gibt es lediglich genau eine Transition in den Zustand $s_3^{Versand}$.

5.4 Normalisierung des Flußprozesses

Nach den bisherigen allgemeinen Definitionen sind die Verweilzeiten einer CTMC zustandsspezifisch. Um dadurch hervorgerufene Schwierigkeiten in späteren Analyseschritten zu vermeiden, benutzen wir die Methode der sogenannten *Normalisierung* (englisch: *uniformization*), um den Flußprozeß $\{\mathcal{F}^t(\tau)\}$ mit zustandsspezifischen Verteilungen für die Verweilzeiten in eine CTMC $\{\bar{\mathcal{F}}^t(\tau)\}$ mit identischen Verteilungen für die Zustandsverweilzeiten zu überführen, die dasselbe zeitliche Verhalten wie der Flußprozeß aufweist [Tij94].

Zunächst bilden wir den Flußprozeß $\{\mathcal{F}^t(\tau)\}$ jedes Workflow-Typs t mit Zustandsmenge $Z^t \cup \{s_A^t\}$, Schrittwahrscheinlichkeiten p_{ij}^t und zustandsspezifischen Abgangsraten ν_i^t auf eine zeitdiskrete Markov-Kette $\{\bar{\mathcal{F}}_n^t\}$ mit identischer Zustandsmenge und den modifizier-

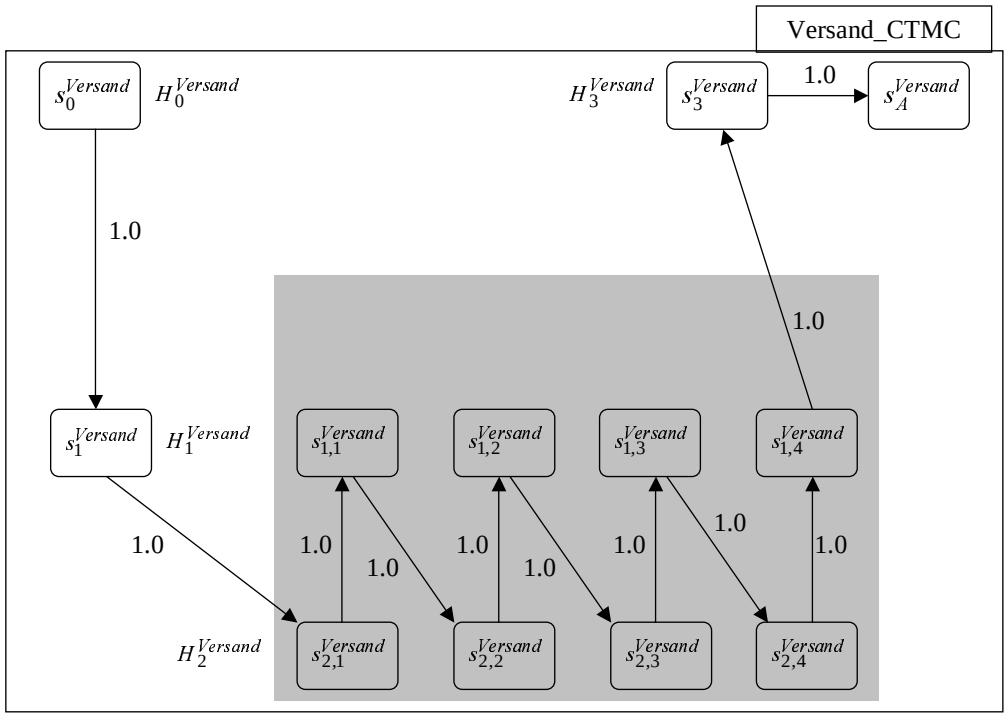


Abbildung 5.4: Expandierte CTMC des Workflow-Typs *Versand* bei konstanter Verteilung der Schleifendurchläufe

ten Schrittwahrscheinlichkeiten

$$\bar{p}_{ij}^t = \begin{cases} \frac{\nu_i^t}{\nu^t} p_{ij}^t & , j \neq i \quad , i \neq A \\ 1 - \frac{\nu_i^t}{\nu^t} & , j = i \quad , i \neq A \end{cases} \tag{5.3}$$

und

$$\bar{p}_{Aj}^t = \begin{cases} 0 & , j \neq A \\ 1 & , j = A \end{cases} \tag{5.4}$$

ab, wobei $\nu^t = \max\{\nu_i^t | s_i^t \in Z^t\}$ die maximale Abgangsrate des Flußprozesses ist. $\left\{\bar{\mathcal{F}}_n^t\right\}$ beschreibt die Schrittwahrscheinlichkeiten einer CTMC $\left\{\bar{\mathcal{F}}^t(\tau) | \tau \geq 0\right\}$.
Die Zeitpunkte für die Ausführung der Schritte von $\left\{\bar{\mathcal{F}}^t(\tau)\right\}$ beschreiben wir durch den Poisson-Prozeß $\{\mathcal{N}(\tau) | \tau \geq 0\}$ mit Rate ν^t, so daß gilt:

$$\bar{\mathcal{F}}^t(\tau) = \bar{\mathcal{F}}_{\mathcal{N}(\tau)}^t \quad , \tau \geq 0. \tag{5.5}$$

Daraus ergibt sich, daß die Verweilzeiten der neuen, normalisierten CTMC $\left\{\bar{\mathcal{F}}^t(\tau)\right\}$ alle identisch, nämlich dem Poisson-Prozeß mit Mittelwert $1/\nu^t$ folgend, verteilt sind. Die zustandsspezifischen Verweilzeiten des Flußprozesses schlagen sich in den zustandsspezifischen "Rückführungen" $\bar{p}_{ii}^t$ nieder. Man kann sich das so vorstellen, daß die normalisierte CTMC immer dann einen Rückführungsschritt von s_i^t in s_i^t ausführt, wenn der Flußprozeß eine weitere Zeiteinheit im Zustand s_i^t verweilen würde.

5.5 Zusammenfassung und Diskussion

In diesem Kapitel haben wir ein Modell vorgestellt, mit dem wir das Kontrollflußverhalten von Workflows stochastisch beschreiben. Das Modell bedient sich der Methode der

zeitkontinuierlichen Markov-Ketten erster Ordnung (CTMC) und basiert auf der Zerlegung von Workflows in sequentielle Ausführungszustände. Als Beispiel für die Modellierung haben wir die CTMC für den Workflow-Typ *Versandhaus* unseres durchgehenden Beispiel-Workflows präsentiert.

Die Schrittwahrscheinlichkeiten zwischen den Zuständen des Flußprozesses müssen initial vom Workflow-Designer angegeben werden. Dabei muß er mit seinem Expertenwissen die Häufigkeit unterschiedlicher Belegungen von Bedingungen und unterschiedlicher Ablaufzweige abschätzen. Wenn das WFMS bereits eine Zeit in Betrieb ist und wir den Nutzen einer Rekonfiguration vorhersagen möchten, können die Schrittwahrscheinlichkeiten auch durch statistisches Auswerten gesammelter Daten über frühere Workflowausführungen (zum Beispiel Log- und Historiendaten) ermittelt werden. Ebenso können auch die Laufzeiten von Aktivitäten bestimmt werden, die die mittleren Verweilzeiten in den Zuständen der CTMC wiedergeben.

Subworkflows repräsentieren wir durch einen Zustand in der CTMC des Elterworkflows. Im Falle von Parallelität repräsentiert der korrespondierende Zustand alle parallelen Subworkflows. Als mittlere Verweildauer für diesen Zustand nehmen wir das Maximum der mittleren Durchlaufzeiten aller in dem Zustand vereinten, parallelen Subworkflows. Die Details dieser Vorgehensweise beschreiben wir in Abschnitt 6.4.

Sind die Dauer einer Aktivität oder eines Subworkflows und damit auch die Verweilzeit im korrespondierenden Zustand der CTMC nicht wie im Modell angenommen exponentialverteilt, so kann man die allgemeine Verteilung der Zustandsverweilzeit wie in Abschnitt 3.3.2 beschrieben durch Substitution des Zustandes durch ein System von Zuständen mit exponentialverteilten Zustandsverweilzeiten approximieren. Ein solches System von Zuständen "simuliert" dann eine der allgemeinen Verteilung "hinreichend ähnliche" verallgemeinerte Erlang-Verteilung für die Verweilzeit in dem substituierten Zustand [Tij94].

Mit diesem CTMC-Modell für das erwartete Kontrollflußverhalten von Workflows ist es uns möglich, die erwartete Anzahl von Aufrufen der verschiedenen Aktivitäten vorherzusagen. Diese ist nämlich identisch zu der erwarteten Anzahl von Besuchen der CTMC in dem zu einer Aktivität korrespondierendem Zustand vor dem Betreten des absorbierenden Zustandes. Dafür verwenden wir Standard-Analysetechniken für Markov-Ketten, deren Anwendung wir im folgenden Kapitel detailliert beschreiben. Schließlich werden wir aus den Ergebnissen dieser Analysen die erwartete Gesamtlast eines Workflows ableiten.

Kapitel 6

Leistungsmodell

In diesem Kapitel entwickeln wir ein Modell, mit dessen Hilfe wir die Leistung eines WFMS abschätzen. Das Modell bedient sich der in Kapitel 5 vorgestellten CTMC, mit der wir das Verhalten des Kontrollflusses modellieren. In Abschnit 6.1 diskutieren wir die Metrik, mit der wir die Leistung eines WFMS beschreiben.

Bei der Entwicklung des Modelles gehen wir in vier Schritten vor.

1. In Abschnitt 6.2 analysieren wir die mittlere *Durchlaufzeit* von Workflows durch die transiente Analyse des Flußprozesses.

2. In Abschnitt 6.3 entwickeln wir ein Modell für die an den Servern anfallende Last. Wir ermitteln die mittlere Anzahl von Service-Aufträgen, die ein Workflow an den verschiedenen Server-Typen initiiert, und leiten daraus die *Ankunftsrate* an dem Server-Typ ab. Dabei erweitern wir das CTMC-Modell zu einem MRM. Danach aggregieren wir für jeden Server-Typ die erwartete Last über alle Workflows aller Workflow-Typen. Dabei gewichten wir die Last durch die einzelnen Workflow-Typen mit der relativen Häufigkeit des Auftretens von Workflows eines Typs. Die Last, die auf einen einzelnen Server fällt, erreichen wir durch gleichmäßige Verteilung der anfallenden Service-Aufträge auf alle Server eines Server-Typs. Daraus leiten wir Aussagen über den maximal erreichbaren *Durchsatz* sowohl an einzelnen Server-Typen als auch von Workflow-Instanzen im WFMS ab.

3. In Abschnitt 6.3 betrachten wir nur Workflow-Typen ohne Subworkflows. Die Erweiterung des Modelles für Subworkflows beschreiben wir in Abschnitt 6.4.

4. Abschließend schätzen wir in Abschnitt 6.5 die mittleren Wartezeiten von Service-Aufträgen an den verschiedenen Server-Typen durch Modellierung einzelner Server als M/G/1-Bedienstationen ab. Die mittlere Wartezeit von Service-Aufträgen wirkt sich unmittelbar auf das Antwortzeitverhalten des WFMS aus, wie es sich dem Benutzer während der Interaktion mit dem WFMS darstellt. Unakzeptabel hohe Wartezeiten sind ein Indiz für eine schlechte Konfiguration des WFMS. Das beschriebene Leistungsmodell ermöglicht uns, die Server-Typen zu ermitteln, die den Flaschenhals des Systems bilden.

6.1 Metrik

Aus betriebswirtschaftlicher Sicht steht beim Einsatz von WFMS die Steigerung der Effizienz von Geschäftsprozessen im Vordergrund. Dies umfaßt jedoch nicht nur monetäre oder zeitliche Größen. Wichtig sind auch Aspekte wie Kundenzufriedenheit, Motivation und Engagement der Mitarbeiter, gleichmäßige Mitarbeiterauslastung ohne wesentliche Überlastung usw.. Eine quantitative Erfassung dieser Aspekte ist jedoch schwierig. Dagegen zielen betriebswirtschaftliche Kennzahlsysteme auf größere Transparenz rein betrieblicher Aspekte ab [Aic95, Sch96]. Die Verwendung von WFMS erlaubt die direkte Erfassung entsprechender Kenngrößen, die bisher nicht explizit verfügbar waren. Beispielsweise können die Mitarbeiterauslastung und die mittlere Bearbeitungszeit ganzer Geschäftsprozesse mit geringem Aufwand erfaßt und statistisch aufbereitet werden.

Um den Einfluß der Leistung verschiedener WFMS-Konfigurationen auf die Effizienz von Geschäftsprozessen systematisch und reproduzierbar zu bestimmen, müssen Leistungsmaße für WFMS definiert werden. Einerseits sollen diese Leistungsmaße sehr eng an den obigen Qualitätskriterien für Geschäftsprozesse angelehnt sein, um leicht interpretierbare Werte zu erhalten, andererseits sollen für unseren Zweck WFMS-externe Parameter wie zum Beispiel die Auswahl und Leistungsfähigkeit der Sachbearbeiter keine Rolle spielen. Ferner müssen die Leistungsmaße systemunabhängig festgelegt sein, damit sie für jedes untersuchte WFMS definiert sind und dieselbe Bedeutung haben. Da ein einzelnes Leistungsmaß nicht alle Anforderungen gleichzeitig erfüllen kann, schlagen wir eine Hierarchie von Leistungsmaßen vor, die von globalen, "makroskopischen" Metriken bis zu Detailgrößen reicht:

1. Durchsatz und Durchlaufzeit ganzer Workflows
2. Antwortzeit einzelner Workflow-Schritte
3. Durchsatz und Wartezeit einzelner Service-Aufträge

Für alle genannten Metriken müssen sowohl die Interferenz der verschiedenen Workflow-Typen als auch das Zusammenspiel der verscheiedenen Server-Typen berücksichtigt werden.

6.2 Durchlaufzeit von Workflows

Wir schätzen die mittlere Durchlaufzeit R_t von Workflows des Typs t mit Hilfe der transienten Analyse des CTMC-Modelles von Kapitel 5 ab.

Definition 6.1. *(Durchlaufzeit)*

> Die Zeit, die die Ausführung einer Instanz eines Workflow-Typs t von ihrer Initiierung bis zur Terminierung im Mittel dauert, heißt **mittlere Durchlaufzeit** R_t des Workflow-Typs t.

Es gilt, daß die mittlere Durchlaufzeit eines Workflows gleich der Zeit ist, die seine korrespondierende CTMC im Mittel benötigt, um den absorbierenden Zustand s_A^t zum ersten

(und einzigen) Mal zu betreten. Somit ist die mittlere Durchlaufzeit R_t eines Workflows vom Typ t gleich der "Mean First Passage Time" m^t_{0A} für s^t_A ausgehend vom Anfangszustand s^t_0:

$$R_t = m^t_{0A}. \tag{6.1}$$

Wir erhalten m^t_{0A} durch Lösen des linearen Gleichungssystems [Tij94]

$$-\nu^t_i m^t_{iA} + \sum_{j \neq A, j \neq i} q^t_{ij} m^t_{jA} = -1 \tag{6.2}$$

nach m^t_{0A}, wobei $s^t_i, s^t_j \in Z^t$ die Zustände der CTMC und

$$q^t_{ij} = \nu^t_i p^t_{ij} \tag{6.3}$$

die Übergangsrate von Zustand s^t_i nach Zustand s^t_j sind.

Die Herleitung des Gleichungssystems 6.2 basiert darauf, daß für jeden Zustand der CTMC die Wahrscheinlichkeitsverteilung der Zeit bis zum ersten Betreten des absorbierenden Zustandes ausgehend von dem Zustand (englisch: *first-pessage time probability*) die Kolmogorov'sche Rückwärtsdifferentialgleichung erfüllt [Tij94].

6.3 Lastmodell

Wir bauen unser Lastmodell hierarchisch auf. Zunächst untersuchen wir, welche Last einzelne Aktivitäten auf den Server-Typen erzeugen. Danach fassen wir mehrere Aktivitäten zusammen und ermitteln mit Hilfe des CTMC-Modelles aus Kapitel 5 für jeden Workflow-Typ die mittlere Last, die die Instanzen des Typs erzeugen. Abschließend zeigen wir, wie sich daraus für jeden Server-Typ die mittlere Ankunftsrate von Service-Aufträgen aus allen Workflow-Typen ableiten läßt.

6.3.1 Aktivitäten-spezifische Last

Die Ausführung eines Workflows umfaßt eine Menge von Aktivitäten, die ihrerseits Service-Aufträge für verschiedene Server-Typen generieren.
Wir beschreiben die Gesamtlast, die jede Ausführung einer Aktivität auf den Server-Typen des WFMS erzeugt, durch einen *Lastvektor*, der für jeden Server-Typ die Anzahl der Service-Aufträge enthällt, die die einmalige Ausführung der Aktivität im Durchschnitt auf dem Server-Typ erzeugt.

Sei k die Anzahl der im WFMS vorhandenen Server-Typen. Dann gibt der Lastvektor

$$L_a = \begin{pmatrix} c_0 \\ c_1 \\ . \\ . \\ . \\ c_{k-1} \end{pmatrix} \tag{6.4}$$

für jeden der k Server-Typen die Anzahl c_x der Service-Aufträge an, die die Ausführung der Aktivität a auf dem Server-Typ x, $x = 0, ..., k-1$, im Mittel initiiert.

6.3.2 Last pro Workflow

Zur Berechnung der Gesamtlast eines Workflows, also der Gesamtzahl der Service-Aufträge, die bei der Ausführung eines Workflows auf den verschiedenen Servern eines WFMS anfallen, entwickeln wir ein MRM mit dem CTMC-Modell aus Kapitel 5 als Basis. Dabei ignorieren wir zunächst die mögliche Existenz von Subworkflows.

Die dem MRM zugrunde liegende Markov-Kette bildet die normalisierte CTMC, die das stochastische Verhalten des Workflow-Typs beschreibt. Als Pauschal-Reward eines Zustandes der CTMC benutzen wir den Lastvektor der Aktivität, die ausgeführt wird, wenn sich die CTMC in dem Zustand befindet.
Die mittlere Anzahl von Service-Aufträgen, die ein Workflow während seiner Ausführung auf den verschiedenen Servern initiiert, entspricht dem akumulierten Reward bis zum Erreichen des absorbierenden Zustandes der CTMC (englisch: *expected reward until absorbtion* [Tij94]).

Wir fassen die Lastvektoren aller an einem Workflow-Typ beteiligten Aktivitäten in einer *Lastmatrix* zusammen. Dabei entspricht die i-te Spalte der Lastmatrix dem Lastvektor der Aktivität, die durch den i-ten Zustand der CTMC repräsentiert wird.

Seien die Anzahl n der an Workflows des Workflow-Typs t beteiligten Aktivitäten sowie für jede der n Aktivitäten deren Lastvektor L_i^t, $i = 0..n-1$, und die t beschreibende CTMC mit den Zuständen $Z^t = \{s_i^t | i \in \{0, ..., n-1, A\}\}$ gegeben. Dann gibt die Lastmatrix

$$\mathbb{L}^t = \left(L_{xi}^t\right) = \left(\begin{array}{ccc} L_0^t & \ldots & L_{n-1}^t \end{array} \right) = \left(\begin{array}{ccc} c_{00} & \cdots & c_{0(n-1)} \\ . & & . \\ . & & . \\ . & & . \\ c_{(k-1)0} & \cdots & c_{(k-1)(n-1)} \end{array} \right) \tag{6.5}$$

für jede der n Aktivitäten von t die Anzahl der Service-Aufträge an, die die Ausführung der Aktivität auf jedem der k Server-Typen des WFMS im Durchschnitt initiiert. Für den absorbierenden Zustand A gibt es in der Lastmatrix keine Spalte, da er keine Aktivität repräsentiert und somit keine Last auf den Servern erzeugt.

Beispiel:

Abbildung 6.1 zeigt die CTMC, die zu dem (Sub-)Workflow-Typ *Bestätigung* aus unserem durchgehenden E-Commerce-Beispiel korrespondiert. Zusätzlich sind beispielhaft die Lastvektoren eingefügt, die die Last der zu den Zuständen $s_0^{Bestätigung}$ bis $s_2^{Bestätigung}$ korrespondierenden Aktivitäten beschreiben. Wir nehmen an, daß das WFMS in diesem Beispiel drei verschiedene Server-Typen hat, so daß die Lastvektoren drei Einträge haben.

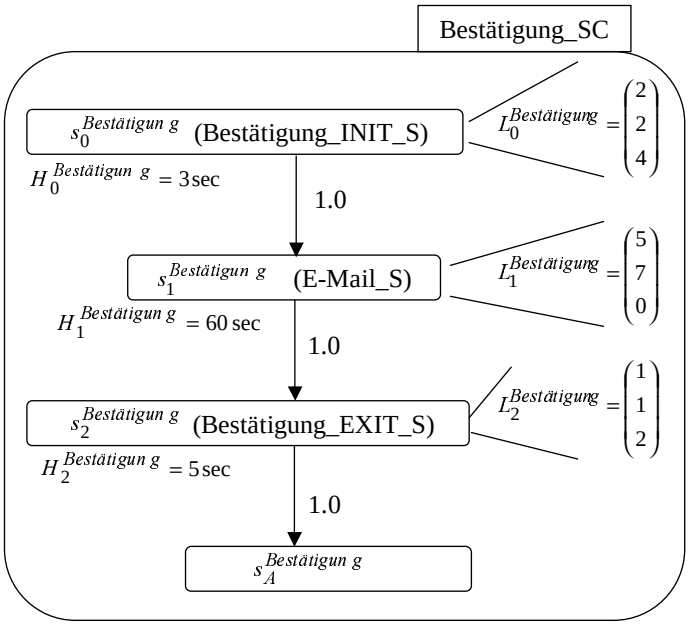


Abbildung 6.1: CTMC des Workflow-Typs *Bestätigung* mit Lastvektoren

Die CTMC aus Abbildung 6.1 hat insgesamt 4 Zustände. Allerdings repäsentiert der absorbierende Zustand keine Aktivität und erzeugt daher keine Last. Die sich ergebende Lastmatrix hat drei Spalten und hat im Beispiel folgendes Aussehen:

$$\mathbb{L}^{Versand} = \begin{pmatrix} 2 & 5 & 1 \\ 2 & 7 & 1 \\ 4 & 0 & 2 \end{pmatrix}$$

Seien für einen Workflow-Typ t

$$\nu^t = \max_{i \neq A} \left\{ \nu_i^t = \frac{1}{H_i^t} \right\} \tag{6.6}$$

die maximale Abgangsrate innerhalb der t beschreibenden CTMC und $\vec{p}_{0i}^{\,t}(z)$ die Tabu-Wahrscheinlichkeit, daß die CTMC nach z Schritten ausgehend vom Zustand s_0^t im Zustand s_i^t ist, ohne zuvor den absorbierenden Zustand s_A^t besucht zu haben [Tij94].
Dann ergibt sich die erwartete Anzahl r_x^t von Service-Aufträgen, die ein Workflow des Workflow-Typs t an einem Server des Server-Typs x während seiner Ausführung initiiert, aus der Formel [Tij94]

$$r_x^t = L_{x0}^t + \frac{1}{\nu^t} \left(\sum_{i \neq A} \sum_{z=0}^{\infty} \vec{p}_{0i}^{\,t}(z) \sum_{j \neq i, j \neq A} q_{ij}^t L_{xj}^t \right), \tag{6.7}$$

wobei q_{ij}^t entsprechend Gleichung 6.3 die Übergangsrate von Zustand s_i^t nach Zustand s_j^t ist.
Die Tabu-Wahrscheinlichkeiten werden mit den Startwerten $\vec{p}_{ii}^{\,t}(0) = 1$ und $\vec{p}_{ij}^{\,t}(0) = 0$ für $i \neq j$ rekursiv berechnet. Für $i \neq A$ und $z \geq 1$ ist

$$\vec{p}_{ij}^{\,t}(z) = \sum_{k \neq A} \vec{p}_{kj}^{\,t} \vec{p}_{ik}^{\,t}(z-1) \tag{6.8}$$

Dabei sind $\vec{p}_{ij}^{\,t}$ die Schrittwahrscheinlichkeiten der zum Flußprozeß korrespondierenden zeitdiskreten Markov-Kette aus den Gleichungen 5.3 und 5.4.

6.3.3 Gesamtlast an Servern und erreichbarer Durchsatz

Wir nehmen an, daß die Instanzen eines Workflow-Typs t mit der mittleren Ankunftsrate α_t von Benutzern gestartet werden. Typischerweise läßt sich ein solcher Ankunftsprozeß unter der Annahme einer hohen Anzahl von initiierenden Clients durch einen Poisson-Prozeß [Nel95] beschreiben.

Die Ankunftsrate l_x^t von Service-Aufträgen einer einzelnen Instanz vom Workflowtyp t am Server-Typ x ergibt sich aus dem Produkt von α_t und der erwarteten Anzahl von Service-Aufträgen r_x^t während der Ausführung einer Workflow-Instanz des Workflow-Typs t:

$$l_x^t = \alpha_t r_x^t \tag{6.9}$$

Die Ankunftsrate l_x von Service-Aufträgen aller Workflow-Typen am Server-Typ x ergibt sich aus der Summe dieses Produktes über alle Workflow-Typen:

$$l_x = \sum_t l_x^t = \sum_t \alpha_t r_x^t \tag{6.10}$$

Unter den Annahmen, daß dem WFMS von jedem Server-Typ x X_x Server zur Verfügung stehen, und daß die an einem Server-Typ auftretende Last gleichmäßig über alle Server des Typs verteilt wird, ergibt sich die erwartete Ankunftsrate von Service-Aufträgen $\tilde{l}_x$ an einem Server des Server-Typs x aus

$$\tilde{l}_x = \frac{l_x}{X_x}. \tag{6.11}$$

Wenn wir darüber hinaus annehmen, daß jeder Service-Auftrag den Server durchschnittlich für eine mittlere Bedienzeit der Dauer b_x beschäftigt, ist der *Durchsatz* $\tilde{d}_x$ an den Servern des Server-Typs x gleich dem maximalen Wert, für den gilt

$$\tilde{d}_x b_x < 1 \tag{6.12}$$

und

$$\tilde{d}_x \leq \tilde{l}_x. \tag{6.13}$$

Bei Ankunftsraten, bei denen $\tilde{l}_x b_x \geq 1$ gilt, können die Server das Lastaufkommen nicht bewältigen, das heißt, sie sind überlastet. Der maximal *erreichbare Durchsatz* $\tilde{d}_x^{max}$ von Service-Aufträgen an einem Server des Server-Typs x ist somit der maximale Wert, für den gilt

$$\tilde{d}_x^{max} b_x < 1, \tag{6.14}$$

und der maximal erreichbare Durchsatz von Service-Aufträgen d_x^{max} an dem Server-Typ x ist

$$d_x^{max} = X_x \tilde{d}_x^{max}. \tag{6.15}$$

Der maximal erreichbare Durchsatz von Workflow-Intanzen im WFMS ist durch den maximal erreichbaren Durchsatz von Service-Aufträgen beschränkt, wobei die Summe der Ankunftsraten von Service-Aufträgen für den Server-Typ x über alle Workflow-Instanzen kleiner oder gleich d_x^{max} sein muß. So gilt für den maximal erreichbaren Durchsatz D_t^{max} von Instanzen des Workflow-Typs t im WFMS, daß für alle x

$$\sum_t D_t^{max} r_x^t \leq d_x^{max} \tag{6.16}$$

gelten muß.

6.4 Berücksichtigung von Subworkflows

Für Workflow-Typen mit Subworkflows können die erwartete Durchlaufzeit und die erwartete Gesamtlast von Workflows bottom-up berechnet werden. Ein Subworkflow sowie eine Menge paralleler Subworkflows werden in der CTMC des Vaterworkflows durch einen einzigen Zustand repräsentiert. Dabei ist die mittlere Verweildauer in diesem Zustand abhängig von den mittleren Durchlaufzeiten aller darin assoziierten Subworkflows, und die Anzahl der zu erwartenden Service-Aufträge in dem Zustand ist gleich der Summe der zu erwartenden Service-Aufträge der Subworkflows.

Seien S die Menge der parallelen Subworkflows, die durch den Zustand s_i^t der das Verhalten des Vater-Workflow-Typs t beschreibenden CTMC repräsentiert sind, R_s die mittlere Durchlaufzeit und r_x^s das mittlere Aufkommen an Service-Aufträgen an den Server-Typ x des Subworkflow-Typs $s \in S$. Bei den Berechnungen der mittleren Durchlaufzeit (Gleichungssystem (6.2)) und der erwarteten Anzahl von Service-Aufträgen von Instanzen des Workflow-Typs t (Gleichung (6.7)) schätzen wir die mittlere Verweildauer H_i^t des Zustandes s_i^t der t beschreibenden CTMC durch das Maximum der mittleren Durchlaufzeiten der Subworkflows in S

$$H_i^t = \max_{s \in S} \{R_s\}, \tag{6.17}$$

ab und setzen für alle Server-Typen x die Anzahl L_{xi}^t der Service-Aufträge an x auf die Summe der zu erwartenden Service-Aufträge aller Subworkflows in S an x, also auf

$$L_{xi}^t = \sum_{s \in S} r_x^s. \tag{6.18}$$

Beispiel:

Abbildung 6.2 zeigt, wie sich dieses Vorgehen bei unserem durchgehenden Beispiel-Workflow auswirkt. Die Abbildung zeigt einen Ausschnitt aus der CTMC von Abbildung 5.1. Der abgebildete Zustand der CTMC repräsentiert den Ausführungszustand des Workflows, der betreten ist, wenn die beiden Subworkflows *Bestätigung* und *Versand* parallel ausgeführt werden. Zu beiden Subworkflows gibt es je eine korrespondierende CTMC, mit deren Hilfe entsprechend der Ausführungen in den Abschnitten 6.2 und 6.3.2 die mittleren Durchlaufzeiten $R_{Bestätigung}$ und $R_{Versand}$ sowie für alle Server-Typen x des WFMS die jeweilige erwartete Anzahl an Service-Aufträgen $r_x^{Bestätigung}$ und $r_x^{Versand}$ ermittelt werden können. Diese werden bei den Berechnungen für den Workflow-Typ *Versandhaus* wie beschrieben eingesetzt.

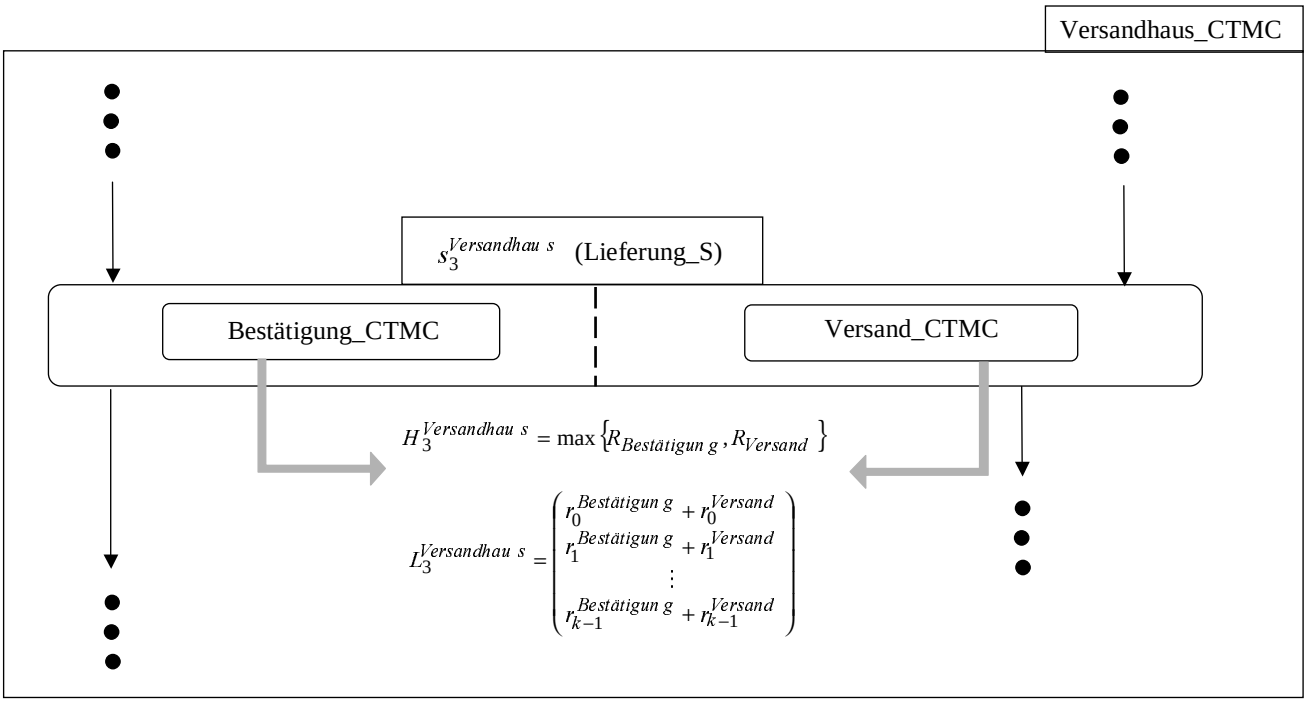


Abbildung 6.2: Bottom-up Berechnung bei Subworkflows

6.5 Abschätzung der Wartezeit von Service-Aufträgen

Wir nehmen an, daß alle Service-Aufträge an einen Server-Typ gleichmäßig auf alle Server des Typs verteilt werden. Dies könnte zum Beispiel durch eine round-robin- oder hash-basierte Verteilung der Service-Aufträge geschehen. In der Praxis geschieht die Zuweisung von Servern bereits beim Start des Workflows, so daß alle zu dem Workflow gehörenden Aktivitäten und Anwendungsprogramme auf denselben Instanzen der jeweiligen Server-Typen arbeiten. Zwar kann diese realistische Art der Lastverteilung zeitweise unbalanciert sein, das heißt, daß einzelne Server deutlich mehr oder weniger Last abbekommen als der Durchschnitt, aber über einen langen Zeitraum betrachtet ist die zu verrichtende Arbeit aller Server eines Typs annähernd gleich.

6.5.1 Modellierung der Server als M/G/1-Bedienstationen

Für die Analyse der mittleren Wartezeit von Service-Aufträgen modellieren wir jeden Server-Typ x als eine Menge von X_x M/G/1-Bedienstationen, wobei X_x die Anzahl der Server des Typs x ist, die dem WFMS zur Verfügung stehen. Durch diese Art der Modellierung benötigen wir als weitere Informationen über die Server lediglich den Erwartungswert und das zweite Moment der Verteilung der Bedienzeit von Service-Aufträgen. Dabei betrachten wir die Verteilung der Bedienzeit von Service-Aufträgen als Server-Typ-spezifisch. Beide Werte können durch statistische Methoden ermittelt werden, zum Beispiel durch Sammeln und Auswerten von Log- oder Historiendaten.

6.5.2 Mittlere Wartezeit von Service-Aufträgen

Seien b_x die mittlere Bedienzeit und $b_x^{(2)}$ das zweite Moment der Bedienzeitverteilung von Service-Aufträgen an den Servern des Server-Typs x. Bei der mittleren Ankunftsrate $\tilde{l}_x$

von Service-Aufträgen (Gleichung (6.11)) ist die Auslastung ρ_x eines Servers des Typs x gegeben durch

$$\rho_x = \tilde{l}_x b_x. \tag{6.19}$$

Für die mittlere Wartezeit w_x von Service-Aufträgen an den Servern des Typs x ergibt sich daraus [All90, Hav98, Tak91]

$$w_x = \frac{\tilde{l}_x b_x^{(2)}}{2\,(1-\rho_x)}. \tag{6.20}$$

Diese mittlere Wartezeit ist die wesentliche Metrik für die Beurteilung des WFMS, da sie alle anderen Metriken direkt beeinflußt und für einen Benutzer bei interaktivem Arbeiten mit dem WFMS unmittelbar spürbar wird. Die Antwortzeit eines Service-Auftrages wird im wesentlichen von der Summe aus der lastunabhängigen Bedienzeit und der lastabhängigen Wartezeit bestimmt.

6.6 Zusammenfassung und Diskussion

In diesem Kapitel haben wir unser Leistungsmodell beschrieben. Als wesentliche Metrik zur Beurteilung eines WFMS benutzen wir die mittlere Wartezeit von Service-Aufträgen, die während der Ausführung von Workflows anfallen. Diese Wartezeiten sind für Benutzer, die aktiv an der Bearbeitung interaktiver Aktivitäten teilnehmen, unmittelbar spürbar. Um die erwartete mittlere Wartezeit von Service-Aufträgen bei gegebener Systemkonfiguration und Systemlast abschätzen zu können, bedienen wir uns der transienten Analyse des im Kapitel 5 eingeführten CTMC-Modelles zur Beschreibung des Kontrollflußverhaltens der Workflow-Instanzen. Damit konnten wir die mittlere Durchlaufzeit sowie die erwartete Ankunftsrate von Service-Aufträgen an den unterschiedlichen Server-Typen des WFMS abschätzen. Diese dienten uns als Parameter bei der Berechnung der mittleren Wartezeiten der Service-Aufträge, zu der wir die einzelnen Server als M/G/1-Bedienstationen modellierten.

Die Systemlast modellieren wir mittels der erwarteten Anzahl der Service-Aufträge, die während der Ausführung einer Aktivität anfallen. Diese Vereinfachung läßt potentielle Einwände aufkommen, die sich jedoch bei genauerer Betrachtung als wenig kritisch herausstellen:

1. Die Einträge in der Lastmatrix scheinen detaillierte Einblicke in die Arbeitsweise und insbesondere das Kommunikationsverhalten des WFMS vorauszusetzen (vergleiche zum Beispiel Abbildung 4.3). Sie können aber auch durch Auswerten gesammelter Laufzeitstatistiken, wie zum Beispiel Logs, ermittelt werden.

2. Nicht jede Instanz einer Aktivität erzeugt die gleiche Anzahl von Service-Aufträgen. Vielmehr hängt ein Teil der initiierten Service-Aufträge von der Laufzeit der Aktivität ab (so zum Beispiel beim Polling einer Aktivität auf dem Datenbestand des Applikations-Servers). Diese Effekte fließen allerdings indirekt in das Modell mit ein, da wir unsere Parameter, also die erwartete Anzahl von Service-Aufträgen durch eine Aktivitäteninstanz, aus statistischen Beobachtungen beziehen. Zudem

interessiert uns letztendlich ohnehin ein aggregierter Wert bezüglich des Rewards über alle Zustände der CTMC, nämlich der Erwartungswert für die Summe der Service-Aufträge für den gesamten Workflow.
Um den Zusammenhang aus Aktivitätendauer und der Anzahl der anfallenden Service-Aufträge dennoch detaillierter zu modellieren, kann man den zugewiesenen Pauschal-Reward im MRM des Abschnitts 6.3 in einen fixen Anteil, der bei allen Instanzen einer Aktivität identisch ist, und eine Rewardrate, die die Anzahl der auftretenden Service-Aufträge pro Zeiteinheit wiedergibt, in der sich die CTMC in dem zur Ausführung der Aktivität korrespondierenden Zustand befindet, aufteilen. Wie sich diese Zweiteilung auf die Berechnung des zu erwartenden Reward bis zum Betreteten des absorbierenden Zustandes (Gleichung (6.7)) auswirkt, wird in [Tij94] beschrieben.

Um die Abschätzung der zu erwartenden Service-Aufträge eines Workflows des Typs t an einen Server-Typen (Gleichung (6.7)) effizient berechnen zu können, werden wir die Anzahl z der Schritte der CTMC durch eine obere Schranke z^t_{max} beschränken müssen. Wir setzen z^t_{max} auf die Anzahl von Zustandsübergängen, die der Workflow bis zur Terminierung mit einer sehr hohen Wahrscheinlichkeit (zum Beispiel 99%) nicht überschreiten wird.

Bei der Abschätzung der mittleren Verweildauer eines Zustandes der CTMC stoßen wir auf das Problem, daß diese von den Antwortzeiten der Service-Aufträge abhängt, die während des Aufenthaltes in dem entsprechenden Ausführungszustand an die einzelnen Server-Typen geschickt werden. Da die Antwortzeit der Service-Aufträge die Wartezeit an den Servern beinhaltet, entsteht eine wechselseitige Abhängigkeit zwischen den mittleren Wartezeiten für Service-Aufträge und der mittleren Verweilzeit in den einzelnen Ausführungszuständen. In einem gut konfigurierten WFMS wird es jedoch so sein, daß die Wartezeit eines Service-Auftrages nur einen geringen Teil an der gesamten Antwortzeit des Auftrages ausmacht, weswegen wir diese Rückkopplung in dieser Arbeit vernachlässigen.

Bei der Approximation der mittleren Verweildauer in einem Zustand, der mehrere Subworkflows repräsentiert (Gleichung (6.17)), durch das Maximum der mittleren Durchlaufzeiten der Subworkflows handelt es sich lediglich um eine untere Schranke der tatsächlichen mittleren Zustandsverweilzeit (siehe Theorem 6.1), so daß die Abschätzung der durch diesen Zustand verursachten Last an den Servern konservativ ist.

Theorem 6.1. *Erwartungswert des Maximums von Zufallsvariablen*

Seien U_1 und U_2 zwei unabhängige Zufallsvariablen mit den Verteilungsfunktionen $F_{U_1}(u)$ und $F_{U_2}(u)$. Dann gilt für die zusammengesetzte Zufallsvariable $V = \max\{U_1, U_2\}$, daß [Nel95]

1. die Verteilungsfunktion von V gleich dem Produkt der Verteilungsfunktionen von U_1 und U_2 ist, und
2. der Erwartungswert von V größer oder gleich dem Maximum der Erwartungswerte von U_1 und U_2 ist.

Wenn die Dauer einer Aktivität oder von Subworkflows und damit auch die Verweildauer im korrespondierenden Zustand der CTMC nicht wie im Modell angenommen exponentialverteilt ist, so kann man die allgemeine Verteilung der Zustandsverweilzeit durch Substitution des Zustandes durch ein System von Zuständen mit exponentialverteilten Zustandsverweilzeiten wie in Abschnitt 3.3.2 beschrieben approximieren. Ein solches System von Zuständen "simuliert" dann eine der allgemeinen Verteilung "ähnliche" Erlang-Verteilung für die Verweilzeit in dem substituierten Zustand [Tij94]. Dabei werden den ersten künstlichen Zuständen der integrierten Pfade jeweils der Lastvektor des ursprünglichen Zustandes und allen übrigen künstlichen Zuständen der Nullvektor als Pauschal-Reward zugewiesen.

Unser Modell der Server des WFMS (Abschnitt 6.5) geht nicht detailiert auf die Architektur und die Konfiguration der Hardware, wie zum Beispiel Prozessorleistung, vorhandener Hauptspeicher, Anzahl von magnetischen Platten u.s.w., ein. Wir nehmen an, daß jeder Server als wohlkonfigurierter Baustein des Gesamtsystems angesehen werden kann. Wir gehen davon aus, daß alle Ressourcen (Prozessoren, Platten, etc.) des Rechners so ausgewogen konfiguriert sind, daß keine von ihnen zum Flaschenhals wird, während die anderen unterlastet sind. Kommerziell verfügbare Standardserver für Informationssysteme sind in der Regel so konfiguriert. Sollten jedoch im betrachteten WFMS Rechner integriert sein, die diesem Anspruch nicht genügen, kann unser Modell dennoch Verwendung finden, indem die Metrik der mittleren Bedienzeit von Service-Aufträgen von den Ressourcen bestimmt wird, die den Flaschenhals des Rechners darstellen.

Eine modulare Erweiterung des Leistungsmodelles ist darüber hinaus, das M/G/1-Modell zur Beschreibung des Serververhaltens durch umfassendere und detailliertere Warteschlangennetze zu ersetzen (vergleiche zum Beispiel [KDH+95]).

Kapitel 7

Verfügbarkeitsmodell

In diesem Kapitel präsentieren wir das Verfügbarkeitsmodell für verteilte WFMS. Es basiert auf dem Systemmodell von Kapitel 4 und dient zur Analyse der Auswirkungen transienter Server-Ausfälle auf die Verfügbarkeit des Gesamtsystems.

In Abschnitt 7.1 beschreiben wir die Metrik, mit der wir die Verfügbarkeit von einzelnen Komponenten und dem System im Ganzen beschreiben. In Abschnitt 7.2 präsentieren wir das Fehlermodell, das den späteren Überlegungen zur Modellierung von Komponentenausfällen innerhalb eines WFMS zu Grunde liegt. In Abschnitt 7.3 entwickeln wir ein Modell zur stochastischen Beschreibung der Verfügbarkeit von Servern. Dabei greifen wir erneut auf die bereits bewährte Methode der CTMC zurück. In Abschnitt 7.4 präsentieren wir abschließend, wie wir das beschriebene Modell analysieren und welche Aussagen wir daraus ableiten können.

7.1 Metrik

Das Maß, mit dem wir die Verfügbarkeit eines WFMS beschreiben, ist die Wahrscheinlichkeit, daß das System zu einem zufällig gewählten Zeitpunkt verfügbar ist, und der daraus resultierende Erwartungswert für die Gesamtausfallzeit pro Jahr. Unter welchen Umständen ein System als verfügbar gilt, kann dabei unterschiedlich definiert sein. Die einfachste Bewertungsmethode ist nachzusehen, ob von jedem Server-Typ des WFMS mindestens ein Server verfügbar ist, also bereit ist, ankommende Service-Aufträge abzuarbeiten.

Definition 7.1. *(verfügbar)*

> Wir bezeichnen ein WFMS als **verfügbar** genau dann, wenn von jedem Server-Typ mindestens ein Server verfügbar ist.

Ein System kann aber auch schon dann als nicht verfügbar gelten, wenn ein Anwender zu lange auf eine Antwort des Systems warten muß, zum Beispiel weil zu viele Komponenten ausgefallen sind und die übrig gebliebenen die Gesamtlast nicht mehr bewältigen können.

Definition 7.2. *(stark verfügbar)*

> Wir bezeichnen ein WFMS als **stark verfügbar** genau dann, wenn es in dem aktuellen Systemzustand in der Lage ist, den herrschenden Durchsatzanforderungen zu genügen.

Das heißt, daß von jedem Server-Typ x eine genügend große Anzahl an Servern X_x verfügbar sein muß, so daß die Ankunftsrate von Service-Aufträgen l_x kleiner oder gleich dem maximal erreichbaren Durchsatz an dem Server-Typ x im aktuellen Systemzustand ist:

$$l_x \leq d_x^{max} \tag{7.1}$$

Entsprechend lassen sich Aussagen über den Prozentsatz an Zeit treffen, in der das System (stark) verfügbar ist.

Definition 7.3. *(Verfügbarkeit, starke Verfügbarkeit)*

> Der Prozentsatz an Zeit, in der ein WFMS (stark) verfügbar ist, heißt **(starke) Verfügbarkeit** des WFMS.

Bei der Konfiguration von WFMS, besonders solchen, die im elektronischen Handel zum Einsatz kommen, ist das Erreichen von starker Verfügbarkeit sehr erstrebenswert. Explodierende Wartezeiten für Benutzer, die auf ein nicht stark verfügbares System treffen, resultieren meist in verärgertem Abwandern zu konkurrierenden Anbietern [BBK00, Wei00a, KSW+01a]. Wie wir sehen werden, ist unser Verfügbarkeitsmodell allgemein genug, daß damit jedem Verständnis von einem verfügbaren System genügt werden kann.

7.2 Fehlermodell

Bei Rechnersystemen werden in den meisten Fällen drei Arten von Fehlern oder Ausfällen unterschieden.

1. *"Soft-Crashes"* oder *"Heisenbugs"*, benannt nach der Heisenberg'schen Unschärferelation [GR93, WV01]. Heisenbugs sind nichtdeterministisch auftretende und nicht reproduzierbare Fehler, wie zum Beispiel Synchronisationsfehler im Bereich der Systemsoftware oder des Betriebssystems bei hoher Last. Bei Soft-Crashes gehen lediglich die Daten verloren, die zum Zeitpunkt des Ausfalles im Hauptspeicher gehalten werden. Daten auf persistentem Speicher (z.B. einer Festplatte) bleiben erhalten. Zur Behebung eines Soft-Crashes genügt meist ein Reboot des Systems mit anschließender Recovery, was sich in einer zu den anderen Ausfallarten relativ kurzen Reperaturzeit (MTTR) niederschlägt.

2. *"Hard-Crashes"* [WV01]. Hard-Crashes zeichnen sich dadurch aus, daß Daten verloren gehen, die als persistent erachtet wurden (z.B. permanente Plattenfehler). Die Reparatur nach einem Hard-Crash kann daher neben einer aufwendigen Fehlersuche auch den Austausch von Hardware-Komponenten, die Restaurierung von Datenbeständen mit Hilfe von Backup-Kopien oder die Neuinstallation von Software-Paketen erfordern.

3. *Service-Shutdowns.* Service-Shutdowns sind geplant und regelmäßiger Natur. Sie dienen zum Update von Software, Aufspielen von Patches oder sonstigen Wartungsarbeiten. Da es in der Hand der Administratoren liegt, wann ein Service-Shutdown stattfindet, kann man davon ausgehen, daß sie nie allzu kritisch für das Gesamtsystem sind. So werden wohl nie alle Server desselben Typs gleichzeitig oder aktuell stark ausgelastete Server(-Typen) heruntergefahren.

In die folgenden Analyseschritte gehen alle drei Arten von Fehlern ein.

7.3 Stochastische Modellierung der Verfügbarkeit von Servern

In der Literatur sind viele Möglichkeiten zur Modellierung der Verfügbarkeit von Rechnersystemen beschrieben, so zum Beispiel Blockdiagramme, Fault Trees und Verfügbarkeitsgraphen [STP96], um nur einige zu nennen.

Wir bedienen uns zur stochastischen Modellierung von Server-Ausfällen und Inbetriebnahme einer CTMC, die wir wie folgt erzeugen:

1. Jeder Zustand der CTMC repräsentiert einen möglichen Systemzustand des WFMS. Analog zu den Systemzuständen beschreiben wir einen Zustand der CTMC als ein k-Tupel, wobei k die Anzahl der verschiedenen Server-Typen des WFMS ist und jeder Eintrag die Anzahl der aktuell verfügbaren Server eines Server-Typs darstellt. Befindet sich die CTMC zum Beispiel im Zustand $(2, 1, 1)$, so bedeutet das, daß zum betrachteten Zeitpunkt 2 Server vom Typ 0 und jeweils 1 Server von den Server-Typen 1 und 2 dem WFMS zur Verfügung stehen, während alle anderen ausgefallen sind und wieder gestartet weden müssen, unabhängig davon, wie die Systemkonfiguration des WFMS gewählt wurde.

2. Wenn ein Server des Server-Typs x ausfällt, wechselt die CTMC in den Zustand, in dem der zu x korrespondierende Wert (also der Wert an der Stelle x innerhalb des Tupels) gegenüber dem Ausgangszustand um eins kleiner ist. So wird zum Beispiel der Zustand $(X_0, ..., X_x, ..., X_{k-1})$ verlassen, wenn ein Server des Typs $x \in \{0, k-1\}$ ausfällt, und der Zustand $(X_0, ..., (X_x - 1), ..., X_{k-1})$ betreten. Entsprechend wechselt die CTMC nach dem Neustart eines Servers des Server-Typs x in den Zustand, in dem der Wert für den Server-Typ x um eins erhöht ist.

3. Die Übergangsraten zwischen den einzelnen Zuständen der normalisierten CTMC werden durch die Ausfallraten λ_x und Reparaturraten μ_x der einzelnen Server-Typen bestimmt [STP96]. So erhalten wir die Übergangsrate für eine Transition, die bei einem Ausfall eines Servers des Typs x beschritten wird, durch

$$X_x \lambda_x$$

und für eine Transition, die bei einem Neustart eines Servers des Typs x beschritten wird, durch

$$(Y_x - X_x)\, \mu_x,$$

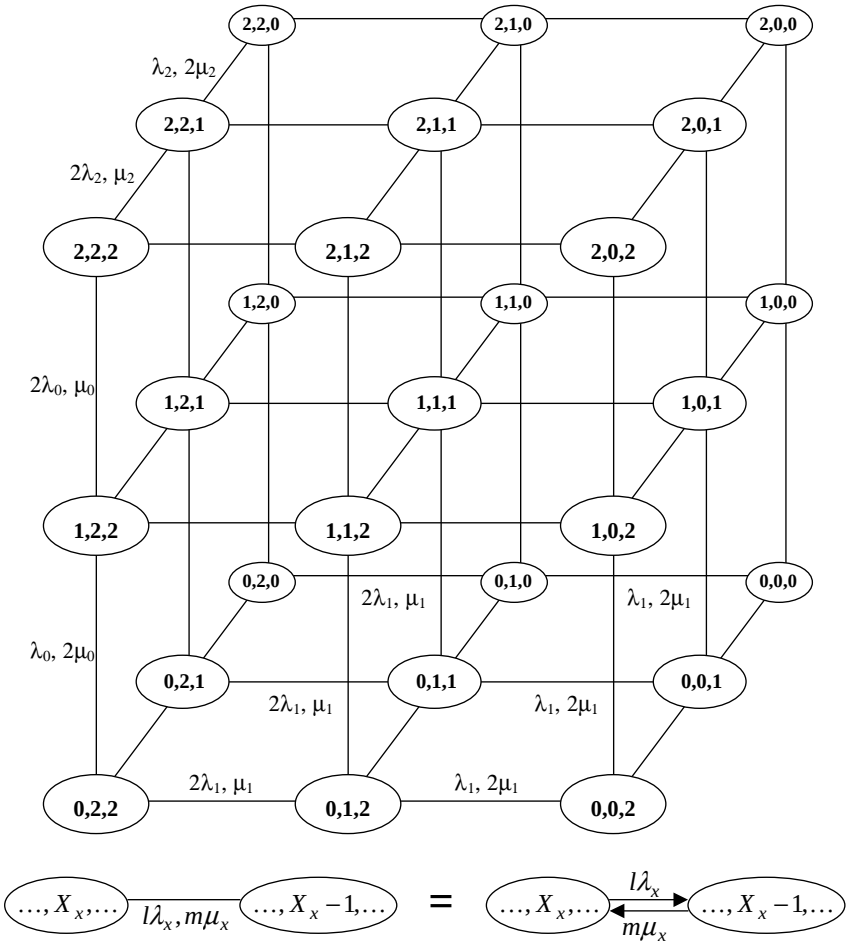


Abbildung 7.1: Beispiel für eine CTMC des Verfügbarkeitsmodelles

wobei X_x jeweils die Anzahl der verfügbaren Server vom Typ x im Ausgangszustand und Y_x die Anzahl der Server vom Typ x in der Systemkonfiguration sind.

Bei dieser Art der Modellierung nehmen wir implizit an, daß nie zwei Ereignisse gleichzeitig eintreffen. Im Modell gibt es zu einem Zeitpunkt maximal einen Ausfall oder Neustart. Die Zeitspanne zwischen zwei aufeinanderfolgenden Ausfällen kann jedoch beliebig nahe bei Null liegen, so daß diese Annahme keinen schwerwiegenden Fehler darstellt.

Beispiel:

Abbildung 7.1 zeigt beispielhaft die Verfügbarkeits-CTMC für ein WFMS mit drei Server-Typen und der Systemkonfiguration $(Y_0, Y_1, Y_2) = (2, 2, 2)$.
Die Kantenbeschriftungen beschreiben die Übergangsraten zwischen zwei "benachbarten" Zuständen. Dabei spezifizieren die Ausdrücke, die ein λ_x enthalten, die Übergangsraten für (weitere) Ausfälle, und die Ausdrücke, die ein μ_x enthalten, die Übergangsraten für einen Neustart eines Servers.
Der Übersichtlichkeit halber sind in Abbildung 7.1 nicht alle Transitionen beschriftet. Darüber hinaus sind keine Transitionen eingezeichnet, deren Übergangsraten gleich Null sind.

Anhand dieser (nachweisbar) ergodischen CTMC können wir die stationären Wahrscheinlichkeiten errechnen, mit denen sich das WFMS in den einzelnen Systemzuständen befin-

det. Dann ergibt sich die Wahrscheinlichkeit, mit der das gesamte WFMS nicht verfügbar ist, aus der Summe der Wahrscheinlichkeiten, daß sich das WFMS in einem Zustand befindet, in dem wenigstens ein Server-Typ nicht verfügbar ist, das heißt, in dem wenigstens an einer Stelle des Zustands-Vektors eine Null vorkommt. Die Einzelheiten der stationären Analyse der CTMC beschreiben wir im folgenden Abschnitt.

7.4 Stationäre Analyse der Verfügbarkeit von Servern

Die stationäre Analyse der Verfügbarkeits-CTMC aus Abschnitt 7.3 liefert uns Informationen über die Wahrscheinlichkeiten, mit denen sich das WFMS in den verschiedenen Systemzuständen befindet.

Seien k die Anzahl der verschiedenen Server-Typen, $(Y_0, ..., Y_{k-1})$ die Systemkonfiguration und $X = \{(X_0, ...X_{k-1}) \,|\, 0 \leq X_x \leq Y_x, 0 \leq x \leq k-1\}$ die endliche Menge der möglichen Systemzustände des WFMS.
Wir bilden X bijektiv auf die Menge $\tilde{X}$ von Integerwerten ab, die im Folgenden die Zustände des CTMC-Modelles aus Abschnitt 7.3 referenzieren:

$$(X_0, ..., X_{k-1}) \mapsto \sum_{j=0}^{k-1} X_j \prod_{l=0}^{j-1} (Y_l + 1) \tag{7.2}$$

So werden zum Beispiel die Zustände $(0,0,0)$, $(1,0,0)$, $(2,0,0)$, $(0,1,0)$ etc. der CTMC aus Abbildung 7.1 auf die Werte 0, 1, 2, 3 etc. abgebildet.

Die *infinitesimale Generatormatrix* der CTMC $Q = (q_{ij})$ mit $i, j \in \tilde{X}$ ist dann wie folgt aufgebaut [STP96]:

- Jedes q_{ij} mit $i \neq j$ ist gleich der Übergangsrate für die Transition vom Zustand $(X_0, ..., X_{k-1}) \in X$, der durch den Wert $i \in \tilde{X}$ repräsentiert wird, in den Zustand $\left(X'_0, ..., X'_{k-1}\right) \in X$, der durch den Wert $j \in \tilde{X}$ repraesentiert wird.

- Die Elemente auf der Diagonalen von Q werden auf

$$q_{ii} = -\sum_j q_{ij}$$

  gesetzt. $-q_{ii}$ ist die Rate, mit der die CTMC den Zustand $i \in \tilde{X}$ verläßt.

Die stationären Zustandswahrscheinlichkeiten π_i, $i \in \tilde{X}$, der CTMC erhalten wir durch Lösen des linearen Gleichungssystems

$$\begin{aligned} \pi Q &= 0 \\ \sum_i \pi_i &= 1, \end{aligned} \tag{7.3}$$

wobei π der Vektor (π_i) ist.

7.5 Zusammenfassung und Diskussion

In diesem Kapitel haben wir ein Modell für die Verfügbarkeit von verteilten WFMS vorgestellt. Wie auch in den vorangegangenen Kapiteln bedienten wir uns dabei der Methode der zeitkontinuierlichen Markov-Ketten (CTMC). Der Aufbau der CTMC basierte auf dem Systemmodell von Kapitel 4. Die stationäre Analyse der CTMC ermöglicht es uns, Aussagen über die erwartete Ausfallrate des WFMS und die Wahrscheinlichkeiten zu treffen, mit denen sich das WFMS in den verschiedenen Systemzuständen befindet.

Das den Analysen zu Grunde liegende Systemmodell ist einfach und grob gehalten. Daher sollten die Ergebnisse der Analyse, insbesondere die Werte über die erwarteten Ausfallzeiten des WFMS, nicht als absolut angesehen werden. Vielmehr sollten die Tendenzen bei der Variation der Konfiguration beobachtet und ausgewertet werden. So können "Quantensprünge" zwischen verschiedenen Konfigurationen erkennbar sein, die bei der Replikation eines bestimmten Server-Typs (zum Beispiel des am unzuverlässigsten) auftreten. Zudem kann die Konfiguration gefunden werden, die am kostengünstigsten einen gesetzten Schwellwert für die Verfügbarkeit erreicht, oder ab der sich weitere Server-Replikationen nur noch marginal auf die erwartete Verfügbarkeit des WFMS auswirken.

Beispiel:

Betrachten wir als Beispiel das folgende Szenario für ein WFMS mit drei verschiedenen Server-Typen. Dabei seien der Server-Typ 0 ein Typ von Workflow-Servern, Server-Typ 1 ein Applikations-Server-Typ und Server-Typ 3 der Kommunikations-Server-Typ. Die Ausfallraten der drei Server-Typen seien wie folgt gegeben:

- Ein Workflow-Server (das heißt ein Server des Server-Typs 0) fällt im Durchschnitt einmal pro Monat aus. Also ist

$$\lambda_0 = \frac{1}{43200} min^{-1}.$$

- Ein Applikations-Server (das heißt ein Server des Server-Typs 1) fällt im Durchschnitt einmal pro Woche aus. Also ist

$$\lambda_1 = \frac{1}{10080} min^{-1}.$$

- Ein Kommunikations-Server (das heißt ein Server des Server-Typs 2) fällt im Durchschnitt einmal pro Tag aus. Also ist

$$\lambda_2 = \frac{1}{1440} min^{-1}.$$

Die eingesetzten Werte sind willkürlich gewählt. Sie reflektieren aber das Verhältnis der Ausfallraten der einzelnen Server-Typen und der damit assoziierten Softwaretechnologien untereinander recht gut.
Die durchschnittliche Zeit bis zur Wiederherstellung der Verfügbarkeit eines Servers, also die MTTR inklusive Speicherallokation, Neustart der Kommunikationsinfrastruktur, Recovery-Maßnahmen, etc., liege für alle Server-Typen

Systemkonfiguration	Erwartete Ausfallzeit pro Jahr
(1,1,1)	71 h
(2,1,1)	65 h
(1,2,1)	62 h
(2,2,1)	60 h
(1,1,2)	11 h
(2,1,2)	5 h
(1,2,2)	86 min
(2,2,2)	26 min
(2,3,2)	25 min
(2,2,3)	21 sec
(2,3,3)	12 sec
(3,3,3)	10 sec

Tabelle 7.1: Erwartete Nicht-Verfügbarkeit des Beispiel-WFMS

bei zehn Minuten. Daraus ergeben sich die Reperaturraten

$$\mu_0 = \mu_1 = \mu_2 = \frac{1}{10} min^{-1}.$$

Ebenfalls in die augenscheinlich hohe MTTR gehen auch selten vorkommende aber dafür umso höhere Zeiten für Hard-Crashes und Service-Shutdowns ein.

Tabelle 7.1 zeigt die Ergebnisse der stationären Analysen der CTMC verschiedener Systemkonfigurationen (Y_0, Y_1, Y_2). Wir erkennen, daß die Zeit, die das WFMS bei einer Konfiguration mit nur einem Server pro Server-Typ erwartungsgemäß pro Jahr nicht verfügbar ist, bei ungefähr 71 Stunden liegt. Durch Hinzunahme von Server-Replikaten kann die Ausfallzeit pro Jahr deutlich verringert werden, zum Beispiel bis hin zu 10 Sekunden bei einer Konfiguration mit drei Servern pro Server-Typ.
Darüber hinaus erkennen wir, daß das Replizieren des unzuverlässigsten Server-Typs (der Kommunikations-Server-Typ in unserem Beispiel) die erwartete Ausfallzeit am signifikantesten verringert.
Schließlich erkennen wir, daß drei Server des Kommunikations-Server-Typs notwendig sind, um die erwartete Ausfallzeit unter einen "akzeptablen" Schwellwert (hier eine Minute pro Jahr) zu drücken. Ein drittes Replikat eines der beiden anderen Server-Typen veringert die erwartete Ausfallzeit hingegen nur noch marginal.

Kapitel 8

Performability-Modell

Das Leistungsmodell aus Kapitel 6 alleine erlaubt uns lediglich Voraussagen über die Wartezeiten von Service-Aufträgen für einen gegebenen Systemzustand des WFMS. Allerdings variieren die Systemzustände eines verteilten WFMS über die Zeit, wie wir es bereits in Kapitel 7 diskutiert haben. Solche Wechsel zwischen den Systemzuständen durch Server-Ausfälle und Neustarts werden im reinen Leistungsmodell nicht berücksichtigt.

In diesem Kapitel kombinieren wir das Leistungsmodell aus Kapitel 6 und das Verfügbarkeitsmodell aus Kapitel 7, um den Einfluß transienter Ausfälle und zeitweiser Nicht-Verfügbarkeit einzelner Server auf das Wartezeitverhalten der Service-Aufträge an den einzelnen Server-Typen darlegen zu können. Diese Kombination aus Leistung (englisch:*performance*) und Verfügbarkeit (englisch: *availability*) ist in der Literatur unter dem Begriff *Performability* [STP96] bekannt.

In Abschnitt 8.1 erörtern wir die Auswirkungen transienter Server-Ausfälle auf die Metrik. In Abschnitt 8.2 präsentieren wir unser Performability-Modell und leiten den stationären Erwartungswert für die Wartezeiten von Service-Aufträgen an den verschiedenen Server-Typen unter Berücksichtigung transienter Server-Ausfälle her. In Abschnitt 8.3 beschreiben wir, wie sich auf analoge Art und Weise der stationäre Erwartungswert für den maximalen Durchsatz von Service-Aufträgen und Workflow-Instanzen ermitteln lassen.

8.1 Metrik

Die Replikation von Servern, also die Existenz mehrerer Instanzen eines Server-Typs, bewirkt, daß bei einem Ausfall eines oder mehrerer Server das gesamte WFMS nach außen hin dennoch verfügbar bleiben kann. Wenn das WFMS den Ausfall eines Servers erkennt, kann es die Service-Aufträge an diesen Server auf die übrigen verfügbaren Instanzen desselben Server-Typs umlieten. Die verbleibenden Server übernehmen so die Arbeit der ausgefallenen Instanz, bis deren Reperatur abgeschlossen und sie wieder verfügbar ist [KAG+96, STP96]. Diese Umverteilung der Service-Aufträge erzeugt eine erhöhte Last an den übernehmenden Servern, was dazu führt, daß sich dort die Warte- und Antwortzeiten von Service-Aufträgen erhöhen und sich der maximal erreichbare Durchsatz des Gesamtsystems verringert.

Definition 8.1. *(Performability)*

> Die Leistung eines WFMS unter Berücksichtigung transienter Komponentenausfälle heißt die **Performability** des WFMS.

Die Metriken, mit denen wir die Performability eines WFMS unter einer gegebenen Systemkonfiguration beschreiben, sind

1. die mittlere Wartezeit und
2. der zu erwartende Durchsatz

von Service-Aufträgen an den verschiedenen Server-Typen, die zu einem beliebigen Zeitpunkt zu erwarten sind, wenn sich das WFMS in einem bezüglich des Ausfallverhaltens der Server "eingeschwungenen" Zustand befindet.

8.2 Stationärer Erwartungswert für Wartezeiten

Unser Performability-Modell ist ein hierarchisches MRM. Als Basis dient uns das CTMC-Modell für die Verfügbarkeit verteilter WFMS aus Abschnitt 7.3. Wir errechnen daraus, mit welcher Wahrscheinlichkeit sich das WFMS in den unterschiedlichen Systemzuständen befindet. Die Zustands-spezifischen Rewardraten liefern uns die Analysen des Leistungsmodelles aus Abschnitt 6.5. Die Ergebnisse der stationären Analysen des MRM können auch als "Erwartungswerte über bedingte Erwartungswerte" interpretiert werden.

Wir benutzen als Rewardrate für die Zustände der CTMC die mittlere Wartezeit von Service-Aufträgen, die zu erwarten ist, wenn sich das WFMS in dem entsprechenden Systemzustand befindet. Dazu müssen wir für jeden möglichen Systemzustand die erwartete Leistung des WFMS bei gegebener Last entsprechend den Ausführungen in Kapitel 6 ermitteln. Durch die stationäre Analyse des MRM erhalten wir den Erwartungswert für die Rewardrate (englisch: *expected reward rate* [STP96]), was in unserem Modell dem Erwartungswert für die mittlere Wartezeit der Service-Aufträge an den verschiedenen Server-Typen des WFMS bei einer gegebenen Systemkonfiguration unter Berücksichtigung von Leistungseinbußen durch Server-Ausfälle entspricht.

Im folgenden betrachten wir ausschließlich die Systemzustände, in denen das WFMS stark verfügbar ist (vergleiche Definition 7.2). Die Konfiguration des WFMS sollte grundsätzlich so vorgenommen werden, daß die Wahrscheinlichkeit, in einen nicht stark verfügbaren Systemzustand zu geraten, marginal wird. Befindet sich das System in einem Zustand, in dem es nicht stark verfügbar ist, kann man nicht mehr von einer stabilen Leistung sprechen.

Seien Y eine gegebene Systemkonfiguration und für alle $i \in \tilde{X}'$ mit $\tilde{X}' \subset \tilde{X}$, die zu einem stark verfügbaren Systemzustand des WFMS korrespondieren, π_i die stationäre Zustandswahrscheinlichkeit für den zu i korrespondierenden Systemzustand (analog zu den Berechnungen in Kapitel 7). Sei darüber hinaus für alle $i \in \tilde{X}'$

$$w(i) = (w_x(i)) \tag{8.1}$$

der Vektor über die mittleren Wartezeiten der Service-Aufträge an den verschiedenen Server-Typen x, die zu erwarten sind, wenn sich das WFMS in dem zu i korrespondierenden Systemzustand befindet (entsprechend den Ausführungen aus Kapitel 6). Der Vektor, der die mittleren Wartezeiten wiedergibt, wenn sich das WFMS in dem Systemzustand befindet, der die Systemkonfiguration Y wiederspiegelt (das heisst, in dem alle Server, mit denen das WFMS konfiguriert wurde, verfügbar sind), bildet dabei die obere Schranke für die Leistung des WFMS unter der gegebenen Systemkonfiguration.

Der Performability-Vektor $W^Y = \left(W_x^Y\right)$ über die erwarteten Wartezeiten von Service-Aufträgen an den verschiedenen Server-Typen x für die Systemkonfiguration Y unter Berücksichtigung von transienten Server-Ausfällen ergibt sich aus dem Produkt aus der normierten Wahrscheinlichkeit, daß sich das WFMS in einem entsprechenden Systemzustand befindet, und dem Vektor über die mittleren Wartezeiten von Service-Aufträgen an den verscheidenen Server-Typen in dem Systemzustand, summiert über alle möglichen Systemzustände des WFMS:

$$W^Y = \sum_{i \in \tilde{X}'} \frac{\pi_i}{\sum\limits_{j \in \tilde{X}'} \pi_j} w(i) = \frac{1}{\sum\limits_{j \in \tilde{X}'} \pi_j} \sum_{i \in \tilde{X}'} \pi_i w(i). \tag{8.2}$$

Die so ermittelten Werte in W^Y sind unsere maßgebliche Metrik für die Qualität einer Systemkonfiguration. Wir sind damit in der Lage, verschiedene Systemkonfigurationen zu vergleichen und die günstigste Konfiguration zu ermitteln, bei der die mittlere Leistung des WFMS unter Berücksichtigung von Server-Ausfällen einen gegebenen Schwellwert nicht unterschreitet.

8.3 Stationärer Erwartungswert für maximalen Durchsatz

Analog zu den Ausführungen in Abschnitt 8.2 können wir den Erwartungswert für den maximalen Durchsatz eines WFMS bei gegebener Systemkonfiguration unter Berücksichtigung transienter Server-Ausfälle bestimmen. Wir benutzen dabei für jeden Systemzustand des WFMS den maximalen erreichbaren Durchsatz des WFMS in dem Systemzustand als Rewardrate.

Sei für alle $i \in \tilde{X}$

$$d^{max}(i) = \left(d_x^{max}(i)\right) \tag{8.3}$$

der Vektor über den jeweils maximal erreichbaren Durchsatz an Service-Aufträgen an den verschiedenen Server-Typen x, der zu erwarten ist, wenn sich das WFMS in dem zu i korrespondierenden Systemzustand befindet (entsprechend den Ausführungen aus Kapitel 6). Der Vektor, der den maximal erreichbaren Durchsatz wiedergibt, wenn sich das WFMS in dem Systemzustand befindet, der die Systemkonfiguration Y wiederspiegelt (das heisst, in dem alle Server, mit denen das WFMS konfiguriert wurde, verfügbar sind), bildet dabei wiederum die obere Schranke für die Leistung des WFMS unter der gegebenen Systemkonfiguration. Für die Systemzustände, in denen einzelne Server-Typen oder das ganze WFMS nicht verfügbar sind, setzen wir an die entsprechenden Stellen innerhalb

des Vektors eine Null.

Der Vektor $D^Y = \left(D_x^Y\right)$ über den zu erwartenden maximalen Durchsatz an Service-Aufträgen an den verschiedenen Server-Typen x für die Systemkonfiguration Y unter Berücksichtigung von transienten Server-Ausfällen ergibt sich aus dem Produkt aus der Wahrscheinlichkeit, daß sich das WFMS in einem entsprechenden Systemzustand befindet, und dem Vektor mit den Werten für den jeweils maximalen Durchsatz von Service-Aufträgen an den verscheidenen Server-Typen in dem Systemzustand, summiert über alle möglichen Systemzustände des WFMS:

$$D^Y = \sum_{i \in \tilde{X}} \pi_i d^{max}(i). \tag{8.4}$$

8.4 Zusammenfassung und Diskussion

In diesem Kapitel haben wir unser Performability-Modell präsentiert. Performability setzt sich zusammen aus Leistung und Verfügbarkeit und beschreibt die Leistung eines Systemes unter Berücksichtigung transienter Komponentenausfälle. Unser Performability-Modell ist ein hierarchisches Markov-Reward-Modell (MRM) mit der Verfügbarkeits-CTMC aus Abschnitt 7.3 und dem Leistungsmodell aus Kapitel 6 als Bausteine. Mit Hilfe der stationären Analyse des MRM haben wir die Berechnung der erwarteten Wartezeiten und des zu erwartenden maximalen Durchsatzes an Service-Aufträgen an den verschiedenen Server-Typen unter Berücksichtigung transienter Server-Ausfälle berechnet. Die erwarteten Wartezeiten bilden unsere maßgebliche Metrik zur Bewertung von Systemkonfigurationen.

Durch das Ausklammern der Systemzustände, in denen das WFMS nicht stark verfügbar ist, aus den Berechnungen der zu erwartenden Performability unterschätzen wir die Auswirkungen beim Hinzufügen von Server-Replikaten. Die absoluten Werte der zu erwartenden Wartezeiten aus den Berechnungen werden kleiner sein als die, die die Benutzer tatsächlich im Mittel erfahren werden, da es vorkommen kann, daß ein Benutzer das System in einem nicht stark verfügbaren Zustand antrifft. Bei einer geeigneten Konfiguration des WFMS, sollte dieses Ereignis allerdings sehr selten sein.

Darüber hinaus werden Warteschlangeneffekte bei den Zustandsübergängen vernachlässigt. Wenn das System durch einen Ausfall oder Wiederinbetriebnahme eines Servers in einen anderen Zustand übergeht, können Service-Aufträge an den betroffenen Server-Typ auf ihre Abarbeitung warten. Solche Überbleibsel nach einem Zustandsübergang sind in unserem Modell nicht berücksichtigt, können aber zu einer Erhöhung der Wartezeit für nachfolgende Service-Aufträge beitragen.

Ähnlich wie bei dem Verfügbarkeitsmodell von Kapitel 7 kann das Performability-Modell zumindest Tendenzen bei der Variation der Systemkonfiguration deutlich machen und "Quantensprünge" zwischen den Wartezeiten bei verschiedenen Systemkonfigurationen identifizieren.

Teil III

Anwendung

Kapitel 9

Entwurf eines Konfigurationswerkzeuges

In diesem Kapitel beschreiben wir den Entwurf eines Konfigurationswerkzeuges für verteilte WFMS basierend auf den Modellen des Teiles II dieser Arbeit [GWW+99, GWW+00]. Kapitel 10 befaßt sich mit der Umsetzung dieser Konzepte bei der Implementierung eines Konfigurationswerkzeuges für das WFMS Mentor-lite.

In Abschnitt 9.1 beschreiben wir die Architektur des Werkzeuges und die Funktionalität der einzelnen Komponenten. Darüber hinaus zeigen wir auf, wie das Werkzeug in die Infrastruktur eines gegebenen WFMS eingebettet werden kann. In Abschnitt 9.2 entwickeln wir einen Algorithmus, mit dessen Hilfe das Konfigurationswerkzeug automatisch Vorschläge zur kostengünstigsten Konfiguration eines WFMS bei gegebener Last und gegebenen Schwellwerten für tolerierbare Wartezeiten und erwartete Verfügbarkeit ermitteln kann.

9.1 Funktionalität und Architektur

Ein Konfigurationswerkzeug für verteilte WFMS soll einen Administrator eines WFMS bei der Suche nach einer geeigneten Konfiguration unterstützen. Dabei sind zwei verschiedene Herangehensweisen seitens des Administrators denkbar.

- Zum einen kann der Administrator die Auswirkungen von hypothetischen Systemkonfigurationen untersuchen, indem er das Werkzeug mit einer Testkonfiguration parametrisiert. Das Konfigurationswerkzeug soll für diesen Fall Vorhersagen über die zu erwartende Leistung, Verfügbarkeit und Performability des so konfigurierten WFMS liefern.

- Zum zweiten soll das Werkzeug eine kostengünstige Konfiguration vorschlagen können, wenn der Administrator Schwellwerte für akzeptable mittlere Wartezeiten und die mindestens zu erreichende (starke) Verfügbarkeit des WFMS angibt.

In beiden Fällen sollen Parameter für die Last und die Beschaffenheit der Server sowohl manuell einstellbar sein als auch automatisch über das Monitoring und die Historiendaten des laufenden WFMS ermittelt werden können.

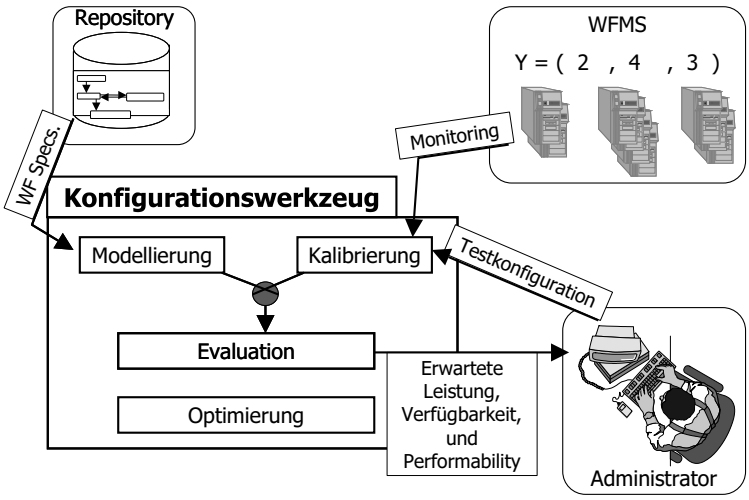


Abbildung 9.1: Vorhersagen für Testkonfigurationen

Abbildung 9.1 gibt einen Überblick über die Architektur des entwickelten Konfigurationswerkzeuges und seine Einbettung in die Infrastruktur eines WFMS. Die Pfeile zeigen an, woher die einzelnen Komponenten des Werkzeuges Eingaben und Parameter erhalten. In der Abbildung ist das Szenario dargestellt, bei dem der Administrator die Auswirkungen einer hypothetischen Konfiguration vom Konfigurationswerkzeug vorhersagen läßt.

Das Konfigurationswerkzeug ist aus den folgenden vier Komponenten aufgebaut:

1. einem Modellierungsmodul zur Abbildung von Workflow-Spezifikationen auf das interne stochastische Modell aus Kapitel 5,
2. einem Kalibrierungsmodul zur Parametrisierung der internen Modelle,
3. einem Evaluationsmodul zur Auswertung der internen Modelle gemäß der eingegebenen Parameter, sowie
4. einem Optimierungsmodul zur Berechnung der kostengünstigsten Konfiguration für gegebene Wartezeit- und Verfgbarkeitsziele unter Berücksichtigung gegebener Parameter.

Im folgenden werden wir die Funktionalität der einzelnen Komponenten genauer erläutern.

9.1.1 Modellierung

Das Modellierungsmodul lädt die Spezifikationen der verschiedenen Workflow-Typen aus dem Repository des WFMS und bildet sie auf das interne stochastische Modell aus Kapitel 5 ab. Für jeden Workflow-Typ wird automatisch eine korrespondierende CTMC erstellt. Die Parametrisierung der CTMC, wie zum Beispiel die Zuordnung von mittleren Zustandsverweildauern und Schrittwahrscheinlichkeiten zwischen den einzelnen Zuständen, erfolgt durch das Kalibrierungsmodul des Konfigurationswerkzeuges.

9.1.2 Kalibrierung

Zur Evaluation der internen Modelle des Konfigurationswerkzeuges müssen Parameter kalibriert werden. Dazu gehören zum einen Parameter, die die Beschaffenheit der Server

beschreiben (die beiden ersten Momente der Bedienzeitverteilungen sowie MTTR und MTTF), und zum anderen Parameter, mit denen Rückschlüsse auf die Last im System vorgenommen werden können (wie zum Beispiel Ankunftsraten von Instanzen der verschiedenen Workflow-Typen etc.).

Mit Hilfe des Kalibrierungsmoduls werden die Parameter automatisch über das Monitoring des laufenden Systems oder manuell über eine Benutzerschnittstelle aufgenommen. Dabei erlaubt ein Mischbetrieb bei der Art der Kalibrierung, daß einzelne Parameter, die durch die Online-Statistiken automatisch bestimmt wurden, durch einen Benutzer des Konfigurationswerkzeuges abgeändert werden können, was vor allem bei der Evaluation hypothetischer Konfigurationen nützlich ist.

Wenn das Konfigurationswerkzeug eingesetzt wird, um eine neue Workflow-Umgebung zu konfigurieren, müssen alle Parameter von einem menschlichen Experten geschätzt werden. Nachdem das WFMS dann eine Weile in Betrieb ist, sollen möglichst alle Parameter durch Online-Statistiken über das laufende System kalibriert werden, so daß das Konfigurationswerkzeug periodisch Vorschläge für eine Rekonfiguration des WFMS machen kann.

9.1.3 Evaluation

Den mathematischen Kern des Konfigurationswerkzeuges bildet das Evaluationsmodul. Es wertet die internen Modelle gemäß der eingegebenen Parameter aus, berechnet den maximal erreichbaren Durchsatz und liefert Vorhersagen über erwartete Warte- und Antwortzeiten sowie die Verfügbarkeit des Systems.

Das Evaluationsmodul erhält für jeden Workflow-Typ eine CTMC vom Modellierungsmodul und die dazugehörigen Parameter vom Kalibrierungsmodul. Dadurch ist es gemäß den Ausführungen in Kapitel 6 in der Lage, die Leistung des Systems in den einzelnen Systemzuständen zu errechnen. Desweiteren kann es mittels der Parameter über die Beschaffenheit der Server des WFMS, die ebenfalls über das Kalibirierungsmodul einfließen, die Verfügbarkeit und die Performability einer potentiellen Systemkonfiguration gemäß den Ausführungen in den Kapiteln 7 und 8 vorher zu sagen. Die Systemkonfiguration, für die die Berechnungen angestellt werden soll, kann einerseits von einem Benutzer des Konfigurationswerkzeuges über das Kalibrierungsmodul eingegeben werden (vergleiche Abbildung 9.1), oder ein zu evaluierendes (Zwischen-)Ergebnis des Optimierungsmoduls sein, das wir in dem folgenden Abschnitt genauer erläutern.

9.1.4 Optimierung

Das Optimierungsmodul berechnet die kostengünstigste Systemkonfiguration für gegebene Wartezeit-, Verfügbarkeits- und Durchsatzziele unter Berücksichtigung gegebener Parameter.

Die Administratoren eines WFMS können folgende Ziele spezifizieren:

1. einen zulässigen Schwellwert für die mittlere Wartezeit von Service-Aufträgen, die den Benutzern des WFMS zugemutet und als akzeptabel angesehen werden,

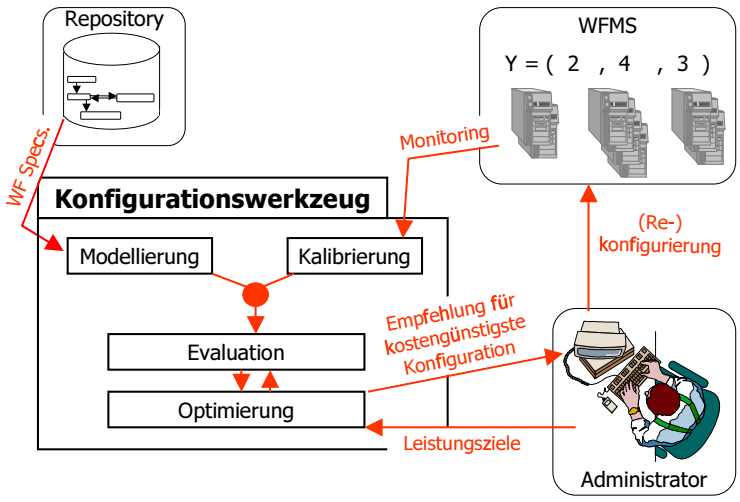


Abbildung 9.2: Vorschläge für kostengünstigste Konfigurationen

2. einen zulässigen Schwellwert für den Anteil an Zeit, in der das WFMS nicht verfügbar ist, oder, in anderen Worten, ein gefordertes Minimum für die Verfügbarkeit des WFMS, sowie

3. einen zu erreichenden Schwellwert für den Anteil an Zeit, in der das WFMS stark verfügbar ist, also einen gegebenen Durchsatz in Workflow-Instanzen pro Zeiteinheit bewältigen kann.

Das erste und das dritte Ziel erfordern die Evaluation des Performability-Modelles, während man für das zweite Ziel lediglich auf das Verfügbarkeitsmodell zurückgreifen muß. Das Optimierungsmodul ist in der Lage, die Berechnungen des Evaluationsmodules für einzelne Systemkonfigurationen anzustoßen. Die Ergebnisse der Berechnungen des Evaluationsmodules werden zur Ermittlung der kostengünstigsten Systemkonfiguration für gegebene Ziele benutzt, worauf wir im nächsten Abschnitt noch genauer eingehen.

Abbildung 9.2 skizziert die Interaktion mit dem Administrator eines WFMS und die internen Kommunikationswege für diese Art des Einsatzes des Konfigurationswerkzeuges.

9.2 Ermittlung der kostengünstigsten Konfiguration

Die Ermittlung der kostengünstigsten Systemkonfiguration eines WFMS, die bei einer gegebenen Last von Workflow-Instanzen von einem Administrator spezifizierte Ziele für Leistung und Verfügbarkeit des WFMS einhält, erfordert das Durchsuchen der Menge aller möglicher Systemkonfigurationen und die Evaluation der internen Modelle des Konfigurationswerkzeuges für alle diese Konfigurationen. Während die korrekte Lösung dieses Problemes eigentlich eines mathematischen Optimierungs-Algorithmus wie zum Beispiel Branch-And-Bound bedarf, beschränken wir uns in dieser Arbeit als ersten Ansatz auf eine einfache Greedy-Heuristik, die sich der optimalen Lösung nähert und stoppt, sobald alle spezifizierten Ziele erreichbar sind. Das Ergebnis der Heuristik ist eine Systemkonfiguration, die die gesetzten Ziele erreicht, aber nicht unbedingt für alle Server-Typen den kostengünstigsten Replikationsgrad ermittelt. Abbildung 9.3 zeigt den Algorithmus dieser

```
1  GreedyAlgorithmus {
2     Berechne initiale Konfiguration (Y1, ..., Yn), so daß jeder
3        Server-Typ den benötigten Durchsatz erfüllen kann.
4
5     Berechne Verfügbarkeit des Systems für diese Konfiguration.
6     Berechne Performability des Systems für diese Konfiguration.
7
8     do {
9        if(Verfügbarkeit zu niedrig) {
10          Ermittle Server-Typ x mit der geringsten Verfügbarkeit.
11          Erhöhe Replikationsgrad Yx des Server-Typs x um 1.
12          Berechne Performability für neue Konfiguration.
13       }
14
15       if(Erwartete Wartezeit zu hoch) {
16          Ermittle den Server-Typ x  mit der höchsten erwarteten
17          Wartezeit mit Hilfe des Performability-Modelles.
18          Erhöhe den Replikationsgrad Yx des Server-Typs x um 1.
19       }
20
21       Berechne Performability und Verfügbarkeit für die neue
22          Konfiguration.
23    }
24    while (Performability- und Verfügbarkeitsziele nicht erreicht)
25 }
```

Abbildung 9.3: Pseudocode des Greedy-Algorithmus

Greedy-Heuristik in Pseudocode-Notation.

Wir nehmen an, daß die Kosten einer Systemkonfiguration proportional zu der Gesamtzahl der Server im WFMS sind. Es ist aber auch möglich, ein verfeinertes Kostenmodell zu erstellen, das zwischen den einzelnen Server-Typen differenziert. Für unseren ersten Ansatz, den im folgenden vorgestellten Greedy-Algorithmus, würde sich allerdings kein Unterschied in der Arbeitsweise ergeben, so daß wir uns vorerst auf diese Annahme beschränken.

Der Greedy-Algorithmus iteriert über potentielle Konfigurationen, indem er die Anzahl der Server-Replikate des kritischsten Server-Typs erhöht, bis sowohl die Verfügbarkeits- als auch die Leistungsziele erreicht sind. Dabei ist zu beachten, daß für das Erreichen der gesetzten Ziele zwei verschiedene Server-Typen kritisch sein können, nämlich einer bezüglich des Verfügbarkeitszieles und einer bezüglich des Leistungszieles. Darüber hinaus beeinträchtigt ein zusätzlicher Server beide Metriken zugleich. Daher werden beide Arten von Zielen in einer Schleife zusammen angegangen. Bei jeder Iteration der Schleife werden sowohl die erwartete Verfügbarkeit als auch die erwartete Performability des WFMS bei der entsprechenden potentiellen Systemkonfiguration mit Hilfe des Evaluationsmodules ermittelt. Nachdem ein Server hinzugefügt wurde, um das erste der beiden Ziele (zum Beispiel das Verfügbarkeitsziel wie in den Zeilen 10 und 11 in Abbildung 9.3) zu erreichen, wird die zweite Metrik erneut evaluiert (Zeile 12 in Abbildung 9.3), bevor eventuell ein zweiter Server inerhalb des selben Iterationsschrittes hinzugefügt wird.

Die erwartete Wartezeit, mit der der vom Benutzer vorgegebene Schwellwert in den Zeilen 15 und 24 des Algorithmus aus Abbildung 9.3 verglichen wird, ist die nach Häufigkeit im Gesamtvolumen gewichtete Summe der erwarteten Wartezeiten der Service-Aufträge an

den verschiedenen Server-Typen x bei der aktuellen Systemkonfiguration Y

$$\sum_x \frac{l_x}{\sum_x l_x} W_x^Y.$$

9.3 Zusammenfassung und Diskussion

Die Funktionalität des in diesem Kapitel konzeptionell erarbeiteten Konfigurationswerkzeuges für verteilte WFMS reicht

- von der mathematisch fundierten Unterstützung eines Administrators bei der Festlegung einer initialen Systemkonfiguration bei Inbetriebnahme einer neuen Workflow-Umgebung basierend auf geschätzten Last- und Systemparametern
- über die Evaluation hypothetischer Systemkonfigurationen eines im Betrieb befindlichen WFMS basierend auf Online-Statistiken
- bis hin zu der Möglichkeit, automatisch turnusmäßig Vorschläge zur Rekonfigurierung eines im Betrieb befindlichen WFMS unter Berücksichtigung spezifizierter Leistungs- und Verfügbarkeitsziele zu ermitteln.

Zukünftige Arbeiten bei der Weiterentwicklung des Konfigurationswerkzeuges werden unter anderem folgende Punkte in Betracht ziehen:

1. Die in Abschnitt 9.2 vorgestellte Heuristik zur Ermittlung der kostengünstigsten Konfiguration resultiert zwar in einer Konfiguration, die die spezifizierten Leistungs- und Verfügbarkeitsziele einhalten kann, die das System aber tendenziell "überkonfiguriert", da nicht für alle Server-Typen unbedingt der kostengünstigste Replikationsgrad gefunden wird. Es ist zu untersuchen, welche bekannten Optimierungs-Algorithmen sich auf unser Problem anwenden und an Stelle des Greedy-Algorithmus in das Konfigurationswerkzeug integrieren lassen.

2. Typischerweise werden für zwei Server verschiedener Server-Typen nicht die gleichen Kosten anfallen. Zum Beispiel werden die Lizenzgebühren für Applikations-Server aufgrund der Vielfalt der installierten Software von den Lizenzgebühren von Workflow-Servern, bei denen alle Komponenten von dem selben Hersteller/Vertreiber geliefert sind, differieren. Auch die Hardware der Rechner muß unterschiedlichen Anforderungen gerecht werden (zum Beispiel brucht ein Workflow-Server viel Hauptspeicher, ein Kommunikations-Server hingegen eher eine sehr schnelle Netzkarte). Es muß daher ein Kostenmodell entwickelt werden, das es den Optimierungs-Algorithmen erlaubt, die kostengünstigste Systemkonfiguration auch bei unterschiedlichen Kosten für verschiedene Server-Typen zu finden.

3. Die Ziele, die der Administrator spezifiziert und zu deren Einhaltung das Konfigurationswerkzeug eine geeignete Konfiguration vorschlägt, können von Server-Typ zu Server-Typ variieren. Wenn ein Benutzer während einer interaktiven Aktivität Daten mit einer Workflow-Engine austauscht, löst er eine Sequenz von Service-Aufträgen am Workflow- und am Kommunikations-Server aus. Die Antwortzeit des Systems, die der Benutzer dabei erfährt, wird von den Wartezeiten aller Aufträge

dieser Sequenz beeinträchtigt. Die akzeptablen Wartezeiten der Server-Typen, die oft in solche interaktiven Sitzungen involviert sind, sind niedriger als die von Server-Typen, die nicht oder weniger in solche interaktiven Sitzungen involviert sind (zum Beispiel Applikations-Server). Es sollte daher möglich sein, verschiedene Schwellwerte für maximal tolerierbare Wartezeiten für verschiedene Server-Typen spezifizieren zu können.

Im nächsten Kapitel beschreiben wir die Umsetzung der hier vorgestellten Konzepte bei der Implementierung eines Konfigurationswerkzeuges für das WFMS Mentor-lite.

Kapitel 10

Implementierung eines Konfigurationswerkzeuges für Mentor-lite

In diesem Kapitel stellen wir die Umsetzung der in Kapitel 9 vorgestellten Konzepte bei der prototypischen Implementierung eines Konfigurationswerkzeuges für unser eigenes WFMS Mentor-lite vor. Die Implementierung wurde im Rahmen einer Diplomarbeit [GWS+00a, Won01] vorgenommen. Wir haben dem Konfigurationswerkzeug den Namen Goliat gegeben, was ein Akronym für den englischen Ausdruck *Goal-Driven Auto-Configuration Tool* ist. Abschnitt 10.1 faßt wichtige Design-Entscheidungen und Interna des Werkzeuges zusammen. Abschnitt 10.2 zeigt den Umgang mit dem Werkzeug anhand einer Beschreibung der Benutzerschnittstellen.

10.1 Implementierung

Goliat wurde in Java 2 Version 1.3 mit dem Java Developer Kit (JDK) implementiert. Aufgrund der Platformunabhängigkeit von Java ist Goliat auf allen Betriebsystemen ausführbar, für die es ein JDK oder eine Java Laufzeitumgebung (englisch: *Java Runtime Environment* (JRE)) gibt. Bei der Implementierung der graphischen Benutzeroberfläche kam das Swing-Toolset der Java Foundation Classes (JFC) zum Einsatz, was der Benutzeroberfläche auf allen Betriebsystemen ein einheitliches "Look and Feel" beschert. Die Implementierung von Goliat umfaßt ca. zehntausend Zeilen Code, wovon ca. zweitausend auf die graphischen Benutzeroberfläche entfallen. Die Lauffähigkeit von Goliat wurde erfolgreich getestet unter den Betriebsystemen Windows9x, WindowsNT, Windows2000, Linux 2.x und Solaris 2.6.

Goliat liest die Statecharts der Workflow-Spezifikationen wie die Workflow-Engine von Mentor-lite mittels eines Parsers ein und speichert sie intern in Form eines gerichteten Graphen. Diese Datenstruktur hat sich bereits bei der Implementierung eines Monitoring-Werkzeuges für Mentor-lite bewährt [Won99]. Die Übereintsimmung der Struktur der Statecharts mit der Struktur der korrespondierenden CTMC macht sich Goliat insofern zu nutze, daß dieser Graph für die interne Abbildung beider Konstrukte benutzt wird. Die Kalibrierung der CTMC der Workflow-Typen und der Systemkonfiguration ist in der momentanen Version von Goliat nur über die Benutzerschnittstelle möglich. Eine spätere

Erweiterung soll es ermöglichen, alle Parameter aus den Historien beendeter Workflow-Instanzen und dem Monitoring von Mentor-lite automatisch extrahieren zu können. Historiendaten können von Mentor-lite selektiv in einer Datenbank hinterlegt und von Goliat gezielt abgefragt werden [KS98, MWG+99b]. Ebenfalls noch offen ist die automatische Extraktion von Schleifen im Kontrollfluß von Workflow-Typen.

Bei Tests mit einer ersten Version von Goliat traten drei Probleme auf, deren Lösung sich auf die Funktionalität von Goliat auswirken können:

1. Die Anzahl der Summanden der unendlichen Summe bei der Berechnung der erwarteten Anzahl r_x^t von Service-Aufträgen, die eine Instanz des Workflow-Typs t an einem Server des Server-Typs x während seiner Ausführung initiiert (Gleichung 6.7), muß beschränkt werden, wodurch sich eine Unterschätzung der Gesamtlast an Service-Aufträgen pro Workflow-Instanz ergibt. Goliat berechnet die Anzahl der Summanden z_{max}^t mit Hilfe der Tabu-Wahrscheinlichkeiten automatisch in Abhängigkeit einer vom Benutzer spezifizierten Wahrscheinlichkeit $\bar{p}_{min}$, mit der eine Workflow-Instanz vom Workflow-Typ t nach z_{max}^t Schritten terminiert ist. z_{max}^t ist die kleinste natürliche Zahl für die gilt, daß

$$\sum_{z=0}^{z_{max}^t} \bar{p}_{0A}^t(z) \geq \bar{p}_{min} \tag{10.1}$$

2. Die Berechnung der Tabu-Wahrscheinlichkeiten eines Workflow-Typs erfolgt rekursiv über die Ausführungszustände des Workflow-Typs. Bei einer zu großen Tiefe des Rekursions-Baumes im Haputspeicher kam es bei den Berechnungen durch Goliat zu Speicherüberläufen. Um die Tiefe des Rekursionsbaumes zu verringern werden alle bereits berechneten Tabu-Wahrscheinlichkeiten $\bar{p}_{ij}^t(z)$ in einem Hash mit dem Schlüssel $i : j : z$ abgespeichert.

3. z_{max}^t wird sehr groß, wenn der Flußprozeß des Workflow-Typs t oft Schleifen durchlaufen muß, was insbesondere durch die Normalisierung der CTMC kein Sonderfall darstellt [Won01]. Bei sehr großem z_{max}^t kam es bei der Berechnung der Gleichung 6.7 ebenfalls zu Speicherüberläufen. Daher überschreitet Goliat in der aktuellen Version eine vom Benutzer in Abhängigkeit von der verwendeten Hardware spezifiziertes z_{max} nicht, auch wenn damit die Wahrscheinlichkeit $\bar{p}_{min}$ noch nicht erreicht worden ist. Auch diese Schranke kann zu einer Unterschätzung der Gesamtlast an Service-Aufträgen von Workflow-Instanzen führen.

10.2 Benutzerschnittstellen

Nach dem Start von Goliat erscheint das in Abbildung 10.1 dargestellte Kontrollfenster. Auf der rechten Seite des Fensters können unten die Parameter der Server-Typen und oben die Spezifikationen der Workflow-Typen geladen werden, die in den Kalkulationen berücksichtigt werden sollen. Die angegebenen Workflow-Spezifikationen, also die Statecharts, werden aus dem Workflow-Repository gelesen und können in dem in Abbilung 10.2 dargestellten Workflow-Editor parametrisiert werden.

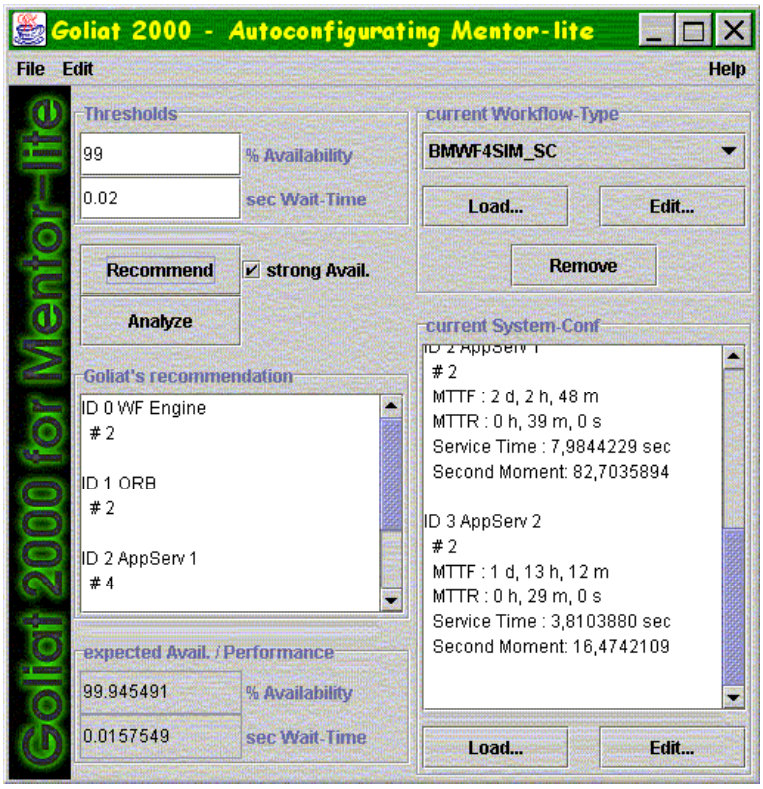


Abbildung 10.1: Kontrollfenster von Goliat

Der Großteil des Workflow-Editors wird durch die Darstellung der Spezifikation des sich gerade in Bearbeitung befindlichen Workflow-Typs eingenommen. Die Zustände des Statecharts werden durch Buttons und die Transitionen dazwischen als Pfeile repräsentiert. Die ECA-Regeln an den Transitionen können nach Bedarf ein oder ausgeblendet werden. Der Benutzer kann durch Anklicken eines geschachtelten Zustandes, der einen Subworkflow enthält, zu der Spezifikation des Subworkflow-Typs wechseln. Geschachtelte Zustände sind mit einem Pfeil/Größerzeichen markiert. Beim Anklicken eines Zustandes, der keinen Subworkflow repräsentiert, öffnet sich eine Dialogbox, in der der Benutzer folgende Parameter für den Zustand angeben kann:

1. den Lastvektor der durch den Zustand repräsentierten Aktivität und

2. die mittlere Verweildauer der Instanzen des Workflow-Typs in dem Zustand.

Zu jeder Transition eines Statecharts gibt es ein Textfeld, in dem die Schrittwahrscheinlichkeit anotiert wird, also die Wahrscheinlichkeit, mit der diese Transition beschritten wird, wenn eine Workflow-Instanz ihre Quelle durchläuft. Wenn ein Zustand nur eine ausgehende Transition besitzt, wird deren Schrittwahrscheinlichkeit automatisch auf 1 gesetzt. In allen anderen Fällen muß sie vom Benutzer angegeben werden. Schließlich muß der Benutzer die Ankunftsrate von Instanzen des Workflow-Typs am WFMS angeben.

Die Kalibrierung der Server-Typen des WFMS erfolgt in dem in Abbildung 10.3 dargestellten System-Editor von Goliat. Der Benutzer kann beliebig Server-Typen, die bei den Kalkulationen berücksichtigt werden, hinzufügen oder entfernen. Zu jedem Server-Typ müssen folgende Parameter angegeben werden:

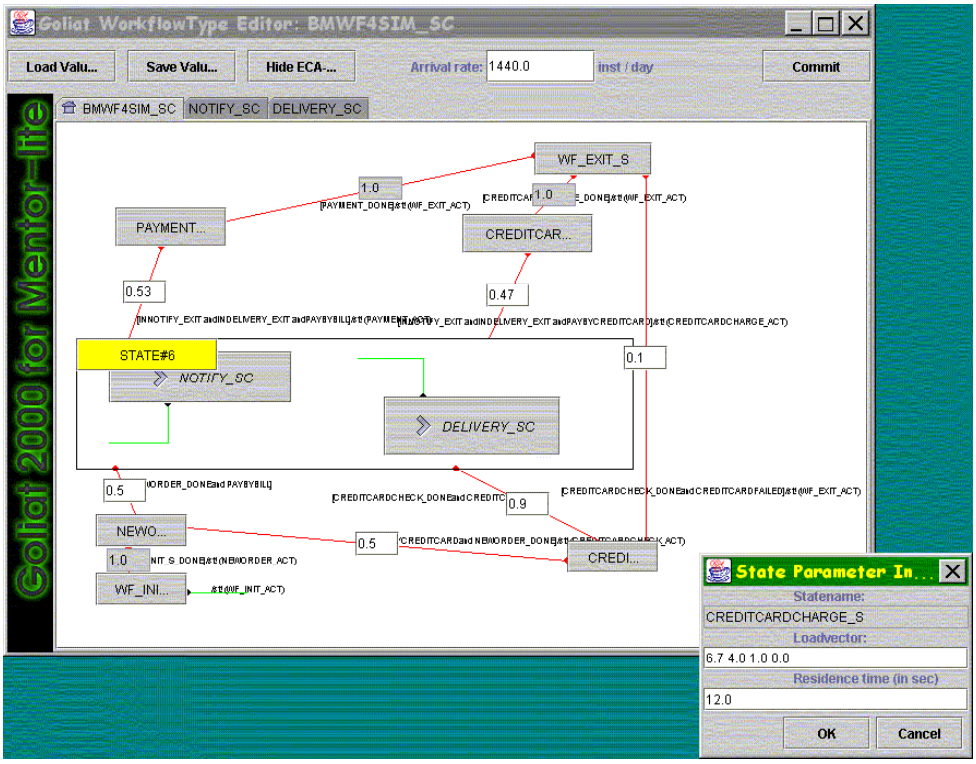


Abbildung 10.2: Workflow-Editor von Goliat

1. Der Name des Server-Typs.
2. Die Anzahl der Server-Replikate dieses Typs.
3. Die MTTF des Server-Typs.
4. Die MTTR des Server-Typs.
5. Der mittlere Bedienzeit von Servic-Aufträgen an dem Server-Typ.
6. Das zweite Moment der Bedienzeit-Verteilung von Service-Aufträgen an dem Server-Typ.

Die linke Seite des Kontrollfensters von Goliat (Abbildung 10.1) dient der Initiierung von Berechnungen unter Berücksichtigung der gegebenen Workflow- und Systemparameter. So kann der Benutzer Schwellwerte für die mindestens zu erreichende Verfügbarkeit und die maximal tolerierbare mittlere Wartezeit für Service-Aufträge im WFMS angeben. Daraufhin kann Goliat einen Vorschlag für die minimale Systemkonfiguration berechnen, die die angegebenen Ziele erreicht. Durch Auswahl der Check-Box neben der "Recommend"-Schaltfläche kann der Benutzer bestimmen, daß das angegebene Verfügbarkeitsziel für die starke Verfügbarkeit des Systems gelten soll. Alternativ kann der Benutzer eine Was-Wäre-Wenn-Analyse für eine von ihm spezifizierte Systemkonfiguration starten. Folgende Ergebnisse der Analyse werden in einem eigenen Fenster (Abbildung 10.4) dargestellt und können im HTML-Format gedruckt und gespeichert werden:

1. Die Erwartungswerte für die Verfügbarkeit und die starke Verfügbarkeit in Prozent sowie die erwartete Ausfallzeit pro Jahr.

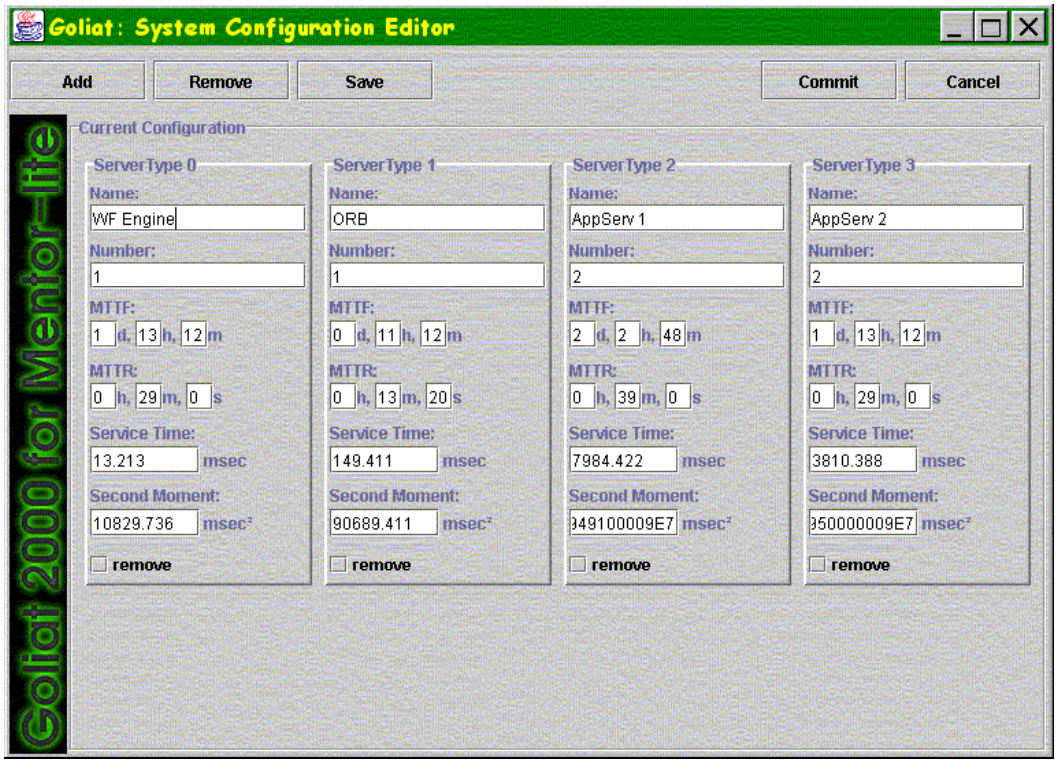


Abbildung 10.3: System-Editor von Goliat

2. Die erwartete Durchlaufzeit jedes Workflow-Typs basierend auf den gegebenen mittleren Zustandsverweildauern und Schrittwahrscheinlichkeiten.

3. Für jeden Server-Typ den maximal erreichbaren und den für die gegebene Workflow-Last benötigten Durchsatz an Service-Aufträgen.

4. Für jeden Workflow-Typ die erwartete Last in Service-Aufträgen pro Workflow-Instanz an den verschiedenen Server-Typen.

5. Die erwarteten mittleren Wartezeiten von Service-Aufträgen an den verschiedenen Server-Typen bei Nicht-Berücksichtigung von Server-Ausfällen.

6. Die erwarteten mittleren Wartezeiten von Service-Aufträgen an den verschiedenen Server-Typen bei Berücksichtigung transienter Server-Ausfälle.

Goliat kann mit Hilfe einer Parameterdatei konfiguriert werden. Ein Beispiel für eine solche Parameterdatei ist im Anhang A gegeben. Die Datei wird beim Start von Goliat eingelesen und kann im laufenden Betrieb manuell erneut eingelesen werden, wenn sie zwischenzeitlich geändert wurde. Die Parameterdatei von Goliat enthält folgende Einträge:

- Die Wahrscheinlichkeit $\bar{p}_{min}$, mit der die Instanzen jedes Workflow-Typs t die Anzahl von z^t_{max} Zustandsübergängen nicht überschreiten wird. z^t_{max} bestimmt so die Anzahl der Summanden der unendlichen Summe bei der Berechnung der erwarteten Anzahl von Service-Aufträgen, die ein Workflow an einem Server-Typ während seiner Ausführung initiiert (siehe Gleichung 6.7). Der Standardwert für $\bar{p}_{min}$ ist für alle t 99%.

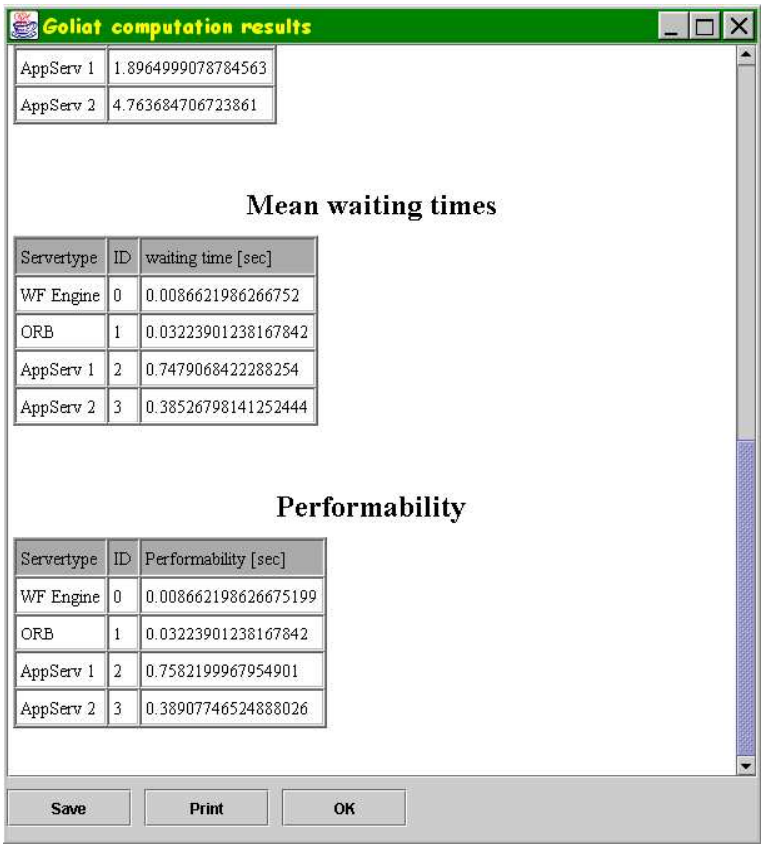


Abbildung 10.4: Ergebnisse der Was-Wäre-Wenn-Analyse

- Eine obere Schranke für die Anzahl der Summanden der unendlichen Summe bei der Berechnung der erwarteten Anzahl von Service-Aufträgen, die ein Workflow an einem Server-Typ während seiner Ausführung initiiert. Diese Schranke muß in Abhängigkeit von der verwendeten Hardware, insbesondere des vorhandenen Hauptspeichers gewählt werden, um Speicherüberläufe zu vermeiden [Won01]. Der Standardwert für die Schranke ist 50.000.
- Ein Skalierungsfaktor für die Darstellungsgröße der Workflows im Workflow-Editor.
- Das Verzeichnis, in dem die Workflow- und Server-Typ-Parameter abgespeichert und geladen werden sollen.

Teil IV

Evaluation

Kapitel 11

Meßaufbau

In diesem Kapitel beschreiben wir den Meßaufbau, mit dem wir Experimente zur Evaluierung der beschriebenen theoretischen Modelle durchgeführt haben. Es handelt sich dabei um eine Kombination aus Simulation und Messung am realen System.

Der Kern des Messaufbaus bildet das Prototyp-WFMS Mentor-lite (vgl. Abschnitt 4.1.5, sowie [MWG+99a, Wei00b, WGR+00]). Es bearbeitet die Workflows in gewohnter Weise, indem es die Workflow-Spezifikationen interpretiert. Auch das Logging und die Initiierung von Synchronisations-Nachrichten bei der verteilten Workflow-Ausführung werden von der Prototyp-Implementierung durchgeführt.
Neben dem Prototypen von Mentor-lite beinhaltet der Meßaufbau folgende Komponenten:

- Ein synthetischer Lastgenerator startet neue Workflow-Instanzen mit einer gegebenen Verteilung für die Zwischenankunftszeiten von Instanzen verschiedener Workflow-Typen. In unserem Falle handelt es sich dabei um Poisson-Verteilungen, deren Mittelwerte Parameter der Experimente sind.

- Ein Aktivitäten-Modul simuliert die Ausführung von Aktivitäten, indem es die Eingabedaten der jeweiligen Aktivität von der Workflow-Engine entgegennimmt, eine durch Parameter des Experimentes spezifizierte Last an Service-Aufträgen auf dem Applikations-Server initiiert oder im Falle von interaktiven Aktivitäten eine durch Parameter des Experimentes spezifizierte Zeit wartet und schließlich kontrollflußrelevante Ergebnisse an die Workflow-Engine zurückliefert.

- Ein Kommunikations-Modul bildet die Funktionalität des in Mentor-lite benutzten ORBs nach. Neben den Aufrufen zwischen den IDL-Schnittstellen von Workflow-Engines und Aktivitäten wird in Mentor-lite auch die transaktionsgeschützte Kommunikation zwischen Workflow-Engines bei verteilter Workflow-Ausführung mittels CORBA organisiert. Im Detail reden die Workflow-Engines dabei nicht direckt miteinander, sondern lesen und schreiben Nachrichten über OTS transaktionsgeschützt aus beziehungsweise in eigens entwickelte Reliable Message Queues. Das Kommunikations-Modul übernimmt während der Simulationen auch die Funktionalität dieser Message Queues als Teil der Kommunikations-Server.

- Ein Crash-Modul simuliert Ausfälle und Reperaturen von Servern. Arbeitsweise und Parametrisierung des Crash-Modules stellen wir in Abschnitt 11.2 vor.

- Ein Monitoring-Modul überwacht die Bedien- und Wartezeiten der Service-Aufträge an den Servern und protokolliert die Eintritts- und Austrittszeiten der Ausführungszustände sowie die Pfade, auf denen die einzelnen Workflow-Instanzen die Kontrollfluß-Spezifikation traversieren.

Zur Abbildung der Abarbeitung von Service-Aufträgen an den verschiedenen Servern haben wir eine Simulationsumgebung entwickelt, in die der Mentor-lite Prototyp und die anderen Komponenten eingebettet wurden. Die Implementierung der Simulationsumgebung basiert auf der Simulationsbibliothek CSIM [CSIM, Sch98], die allgemeine Primitive zur Beschreibung des zeitlichen Verhaltens von Prozessen und Bedienstationen enthält. Im folgenden werden wir detailliert die Simulation von Service-Aufträgen (Abschnitt 11.1) und Server-Ausfällen (Abschnitt 11.2) erläutern.

Im Rahmen einer Diplomarbeit [Sch01] wurde der gesamte Messaufbau derart in die Prototyp-Implementierung von Mentor-lite integriert, daß mit demselben Code sowohl Simulationen als auch die reale Ausführung von Workflow-Instanzen durchgeführt werden können. Bei einer Simulation werden virtuelle Ressourcen angesprochen. Dadurch erhält man bei Messungen automatisch reale Werte zur Kalibrierung der Meßläufe (zum Beispiel Anzahl und Dauer von Service-Aufträgen an Workflow-Server, Anzahl und Reihenfolge von Aufträgen an Kommunikations-Server etc.) und kann gleichzeitig im Simulationsmodus einfacher und effizienter reproduzierbare Leistungsbeobachtungen festhalten.

11.1 Abbildung von Service-Aufträgen

Während der Abarbeitung von Workflow-Instanzen fallen an verschiedenen Servern Service-Aufträge zur Bearbeitung an. Workflow-Engines initiieren Service-Aufträge an Workflow-Servern bei der Interpretation des Kontrollflusses und an Kommunikations-Servern beim Starten externer Anwendungsprogramme als automatische Aktivitäten oder beim Austausch von Synchronisationsnachrichten mit anderen Engines. Anwendungsprogramme belasten Applikations-Server mit Service-Aufträgen und liefern Resultate über Kommunikations-Server an Workflow-Engines.

In den Simulationen bilden jede Workflow-Engine und jede Aktivität einen eigenen Prozeß. Diese Prozesse initiieren Aufträge an Bedienstationen, die die einzelnen Server repräsentieren. Für jeden Server des WFMS gibt es eine Bedienstation mit einer Warteschlange für Service-Aufträge [CSIM]. Die Aufträge in jeder Warteschlange werden nach dem First-Come-First-Serve (FCFS) Scheduling-Verfahren abgearbeitet.

Die Generierung der Aufträge an die Bedienstationen der verschiedenen Server-Typen erfolgt folgendermaßen:

1. Workflow-Server: Eine Workflow-Engine kennt den Workflow-Server, auf dem sie läuft. Die Interpretation der Workflow-Spezifikation, also die Abarbeitung einer Workflow-Instanz, wird durch den Prototypen von Mentor-lite gesteuert. Zur Generierung von Service-Aufträgen an den Workflow-Server haben wir Messpunkte in den Code des Prototypen eingebaut, mit deren Hilfe wir identifizieren können, wann die Workflow-Engine Arbeit verrichtet. An solchen Checkpunkten messen wir die CPU-Zeit bis zum Erreichen des nächsten und generieren dort einen Auftrag

mit der gleichen Bedienzeit an die simulierte Bedienstation, die den entsprechenden Workflow-Server repräsentiert.

2. Kommunikations-Server: Der Prototyp von Mentor-lite kommuniziert mit Anwendungsprogrammen und anderen Workflow-Engines mittels CORBA. In den Simulationen fangen wir solche Aufrufe an die CORBA-Server ab und generieren stattdessen Aufträge an simulierte Bedienstationen, die verfügbare Kommunikations-Server repräsentieren. Ebenso verfahren wir mit Aufrufen von Anwendungsprogrammen (in unserem Fall vom Aktivitäten-Modul der Simulationsumgebung), die Daten über die IDL-Schnittstelle der Workflow-Engine liefern oder abrufen. Für jede Art von CORBA-Aufrufen, die während der Abarbeitung eines Workflows mit dem Prototypen von Mentor-lite auftreten kann, kann ein Mittelwert für die Dauer solcher Aufträge parametrisiert werden, der während der Simulation als Mittelwert für eine Exponentialverteilung für die Bedienzeit der Aufträge an den Bedienstationen dient.

3. Applikations-Server: Aufträge an die die Applikations-Server repräsentierenden Bedienstationen werden ausschließlich vom Aktivitäten-Modul initiiert. Jeder Aktivität eines Workflow-Typs ist ein Typ von Applikations-Servern zugeordnet. Das Aktivitäten-Modul generiert nach dem Start einer Aktivität, der durch eine Workflow-Engine veranlaßt wurde, einen Auftrag an eine entsprechende verfügbare Bedienstation. Die Größe des Auftrages wird so festgelegt, daß die Aufträge einer Aktivität über alle Workflow-Instanzen, bei der die Aktivität aufgerufen wird, mit einem zu parametrisierenden Mittelwert exponentialverteilt sind.

Die Tatsache, daß wir bei den Bedienzeiten an den Kommunikations-Servern und den Applikations-Servern auf Exponentialverteilungen zurückgreifen, hat keine Auswirkungen auf die Genauigkeit der Vorhersagen der analytischen Modelle, mit denen wir die gemessenenen Werte vergleichen werden. Das M/G/1-Modell, das wir zur Modellierung der Server benutzen, ist in der Lage, jede beliebige Verteilung für die Bedienzeit der Service-Aufträge zu berücksichtigen. Die Parameter, mit denen das Modell kalibriert wird, sind die ersten beiden Momente der Verteilung, die wir von dem Monitoring-Modul der Meßumgebung beziehen.

11.2 Abbildung von Server-Ausfällen

Die CSIM-Bibliothek bietet die Möglichkeit, Prozesse zu priorisieren und deren Aufträge an den Bedienstationen bevorzugt zu behandeln. Wir simulieren Server-Ausfälle durch einen hoch priorisierten Service-Auftrag des Crash-Modules an den entsprechenden Server, der ausfallen soll. Die Bedienzeit dieses Auftrages ist die Zeit, die vergeht, bis der Server wieder verfügbar ist.

Das Scheduling-Verfahren, das wir dafür an den Bedienstationen der Simulationen verwenden, wird in der CSIM-Terminologie als "preempt resume" [CSIM] bezeichnet. Dabei werden die Aufträge niedriger priorisierter Prozesse (in unserem Fall Service-Aufträge laufender Workflow-Instanzen wie in Abschnitt 11.1 beschrieben) eingefroren, wenn Aufträge höher priorisierter Prozesse (in unserem Fall die Aufträge des Crash-Modules) an der Warteschlange eintreffen, und erst nach deren vollständiger Abarbeitung fortgesetzt. Aufträge mit gleicher Priorität werden auch hier nach dem First-Come-First-Serve (FCFS)

Scheduling-Verfahren abgearbeitet.

Das Crash-Modul erhält für jeden Server-Typ Parameter für zwei Arten von Nicht-Verfügbarkeit:

1. Für die Zeit zwischen zwei aufeinanderfolgender Soft- oder Hard-Crashes nehmen wir eine Exponentialvereilung an. Die Zeit zur Behebung eines solchen Fehlers bis zur Wiederverfügbarkeit des Servers sei ebenfalls exponentialverteilt. Beide Verteilungen werden durch ihren Mittelwert parametrisiert.

2. Als weiteren Grund für Ausfälle nehmen wir Service-Shutdowns von Servern in die Simulationen auf. Die Zeit, die zwischen zwei aufeinanderfolgenden Service-Shutdowns eines Server-Typs liegt, ist kontinuierlich gleichverteilt zwischen zwei zu parametrisierenden Grenzen. Die Dauer aller Service-Shutdowns für einen Server-Typ nehmen wir als konstant an.

Durch diese realitätsnahe Unterscheidung von Ursachen für die Nicht-Verfügbarkeit von Servern erhalten wir eine hohe Varianz in den Verteilungen für die Zeit zwischen zwei aufeinanderfolgenden Ausfällen von Servern eines Server-Typs.

Nachdem ein Server ausgefallen ist, werden alle Service-Aufträge, die zum Zeitpunkt des Ausfalles in seiner Warteschlange stehen, auf die restlichen, noch verfügbaren Server umgeschichtet. Im Falle von Workflow-Servern migrieren die gesamten Workflow-Instanzen, die dem ausgefallenen Server zugeteilt waren. Dieses Vorgehen wird gerechtfertigt durch die Annahme, daß die Warteschlangen von den Prozessen getrennt verwaltet werden. Diese Methodik wird zum Beispiel auch bei *Web Server Farms* angewendet. In unserer Simulationsumgebung werden diese Migrationen von dem Crash-Modul vorgenommen, nachdem es einen Ausfall initiiert hat.

Kapitel 12

Meßreihen und Parametrisierung

In diesem Kapitel diskutieren wir die Wahl der Parameter für unsere Experimente zur Evaluierung der mathematischen Modelle. Abschnitt 12.1 erörtert die Meßreihen, die wir mit der Simulationsumgebung durchgeführt haben, und welche Parameter wir dabei variiert haben. In Abschnitt 12.2 stellen wir die Parameter vor, deren Werte für alle Experimente identisch gehalten wurden.

12.1 Meßreihen

Wir haben zwei Freiheitsgrade betrachtet, die wir kreuzweise zu vier Meßreihen kombiniert haben:

1. Zentrale gegenüber verteilter Workflow-Ausführung

   Zentrale und verteilte Workflow-Ausführungen stellen unterschiedliche Anforderungen an das WFMS, die sich auch in der Konfiguration niederschlagen. So resultiert die verteilte Workflow-Ausführung in einer höheren Last an den Kommunikations-Servern, da die Synchronisation der an einer Workflow-Instanz beteiligten Workflow-Engines hinzukommt. Darüber hinaus ordnen wir jedem Subworkflow eines Workflow-Typs einen eigenen Server-Typ zu. Dies deckt sich mit der Realität, da zum Beispiel Subworkflows einzelner Abteilungen auch jeweils von den Workflow-Engines dieser Abteilungen gesteuert werden. Wir werden daher sowohl Meßreihen mit zentraler Workflow-Ausführung (ein Typ von Workflow-Servern steuert den gesamten Workflow) als auch Meßreihen mit verteilter Workflow-Ausführung (zwei Typen von Workflow-Servern steuern jeweils einen Teil des Workflows) vorstellen.

2. Stabiles WFMS ohne Server-Ausfälle gegenüber Berücksichtigung von transienten Server-Ausfällen

   Zur Evaluation des Leistungsmodelles aus Kapitel 6 müssen wir von einem stabilen System ohne Server-Ausfälle ausgehen. Demgegenüber werden Server-Ausfälle im Performability-Modell aus Kapitel 8 explizit berücksichtigt. Wir werden daher sowohl Meßreihen ohne als auch Meßreihen mit simulierten Server-Ausfällen durchführen, indem wir das Crash-Modul der Simulationsumgebung deaktivieren beziehungsweise aktivieren.

Wir messen die mittleren Wartezeiten der Service-Aufträge an den einzelnen Server-Typen in Abhängigkeit von der Ankunftsrate von Workflow-Instanzen bei einer festen Systemkonfiguration. Um aussagekräftige Ergebnisse präsentieren zu können, variieren wir die

Systemkonfiguration des WFMS zwischen den Meßreihen. Welche Konfiguration im Einzelfall benutzt wird, beschreiben wir bei der Diskussion der Meßergebnisse in Kapitel 13.

Für alle Meßreihen stellen wir die in den Simulationen gemessenen Ergebnisse denen gegnüber, die Goliat unter gleicher Parametrisierung aus den theoretischen Modellen ermittelt. Dadurch können wir die Genauigkeit der Vorhersagen evaluieren, die wir mit den Modellen berechnen. Bei den Berechnungen benutzte Goliat die Einstellungen aus der Parameterdatei aus Anhang A.

12.2 Feste Parameter

In diesem Abschnitt stellen wir die Parameter vor, deren Werte für alle Experimente fix sind. Dabei beschreiben wir zunächst die Parametrisierung des Workflow-Typs, der bei den Experimenten benutzt wurde. Es handelt sich dabei um den Benchmark-Workflow, an dem bereits die mathematischen Modelle illustriert wurden. Danach beschreiben wir die Parametrisierung der Server-Typen des simulierten WFMS bezüglich ihrer Bedienzeiten und ihres Ausfallverhaltens.

12.2.1 Parametrisierung des Benchmark-Workflows

Der Workflow-Typ, den wir zur Evaluation der mathematischen Modelle heranziehen, ist der Benchmark-Workflow aus Abschnitt 3.2. Die variablen Parameter, die der Benchmark zur Verfügung stellt, setzen wir wie folgt fest:

1. Aktivitätendauer:

   Tabelle 12.1 zeigt die mittlere Dauer der Aktivitäten des Benchmark-Workflows in den Simulationen. Dabei sind *Bestellung_ACT* und *Bezahlung_ACT* interaktive Aktivitäten, also Aktivitäten, die von menschlichen Benutzern auf ihren Arbeitsplatzrechnern ausgeführt werden und somit die Applikations-Server des Systems nicht belasten. Die übrigen Aktivitäten werden vollautomatisch auf den Applikations-Servern ausgeführt. Während der Messungen wird die Simulation der vollautomatischen Aktivitäten von dem Aktivitäten-Modul der Meßumgebung übernommen. In die mittlere Dauer der interaktiven Aktivitäten ist die Zeit bis zum tatsächlichen Beginn der Bearbeitung durch einen Benutzer (also die Zeit, in der die Aktivität in der Worklistlist des Benutzers steht) eingerechnet.

2. Schleifendurchläufe:

   Die Anzahl der Schleifendurchläufe im Subworkflow-Typ *Versand* des Benchmark-Workflows ist identisch zu der Anzahl der verschiedenen Produkte, die in dem aktuellen Vorgang (also der aktuellen Workflow-Instanz) bestellt wurden. In den Simulationen folgt diese Anzahl einer diskreten Gleichverteilung über die Werte 1 bis 3.

3. Häufigkeit von Bezahlung per Rechnung beziehungsweise Kreditkarte:

   Die Wahl der Zahlungsmodalität bewirkt im Benchmark-Workflow an zwei Stellen eine Aufspaltung des Kontrollflusses, die die Lebensdauer von Workflow-Instanzen

Aktivität	Art	mittlere Dauer
Bestellung_ACT	interaktiv	20 min
KreditkartenPrüfung_ACT	automatisch	2 sec
E-Mail_ACT	automatisch	700 msec
LagerErmittlung_ACT	automatisch	500 msec
LagerInstruierung_ACT	automatisch	700 msec
KreditkartenBelastung_ACT	automatisch	3 sec
Bezahlung_ACT	interaktiv	45 min

Tabelle 12.1: Mittlere Dauer der Aktivitäten

und die Last auf den Applikations-Servern beeinflussen. Im Falle einer Kreditkartenzahlung werden zwei kurz dauernde automatische Aktivitäten durchlaufen, während im Falle von Bezahlung per Rechnung statt dessen eine lange dauernde interaktive Aktivität ausgeführt wird. In den Simulationen wird die Zahlung per Rechnung beziehungsweise per Kreditkarte bei allen Workflow-Instanzen mit einer Häufigkeit von jeweils 50% gewählt.

4. Häufigkeit von Kreditkartenfehlern:

   Das Ergebnis der Aktivität *KreditkartenPrüfung_ACT* des Benchmark-Workflows ist entweder positiv, wenn die Kreditkarte gültig und gedeckt ist, oder negativ. Abhängig von diesem Ergebnis wird der Kontrollfluß des Workflows gesplittet. Im Falle eines negativen Ausgangs der Kreditkartenprüfung wird die Workflow-Instanz unmittelbar beendet. Wir erhalten so die Möglichkeit, auch extrem kurzlebige Workflow-Instanzen in die Evaluation miteinzubeziehen. In den Simulationen erfolgt das Auftreten eines Kreditkartenfehlers bei allen Workflow-Instanzen, bei denen eine Bezahlung per Kreditkarte angenommen wird, mit einer Häufigkeit von 10%.

Für die Meßreihen mit verteilter Workflow-Ausführung partitionieren wir den Benchmark-Workflow in zwei Teile. Dabei bildet der Subworkflow-Typ *Versand* die eine und der Rest den zweiten Teil. Für jeden Teil gibt es einen eigenen Typ von Workflow-Servern und einen eigenen Typ von Applikations-Servern. Im Falle der zentralen Workflow-Ausführung bleibt die Zuordnung der automatischen Aktivitäten auf die Applikations-Server-Typen erhalten.

12.2.2 Bedienzeiten und Ausfälle der Server-Typen

In diesem Abschnitt beschreiben wir, wie wir die Eigenschaften der Server-Typen des WFMS simulieren. Wir stellen dazu zunächst vor, wie wir die Bedienzeiten an den einzelnen Server-Typen festlegen. Die Strategien, mit denen wir die Werte für die Bedienzeiten von Aufträgen festlegen, sind abhängig davon, ob es sich um Service-Aufträge an einen Workflow-Server, an einen Kommunikations-Server oder an einen Applikations-Server handelt. Schließlich geben wir die Werte der Parameter an, die das Ausfallverhalten der unterschiedlichen Server-Typen beschreiben.

Die Basis der in den Experimenten gewählten Systemkonfigurationen wird von 5 Server-Typen gebildet werden, wobei die Systemkonfiguration selbst, also der Replikationsgrad

der einzelnen Server-Typen, von Meßreihe zu Meßreihe variiert und daher in den entsprechenden Abschnitten von Kapitel 13 vorgestellt wird. Bei den fünf Server-Typen handelt sich um

- 2 verschiedene Typen von Workflow-Servern (im folgenden WFS1 und WFS2),
- 1 Typ von Kommunikations-Servern (im folgenden KS) und
- 2 verschiedene Typen von Applikations-Servern (im folgenden AS1 und AS2).

Den zweiten Typ von Workflow-Servern benötigen wir nur für die Experimente mit verteilter Workflow-Ausführung. Dabei wird die Steuerung des Subworkflow-Typs *Versand* auf den Server-Typ WFS2 ausgelagert. Im Falle der zentralen Workflow-Ausführung wird die gesamte Ausführung der Workflow-Instanzen von dem Server-Typ WFS1 gesteuert. Analog werden die Aktivitäten LagerErmittlung_ACT und LagerInstruierung_ACT von dem Server-Typ AS2 und die restlichen Aktivitäten von dem Server-Typ AS1 ausgeführt.

Bedienzeit für Service-Aufträge an die Workflow-Server

Die Bedienzeit für Aufträge an einen Workflow-Server ist nicht als externer Parameter eines Simulationslaufes zu spezifizieren. Jeder Auftrag an einen Workflow-Server wird von dem in die Simulationsumgebung eingebetteten Prototypen von Mentor-lite initiiert. Die Bedienzeit eines solchen Auftrages stimmt mit der tatsächlichen CPU-Zeit überein, die der Service-Auftrag bei der realen Workflow-Ausführung beanspruchen würde.

Bedienzeit für Service-Aufträge an die Kommunikations-Server

Die mittlere Dauer der Aufträge an die Kommunikations-Server orientiert sich an empirischen Erfahrungen mit der CORBA-Implementation Orbix der Firma IONA [IONA], die wir mit Mentor-lite sammeln konnten [Sch01]. Tabelle 12.2 beschreibt die Service-Aufträge, die während der (verteilten) Ausführung eines Workflows mit Mentor-lite vorkommen, und gibt ihre mittlere Bedienzeit wieder.

Bedienzeit für Service-Aufträge an die Applikations-Server

Pro vollautomatischer Aktivität gibt es genau einen Auftrag an einen Applikations-Server. Die mittlere Dauer der Aufträge einer Aktivität entspricht der mittleren Durchlaufzeit der Aktivität (vergleiche Tabelle 12.1). Interaktive Aktivitäten werden typischerweise von menschlichen Benutzern auf ihren Arbeitsplatzrechnern ausgeführt und belasten die Applikations-Server nicht.

Ausfälle und Reparaturen von Servern

Die Palette an Gründen für die Nicht-Verfügbarkeit von Rechnersystemen ist groß. Sie reicht von leicht zu behebenden Soft-Crashes wie kurzzeitigen Stromausfällen bis hin zu schwerwiegenden Hard-Crashes wie Ausfällen von Festplatten oder kompletter Rechner. Auch Angriffe von außen auf Rechner oder Systemkomponenten, die über das Internet zugänglich sind, führen oft zu massiven Leistungsverlusten, die das System für einen Benutzer praktisch unbrauchbar machen (englisch: *denial of service attacks*). Schließlich fallen auch geplante Zeiten der Nicht-Verfügbarkeit von Systemkomponenten an, wie zum

Service-Auftrag	mittlere Bedienzeit
Verbindungsaufbau zum Kommunikations-Server	100 msec
Verbindungsaufbau zu einer automatischen Aktivität auf dem Applikations-Server durch die Workflow-Engine	500 msec
Starten von automatischen Aktivitäten auf dem Applikations-Server durch die Workflow-Engine	75 msec
Variablenübergabe an die Workflow-Engine via IDL-Schnittstelle	75 msec
Löschen des Regestry-Eintrages beim Kommunikations-Server	50 msec
Erzeugen einer Message Queue	200 msec
Beginn einer Transaktion in OTS	100 msec
transaktionsgeschütztes Schreiben einer Nachricht in eine Message Queue	150 msec
transaktionsgeschütztes Lesen einer Nachricht aus einer Message Queue	150 msec
Rollback einer Transaktion durch OTS	100 msec
Commit einer Transaktion in OTS	100 msec

Tabelle 12.2: Mittlere Bedienzeiten der Service-Aufträge an Kommunikations-Server

Beispiel Service-Shutdowns zum Aufspielen neuer Versionen von Software-Komponenten.

Gerade bei häufig vorkommenden Ausfällen von Komponenten eines WFMS ist eine genaue Abschätzung der Leistung, die das System im langfristigen Mittel zu leisten im Stande ist, besonders schwierig, aber auch besonders wichtig für die Beurteilung der Qualität einer gewählten Systemkonfiguration. Um die Genauigkeit der Vorhersagen unserer Modelle zu evaluieren, werden wir alle Experimente als überspitzte Streßtests, also mit sehr häufigen und langen Ausfällen parametrisieren.

Wir wählen die MTTF und MTTR von Soft- und Hard-Crashes für ein WFMS in extrem ungünstiger Situation. Die Kommunikations-Server stellen wir dabei besonders schlecht dar, da sie stark belastet und oft Opfer von Angriffen von außen sind. Tabelle 12.3 zeigt die Werte für die MTTF und MTTR, mit denen wir das Crash-Modul für die an den Experimenten beteiligten Server-Typen parametrisieren. Zudem simuliert das Crash-Modul für jeden Server-Typ Service-Shutdowns einer Dauer von 60 Minuten. Die Zeit zwischen zwei Service-Shutdowns eines Server-Typs ist dabei kontinuierlich gleichverteilt von 6 bis 8 Tagen, also im Mittel ein Service-Shutdown pro Woche und Server-Typ.

Server-Typ	MTTF	MTTR
WFS1	2 d	15 min
WFS2	2 d	20 min
KS1	6 h	10 min
APS1	1 d	15 min
APS2	12 h	15 min

Tabelle 12.3: MTTF und MTTR der Soft- und Hard-Crashes an den Server-Typen

Kapitel 13

Diskussion der Meßergebnisse

In diesem Kapitel diskutieren wir die Meßergebnisse der Simulationsläufe. In Abschnitt 13.1 zeigen wir die Genauigkeit des Lastmodelles aus Abschnitt 6.3, indem wir die gemessene Anzahl von Service-Aufträgen pro Workflow-Instanz der von Goliat erwarteten Anzahl gegenüberstellen. In Abschnitt 13.2 diskutieren wir die Ergebnisse der Meßläufe, in denen wir das Crash-Modul ausgeschaltet liesen. Die Evaluierung des Performability-Modelles durch Messungen mit eingeschaltetem Crash-Modul folgt in Abschnitt 13.3. In Abschnitt 13.4 diskutieren wir die Genauigkeit des Optimierungsmoduls von Goliat anhand von Messungen, in denen wir eine von Goliat vorgeschlagene Systemkonfiguration benutzten.

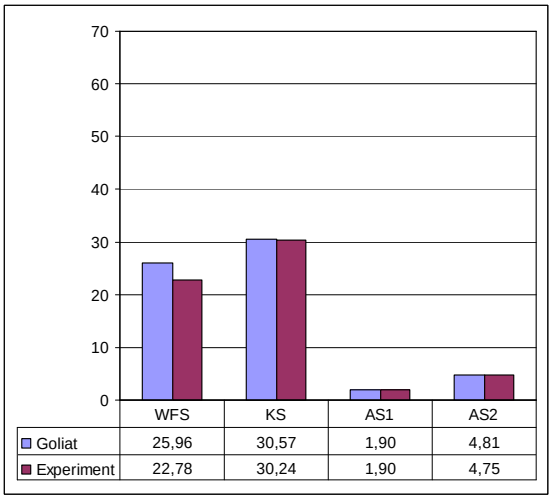
Für alle Meßreihen stellen wir die in den Simulationen gemessenen Ergebnisse denen gegnüber, die Goliat unter gleicher Kalibrierung aus den theoretischen Modellen ermittelt. Dadurch können wir die Genauigkeit der Vorhersagen evaluieren, die wir mit den Modellen berechnen. Bei den Berechnungen benutzte Goliat die Einstellungen aus der Parameterdatei aus Anhang A.

13.1 Last

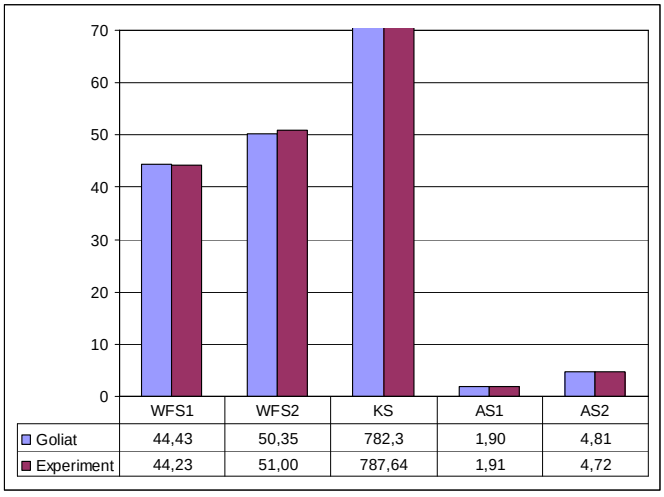
In diesem Abschnitt diskutieren wir die Genauigkeit der Abschätzung der Anzahl der Service-Aufträge, die eine Workflow-Instanz auf den verschiedenen Server-Typen initiiert.

Abbildung 13.1 stellt für jeden Server-Typ die aus dem Lastmodell aus Abschnitt 6.3 resultierenden Erwartungen den beobachteten Mittelwerten aus allen Meßläufen gegenüber, die wir mit dem Benchmark-Workflow durchgeführt haben. Abbildung 13.1(a) zeigt diese Gegenüberstellung für die Meßläufe mit zentraler Workflow-Ausführung und Abbildung 13.1(b) für die Meßläufe mit verteilter Workflow-Ausführung. Da bei der zentralen Workflow-Ausführung die Workflow-Instanzen komplett auf einem Typ von Workflow-Servern abgearbeitet werden, ist in Abbildung 13.1(a) nur ein Typ von Workflow-Servern (bezeichnet mit *WFS*) dargestellt.

Sowohl bei zentraler als auch bei verteilter Workflow-Ausführung nähern die analytischen Vorhersagen die gemessenen Werte sehr gut an. Die erhöhte Anzahl an Service-Aufträgen an den Workflow-Servern und den Kommunikations-Servern bei verteilter Workflow-Ausführung resultieren daraus, daß die Kommunikationsmanager von Mentor-lite ihre Reliable Message Queues pollen, wenn sie Synchronisationsnachrichten von anderen Workflow-



(a) zentrale Workflow-Ausführung



(b) verteilte Workflow-Ausführung

Abbildung 13.1: Service-Aufträge pro Workflow-Instanz

Engeines erwarten. Sicherlich gibt es hier Potential, den Mentor-lite-Prototypen zu verbessern, was aber in dieser Arbeit nicht weiter verfolgt werden soll. Eine Konsequenz, die sich aus diesem Phänomen ergibt, ist, daß die Anzahl der Replikate des Kommunikations-Server-Typs in den Messungen mit verteilter Workflow-Ausführung größer gewählt werden muß als in den Messungen mit zentraler Workflow-Ausführung, um den benötigten Durchsatz zu erhalten.

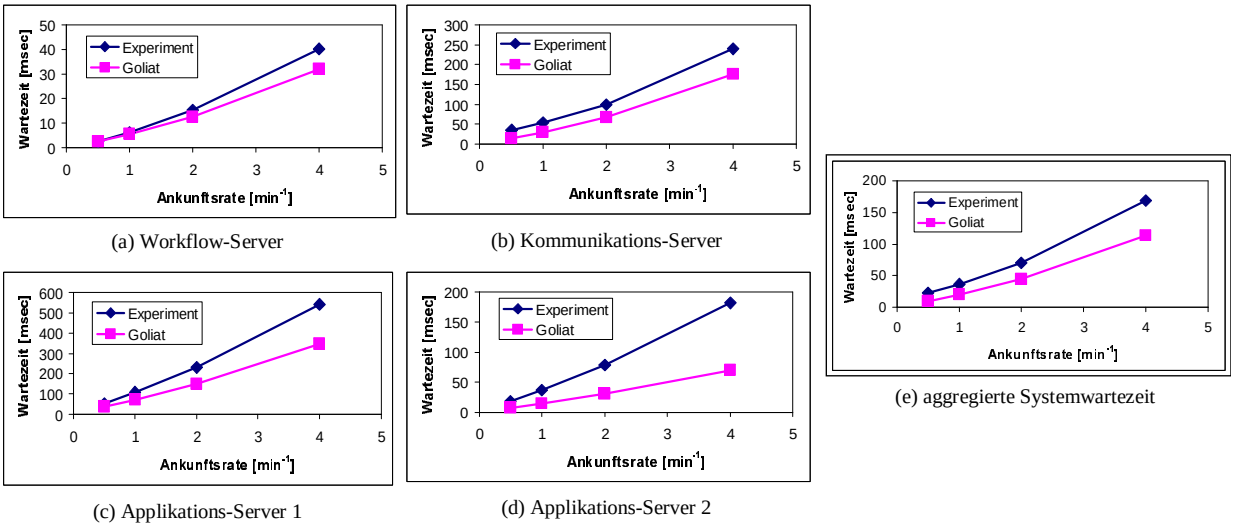
13.2 Leistung

Die in diesem Abschnitt gezeigten Experimente dienen der Evaluierung des Leistungsmodelles aus Kapitel 6. Das Crash-Modul der Meßumgebung war dabei ausgeschaltet.

Abbildung 13.2 zeigt die Ergebnisse der Meßläufe mit zentraler Workflow-Ausführung. Die Systemkonfiguration wurde so gewählt, daß von jedem Server-Typ ein Server vorhanden war. Neben den gemessenen und analytisch vorhergesagten Wartezeiten an den verschiedenen Server-Typen zeigt Abbildung 13.2(e) die aggregierte Systemwartezeiten, also die Summe der nach dem Gesamtauftreten von Service-Aufträgen gewichteten Wartezeiten der Server-Typen.

Die Diagramme zeigen, daß Goliat die Wartezeiten an allen Server-Typen unterschätzt. Diese Unterschätzung ist im Bezug auf die aggregierten Systemwartezeiten jedoch nicht kritisch und bleibt in einem akzeptablen Bereich nahe an den experimentell ermittelten Werten.

Abbildung 13.3 zeigt die Ergebnisse der Meßläufe mit verteilter Workflow-Ausführung. Wegen der in Abschnitt 13.1 angesprochenen höheren Belastung des Kommunikations-Server-Typs haben wir die Anzahl der Server dieses Server-Typs auf fünf erhöht. Die Anzahl der Server des zusätzlichen Typs von Workflow-Servern, der in unserem Szenario für die Steuerung des Subworkflow-Typs *Versand* zuständig ist, war ebenso wie bei den anderen eins.

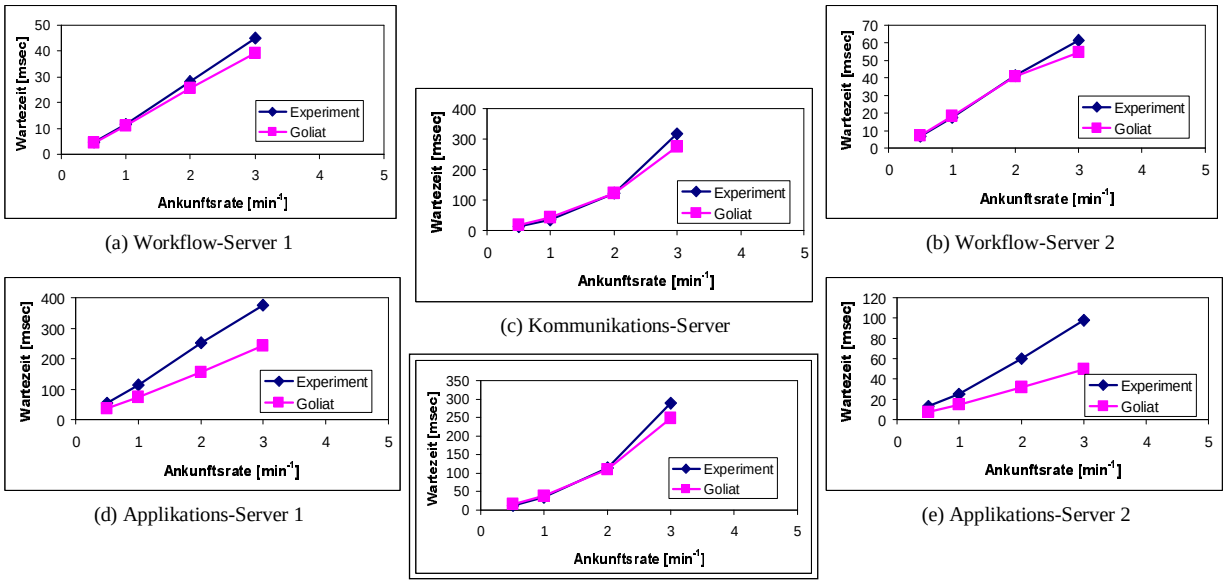


(a) Workflow-Server (b) Kommunikations-Server

(e) aggregierte Systemwartezeit

(c) Applikations-Server 1 (d) Applikations-Server 2

Abbildung 13.2: Leistung des stabilen WFMS bei zentraler Workflow-Ausführung



(a) Workflow-Server 1 (b) Workflow-Server 2

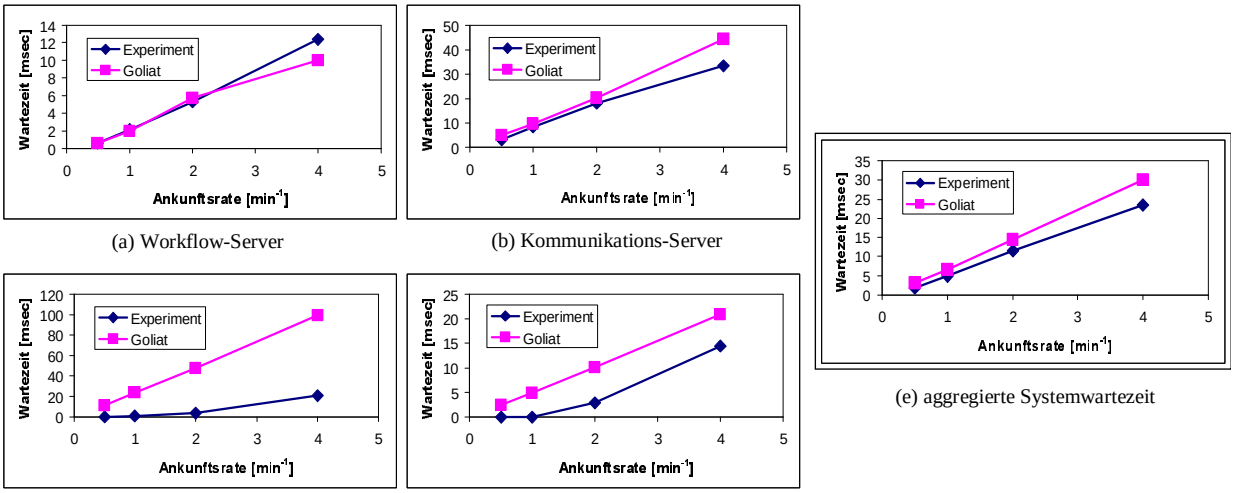
(c) Kommunikations-Server

(d) Applikations-Server 1 (e) Applikations-Server 2

(f) aggregierte Systemwartezeit

Abbildung 13.3: Leistung des stabilen WFMS bei verteilter Workflow-Ausführung

Da sich die Last an den Applikations-Servern gegenüber den Experimenten mit zentraler Workflow-Ausführung nicht verändert hat, sind sowohl die gemessenen als auch die vorhergesagten Werte für die Wartezeiten an diesen beiden Server-Typen bei beiden Meßläufen gleich. Die analytischen Vorhersagen von Goliat für die Wartezeiten an den Workflow-Servern sind so genau, wie sie es bereits bei den Messungen mit zentraler Workflow-Ausführung waren. Im Falle der Kommunikations-Server werden die Vorhersagen genauer, wobei Goliat die Wartezeiten bei kleinen Ankunftsraten etwas überschätzt. Diese Überschätzung liegt daran, daß die Verteilung der Service-Aufträge zwischen den fünf Servern dieses Server-Typs nach dem Round-Robin-Verfahren geschieht. Dadurch wird die anfallende Last besser balanciert als in dem analytischen Modell angenommen. Das analytische Modell geht davon aus, daß alle Service-Aufträge an den Server-Typ in fünf gleiche Poisson-Ankunftsprozesse mit einer Ankunftsrate von einem fünftel der Gesamtankunftsrate aufgeteilt werden, was signifikante Lastschwankungen zur Folge haben



(a) Workflow-Server

(b) Kommunikations-Server

(e) aggregierte Systemwartezeit

(c) Applikations-Server 1

(d) Applikations-Server 2

Abbildung 13.4: Performability bei zentraler Workflow-Ausführung

kann. In den Experimenten zu Abbildung 13.3 folgt die Zeit zwischen zwei aufeinanderfolgenden Ankünften von Service-Aufträgen an den Kommunikations-Servern nicht mehr einer Exponentialverteilung, sondern einer Verteilung mit geringerer Varianz. Dieser Effekt zeigt sich auch in den folgenden Meßreihen zum Performability-Modell.

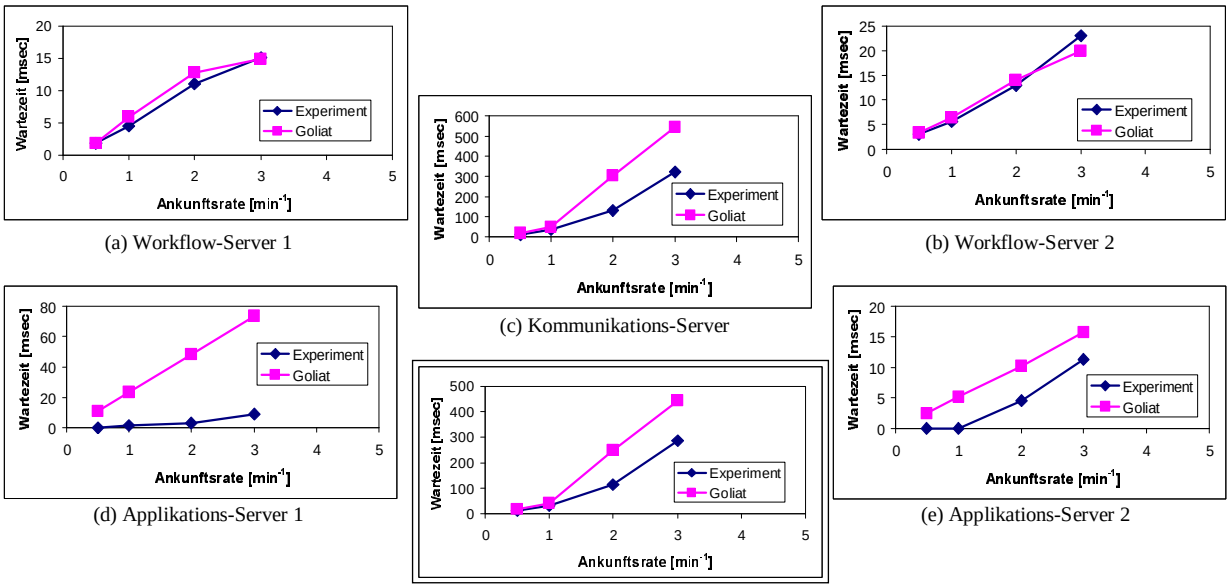
13.3 Performability

In diesem Abschnitt diskutieren wir die Experimente, die mit eingeschaltetem Crash-Modul durchgeführt wurden. Die Experimente dienen der Evaluation des Performability-Modelles aus Kapitel 8.

Abbildung 13.4 zeigt die Ergebnisse der Messungen mit eingeschaltetem Crash-Modul und zentraler Workflow-Ausführung. Es gab drei Server von jedem Server-Typ. Mit Ausnahme des Workflow-Server-Typs überschätzt Goliat die mittleren Wartezeiten von Service-Aufträgen an allen Server-Typen. Der Grund ist die bereits diskutierte, gegenüber der Annahmen der analytischen Modelle bessere Lastbalancierung des Round-Robin-Zuweisungsverfahrens. Die mittleren Wartezeiten an den Workflow-Servern werden von Goliat leicht unterschätzt, da hier alle Service-Aufträge einer Workflow-Instanz an denselben Server gerichtet werden und nicht wie bei den anderen Server-Typen verteilt werden. Die Unterschätzung wächst mit steigender Last, da das Performability-Modell transiente Warteschlangeneffekte bei Übergängen zwischen den Systemzuständen vernachlässigt:

1. Bei einem Ausfall eines Servers werden die Workflow-Instnzen, die ihm zugeordnet sind, auf die restlichen noch verfügbaren Server umverteilt. Die Service-Aufträge, die bereits auf ihre Abarbeitung gewartet haben, werden dabei in der Warteschlange des neuen Servers hinten angehängt und müssen daher eine längere Wartezeit hinnehmen als erwartet.

2. Bei der Wiederinbetriebnahme eines Servers bleibt die Zuordnung der aktiven Workflow-Instanzen zu den anderen Servern erhalten. Die erhöhte Last, die die Server, die nicht ausgefallen waren, aufgrund des Ausfalles bewältigen mußten, bleibt daher auch nach der Wiederinbetriebnahme bestehen und verringert sich nur langsam, bis



(a) Workflow-Server 1

(b) Workflow-Server 2

(c) Kommunikations-Server

(d) Applikations-Server 1

(e) Applikations-Server 2

(f) aggregierte Systemwartezeit

Abbildung 13.5: Performability bei verteilter Workflow-Ausführung

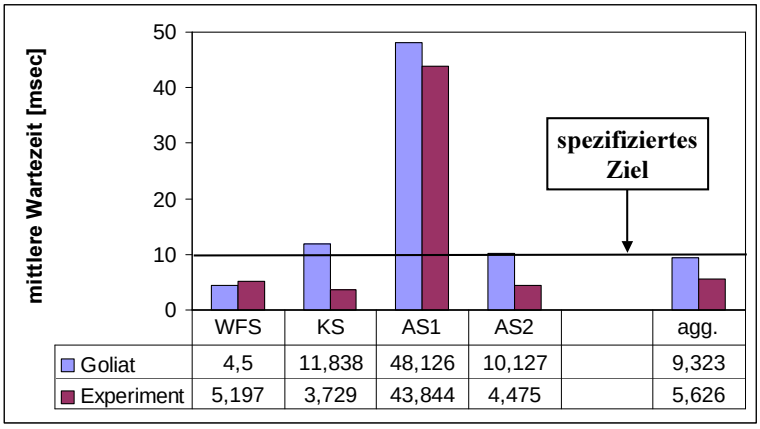
wieder allen Servern ungefähr die gleiche Anzahl an Workflow-Instanzen zugewiesen ist. In diesem Übergangszeitraum erfahren die Service-Aufträge an den einen, wieder angelaufenen Server geringere und die Service-Aufträge an die restlichen Server höhere Wartezeiten als im Modell angenommen.

Abbildung 13.5 zeigt die Diagramme zu den Meßläufen unter verteilter Workflow-Ausführung. Die Systemkonfiguration bei diesen Messungen bestand aus fünf Kommunikations-Servern und jeweils drei Servern der anderen Server-Typen. Die Überschätzung der Wartezeiten an den Kommunikations-Servern steigt aufgrund der gestiegenen Belastung bei verteilter Workflow-Ausführung an. Die bereits in Abschnitt 13.2 beobachtete, mit der Auslastung der Server wachsende Überschätzung der Wartezeiten potentiert sich durch die Berücksichtigung der Server-Ausfälle. In Systemzuständen, in denen ein oder mehrere Kommunikations-Server ausgefallen sind, wird die Belastung jedes einzelnen Servers und somit die Überschätzung durch das zu Grunde liegende Leistungsmodell aus Kapitel 6 höher, je mehr Server ausgefallen sind. Da in dem Szenario von Abbildung 13.5 der Kommunikations-Server-Typ der, was die Ausfälle von Servern angeht, unzuverlässigste Server-Typ ist, sind die Auswirkungen besonders deutlich sichtbar.

13.4 Kostengünstigste Konfiguration

In diesem Abschnitt diskutieren wir die Fähigkeit von Goliat, mit Hilfe der analytischen Modelle eine kostengünstige Systemkonfiguration vorzuschlagen, die angeforderte Durchsatz- und Wartezeitziele erreicht.

Dazu machten wir eine Messung mit zentraler Workflow-Ausführung, bei der wir eine von Goliat vorgeschlagene Systemkonfiguration verwendeten. Die Parameter, mit denen wir Goliat kalibrierten, waren dieselben wie bei den bisher gezeigten Meßreihen. Die Ankunftsrate von Workflow-Instanzen betrug zwei pro Minute. Die Ziele, die die Systemkonfiguration einhalten sollte, waren folgende:



	WFS	KS	AS1	AS2		agg.
Goliat	4,5	11,838	48,126	10,127		9,323
Experiment	5,197	3,729	43,844	4,475		5,626

Abbildung 13.6: Performability unter der von Goliat vorgeschlagenen Systemkonfiguration Y=(3,5,3,3)

1. Die starke Verfügbarkeit des WFMS, also der Anteil der Zeit, in der der benötigte Durchsatz vom WFMS aufrecht erhalten werden kann, soll bei mindestens 99,99% liegen.

2. Die aggregierte Systemwartezeit, also die mittlere Wartezeit aller Service-Aufträge, soll nicht höher sein als 10 Millisekunden.

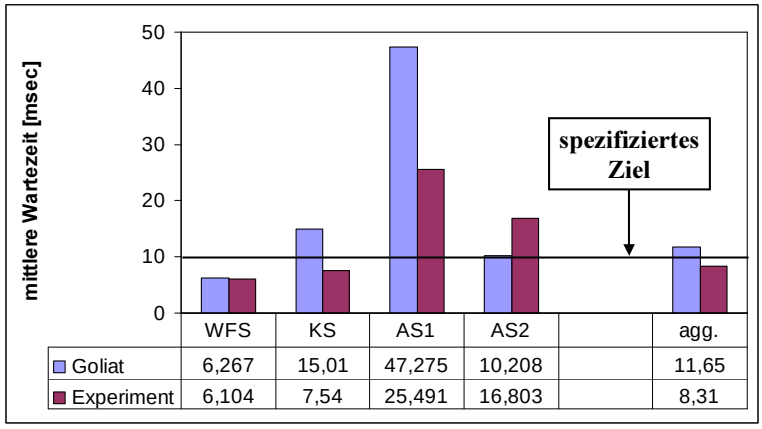
Die Systemkonfiguration, die Goliat vorgeschlagen hatte war:

- 3 Workflow-Server,
- 5 Kommunikations-Server und
- 3 Server von jedem der beiden Applikations-Server-Typen.

Der Kommunikations-Server-Typ bildet sowohl in Hinsicht auf die Verfügbarkeit als auch bezüglich der Performability den Flaschenhals unter den Server-Typen.

Abbilding 13.6 zeigt die mittleren Wartezeiten der Service-Aufträge an den verschiedenen Server-Typen und die aggregierte Systemwartezeit, die bei der Messung mit der von Goliat vorgeschlagenen Systemkonfiguration beobachtet wurden. Zum Vergleich zeigt Abbildung 13.6 die Wartezeiten, die Goliat mit Hilfe der analytischen Modelle vorhergesagt hat.

Wie bereits in Abschnitt 13.3 diskutiert wurde, überschätzt Goliat die aggregierte Systemwartezeit. Daraus resultiert zum einen, daß die spezifizierten Ziele durch die vorgeschlagene Konfiguration eingehalten werden. Zum zweiten bewirkt die Überschätzung aber auch, daß das WFMS leicht überkonfiguriert wird. Die Ergebnisse eines Experimentes mit einer Systemkonfiguration, in der die Anzahl der Workflow-Server und der Applikations-Server bei jeweils drei gleich blieben aber nur vier Kommunikations-Server eingesetzt wurden, sind in Abbildung 13.7 dargestellt. Sie zeigen, daß diese Systemkonfiguration unter gleichen Bedingungen zum Erreichen der Ziele ausreicht und somit die tatsächlich optimale Lösung ist. Die vom Optimierungsmodul von Goliat vorgeschlagene Systemkonfiguration ist also um einen Server zu teuer, was jedoch nur ein vierzehntel der Gesamtkosten der Systemkonfiguration ausmacht.



	WFS	KS	AS1	AS2		agg.
Goliat	6,267	15,01	47,275	10,208		11,65
Experiment	6,104	7,54	25,491	16,803		8,31

Abbildung 13.7: Performability unter der optimalen Systemkonfiguration Y=(3,4,3,3)

13.5 Zusammenfassung und Diskussion

In diesem Kapitel haben wir Ergebnisse von Messungen diskutiert, die wir zur Evaluierung der im zweiten Teil dieser Arbeit vorgestellten analytischen Modelle durchgeführt haben.

Die analytischen Modelle erzielen eine akzeptable Genauigkeit sowohl bei den Leistungsvorhersagen eines stabilen WFMS als auch bei Performability-Vorhersagen unter Berücksichtigung transienter Server-Ausfälle. Das Konfigurationswerkzeug Goliat, dessen Berechnungen auf den Modellen basieren, überschätzt die anfallenden Wartezeiten des Gesamtsystems leicht. Dennoch führen die Performability-Vorhersagen zu Vorschlägen von Systemkonfigurationen, die für gegebene Lastaufkommen und Ausfallverhalten die anfallenden Kosten gering halten und dabei von einem Benutzer spezifizierte Durchsatz- und Wartezeitziele erreichen.

Teil V

Fazit

Kapitel 14

Zusammenfassung der Arbeit

Eine Systemkonfiguration eines verteilten Workflow-Management-Systems (WFMS) besteht aus verschiedenen Typen von Workflow-Servern und Applikations-Servern sowie einem Typ von Kommunikations-Server (zum Beispiel ein ORB). Jeder Server-Typ der ersten beiden Kategorien ist einer bestimmten Menge von Aktivitäten beziehungsweise externen Anwendungsprogrammen zugeordnet. Alle Server-Typen inklusive dem Kommunikations-Server können zur Erhöhung der Leistung und der Verfügbarkeit auf mehreren Rechnern repliziert werden. Durch diese Flexibilität (die im übrigen auf die ein oder andere Art und Weise auch von kommerziellen WFMS unterstützt wird) wird die Wahl einer geeigneten Systemkonfiguration eines WFMS zu einem schweren Problem, wenn gesetzten Anforderungen bezüglich des Durchsatzes, der Antwortzeit bei interaktiven Sitzungen und der Verfügbarkeit genügt werden soll. Darüber hinaus kann es erforderlich werden, eine initiale Konfiguration an Änderungen der Workflow-Last, zum Beispiel durch Hinzunahme neuer Workflow-Typen, anzupassen.

In dieser Dissertation wurde ein System analytischer Modelle basierend auf stochastischen Methoden wie zeitkontinuierlichen Markov-Ketten und Markov-Reward-Modellen entwickelt, das die Vorhersage von Leistung, Verfügbarkeit und Performability eines verteilten WFMS für eine gegebene Workflow-Last ermöglicht. Das Leistungsmodell schätzt den maximal erreichbaren Durchsatz an Workflow-Instanzen und die mittlere Wartezeit von Service-Aufträgen basierend auf einem Markov-Modell für das stochastische Kontrollflußverhalten verschiedener Workflow-Typen. Das Verfügbarkeitsmodell schätzt die mittlere Ausfallzeit des WFMS bei gegebenen Ausfall- und Reperaturraten für die verschiedenen Server-Typen. Schließlich kombiniert das Performability-Modell die beiden vorher genannten Modelle und berücksichtigt somit Leistungseinbußen während transienter Server-Ausfälle. Das Performability-Modell schätzt die tatsächliche mittlere Wartezeit von Service-Aufträgen unter expliziter Berücksichtigung der Zeiträume, in denen nur ein Teil der Server, mit denen das WFMS konfiguriert wurde, verfügbar ist. Diese Modelle, die wir durch systematische Messungen in einer realitätsnahen, einen echten WFMS-Prototypen (das WFMS Mentor-lite) umspannende Simulationsumgebung evaluiert haben, bilden die Grundlage eines Konfigurationswerkzeuges für verteilte WFMS.

Das Konfigurationswerkzeug bezieht seine Daten aus Statistiken über die Workflow-Last und die Historie von Workflow-Instanzen. Es überträgt die Informationen in die internen Modelle und dient so einer Was-Wäre-Wenn-Analyse für hypothetische Systemkonfigurationen. Darüber hinaus kann das Konfigurationswerkzeug durch Variation von hy-

pothetischen Konfigurationen Vorschläge für die (analytisch) beste Systemkonfiguration erarbeiten, nämlich die Systemkonfiguration, die mit den geringsten Replikationsgraden für die verschiedenen Server-Typen des WFMS gegebene Leistungs-, Verfügbarkeits- und Performability-Ziele erreicht.

Kapitel 15

Ausblick

In diesem Kapitel präsentieren wir Überlegungen, wie Erweiterungen und Verbesserungen des vorgestellten Konfigurationswerkzeuges und seiner internen analytischen Methoden neben den bereits in den Diskussionen der einzelnen Kapitel aufgeführten Punkten angegangen werden können.

An vorderster Stelle steht die Evaluierung, daß die internen Modelle des Konfigurationswerkzeuges auch bei anderen, insbesondere kommerziellen WFMS, anwendbar sind. Dazu muß die Meßumgebung auf solche Systeme angepaßt werden. Zwei Vorgehensweisen sind denkbar:

1. Der Kern von Mentor-lite, um den herum die Simulationskomponenten aufgebaut sind, wird modular durch die Workflow-Engines der zu untersuchenden WFMS ausgetauscht. Dieser Ansatz ändert jedoch nichts an der Simulation der anderen Komponenten. Der Quellcode des WFMS müßte verfügbar sein und eventuell angepaßt werden (z.B. einbauen von Checkpoints zur Identifikation von Service-Aufträgen an einen Workflow-Server), was insbesondere bei kommerziellen WFMS nur durch das Mitwirken von Entwicklern des Systems erreichbar ist.

2. Die gesamte Meßumgebung wird generisch weiterentwickelt, so daß es möglich ist, die Meßläufe alleine durch die von anderen WFMS zur Verfügung gestellten Monitoring- und Historiendaten realitätsnah nachzuvollziehen. Dazu müßte untersucht werden, welche Werte bei den WFMS zu erfassen sind.

Orthogonal zu diesen Erweiterungen stellen Fallstudien mit im Einsatz befindlichen WFMS über längere Zeiträume hinweg die beste Art der Evaluierung dar.

Innerhalb des Leistungsmodelles werden die Server als eine Gruppe alleinstehender M/G/1-Bedienstationen modelliert. Es bleibt zu untersuchen, welche anderen Arten der Modellierung von Rechnersystemen (zum Beispiel Warteschlangennetze oder stochastische Petri-Netze [Bol89, STP96]) die Vorhersagen für die Leistung der Server verbessern können. Dabei muß aber darauf geachtet werden, daß die Modelle generisch genug bleiben, sodaß alle WFMS in einfacher Weise auf sie abgebildet werden können.

Die Metriken, auf die das Konfigurationswerkzeug abziehlt, sind die Wartezeit, die ein potentieller Benutzer des WFMS ertragen muß, wenn er interktiv mit dem System arbeitet, und der Durchsatz an Workflows verschiedener Workflow-Typen. Beide Metriken sind

direkt von den mittleren Wartezeiten der Service-Aufträge an den verschiedenen Server-Typen des WFMS abhängig. Eine Verschärfung der Optimierungsziele bei der Suche nach einer geeigneten Systemkonfiguration bilden stochastische Leistungsgarantien [Wei00a]. Eine stochastische Leistungsgarantie könnte zum Beispiel sein, daß 90% aller Service-Aufträge an einen Server-Typ eine Wartezeit von weniger als 20 Millisekunden aufweist. Die Berechnungen solcher Leistungsgarantien bedienen sich der Laplace-Transformierten für die Verteilungen der Bedienzeiten an den Servern. Allerdings ist die analytische Bestimmung einer Laplace-Transformierten typischerweise nicht ohne anwendungsspezifische Vereinfachungen in der Modellierung möglich. Bei stochastischen Leistungsgarantien über mehrere Server-Typen mit verschiedenen Verteilungen für die Bedienzeiten müssen diese Verteilungen gefaltet werden. Es bleibt zu untersuchen, ob es möglich ist, stochastische Leistungsgarantien aus den analytischen Modellen abzuleiten. Insbesondere ist darauf zu achten, welche Parameter zur Kalibrierung der Modelle benötigt werden und zur Verfügung gestellt werden können.

Letztlich ist auch zu untersuchen, inwiefern von stationären Aussagen abgewichen werden muß und transiente Effekte beim Lastaufkommen oder beim Ausfallverhalten von Servern berücksichtigt werden müssen. So könnte zum Beispiel die Ankunftsrate von Workflow-Instanzen von der Tageszeit anhängig sein, oder es könnten mehrere Service-Shutdowns zur Aktualisierung der Software für Server desselben Server-Typs unmittelbar hintereinander durchgeführt werden.

Kapitel 16

Summary of the Thesis

A system configuration of a distributed workflow management system (WFMS) consists of different types of workflow servers (i.e., workflow engines), application servers, and one type of communication server (e.g., an ORB). Each server of the first two categories can be dedicated to a specified set of workflow activities or external applications on a per type basis. Each of these dedicated servers and also the communication server can be replicated across multiple computers for enhanced performance and availability. Given this flexibility (which is provided in similar ways also by some commercial WFMSs), it is a difficult problem to choose an appropriate configuration for the entire WFMS that meets all requirements with regard to throughput, interaction response time, and availability. Moreover, it may be necessary to adapt an initial configuration over time due to changes of the workflow load, e.g., upon adding new workflow types.

To solve this configuration problem, we have developed a suite of analytic models, using stochastic methods like continuous-time Markov chains and Markov reward models, to predict the performance, availability, and performability under a given load. The performance model estimates the maximum sustainable throughput in terms of workflow instances per time unit and the mean waiting time for service requests on the basis of a Markov chain model for the statistical behavior of the various workflow types. The availability model estimates the mean downtime of the entire system for given failure and restart rates for the various components. Finally, the performability model takes into account the performance degradation during transient server down times and estimates the effective mean waiting time for service requests with explicit consideration of periods during which only a subset of a server type's replicas are running. These models, which are evaluated by a set of systematic measurements in a realistic simulation environment with a real WFMS prototype (the WFMS Mentor-lite), form the core of an auto-configuration tool for distributed WFMSs.

The auto-configuration tool is driven by statistics on the workload and the workflow history. It can feed this information into its analytic models for the what-if analysis of a hypothetical configuration. By systematic variation of the parameters for such configurations the tool is able to derive the (analytically) best configuration, i.e., the minimum degree of replication of the involved server types to meet given availability and performance or performability goals, and recommend appropriate reconfigurations.

Teil VI

Anhang

Anhang A

Parameterdatei von Goliat

In diesem Teil des Anhangs präsentieren wir ein Beispiel für eine Datei, mit der das Konfigurationswerkzeug Goliat parametrisiert werden kann. Goliat liest die Datei beim Start automatisch ein. Ein nachträgliches Einlesen der geänderten Datei kann über die Schaltleiste *Edit* > *Reload Preferences* im Kontrollfenster von Goliat initiiert werden.

Mit # gekennzeichnete Zeilen werden von Goliat als Kommentare interpretiert und ignoriert.

```
# Beispiel für eine Parameterdatei von Goliat
# im Home-Verzeichnis als Goliat.cfg speichern und anpassen

# Die minimal zu erreichende Wahrscheinlichkeit, mit der
# die Workflow-Instanzen terminieren, bei der Berechnung
# der unendlichen Summe (anpaßbar wegen Genauigkeit der
# Vorhersage der erzeugten Last) (double)
Min_Probability = 0.99

# Die maximale Anzahl an Iterationen bei der Abschätzung der
# benötigten Schritte, mit denen der WF mit der spezifizierten
# Wahrscheinlichkeit terminiert (anpassbar wegen Speicher) (int)
Max_Iterations = 50000

# Die Darstellungsgröße der Workflows im Workflow-Editor (double)
Zoom_Factor = 24.0

# Das Verzeichnis, in dem die Workflow- und Server-Typ-Parameter
# abgespeichert und geladen werden (letzter Separator ist wichtig!)
Savepath = d:/user/gillmann/goliat/parameters/
```

Anhang B

Glossar

α_t Ankunftsrate von Workflows des Workflow-Typs t (siehe Abschnitt 6.3.3).

λ_x Ausfallrate von Servern des Server-Typs x (siehe Definition 4.3).

μ_x Reperaturrate von Servern des Server-Typs x (siehe Definition 4.4).

ν_i^t Abgangsrate des Ausführungszustandes s_i des Workflow-Typs t (siehe Definition 5.3).

Abgangsrate Rate, mit der ein stochastischer Prozeß einen Zustand verläßt (siehe Definition 5.3).

Absorbierender Zustand Zustand des Flußprozesses, der die Zeit nach der Terminierung des Workflows repräsentiert (siehe Definition 5.2).

Anfangswahrscheinlichkeit Wahrscheinlichkeit, mit der ein Zustand einer Markov-Kette zum Zeitpunkt Null betreten ist (siehe Definition 3.7).

Antwortzeit Zeit zwischen Initiierung und Beendigung der Abarbeitung eines Service-Auftrages (siehe Definition 3.13).

Ausfallrate Kehrwehrt der MTTF eines Server-Typs (siehe Definition 4.3).

Ausführungszustand Gegeben durch in Ausführung befindliche Aktivität oder die nichtleere Menge in Ausführung befindlicher paralleler Subworkflows (siehe Definition 5.1).

CTMC (continuous time Markov chain) Zeitkontinuierliche Markov-Kette (siehe Definition 3.3).

Durchlaufzeit Dauer der Ausführung eines Workflows (siehe Definition 6.1).

DTMC (discrete time Markov chain) Zeitdiskrete Markov-Kette (siehe Definition 3.3).

Flußprozeß CTMC zur Modellierung des Kontrollfluß-Verhaltens von Workflows (siehe Definition 5.3)

H_i^t Mittlere Verweildauer eines Workflows vom Typ t im Ausführungszustand s_i^t (siehe Definition 5.3).

Hard-Crash Ausfall mit Verlust persistenter Daten und hoher MTTR (siehe Abschnitt 7.2).

L_{xi}^t Anzahl der Service-Aufträge im Ausführungszustand s_i^t an Server-Typ x (siehe Abschnitte 6.3.2 und 6.4).

L_a Lastvektor der Aktivität a (siehe Abschnitt 6.3.1).

$\mathbb{L}^t$ Lastmatrix des Workflow-Typs t (siehe Abschnitt 6.3.2).

Last Anzahl an Service-Aufträgen an einen Server-Typ während eines Beobachtungszeitraumes (siehe Definition 4.2).

Markov-Kette Mathematisches Modell zur Beschreibung eines stochastischen Prozesses, der gemäß gegebener Übergangswahrscheinlichkeiten von Zustand zu Zustand wechselt (siehe Abschnitt 3.3.1).

MRM Markov-Reward-Modell – Ordnet jedem Zustand einer Markov-Kette für dessen Betreten eine "Belohnung" (englisch: *reward*) zu (siehe Abschnitt 3.3.3).

MTTF (mean time to failure) Mittlere Dauer bis zum nächsten Ausfall eines Servers nach Inbetriebnahme (siehe Definition 4.3).

MTTR (mean time to repair) Mittlere Dauer bis zur Wiederverfügbarkeit eines Servers nach einem Ausfall (siehe Definition 4.4).

p_{ij}^t Schrittwahrscheinlichkeiten des Flußprozeses für Workflow-Typ t (siehe Definitionen 3.6 und 5.3).

$\vec{p}_{ij}^t$ Schrittwahrscheinlichkeiten der aus der Normalisierung des Flußprozesses für Workflow-Typ t resultierenden zeitdiskreten Markov-Kette (siehe Abschnitt 5.4).

$\vec{p}_{ij}^t\,(z)$ Tabu-Wahrscheinlichkeit, daß der Flußprozeß für Workflow-Typ t nach z Schritten beginnend in Zustand s_i^t in Zustand s_j^t ist, ohne zuvor den absorbierenden Zustand s_A^t betreten zu haben (siehe Gleichung 6.8).

Performability Leistung eines Systems unter Berücksichtigung transienter Komponentenausfälle (siehe Definition 8.1).

Reparaturrate Kehrwehrt der MTTR eines Server-Typs (siehe Definition 4.4).

s_i^t Ausführungszustände des Workflow-Typs t sowie Zustände der t repräsentierenden CTMC (siehe Definition 5.1 sowie Abschnitt 5.2).

s_A^t Absorbierender Zustand der CTMC für Workflow-Typ t (siehe Definition 5.2).

Schrittwahrscheinlichkeit Wahrscheinlichkeit, mit der ein stochastischer Prozeß in genau einem Schritt von einem Zustand in einen anderen wechselt (siehe Definition 3.6).

Server Instanz eines Server-Typs (siehe Definition 4.1).

Server-Typ Spezieller Typ von Workflow-Servern, Applikations-Servern oder Kommunikations-Servern (siehe Definition 4.1).

Service-Shutdown geplante, kontrollierte Nicht-Verfügbarkeit von Rechnern (siehe Abschnitt 7.2).

Soft-Crash nichtdeterministisch auftretender, nicht reproduzierbarer Ausfall mit geringer MTTR (siehe Abschnitt 7.2).

Starke Verfügbarkeit Prozentsatz an Zeit, in der ein System den geforderten Durchsatz an Aufträgen erzielen kann (siehe Definition 7.3).

Systemkonfiguration Tupel aus der Anzahl der Server jeden Server-Typs (siehe Definition 4.5).

Systemzustand Tupel aus der Anzahl der verfügbaren Server eines jeden Server-Typs (siehe Definition 4.6).

Verfügbarkeit Prozentsatz an Zeit, in der ein System Aufträge bedienen kann (siehe Definition 7.3).

Verweildauer Zeit, die der Flußprozeß in einem Zustand verweilt, bevor er in den nächsten Zustand übergeht (siehe Definition 5.3).

W^Y Performability-Vektor für Systemkonfiguration Y (siehe Abschnitt 8.2).

W_x^Y Eintrag im Performability-Vektor für Server-Typ x bei Systemkonfiguration Y (siehe Abschnitt 8.2).

Wartezeit Zeit bis zum Beginn der Abarbeitung eines Service-Auftrages (siehe Definition 3.12).

WFMS Workflow-Management-System.

X Menge aller möglichen Systemzustände (siehe Definition 4.6).

$\tilde{X}$ Menge der ganzzahligen Darstellungen aller möglichen Systemzustände (siehe Abschnitt 7.4).

Y Systemkonfiguration (siehe Definition 4.5).

Z^t Menge aller Ausführungszustände des Workflow-Typs t sowie Menge der Zustände des Flußprozesses für t (siehe Abschnitte 5.1 und 5.2).

Literaturverzeichnis

[AAA+97] G. Alonso, D. Agrawal, A. Abbadi, C. Mohan: *Functionality and Limitations of Current Workflow Management Systems*, IEEE Expert, Ausgabe 12, Nummer 5, 1997

[ADEPT] ADEPT Project, http://www.informatik.uni-ulm.de/dbis/f&l/forschung/workflow/ftext-adept_e.html

[Aic95] C. Aichele: *Geschäftsprozeßanalyse auf Basis von Kennzahlensystemen*, Dissertation, Rechts- und Wirtschaftswissenschaftliche Fakultät, Universität des Saarlandes, Saarbrücken, 1995

[All90] A.O. Allen: *Probability, Statistics, and Queueing Theory with Computer Science Applications*, Second Edition, Academic Press, 1990

[BBK00] N. Bhatti, A. Bouch, A. Kuchinsky: *Integrating User-Perceived Quality into Web Server Design*, Int'l. World Wide Web Conf. (WWW), Amsterdam, Niederlande, 2000

[BD97] T. Bauer, P. Dadam: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration*, IFCIS Int'l Conf. on Cooperative Information Systems (CoopIS), Kiawah Island, Süd-Carolina, 1997

[BD99a] T. Bauer, P. Dadam: *Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation*, Informatik Forschung und Entwicklung, Ausgabe 14, Nummer 4, 1999

[BD99b] T. Bauer, P. Dadam: *Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation*, Ulmer Informatik-Berichte, Nummer 99-02, Ulm, 1999

[BD99c] T. Bauer, P. Dadam: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows*, GI Workshop Unternehmensweite und unternehmensübergreifende Workflows, Paderborn, 1999

[BD00] T. Bauer, P. Dadam: *Efficient Distributed Workflow Management Based on Variable Server Assignment*, Conf. on Advanced Information Systems Engineering (CAiSE), Stockholm, Schweden, 2000

[BGM+98] G. Bolch, S. Greiner, H. de Meer, K.S. Trivedi: *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications*, John Wiley and Sons, 1998

[Bol89] G. Bolch: *Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle*, B.G. Teubner Verlag, 1989

[BRD01a] T. Bauer, M. Reichert, P. Dadam, *Adaptives und verteiltes Workflow-Management*, GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Oldenburg, 2001

[BRD01b] T. Bauer, M. Reichert, P. Dadam, *Effiziente Übertragung von Prozessinstanzdaten in verteilten Workflow-Management-Systemen*, Informatik Forschung und Entwicklung, Ausgabe 16, Nummer 2, 2001

[BSR96] A. Bonner, A. Shrufi, S. Rozen: *LabFlow-1: a Database Benchmark for High-Throughput Workflow Management*, Int'l Conf. on Extending Database Technology (EDBT), Avignon, Frankreich, 1996

[CHR+98] A. Cichoki, A. Helal, M. Rusinkiewicz, D. Woelk: *Workflow and Process Automation - Concepts and Technology*, Kluwer Academic Publisher, 1998

[CKS+01] D. Christensen, A. Kraiß, A. Syri, G. Weikum: *Automatische Übersetzung von Geschäftsprozeßmodellen in ausführbare Workflows*, GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Oldenburg, 2001

[CGS97] S. Ceri, P. Grefen, G. Sanchez: *WIDE - A Distributed Architecture for Workflow Management*, Int'l Workshop on Research Issues in Data Engineering (RIDE), Birmingham, England, 1997

[CSIM] Mesquite Software Homepage, http://www.mesquite.com

[DGA+98] A. Dogac, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, M. Altinel: *Design and Implementation of a Distributed Workflow Management System: METUFlow*, in [DKO+98]

[DHK+97] J. Doppelhammer, T. Höppler, A. Kemper, D. Kossmann: *Database Performance in the Real World - TPC-D and SAP R/3*, ACM SIGMOD Int'l Conf. on Modeling of Data (SIGMOD), Tucson, Arizona, 1997

[DKM+97] S. Das, K. Kochut, J. Miller, A. Sheth, D. Worah: *ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR_2*, Technical Report, University of Georgia, Athens, Georgia, 1997

[DKO+98] A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Hrsg.): *Workflow Management Systems and Interoperability*, NATO Advanced Study Institute, Springer Verlag, 1998

[DR98] P. Dadam, M. Reichert, *The ADEPT WfMS Project at the University of Ulm*, European Workshop on Workflow and Process Management (WPM), Zürich, Schweiz, 1998

[EGL98] J. Eder, H. Groiss, W. Liebhart: *The Workflow Management System Panta Rhei*, in [DKO+98]

[GBL+98] A. Geppert, M. Berndtsson, D. Lieuwen, C. Roncancio: *Performance Evaluation of Object-Oriented Active Database Management Systems Using the BEAST Benchmark*, Theory and Practice of Object Systems (TAPOS), Ausgabe 4, Nummer 3, 1998

[GCS+98] P. Grefen, S. Ceri, G. Sanchez: *Trabsaction and Rule Support for Workflow Management - A Retrospective on the WIDE Architecture*, European Workshop on Workflow and Process Management (WPM), Zürich, Schweiz, 1998

[GHS95] D. Georgakopoulos, M. Hornick, A. Sheth:*An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*, Distributed and Parallel Databases, Ausgabe 3, Nummer 2, 1995

[GMW+99] M. Gillmann, P. Muth, G. Weikum, J. Weißenfels: *Benchmarking von Workflow-Management-Systemen*, GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Freiburg, 1999

[GMW00] M. Gillmann, R. Mindermann, G. Weikum: *Benchmarking and Configuration of Workflow Management Systems*, IFCIS Int'l Conf. on Cooperative Information Systems (CoopIS), Eilat, Israel, 2000

[GR93] J. Gray, A. Reuter: *Transaction Processing - Concepts and Technics*, Morgan Kaufmann Publishers, 1993

[GT96] M. Greiner, G. Tinhofer: *Stochastik für Studienanfänger der Informatik*, Carl Hanser Verlag, 1996

[GT98] A. Geppert, D. Tombros: *Event-based Distributed Workflow Execution with EVE*, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, The Lake District, England, 1998

[GWS+00a] M. Gillmann, J. Weissenfel, G. Shegalov, W. Wonner, G. Weikum: *A Goal-driven Auto-Configuration Tool for the Distributed Workflow Management System Mentor-lite*, ACM SIGMOD Int'l Conf. on Modeling of Data (SIGMOD), Dallas, Texas, 2000

[GWS+00b] M. Gillmann, J. Weissenfels, G. Shegalov, W.Wonner, G. Weikum: *Mentor-lite Customizability: Tailoring a Light-Weight Workflow Management System to Workflow Application and Organizational Needs*, Software Demonstrations Track, Int'l. Conf. on Extending Database Technology (EDBT), Konstanz, 2000

[GWW+99] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiß: *Performance Assessment and Configuration of Enterprise-Wide Workflow Management Systems*, GI Workshop Unternehmensweite und unternehmensübergreifende Workflows, Paderborn, 1999

[GWW+00] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiss: *Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems*, Int'l Conf. on Extending Database Technology (EDBT), Konstanz, 2000

[HA98] C. Hagen, G. Alonso: *Backup and Process Migration Mechanisms in Process Support Systems*, Technischer Report, Nummer 304, ETH Zürich, Zürich, Schweiz, 1998

[HA99] C. Hagen, G. Alonso: *Highly Available Process Support Systems: Implementing Backup Mechanisms*, IEEE Symposium on Reliable Distributed Systems (SRDS), Lausanne, Schweiz, 1999

[Här97] T. Härder (Hrsg.): *Informatik Forschung und Entwicklung – Themenheft "Workflow-Management"*, Ausgabe 12, Nummer 2, 1997

[Har87] D. Harel: *State Charts: A Visual Formalism for Complex Systems*, Science of Computer Programming, Ausgabe 8, Nummer 3, 1987

[Hav98] B.R. Haverkort: *Performance of Computer Communication Systems*, John Wiley and Sons, 1998

[HG97] D. Harel, E. Gery: *Executable Object Modeling with Statecharts*, IEEE Computer, Ausgabe 30, Nummer 7, 1997

[HHJ+00] C. Hahn, S. Horn, S. Jablonski, R. Lay, J. Neeb, R. Schamburger, M. Schlundt: *Demonstration: How to Enact Flexible Workflow Management: The MOBILE Approach*, Software Demonstrations Track, Int'l. Conf. on Extending Database Technology (EDBT), Konstanz, 2000

[HRB+00] C. Hensinger, M. Reichert, T. Bauer, T. Strzeletz, P. Dadam: *Adept$_{workflow}$ - Advanced Workflow Technology for the Efficient Support of Adaptive, Enterprise-wirde Processes*,Software Demonstrations Track, Int'l. Conf. on Extending Database Technology (EDBT), Konstanz, 2000

[IBM] IBM Library: *MQSeries Workflow: Concepts and Architecture*, http://www-4.ibm.com/software/ts/mqseries/library/manualsa/#Workflow

[IONA] IONA Technologies PLC, Handbücher zu Orbix 2.3, 1997

[Jai91] R. Jain: *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, 1991

[JB96] S. Jablonski, C. Bussler: *Workflow-Management, Modeling Concepts, Architecture and Implementation*, International Thomson Computer Press, 1996

[JBS97] S. Jablonski, M. Böhm, W. Schulze (Hrsg.): *Workflow-Management - Entwicklung von Anwendungen und Systemen*, dpunkt Verlag, 1997

[KAG+96] M. Kamath, G. Alonso, R. Günthör, C. Mohan: *Providing High Availability in Very Large Workflow Management Systems*, Int'l Conf. on Extending Database Technology (EDBT), Avignon, Frankreich, 1996

[KDH+95] R. Klar, P. Dauphin, F. Hartleb, R. Hofmann, B. Mohr, A. Quick, M. Siegle: *Messung und Modellierung paralleler und verteilter Rechensysteme*, B.G. Teubner Verlag, 1995

[Kle76] L. Kleinrock: *Queueing Systems Volume II: Computer Applications*, John Wiley and Sons, 1976

[Kra98] A. Kraiß: *Hierarchische Speicherverwaltung für Informationssysteme mit Tertiärspeicher*, Dissertation, Technische Fakultät, Universität des Saarlandes, Saarbrücken, 1998

[KS98] A. Kläser, K. Stichter: *Konzeption und Implementierung eines Workflow-Log-Managers und eines Workflow-History-Managers*, Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1998

[KSW+01a] A. Kraiß, F. Schön, G. Weikum, U. Deppisch: *Middleware-Antwortzeitgarantien für e-Services*, GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), Oldenburg, 2001

[KSW+01b] A. Kraiss, F. Schoen, G. Weikum, U. Deppisch: *Towards Response Time Guarantees for e-Service Middleware*, Data Engineering Bulletin special issue on Infrastructure for Advanced E-Services, Ausgabe 24, Nummer 1, 2001

[KW97] A. Kraiss, G. Weikum: *Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions*, Int'l Conf. on Very Large Data Bases (VLDB), Athen, Griechenland, 1997

[KW98] A. Kraiss, G. Weikum: *Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions*, VLDB Journal, Ausgabe 7, Nummer 3, 1998

[KWA99] J. Klingemann, J. Waesch, K. Aberer: *Deriving Service Models in Cross-Organizational Workflows*, Int'l Workshop on Reasearch Issues in Data Engineering (RIDE), Sydney, Australien, 1999

[Lan92] H. Langendörfer: *Leistungsanalyse von Rechensystemen*, Carl Hanser Verlag, 1992

[LR99] F. Leymann, D. Roller: *Production Workflow*, Prentice Hall, 1999

[MAG+95] C. Mohan, G. Alonso, R. Gnthr, M. Kamath: *Exotica: A Research Perspective on Workflow Management Systems*, Data Engineering Bulletin, Ausgabe 18, Nummer 1, 1995

[Mentor] Mentor-lite Project, http://www-dbs.cs.uni-sb.de/-mlite

[Meteor] Meteor, http://lsdis.cs.uga.edu/proj/meteor/meteor.html

[Min00] R. Mindermann: *Benchmarking von Workflow-Management-Systemen*, Diplomarbeit, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2000

[Moh99] C. Mohan: *Workflow Management in the Internet Age*, Tutorial, http://www-rodin.inria.fr/ mohan

[MP96] K. Moore, M. Peterson: *A Groupware Benchmark Based on Lotus Notes*, IEEE CS Int'l Conf. on Data Engineering (ICDE), New Orleans, Louisiana, 1996

[MPS+98] J. Miller, D. Palaniswami, A. Sheth, K. Kochut, H. Singh: *WebWork: METEOR2's Web-Based Workflow Management System*, Journal of Intelligent Information Systems, Ausgabe 10, Nummer 2, 1998

[MWG+98] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: *Mentor-lite: Integrating Light-Weight Workflow Management Systems within Business Environments*, European Workshop on Workflow and Process Management (WPM), Zürich, Schweiz, 1998

[MWG+99a] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: *Integrating Light-Weight Workflow Management Systems within Existing Business Environments*, IEEE CS Int'l Conf. on Data Engineering (ICDE), Sydney, Australien, 1999

[MWG+99b] P. Muth, J. Weissenfels, M. Gillmann, G. Weikum: *Workflow History Management in Virtual Enterprises using a Light-Weight Workflow Management System*, Int'l Workshop on Research Issues in Data Engineering (RIDE), Sydney, Australien, 1999

[Nel95] R. Nelson: *Probability, Stochastic Processes, and Queueing Theory*, Springer Verlag, 1995

[NFZ98] M. Nüttgens, T. Feld, V. Zimmermann: *Business Process Modeling with EPC and UML: Transformation or Integration*, Workshop des Arbeitskreises "Grundlagen objektorientierter Modellierung" (GROOM) der GI-Fachgruppe 2.1.9 ("Objektorientierte Softwareentwicklung"), Heidelberg, 1998

[Nor97] J.R. Norris: *Markov Chains*, Cambridge University Press, 1997

[RBD99] M. Reichert, T. Bauer, P. Dadam, *Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows*, GI Workshop Unternehmensweite und unternehmensübergreifende Workflows, Paderborn, 1999

[RBD00] M. Reichert, T. Bauer, P. Dadam, *ADEPT - Realisierung flexibler und zuverlässiger unternehmensweiter Workflow-Anwendungen*, Knowledge Engineering, Management and Training (KnowTech), Leipzig, 2000

[RD97] M. Reichert, P. Dadam: *A Framework for Dynamic Changes in Workflow Management Systems*, Int'l Workshop on Database and Expert Systems Applications (DEXA Workshop), Toulouse, Frankreich, 1997

[RD98] M. Reichert, P. Dadam: *ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control*, Journal of Intelligent Information Systems, Special Issue on Workflow Management, Ausgabe 10, Nummer 2, 1998

[RDR97] F. Rump, J. Desel, H. Reichel (Hrsg.): *Proc. GI-Workshop "Grundlagen der Parallelität"*, Aachen, 1997

[RHD98] M. Reichert, C. Hensinger, P. Dadam, *Supporting Adaptive Workflows in Advanced Application Environments*, EDBT Workshop on Workflow Management Systems, Valencia, Spanien, 1998

[Sch96] A.W. Scheer: *Benchmarking Business Process*, IFIP Int'l Conf. on Production Management Systems (APMS), Kyoto, Japan, 1996

[Sch98] H.D. Schwetmann: *Model-based Systems Analysis Using CSIM18*, Winter Simulation Conference (WSC), Washington DC, New York, 1998

[Sch01] T. Schillo: *Verifikation und Evaluierung von Konfigurationen verteilter Workflow-Management-Systeme*, Diplomarbeit, Fachrichtung Informatik, Universität des Saarlandes, Saarbrücken, 2001

[SGW00] G. Shegalov, M. Gillmann, G. Weikum: *XML-enabled Workflow Management for E-Services across Heterogeneous Platforms*, Workshop on Technologies for E-Services (TES), Kairo, Ägypten, 2000

[SGW01] G. Shegalov, M. Gillmann, G. Weikum: *XML-enabled Workflow Management for E-Services across Heterogeneous Platforms*, erscheint in VLDB Journal Special Issue on E-Services, Springer Verlag, 2001

[SH97] H. Schuster, P. Heinl: *A Workflow Data Distribution Strategy for Scalable Workflow Management Systems*, ACM Symposium on Applied Computing (SAC), San Jose, Kalifornien, 1997

[SK98] A. Sheth, K.J. Kochut: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*, in [DKO+98]

[SNS99] H. Schuster, J. Neeb, R. Schamburger: *A Configuration Management Approach for Large Workflow Management Systems*, Int'l Joint Conf. on Work Activities Coordination and Collaboration (WACC), San Francisco, Kalifornien, 1999

[Staff] Staffware, http://www.staffware.com

[STP96] R. A. Sahner, K. S. Trivedi, A. Puliafito: *Performance and Reliability Analysis of Computer Systems*, Kluwer Academic Publishers, 1996

[Tak91] H. Takagi: *Queueing Analysis - Volume 1: Vacation and Priority Systems, Part 1*, North-Holland, 1991

[Tij94] H.C. Tijms: *Stochastic Models - An Algorithmic Approach*, John Wiley and Sons, 1994

[TPC] Transaction Processing Performance Council, http://www.tpc.org

[UML] Unified Modeling Language (UML), http://www.rational.com/uml/documentation.html

[VB96] G. Vossen, J. Becker (Hrsg.): *Geschäftprozeßmodellierung und Workflow-Management - Modelle, Methoden, Werkzeuge*, International Thomson Publishing, 1996

[VW98] G. Vossen, M. Weske: *The WASA Approach to Workflow Management for Scientific Applications*, in [DKO+98]

[VW99] G. Vossen, M. Weske: *The WASA2 Object-Oriented Workflow Management System*, ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD), Philadelphia, Philadelphia, 1999

[Wei00a] G. Weikum: *The Web in 2010: Challenges and Opportunities for Database Research*, Conf. at the Occasion of Dagstuhl's 10th Anniversary "Informatics - 10 Years Back, 10 Years Ahead", Dagstuhl, erschienen in Lecture Notes in Computer Science (LNCS), Nummer 2000, Springer Verlag, 2000

[Wei00b] J. Weißenfels: *Architektur erweiterbarer Workflow-Management-Systeme*, Dissertation, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, 2000

[WFMC] Workflow Management Coalition, http://www.wfmc.org

[WGR+00] J. Weißenfels, M. Gillmann, O. Roth, G. Shegalov, W. Wonner: *The Mentor-lite Prototype: A Light-Weight Workflow Management System*, IEEE CS Int'l. Conf. on Data Engineering (ICDE), San Diego, Kalifornien, 2000

[Wod97] D. Wodtke: *Modellbildung und Architektur von verteilten Workflow-Management-Systemen*, Dissertation, Technische Fakultät, Universität des Saarlandes, Saarbrücken, 1997

[Won99] W. Wonner: *MoM - Ein Monitoring-Tool für Mentor-lite*, Dokumentation zum Fortgeschrittenenpraktikum, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1999

[Won01] W. Wonner: *Goliat - Ein Konfigutations-Tool für Mentor-lite*, Diplomarbeit, Fachrichtung Informatik, Universität des Saarlandes, Saarbrücken, 2001

[WV98] M. Weske, G. Vossen: *The WASA Project: A Survey*, European Workshop on Workflow and Process Management (WPM), Zürich, Schweiz, 1998

[WV01] G. Weikum, G. Vossen: *Transactional Information Systems - Theory, Algorithms, and the Practice of Concurrency Control and Recovery*, Morgan Kaufmann Publishers, 2001

[WW97] D. Wodtke, G. Weikum: *A Formal Foundation for Distributed Workflow Execution Based on State Charts*, Int'l Conf. on Database Theory (ICDT), Delphi, Griechenland, 1997

[WWK+97] G. Weikum, D. Wodtke, A. Kotz-Dittrich, P. Muth, J. Weißenfels: *Spezifikation, Verifikation und verteilte Ausführung von Workflows in MENTOR*, in: [Här97]