Ein konstruktiver Ansatz zur systematischen Entwicklung von Ausführungsanweisungen für Workflows

Gemeinsamer Technischer Bericht der

Technischen Universität Dresden
Lehrstuhl Datenbanken
(Technischer Bericht TUD/FI/98/05- April '98)

Universität Zürich
Institut für Informatik
(Technischer Bericht 98.08)

Universität Erlangen-Nürnberg
Lehrstuhl für Datenbanksysteme
(Technischer Bericht TR-I6-1998-3)

M. Böhm
Technische Universität Dresden
markus.boehm@inf.tu-dresden.de

unter Mitarbeit von

Prof. Dr. habil. K. Meyer-Wegener
Technische Universität Dresden
kmw@inf.tu-dresden.de

Prof. Dr. habil. C. Cap
Universität Zürich, jetzt Universität Rostock
cap@informatik.uni-rostock.de

Prof. Dr. S. Jablonski
Universität Erlangen-Nürnberg
stefan.jablonski@informatik.uni-erlangen.de

Vorwort

Der Technische Bericht stellt Ergebnisse vor, die bei Forschungsaufenthalten an der Universität Zürich (Sommer 1997), der Universität Erlangen-Nürnberg (Herbst 1997) und meiner Arbeit an der Technischen Universität Dresden entstanden sind. Der Aufenthalt am Institut für Informatik (IFI) der Universität Zürich markiert den Beginn des Versuchs, bislang nicht untersuchte Schritte bei der Entwicklung von Workflow-Schemata mit formalen Mitteln zu beschreiben und ausgehend von der formalen Darstellung auf Eigenschaften der Workflow-Management-Anwendung zu schließen.

Prof. Dr. Cap, auf dessen Einladung ich nach Zürich kam, hat dieses Vorhaben mit großem Interesse, umfassender Betreuung und hohem persönlichen Engagement unterstützt. Ihm bin ich für die fachliche Begleitung und freundschaftliche Aufnahme zu großem Dank verpflichtet, dem ganzen IFI in gleichem Maß für die unvergleichliche Mischung aus professioneller Arbeitsatmosphäre und kameradschaftlichem Umgang miteinander. Ich weiß diese außergewöhnlich seltene Kombination zu schätzen.

Der anschließende Aufenthalt am Lehrstuhl für Datenbanksysteme der Universität Erlangen-Nürnberg bot die Gelegenheit, Ideen aus Zürich zur Diskussion zu stellen. Prof. Dr. Jablonski und seine Mitarbeiter haben reges Interesse gezeigt und durch ihre Erfahrung im Bereich Workflow-Management eine Vielzahl konstruktiver Hinweise gegeben, gegründet auf dem tiefen Verständnis der Themas und der Absicht, tragfähige und langfristig überlegene Konzepte zu entwickeln. Die bemerkenswerte und fachübergreifende Diskussionskultur am Lehrstuhl sowie das mir entgegengebrachte Interesse haben den Aufenthalt positiv geprägt.

Über den gesamten Zeitraum hat Prof. Dr. Meyer-Wegener die Arbeit begleitet und mit richtungsweisenden Ideen, konstruktiven Hinweisen und ordnenden Ratschlägen unterstützt. Für sein konstantes Interesse am Thema und die Beharrlichkeit, nach den workflow-spezifischen Aussagen zu suchen, bin ich ihm zu ganz besonderem Dank verpflichtet. Auf ihn geht auch die Anregung zurück, Diskussionspartner außerhalb der eigenen Hochschule zu suchen, um entwickelte Ansätze nicht nur in Form von Publikationen vorzustellen.

Da es zum jetzigen Zeitpunkt weder möglich noch sinnvoll scheint, die Ergebnisse nach ihrem Entstehungsort aufzulösen, wurde die Form eines gemeinsamen Technischen Berichts gewählt, der an allen drei Instituten verfügbar ist.

Dresden, im März 1998

Markus Böhm

Kurzfassung

Die Definition und Implementierung von Ausführungsanweisungen für Workflows war bisher eine Aufgabe für die Entwickler von Workflow-Management-Systemen (WFMS), die die Workflow-Sprache mit Ausdrucksmitteln für unterschiedliche Formen der Workflow-Ausführung ausgestattet haben. Durch die Verfügbarkeit von WFMS mit *erweiterbaren* Workflow-Sprachen kann die Aufgabe der Sprachdefinition oder Spracherweiterung jetzt dem Entwickler der Workflow-Management-Anwendung zufallen, der zuvor nur die angebotenen Ausdrucksmittel der Workflow-Sprache zur Erstellung von Workflow-Schemata genutzt hat.

Dieser Technische Bericht stellt einen Ansatz für die Entwicklung neuer Ausführungsanweisungen vor. Im Mittelpunkt steht ein Funktionsmodell, das die Funktionsweise dieser Sprachelemente auf konzeptioneller Ebene erklärt. Danach sind Ausführungsanweisungen zustandstragende reaktive Objekte, die mit den Workflow-Instanzen interagieren, deren Ausführung sie kontrollieren. Die Ausführungsanweisung definiert in Abhängigkeit von der intendierten Workflow-Ausführung zulässige Konstellation von Zuständen und Zustandsübergängen der kontrollierten Workflow-Instanzen. Das WFMS nutzt diese Information für Steuerungs- und Kontrollaufgaben bei der Ausführung der Workflows.

Mit Hilfe Verteilter Transitionssysteme wird das Funktionsmodell für eine konkrete Ausführungsanweisung mit formalen Mitteln beschrieben. Symbolische Bezeichner für Zustände der Workflow-Instanzen und der Ausführungsanweisung werden zu Transitionen zusammengefaßt; das Schalten einer Transition gibt zulässige Veränderungen des Gesamtsystems aus Workflow-Instanzen und Ausführungsanweisung wieder. Die Darstellung von Ausführungsanweisungen durch ein Verteiltes Transitionssystem ermöglicht weiterhin, Eigenschaften der Ausführungsanweisung abzuleiten und auf Übereinstimmung mit den Anforderungen des Entwicklers an die Workflow-Ausführung zu prüfen.

Über die Entwicklung einzelner Ausführungsanweisungen hinaus werden exemplarisch Ausführungsanweisungen für eine Workflow-Sprache entworfen. Im Gegensatz zu den Ausführungsanweisungen bisheriger Workflow-Sprachen können damit differenzierte Formen der Nebenläufigkeit von Workflows spezifiziert werden.

Abschließend geht der Bericht kurz auf die Implementierung von Ausführungsanweisungen ein. Als Ausführungsumgebung wird ein WFMS mit einem dedizierten Kontrollfluß-Server unterstellt. Der Kontrollfluß-Server kommuniziert mit anderen Komponenten des WFMS und entscheidet, welche Workflow-Instanzen ausgeführt werden und welche Ereignisse bei der Ausführung relevant sind und verarbeitet werden müssen. Vor dem Hintergrund dieser Ausführungsumgebung wird untersucht, welchen Beitrag Beschreibungen der Ausführungsanweisungen als Verteilte Transitionssysteme leisten können.

Abbildungen

Abbildung 2.1: Direkte Erstellung ausführbarer Repräsentationen von Workflow-Schemata 6

Abbildung 2.2: Indirekte Erstellung ausführbarer Repräsentationen von Workflow-Schemata 7

Abbildung 2.3: Übersetzung in eine zusammenhängende Workflow-Schema-Repräsentation 8

Abbildung 2.4: Übersetzung in eine fragmentierte Repräsentation 9

Abbildung 2.5: Beispiele für die Verwendung von Kontrollflußprimitiven 18

Abbildung 2.6: Möglichkeiten zur Beschreibung der alternativen Ausführung zweier Workflows 20

Abbildung 3.1: Realisierungsvarianten einer Workflow-Ausführung 24

Abbildung 3.2: Beziehungen zwischen Zuständen und Zustandsübergängen 27

Abbildung 3.3: Beispiele (1) bis (4) 32

Abbildung 3.4: Transitionen zur Beschreibung des Verhaltens einer elementaren Workflow-Instanz 40

Abbildung 3.5: Instanziierung und Deinstanziierung von Workflows 41

Abbildung 3.6: Entziehen einer Voraussetzung durch Verlassen eines Zustands 42

Abbildung 3.7: Schaffen einer Voraussetzung durch Erreichen eines Zustands 43

Abbildung 3.8: Doppelt restriktiver Zustandsübergang 43

Abbildung 3.9: Synchronisierte Zustandseinnahme 44

Abbildung 3.10: Synchronisierte Zustandsübergänge 44

Abbildung 3.11: Alternative 44

Abbildung 3.12: Ausführungsanweisung SEQUENCE 46

Abbildung 4.1: Ausführungsanweisungen im Kontext eines Workflow-Schemas 52

Abbildung 4.2: Ausführungsmodell für Workflow-Instanzen 56

Abbildung 4.3: OPT(A), Variante II 57

Abbildung 4.4: OPT(A), Variante II 57

Abbildung 4.5: OPT_ASYNC_END(A) 57

Abbildung 4.6: MULT(A) 57

Abbildung 4.7: ALT(A, B) 58

Abbildung 4.8: SEQ(A, B) 58

Abbildung 4.9: SEQ_COMMIT(A, B) 59

Abbildung 4.10: PAR(A, B) 59

Abbildung 4.11: SER(A, B) 60

Abbildung 4.12: ALL(A, B) 61

Abbildung 4.13: DELAY(A, B) 61

Abbildung 4.14: ENFORCE_OVERLAPS(A, B) 62

Abbildung 4.15: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_OVERLAPS(A, B) 62

Abbildung 4.16: OPTIONAL_DURING(A, B) 63

Abbildung 4.17: Intervalldarstellung der Workflow-Ausführung durch OPTIONAL_DURING(A, B) 63

Abbildung 4.18: ENFORCE_ENDS(A, B) 63

Abbildung 4.19: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_ENDS(A, B) 63

Abbildung 4.20: ENFORCE_SWITCH(A, B) 64

Abbildung 4.21: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_SWITCH(A, B) 64

Abbildung 4.22: Beispiel für eine Klassifikation von Ausführungsanweisungen 66

Abbildung 4.23: Schablone der SEQ-Ausführungsanweisung 69

Abbildung 4.24: Schablone der ALT-Ausführungsanweisung 69

Abbildung 4.25: Schablone der PAR-Ausführungsanweisung 69

Abbildung 4.26: Traversieren einer kompositen Ausführungsanweisung 70

Abbildung 4.27: Transitionen für SEQ(ALT(A, B), PAR(C, SEQ(D, E))) 70

Abkürzungen

AA	Ausführungsanweisung
ECA	Event Condition Action
FDL	FlowMark Definition Language (Workflow-Sprache des WFMS FlowMark)
KFK	Kontrollflußkonstrukt
KFP	Kontrollflußprimitiv
OIS	Office Information System
PN	Petri-Netz
S/S	State/State (Beziehung zwischen Zuständen)
S/T	State/Transition (Beziehung zwischen Zustand und Zustandsübergang)
T/S	Transition/State (Beziehung zwischen Zustandsübergang und Zustand)
T/T	Transition/Transition (Beziehung zwischen Zustandsübergängen)
WF	Workflow
WFMA	Workflow-Management-Anwendung
WfMC	Workflow Management Coalition
WFMS	Workflow-Management-System

Inhaltsübersicht

Vorwort i

Kurzfassung ii

Abbildungen iii

Abkürzungen v

Inhaltsverzeichnis vi

Problemstellung und Motivation viii

Ergebnisse in der Übersicht ix

1 Einführung 1

2 Grundlagen 5

2.1 Methoden zur Erstellung von Workflow-Schemata 5

2.1.1 Direkte Erstellung von Workflow-Schemata als Ausdrücke operationaler Zwischensprachen 6

2.1.2 Indirekte Erstellung von Workflow-Schemata als Ausdrücke operationaler Zwischensprachen 7

2.1.3 Übersetzung in eine kompakte Repräsentation des Workflow-Schemas 8

2.1.4 Übersetzung in eine fragmentierte Repräsentation des Workflow-Schemas 8

2.2 Ausführung von Workflows aus konzeptioneller Sicht 10

2.2.1 Ausführungsmodelle von Workflow-Instanzen 10

2.2.2 Faktoren zur Bestimmung ausführbereiter Workflows 14

2.2.3 Kontrollflußzentrierte Ausführung von Workflows 15

2.3 Kontrollflußdefinition in Workflow-Schemata 16

2.3.1 Kontrollflußprimitive 17

2.3.2 Kontrollflußkonstrukte 19

2.3.3 Ausführungsanweisungen 19

2.3.4 Gegenüberstellung am Beispiel 20

2.3.5 Vergleichende Bewertung 21

3 Entwicklung neuer Ausführungsanweisungen 24

3.1 Motivation für die Entwicklung einer Ausführungsanweisung 24

3.2 Funktionsmodell für Ausführungsanweisungen 25

3.2.1 Ausdrucksmöglichkeiten im Funktionsmodell 26

3.2.2 Beispiele für Beziehungstypen bei der Ausführung von Workflows 29

3.2.3 Funktionsmodell-orientierte Entwicklung von Ausführungsanweisungen 33

3.2.4 Zusammenfassung 36

3.3 Verteilte Transitionssysteme 37

3.3.1 Informelle Einführung 38

3.3.2 Verwendung zum Entwurf neuer Ausführungsanweisungen 38

3.4 Modellierung einer Ausführungsanweisung 40
3.4.1 Beschreibung des Verhaltens von Workflow-Instanzen 40
3.4.2 Formulierung von Anforderungen an die Workflow-Ausführung 40
3.4.3 Beschreibung einer Ausführungsanweisung als Transitionssystem 45
3.4.4 Aussagen über die Ausführungsanweisung 46

4 Konzeption von Ausführungsanweisungen für eine Workflow-Sprache 51

4.1 Entscheidungen zum Design der Ausführungsanweisungen 51
4.1.1 Kontrollflußdefinition im Workflow-Schema 52
4.1.2 Entwurfsentscheidungen für Ausführungsanweisungen 52

4.2 Ausführungsanweisungen für ausgewählte Formen der Workflow-Ausführung 54
4.2.1 Ausführungsmodell der Workflow-Instanzen 55
4.2.2 Elementare Ausführungsformen 56
4.2.3 Elementare Ausführungsfolgen 58
4.2.4 Deskriptive Serialisierung 60
4.2.5 Präskriptive Parallelausführung 61
4.2.6 Klassifikationsmöglichkeiten für Ausführungsanweisungen 64

4.3 Simulation der Funktionsweise von Ausführungsanweisungen 68
4.3.1 Aufbau zusammenhängender Ausführungsanweisungen 68
4.3.2 Beispiel einer automatisch erzeugten Kontrollflußbeschreibung 68

4.4 Implementierung der Ausführungsanweisungen 71

5 Abgrenzung von anderen Arbeiten 73

5.1 Kontrollflußprimitive, Kontrollflußkonstrukte, Ausführungsanweisungen 73

5.2 Rahmenwerke zur Spezifikation von Ausdrucksmitteln 74

5.3 Spezifikation von Task-Dependencies 75

5.4 Simulation der Workflow-Ausführung 76

6 Offene Probleme und weitere Aktivitäten 78

7 Zusammenfassung 80

Problemstellung und Motivation

Das wichtigste Gestaltungsmittel bei der Entwicklung von Workflow-Management-Anwendungen ist die Workflow-Sprache. Mit der Workflow-Sprache beschreibt der Anwendungsentwickler die Arbeitsvorgänge, die das Workflow-Management-System unterstützen soll. Ist die Workflow-Sprache — wie in allen kommerziellen WFMS und den meisten Forschungsprototypen — in Syntax und Semantik unveränderbar festgelegt, setzt das den Gestaltungsmöglichkeiten enge Grenzen. Erweiterbare Workflow-Sprachen heben diese Beschränkung auf; sie ermöglichen die Einführung neuer Ausdrucksmittel, ganz nach den Wünschen und Bedürfnissen des Anwendungsentwicklers. Erkauft wird diese Freiheit durch (noch) mehr Komplexität bei der Anwendungsentwicklung. Der Anwendungsentwickler muß nun zwei Phasen planen: Sprach*entwurf* und Sprach*verwendung*. Zur Verwendung von Workflow-Sprachen gibt es inzwischen Erfahrungen, der Entwurf von Workflow-Sprachen ist jedoch kaum untersucht, noch in Praxisprojekten zum Einsatz gekommen. Dabei stellen sich sehr interessante Fragen, z.B.:

- Wann ist die Einführung eines neues Ausdrucksmittels überhaupt sinnvoll?
- Kann durch Kombination vorhandener Ausdrucksmittel das gleiche Ziel erreicht werden?
- Wie plant und entwickelt man ein neues Konstrukt für eine Workflow-Sprache?
- Welche neuen Ausdrucksmöglichkeiten entstehen durch das neue Konstrukt?
- Wie stellt man sicher, daß neue und alte Konstrukte zusammenpassen?
- Wie testet, analysiert und simuliert man Funktion und Verhalten des neuen Konstrukts?
- Wie bildet man anwendungsbezogene Anforderungen auf den Entwurfsvorgang ab?

Diesen Fragen geht der vorliegende Technische Bericht nach. Er fokussiert dazu das eng umrissene Problem der Workflow-Ausführung, also der Festlegung einer Reihenfolge, in der Subworkflows eines kompositen Workflows zur Ausführung gelangen. Vorgaben zur Reihenfolge entstammen dem Arbeitsvorgang, der Anwendungsentwickler braucht daher Sprachmittel, diese Vorgaben im Workflow-Schema zu verankern. Das geschieht mit Hilfe von Ausführungsanweisungen wie z.B. SEQUENCE(Workflow-1, Workflow-2) für die Hintereinanderausführung von zwei Workflows. Verlangt der Arbeitsvorgang keine Reihenfolge, wohl aber die Ausführung beider Workflows, ist SEQUENCE offensichtlich ungeeignet. Sinnvoller wäre ein Konstrukt BOTH_WITHOUT_ORDER(Workflow-1, Workflow-2). Mit Blick auf obige Fragen heißt das: Konzeption, Planung, Entwurf und Test einer *neuen* Ausführungsanweisung für die Ausführung zwei Workflows ohne Reihenfolge. Allerdings liegt das Ziel dieses Berichts weniger in der Beantwortung der Fragen für den Einzelfall, vielmehr werden allgemeingültige Zusammenhänge von Workflow-Management-System, Workflow-Sprache und anwendungsspezifischen Spracherweiterungen herausgearbeitet. Diesbezügliche Ergebnisse werden auf der nächsten Seite stichwortartig vorgestellt, sie sind bewußt auf das Wesentliche zugespitzt und nicht als Ersatz für den Bericht gedacht.

Ergebnisse

Folgende Punkte stellen in Kurzform ausgewählte Ergebnisse vor; die ausführliche Behandlung erfolgt im Bericht, auf die jeweiligen Stellen wird verwiesen.

1. Workflow-Management-Anwendungen erfordern erweiterbare Workflow-Sprachen, die dem Anwendungsentwickler bei Bedarf die Einführung spezifischer Ausdrucksmittel erlauben. Jeder vordefinierte Sprachumfang bevorzugt eine Anwendungsdomäne oder eine Klasse von Arbeitsvorgängen. Workflow-Sprachen ohne Erweiterungsmöglichkeit mißachten die Grundsätze des Workflow-Managements: Flexibilität, Anpassungsfähigkeit und Anwendungsneutralität (→ Abschnitt 3.1, Abschnitt 4.2).

2. Erweiterbare Workflow-Sprachen eröffnen einen neuartigen Weg, Funktionen einer Workflow-Management-Anwendung zu realisieren. Als WFMS-interne Umgebung zur Programmierung reduzieren sie den Bedarf, Steuerungsfunktionen in externe Applikationen auszulagern oder auf störende Weise im Workflow-Schema zu realisieren. Der Anteil WFMS-kontrollierter Anwendungslogik wird durch erweiterbare Workflow-Sprachen weiter gesteigert, weil auf die Notlösung, zusätzliche Steuerungsprogramme zu erstellen, verzichtet werden kann (→ Abschnitt 4.1).

3. Als Alternative zu Kontrollflußprimitiven und Kontrollflußkonstrukten werden Ausführungsanweisungen eingeführt. Ausführungsanweisungen definieren das Verhalten kompositer Workflows ergebnisorientiert; die Kontrollflußdefinition zwischen Subworkflows entfällt. Die Einführung neuer Ausführungsanweisungen erlaubt es, auf ein einziges Workflow-Schema oder eine ganze Workflow-Management-Anwendung zugeschnittene Funktionalität zu kapseln und dem Anwendungsentwickler als Spracherweiterung anzubieten (→ Abschnitt 2.3.3, Kapitel 3).

4. Ein konzeptionelles Funktionsmodell für Ausführungsanweisungen wird definiert und als Entwurfshilfsmittel eingesetzt. Das Funktionsmodell dient zudem als Bezugssystem, um unterschiedliche Ansätze zur Kontrollflußdefinition in Workflows zu vergleichen und Aussagen über ihre Ausdrucksmächtigkeit abzuleiten (→ Abschnitt 3.2).

5. Die bestimmende Rolle der Ausführungsmodelle von Workflow-Instanzen bei der Gestaltung neuer Ausführungsanweisungen wird herausgearbeitet. Aus dem identifizierten Zusammenhang von Zuständen im Ausführungsmodell und Formen der Workflow-Ausführung geht hervor, wie sich Design oder Konfiguration eines WFMS auf seine Einsatzbreite auswirkt (→ Abschnitt 3.2.3).

6. Verteilte Transitionssysteme spezifizieren Verhalten und Funktionsweise von Ausführungsanweisungen. Die Spezifikation ist ausführbar und einer interaktiven Simulation oder Analyse zugänglich. Wichtige Eigenschaften der Ausführungsweisung oder der von ihr erzeugten Workflow-Ausführung können damit validiert werden, bevor der Anwendungsentwickler mit der Implementierung beginnt (→ Abschnitt 3.3, Abschnitt 3.4.4).

7. Ausführungsanweisungen werden aus Bausteinen zusammengesetzt, die einzelne Facetten der intendierten Workflow-Ausführung beschreiben, insbesondere Instanziierung, Reihenfolge, Interaktion, Abhängigkeiten, Synchronisation und Terminierung. Formuliert werden die Ausprägungen der genannten Facetten über Zustandskonstellationen der zu kontrollierenden Workflow-Instanzen. Durch unterschiedliche Bausteinkombinationen nimmt der Entwickler auf die Eigenschaften der Ausführungsanweisung gezielt Einfluß (→ Abschnitt 3.4.1 bis Abschnitt 3.4.3).

8. Für eine fiktive Workflow-Sprache wird ein initialer Satz von Ausführungsanweisungen entworfen. Als Beispiel für die anwendungsspezifische Ausrichtung dieser Workflow-Sprache gibt es zusätzliche Ausführungsanweisungen, mit denen die Parallelausführung zweier Workflows sehr differenziert beschrieben wird. Die Workflows werden nicht, wie bisher üblich, nur gleichzeitig bereitgestellt, sondern das Interaktionsmuster während der Ausführungszeit ist ebenfalls Teil der Festlegung (→ Abschnitt 4.2).

9. Die Simulation von Funktion und Verhalten erstreckt sich nicht nur auf eine einzige Ausführungsanweisung, sondern auch auf komposite Ausdrücke mehrerer Ausführungsanweisungen. Das bedeutet für den Entwickler erstmals die Möglichkeit, bereits vor der Implementierung eines Workflow-Schemas verschiedene Kontrollflußdefinitionen zu vergleichen und in ihrer Auswirkung auf die zukünftige Workflow-Ausführung abzuschätzen (→ Abschnitt 4.3).

10. Charakteristische Merkmale der Ausführung von Workflows werden identifiziert und zu einem Klassifikationsschema kombiniert. Die Art der Merkmalskombination liefert verschiedenartige Klassifikationsschemata; jedes Klassifikationsschema gliedert die Ausführungsanweisungen nach anderen Gesichtspunkten. Zielgruppengerechte oder verwendungsspezifische Klassifikationsschema sind damit möglich (→ Abschnitt 4.2.6).

Fazit

Die Ergebnisse lassen den Schluß zu, daß die praktische Nutzung erweiterbarer Workflow-Sprachen erheblichen Zusatzaufwand für den Anwendungsentwickler bedeuten kann. Der Aufwand fällt einmalig bei der Entwicklung neuer Ausdrucksmittel an, trägt danach aber zur Verbesserung der Ausdrucksmöglichkeiten und zur Erhöhung der Beschreibungsqualität im Workflow-Schema bei. Die Entwicklung neuer Ausdrucksmittel ist daher als „Investition“ in die Fortentwicklung der Workflow-Sprache zu sehen und aus diesem Grund vertretbar.

1 Einführung

Eine wichtige Aufgabe der angewandten Informatik ist die Entwicklung betriebswirtschaftlicher Anwendungssysteme, mit denen Unternehmen ihre Geschäftsprozesse effizienter gestalten und neue Geschäftsfelder erschließen. Für die Implementierung dieser Anwendungssysteme müssen Arbeitsvorgänge, Organisationsstrukturen und Ressourcen des Unternehmens mit einer Sprache beschrieben werden, die ein EDV-System interpretieren kann. Dem jeweiligen Stand der Informatik entsprechend, sind Sprachen unterschiedlicher Generationen, Entwicklungslinien und Programmierparadigmata in Gebrauch. Von Ausnahmen abgesehen sind diese Sprachen nicht spezialisiert; allenfalls dominieren sie einzelne Anwendungsgebiete (z.B. Fortran die Naturwissenschaften oder früher COBOL betriebswirtschaftliche Anwendungen). Als Folge der universellen Einsetzbarkeit muß der Anwendungsentwickler die Kluft zwischen Programmiersprache und Anwendung überbrücken — oft durch „programming-in-the-*small*“ [Dere76], was die verfügbaren Ausdrucksmittel und die notwendige Tätigkeit hinreichend deutlich charakterisiert.

Seit Beginn der 80-er Jahre wird versucht, administrative (Büro)-Vorgänge im Unternehmen für betriebswirtschaftliche Anwendungssysteme nicht mehr durch universelle Programmiersprachen zu beschreiben. Zunächst übernahm man vollkommen unkritisch Modellierungskonzepte für Abläufe in technischen Systemen; erst später wurde die Notwendigkeit spezieller Ausdrucksmittel für Vorgänge, Mitarbeiter, Aufgaben, Dokumente, Zuweisungsregeln, Durchführungsrichtlinien und Hilfsmitteln zur Aufgabenerledigung zumindest erkannt. Da anfangs die Leistungsfähigkeit der Hard- und Software sowie das theoretische Hintergrundwissen für eine wirkliche Unterstützung von gemeinschaftlich und verteilt ausgeführten Arbeitsvorgängen nicht ausreichten, setzten sich die sogenannten *Office Information Systems* (OIS) nicht durch und blieben ein Forschungsthema (siehe [Barb83, Crof84, Maze84, Nieh85, Ronz85, Besl87, Woet87, Loch88, Lutz88, Nier89], für Vergleiche einzelner Ansätze und Prototypen [Brac84, S. 153–166], [Hase87, S. 77–131], [Schw93, S. 29–57], [Rein93, S. 75–89] und [Ober96, S. 34–46]).

OIS sind, neben Systemen zur Unterstützung der Software-Erstellung, Dokumenten-Management und erweiterten Transaktionskonzepten, eine Keimzelle heutiger *Workflow-Management-Systeme* (WFMS). Die inzwischen unüberschaubare Literatur erlaubt nur den Verweis auf Standardwerke [Rein93, Jabl96a, Ober96, Jabl97] oder Aufsatzsammlungen (z.B. zur Geschäftsprozeßmodellierung [Voss96], Praxiserfahrung [Öste96] oder Einführungsvorbereitung [Herr98] von WFMS). Für WFMS werden Arbeitsvorgänge mit *Workflow-Sprachen* explizit beschrieben und vom anwendungsunabhängigen Anteil des Programmcodes einer *Workflow-Management-Anwendung* (WFMA) separiert. Diese Trennung erleichtert es, auf Änderungen im Arbeitsvorgang zu reagieren. Allerdings bieten heutige Workflow-Sprachen mangels allgemein akzeptierter Standards unterschiedlich geartete Sprachmittel und Konzepte an, um relevante Inhalte des Arbeitsvorgangs für die Unterstützung durch ein WFMS auszudrücken. Allen Workflow-Sprachen ist jedoch gemein, daß sie für Objekte der Anwendungswelt eigene Sprachmittel aufweisen und vom Anwendungsentwickler keine Rekonstruktion auf der Basis elementarer Datenstrukturen und Programmanweisungen verlangen. Im Idealfall erlauben

Workflow-Sprachen das „programming-in-the-*large*“ nach [Dere76] und führen so zu einer neuartigen Form der Anwendungsentwicklung [Wede96].

Von schlecht umgesetzten oder grundsätzlich ungeeigneten Konzepten abgesehen, verkleinern Workflow-Sprachen den Abstand von Beschreibungsmittel und Anwendungswelt tatsächlich. Der Entwickler einer WFMA profitiert von Anwendungsnähe und Problemorientierung der Workflow-Sprache sowohl bei der Erstellung der Anwendung als auch bei Modifikationen und Erweiterungen in Folge von Änderungen der Arbeitsvorgänge — solange die Workflow-Sprache zur Beschreibung der Arbeitsvorgänge ausreicht. Weichen die Eigenschaften der zu unterstützenden Arbeitsvorgänge von dem ab, was von den Entwicklern der Workflow-Sprache vorgesehen wurde, entsteht z.B. folgendes Problem: Arbeitsvorgänge mit schwach strukturierten Aktivitäten, nur bedingt planbaren Phasen oder komplizierten Sonderfällen werden nur unzureichend beschrieben, wenn die Workflow-Sprache lediglich Ausdrucksmittel für klar strukturierte und häufig wiederholte Tätigkeiten vorsieht (zur Behandlung unterschiedlicher Klassen von Arbeitsvorgängen siehe [Deit96]). Ein anderes Defizit bei Workflow-Sprachen sind Ausdrucksmittel, die zwar den Arbeitsvorgängen gerecht werden, aber zu schlecht wartbaren und damit fehleranfälligen Beschreibungen führen oder dem Anwendungsentwickler viel technisches Detailwissen über das WFMS abverlangen.

Defizite in den Ausdrucksmitteln und den pragmatischen Qualitäten der Workflow-Sprache verhindern eine adäquate Beschreibung des Arbeitsvorgangs. Das führt bei dynamischen Arbeitsvorgängen dazu, daß die Workflow-Sprache nur eine von mehreren Durchführungsvarianten beschreiben kann. Die Anwender sind dann gezwungen, ungeachtet der aktuellen Situation, nach dieser möglicherweise unpassenden Variante zu arbeiten. Alternativ kann der Anwendungsentwickler mit einem „work-around“ fehlende Ausdrucksmittel der Workflow-Sprache kompensieren und erforderliche Funktionalität außerhalb des WFMS realisieren. Beispiele für außerhalb des WFMS realisierte Funktionalität sind Batch-Programme, die die Steuerung des Arbeitsvorgangs anstelle des WFMS übernehmen. Rufen diese WFMS-externen Steuerungsprogramme weitere Anwendungsprogramme auf, verändern sie ablaufrelevante Daten und beeinflussen sie den Fortgang der Arbeit ohne Beteiligung des WFMS, sind inkonsistente Bearbeitungszustände und divergierende Teilergebnisse im Arbeitsvorgang sowie unvollständige Protokolle im WFMS die Folge. Selbst durch die Einführung zusätzlicher Kontrollmechanismen bleiben weitere Nachteile: Die Aufsplittung von Applikationslogik zwischen WFMS und externen Steuerungsprogrammen führt zurück zu einem schwer wartbaren, heterogenen Applikationsverbund. Noch ein dritter „Ausweg“ ist bei unzureichenden Sprachmitteln möglich: Eine fehlende Funktion wird mit der Workflow-Sprache und ohne „work-around“ erbracht, durch Mißbrauch oder Uminterpretation vorhandener Ausdrucksmittel. Die drei Vorschläge verhindern eine leistungsfähige, flexible und wartbare WFMA und sollten vermieden werden.

Die Schwierigkeiten bei der Beschreibung von Arbeitsvorgängen für WFMS haben zwei Ursachen: Zum einen werden viele Workflow-Sprachen von Entwicklern konzipiert, die ein eingeschränktes Verständnis von Workflow-Management haben, das z.B. aus der Gleichsetzung mit (erweiterten) Transaktionen wie u.a. in [Brei93, Geor94, Kris94, Alon96, Puus97] resultiert. Die Einschränkung äußert sich oftmals in ganzen Klassen von Arbeitsvorgängen, die a

priori von der Unterstützung durch ein WFMS ausgeschlossen werden (vgl. [Böhm97a, S. 77ff.]). Die Folge ist ein primitives bzw. einseitig ausgerichtetes Workflow-Metaschema der Workflow-Sprache. Zum anderen ist die Workflow-Sprache häufig unveränderbarer Bestandteil des WFMS und damit keiner Erweiterung oder nachträglichen Anpassung zugänglich, obgleich gerade Anpassungsfähigkeit und Flexibilität zentrale Konzepte des Workflow-Managements sind (vgl. [Jabl96a, S. 110ff.]).

Einen Ausweg bieten *erweiterbare* Workflow-Sprachen, die dem Anwendungsentwickler die Einführung neuer Sprachelemente erlauben. Die für Erweiterungen notwendige Offenheit des WFMS gestattet es, die Semantik der vorhandenen Sprachelemente neuen Anwendungssituationen anzupassen. Idealerweise wird die Funktion des neuen Sprachelements nicht als Makro auf der gleichen Sprachebene realisiert, sondern im WFMS-Kern verankert. Die Details der Implementierung müssen dem Anwendungsentwickler verborgen bleiben, damit er das neue Konstrukt als homogene Erweiterung der Workflow-Sprache empfindet und einsetzen kann.

Der Vorteil einer erweiterbaren Workflow-Sprache besteht nicht nur in der Einführung neuer Konstrukte: Leistungen des WFMS können auf einem neuen (dritten) Weg realisiert werden, nämlich erstens außerhalb des WFMS in externen Anwendungsprogrammen, die vom WFMS aufgerufen werden und die Kontrolle nach Ausführungsende an das WFMS zurückgeben (trotz der eingangs genannten Nachteile ist dies manchmal unvermeidbar), zweitens mit Hilfe der Workflow-Sprache durch Kombination *mehrerer* vorhandener Sprachelemente oder drittens durch die Realisierung der bislang fehlenden Funktion in *einem* neuen Sprachelement. Diese Alternativen stellen vollkommen verschiedene Wege der Umsetzung dar, auch hinsichtlich nichtfunktionaler Eigenschaften wie Wartbarkeit, Wiederverwendbarkeit usw. Für die Entscheidung, auf welchem Weg eine Funktion zu realisieren ist, gibt es keine allgemeingültigen Regeln; allenfalls lassen sich Empfehlungen geben: Ausgefallene Funktionen sollten außerhalb des WFMS realisiert werden, um die Übersichtlichkeit der Workflow-Sprache zu erhalten. Gleiches gilt für Funktionen, die den Arbeitsvorgang nicht direkt betreffen und deren Durchführung nicht vom WFMS überwacht werden muß. Läßt sich eine Funktion von der speziellen Verwendung lösen, so daß ihre Implementierung im WFMS auch anderen Anwendungen nutzt, ist die Erweiterung der Workflow-Sprache sinnvoll.

In fast jeder Anwendungssituation für WFMS müssen temporale und kausale Abhängigkeiten, Synchronisations- und Koordinationsbeziehungen zwischen einzelnen Aufgaben beschrieben werden. Workflow-Sprachen, die nur unspezifische Beschreibungsprimitive für diese Abhängigkeiten anbieten, führen zu komplexen, unübersichtlichen und wartungsunfreundlichen Workflow-Schemata. Auch die Beschränkung auf Ausdrucksmittel für sequentielle, alternative und parallele Ausführung von Workflows ist für den praktischen Einsatz unzureichend. Hier ist zu beachten, daß es nicht um die Beschreibung atomarer und zeitloser Aktionen geht und das weiterhin nicht die für theoretische Betrachtungen sinnvolle Reduktion auf einen kleinen Satz von Beschreibungsmitteln angestrebt wird. Die Beziehungen in der Realität sind erstens zu kompliziert und zweitens zu unterschiedlich, als daß ein fester Satz an elementaren Beschreibungsmitteln ausreichend wäre. Zudem hat die Anschaulichkeit, Lesbarkeit und Änderungsfähigkeit durch den Anwendungsentwickler einen zu großen Stellenwert, um eine theoretisch durchaus mögliche Reduzierung auf Beschreibungsmittel für Alternative, Sequenz

und Parallelität zu gestatten. Minimalität und Vollständigkeit von Beschreibungsmitteln für Prozesse (jetzt im Verständnis wie z.B. in ACP [Baet90] und CSP [Hoar85]) sind daher für die Beschreibung von Workflows neu zu interpretieren und nicht einfach zu übertragen.

Hat die Workflow-Sprache hingegen nur Anweisungen für ausgewählte Ablaufsituationen eines bestimmten Anwendungsgebiets, ist dies auf den ersten Blick zwar komfortabel, schränkt aber die Vielseitigkeit und Einsetzbarkeit des WFMS ein. In beiden Fällen schaffen erweiterbare Workflow-Sprachen Abhilfe; die Einführung von Ausführungsanweisungen für Workflows erlaubt es, situationsgerechte Ausdrucksmittel zu schaffen und einzusetzen. Die vor der Einführung notwendige Entwicklung von Ausführungsanweisungen umfaßt eine Reihe von Entwurfsentscheidungen, die zu unterschiedlichen Realisierungsvarianten dieser Spracherweiterungen führen und Gegenstand dieses Technischen Berichts sind. Die Entwicklung von Ausführungsanweisungen wird stellenweise anhand von MOBILE [Jabl96a] erläutert, weil dieses WFMS die Erweiterung seiner Workflow-Sprache erlaubt und zudem gut dokumentiert ist. Im Vordergrund stehen die Vorgehensweise, der Einsatz formaler Hilfsmittel bei Entwurf und Test von Ausführungsanweisungen sowie eine ausführliche Behandlung der vorgestellten Beispiele. Die Schritte bei Entwurf und Gestaltung neuer Ausführungsanweisungen sind unabhängig von MOBILE formuliert und damit auf andere WFMS übertragbar.

Der Technische Bericht ist wie folgt gegliedert: Kapitel 2 behandelt Grundlagen der Kontrollflußdefinition für Workflows, klassifiziert und bewertet unterschiedliche Ansätze dahinter und motiviert die weitere Beschäftigung speziell mit Ausführungsanweisungen. Ausführungsanweisungen werden mit Hilfe eines Funktionsmodells in Kapitel 3 erklärt sowie bezüglich Funktionsweise und zu erwartender Workflow-Ausführung erläutert. Kapitel 3 führt weiterhin Verteilte Transitionssysteme ein, um das Funktionsmodell einer Ausführungsanweisung formal zu beschreiben. Kapitel 4 diskutiert die Konzeption von Ausführungsanweisungen für eine Workflow-Sprache und stellt eine Auswahl von Anweisungen für unterschiedliche Arten der Workflow-Ausführung vor. Kapitel 5 stellt Arbeiten vor, die im Technischen Bericht berührte Themen behandeln. Offene Probleme und die Richtung der weiteren Forschung werden in Kapitel 6 dargelegt, bevor Kapitel 7 eine Zusammenfassung bringt. Zentrale Aussagen und Ergebnisse sind zudem am Anfang des Berichts in einer Übersicht stichpunktartig aufgelistet.

2 Grundlagen

Die *Entwicklung von Ausführungsanweisungen* ist ein spezielles Thema im Workflow-Management, dessen Behandlung eine systematische Vorbereitung erfordert. Zum einen sind *Ausführungsanweisungen* nur eine Möglichkeit, das Verhalten der Workflows zum Ausführungszeitpunkt zu bestimmen — eine Abgrenzung von anderen Möglichkeiten ist deshalb notwendig. Zum zweiten ist die *Entwicklung* von Ausführungsanweisungen ein neues Gebiet, da nur das WFMS MOBILE als Forschungsprototyp diese Form der Erweiterbarkeit anbietet — dies erfordert eine Behandlung WFMS-interner Mechanismen, die für den Entwickler einer Workflow-Management-Anwendung bislang ohne Bedeutung waren.

Das Kapitel nimmt die notwendige Einordnung des Themas in drei Schritten vor: Abschnitt 2.1 geht auf die Erstellung von Workflow-Schemata ein, weil die spätere Verwendung von Ausführungsanweisungen in diesen Bereich (der „Buildtime“) fällt. Abschnitt 2.2 wechselt den Betrachtungszeitpunkt (zur „Runtime“), behandelt die Ausführung von Workflows aus konzeptioneller Sicht und arbeitet die Rolle der Ausführungsanweisungen heraus. Abschnitt 2.3 stellt alternative Möglichkeiten zur Festlegung der Workflow-Ausführung vor und begründet durch eine vergleichende Bewertung die Entscheidung für Ausführungsanweisungen.

2.1 Methoden zur Erstellung von Workflow-Schemata

Workflow-Schemata sind ausführbare Repräsentationen von Arbeitsvorgängen. Workflow-Schemata werden vom WFMS zur Laufzeit interpretiert und zur Steuerung und Kontrolle der Workflow-Ausführung verwendet. Als zentrale Beschreibung beinhaltet das Workflow-Schema steuerungsrelevante Informationen zum Arbeitsvorgang, der in seiner WFMS-gestützten Ausführung *Workflow* heißt und WFMS-intern durch eine oder mehrere *Workflow-Instanzen* repräsentiert wird [Ortn97, S. 491]. Daß keine Übereinkunft über Inhalte und Darstellung von Workflow-Schemata besteht, beweisen die unterschiedlichen Vorschläge, die derzeit in der Literatur diskutiert werden. Die zur Zeit umfassendste Sicht auf Workflows bieten die von [Jabl94] eingeführten Erweiterungen der Aspekte, die in [Curt92] und [Malo94] zur Charakterisierung von Prozessen vorgeschlagen wurden.

Die unterschiedlichen Inhalte eines Workflow-Schemas werden zudem mit einer Vielzahl von Mitteln beschrieben (für eine Übersicht siehe z.B. [Rein93, S. 69ff.], [Jabl96a, S. 114ff.], [Böhm97a, S. 77ff.]). Einen Abriß der Methoden zur Erstellung von Workflow-Schemata geben die folgenden Abschnitte, allerdings nur in dem Maß, wie es das Thema „Ausführungsanweisungen“ erfordert. Ausführungsanweisungen kommen nicht bei allen Methoden vor, insbesondere fehlen sie beim Einsatz workflow-unspezifischer Beschreibungsmittel. Diese Methoden werden nicht weiter betrachtet, weiterführende Literatur ist in den jeweiligen Abschnitten angegeben.

2.1.1 Direkte Erstellung von Workflow-Schemata als Ausdrücke operationaler Zwischensprachen

In einigen WFMS hat der Entwickler keine Workflow-Sprache zur Verfügung, sondern er selbst muß Workflow-Schemata in eine Repräsentation überführen, die von der Ausführungskomponente (Basismaschine) interpretiert wird. Das bedeutet, Workflow-Schemata als Anordnung *workflow-spezifischer* Begriffe einer Sprache existieren bei dieser Methode eigentlich nicht.

Statt dessen erfolgt die Beschreibung eines „Workflow-Schemas“ direkt in einer Zwischensprache, die eher an den Eigenschaften der vorhandenen Ausführungsumgebung als denen des Entwicklers orientiert ist. Das führt dazu, daß der Entwickler durch Kombination der angebotenen Ausdrucksmittel die Aspekte eines Workflows nach eigenem Ermessen repräsentiert, jeder Entwickler dazu noch auf seine Art, mit entsprechenden Konsequenzen für die Portabilität. Das Problem tritt immer dann auf, wenn für Workflow-Schemata Beschreibungsmittel verwendet werden, die für andere Zwecke entwickelt wurden oder absichtlich für eine universelle Verwendung ausgelegt sind. So haben z.B. (höhere) Petri-Netze [Reis82], State-Activity-Charts [Hare87, Hare88] usw. nichts mit Workflows zu tun, auch die in einigen WFMS eingesetzten ECA-Regeln sind keine WFMS-spezifischen Konzepte. Obwohl es sich bei den genannten Beispielen nur um *Basismechanismen* zur Realisierung einzelner Teilaufgaben handelt, gibt es Versuche, mit diesen Mitteln ganze Workflow-Schemata zu erstellen (z.B. mit Petri-Netzen: [Ober96], mit State-Activity-Charts: [Wodt95], mit ECA-Regeln: [Barb94, Casa95, Eder94, Kapp95a]). In der Folge dieses Vorgehens dienen dann Petri-Netz-Interpreter, Simulationsumgebungen für State-Actvitity-Charts (z.B. STATEMATE [Hare90] in [Wodt95]), aktive Datenbanken (z.B. GEMSTONE [Butt91] in TRIGS$_{flow}$ [Kapp95a]) als Ausführungsumgebung, die mit hohem Aufwand an die Erfordernisse von WFMS angepaßt werden.

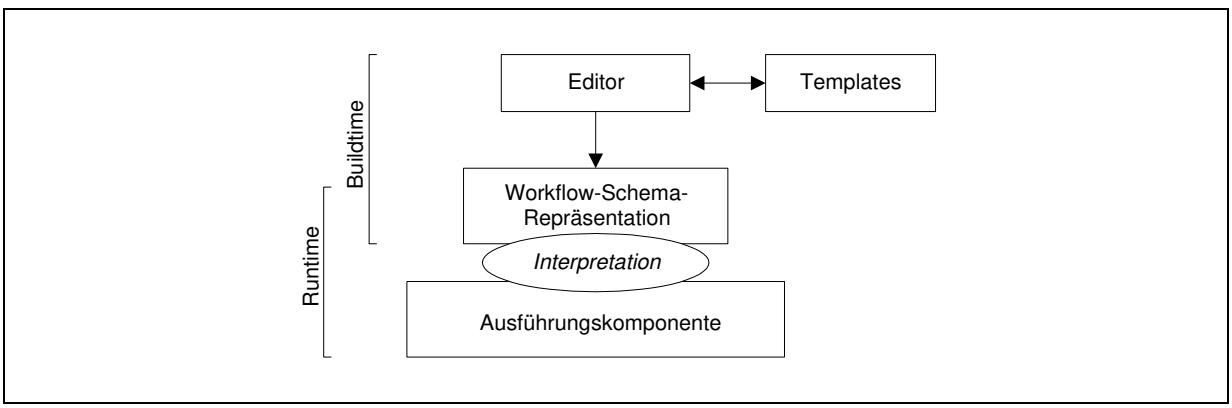


Abbildung 2.1: Direkte Erstellung ausführbarer Repräsentationen von Workflow-Schemata

Abbildung 2.1 zeigt schematisch, wie mit Templates und einem Editor eine Repräsentation eines Workflow-Schemas erstellt wird. Daß der Anwendungsentwickler dazu eine Sprache verwendet, die von der Ausführungsumgebung interpretiert wird, bringt die Überschneidung der Zeitabschnitte „Buildtime“ und „Runtime“ zum Ausdruck. Beschreibungen einzelner Aspekte eines Workflows sind nicht voneinander zu unterscheiden und werden von der Ausführungskomponente zwangsläufig gleich behandelt (für weitere Nachteile siehe [Schu97, S.

27ff.]). Durch das Fehlen einer eigenen Workflow-Sprache entfallen auch Ausführungsanweisungen, weshalb WFMS mit dieser Konzeption hier nicht weiter betrachtet werden.

2.1.2 Indirekte Erstellung von Workflow-Schemata als Ausdrücke operationaler Zwischensprachen

Ein erster Abstraktionsschritt von der Sprache der Ausführungskomponente besteht darin, dem Entwickler *Workflow-Schema-Konstrukte* anzubieten. Workflow-Schema-Konstrukte bestehen aus vordefinierten Teilausdrücken der Sprache, die von der Ausführungsumgebung interpretiert wird (Abbildung 2.2). Im Gegensatz zur direkten Erstellung verwendet der Entwickler vorwiegend diese Konstrukte und wird zusätzlich durch einen Mechanismus zur syntaktisch korrekten Komposition dieser Teilausdrücke entlastet, die z.B. in einem Editor implementiert sind. Workflow-Schema-Konstrukte haben den Charakter von Makros, die entweder durch einen Präprozessor aufbereitet oder bereits vom Kompositionsmechanismus expandiert werden. Ein Beispiel für diesen Ansatz sind Workflow-Schemata, in denen die Ausführungsfolge von Aktivitäten durch ECA-Regelgruppen beschrieben werden, die der Entwickler als Templates verwendet oder die aus einem Ablaufgraphen generiert werden. (vgl. [Zuku96, Jasp97]). Man beachte, daß auch bei dieser Methode das „workflow-spezifische“ nur durch den Entwickler und den von ihm vorgenommenen Gebrauch des Beschreibungsmittel entsteht; weder die Syntax der verwendeten Sprache noch die Semantik der einzelnen Ausdrucksmittel sind dafür gemacht worden.

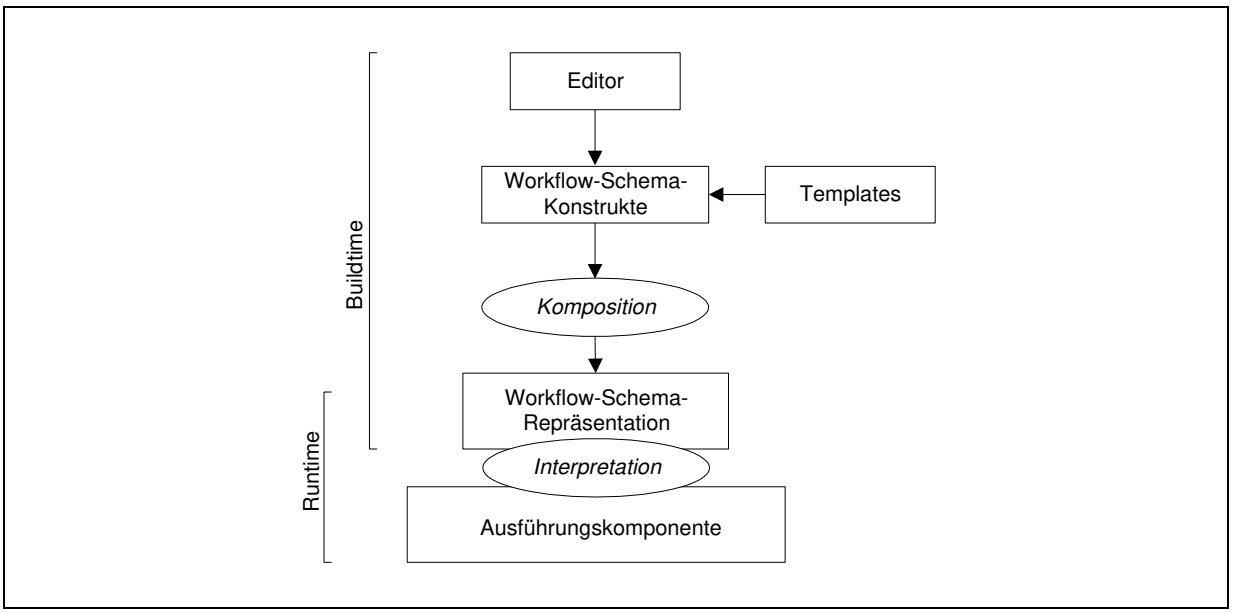


Abbildung 2.2: Indirekte Erstellung ausführbarer Repräsentationen von Workflow-Schemata

Gleichgeblieben ist die Behandlung durch die Ausführungskomponente, die wieder eine homogene Workflow-Schema-Repräsentation vorfindet und interpretiert. Spracherweiterungen (z.B. neue Ausführungsanweisungen) sind demnach neue Workflow-Schema-Konstrukte. Sie werden einmalig unter Beachtung des Kompositionsmechanismus als Schablone definiert, erfordern aber keine Erweiterung oder Veränderung der Ausführungskomponente. Auch diese Methode wird nicht weiter betrachtet, die Gründe hat Abschnitt 2.1.1 genannt.

2.1.3 Übersetzung in eine kompakte Repräsentation des Workflow-Schemas

Ein zusätzlicher Übersetzungsvorgang erlaubt es, eine eigene Workflow-Sprache zur Erstellung von Workflow-Schemata einzuführen, die dann unabhängig von der Sprache der Ausführungskomponente erstellt werden. Erst beim Übersetzungsvorgang greift der Compiler auf die Definition der Workflow-Sprache zurück (Abbildung 2.3).

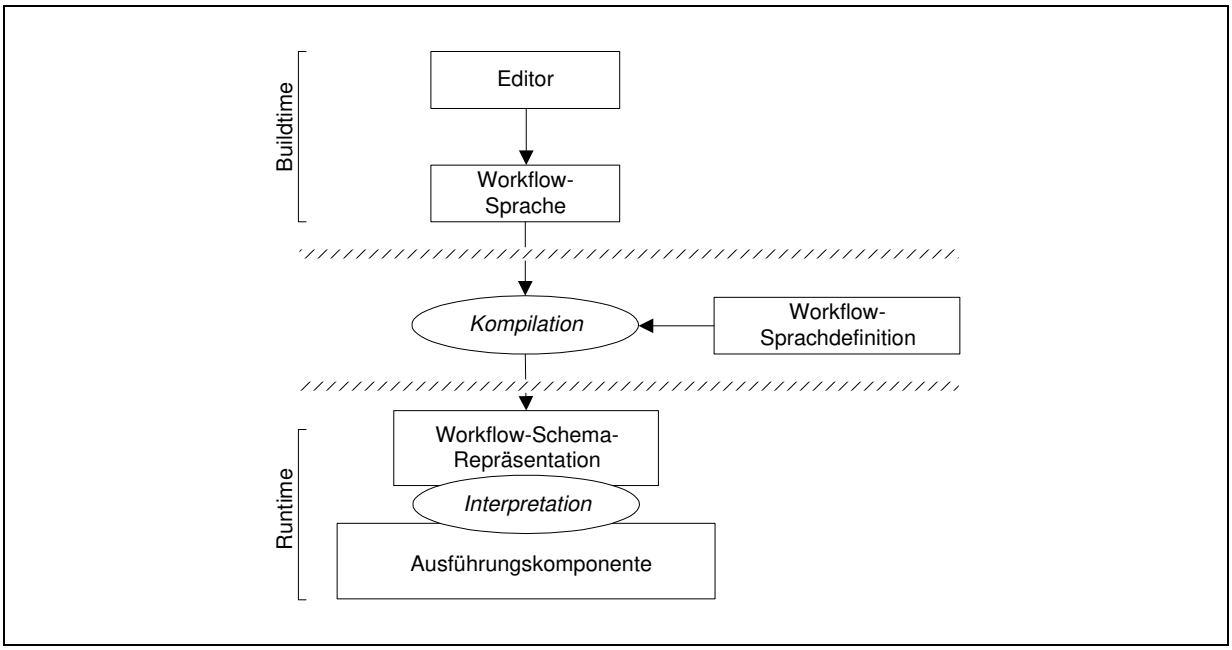


Abbildung 2.3: Übersetzung in eine zusammenhängende Workflow-Schema-Repräsentation

Bei dieser Methode bringt die eigene Workflow-Sprache Vorteile für die Lesbarkeit des Workflow-Schemas, allerdings ist die Ausdrucksstärke der Workflow-Sprache von der Sprache der Ausführungskomponente abhängig. Auch besteht die Gleichbehandlung aller Aspekte eines Workflows weiterhin. Das ist unproblematisch, wenn die erzeugte Repräsentation von Workflow-Schemata in einer funktional reichhaltigen Sprache stattfindet und die Ausführungskomponente diese Funktionen unterstützt. Das ist beispielsweise der Fall, wenn die Übersetzung in konventionelle Programmiersprachen erfolgt, hingegen bringt dieser Ansatz nichts, wenn trotz eigener Workflow-Sprache wieder Petri-Netze oder ECA-Regeln entstehen.

2.1.4 Übersetzung in eine fragmentierte Repräsentation des Workflow-Schemas

Workflow-Schemata können mit Workflow-Sprachen erstellt werden, die nach der Übersetzung zwei oder mehr „Fragmente“ für verschiedene Ausführungskomponenten liefern (Abbildung 2.4). Für die Fragmentierung gibt es unterschiedliche Motive, beispielsweise eine verteilte Ausführung der Workflows. Ein Workflow-Schema wird so übersetzt, daß Subworkflows auf unterschiedlichen Knoten einer verteilten Umgebung ausgeführt werden können (z.B. in [Kotz97]). Eine andere Art der Fragmentierung zerlegt Workflow-Schemata gemäß den enthaltenen Aspekten. Dahinter steht die Erkenntnis, daß unterschiedliche Aspekte eines Workflows auch unterschiedlich im WFMS repräsentiert werden müssen (z.B. verlangt

ein Regelwerk für die Zuordnung von Workflow-Instanzen zu Akteuren [Bußl97] eine andere Beschreibung als die Zuordnung externer Applikationen zu Workflows). Der Übersetzungsvorgang bei dieser Methode erzeugt für unterschiedliche Aspekte im Workflow-Schema auch *unterschiedliche* Repräsentationen, die von *verschiedenen* logischen Ausführungskomponenten verarbeitet werden.

Kombinationen der Ansätze (aspektorientierte Fragmentierung für die verteilte Ausführung) sind möglich und beispielsweise im WFMS MOBILE [Jabl96a] realisiert, so daß für jeden Aspekt eine oder sogar mehrere Ausführungskomponenten zur Verfügung stehen, die zur Ausführungszeit untereinander kommunizieren [Bußl97, Schu97]. Da dieser Ansatz die Möglichkeit bietet, Ausführungsanweisungen als echte Spracherweiterung zu definieren und in der zugehörigen Ausführungskomponente zu implementieren, wird die Entwicklung neuer Ausführungsanweisungen vor diesem Hintergrund behandelt und MOBILE als Beispiel gewählt.

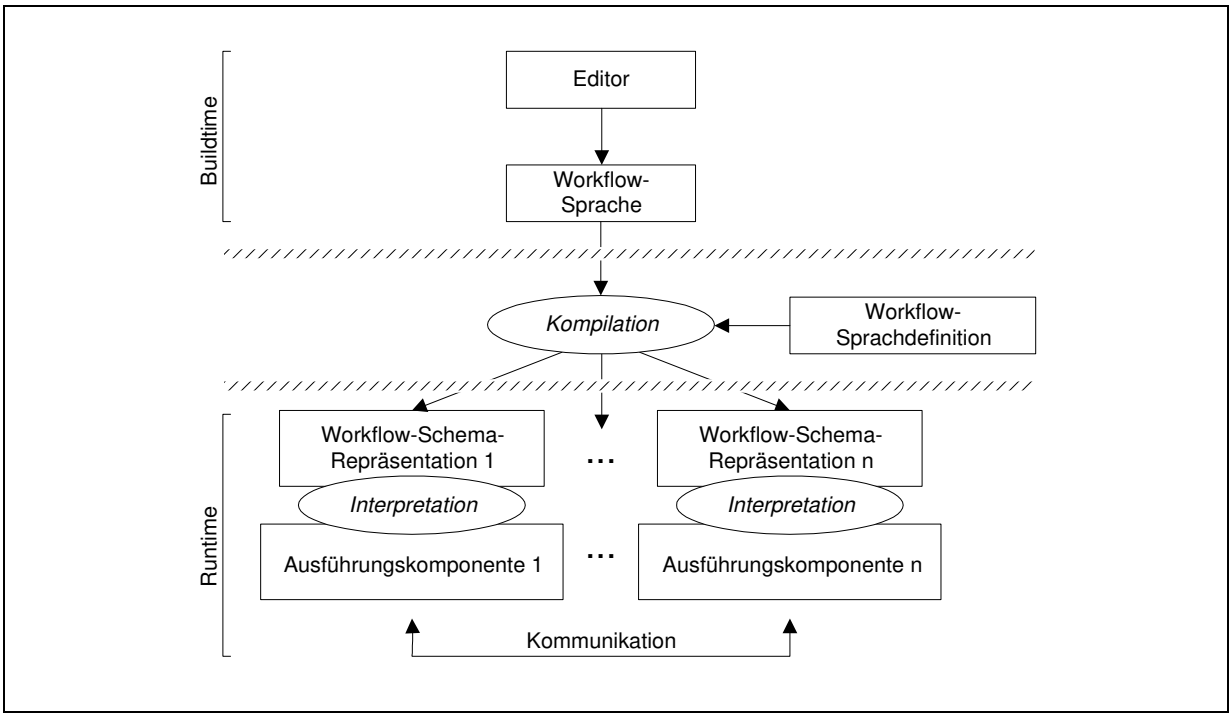


Abbildung 2.4: Übersetzung in eine fragmentierte Repräsentation

Ausgeklammert blieb bisher, woraus Workflow-Schemata abgeleitet werden können und welche Ansätze zur (inhaltlichen) Gestaltung es gibt (vgl. dazu z.B. [Ambe95], [Kara96], [Stei97a], [Stei97b]). Für den technischen Vorgang der Schema-Erstellung ist dies unerheblich; allerdings können die skizzierten Methoden danach beurteilt werden, wieviel Gestaltungsmöglichkeiten sie dem Anwendungsentwickler lassen, insbesondere wenn die Anwendung komplizierte Formen der Workflow-Ausführung erfordert oder besondere Freiheitsgrade für die zukünftigen Benutzer der WFMA verlangt. Abschnitt 3.1 geht auf diese Situation ein; das weitere Kapitel 2 behandelt zuvor noch die Ausführung von Workflows (Abschnitt 2.2) und insbesondere die Kontrollflußdefinition (Abschnitt 2.3).

2.2 Ausführung von Workflows aus konzeptioneller Sicht

Als Geschehen in der Realität müssen Workflows auf Zeichenebene repräsentiert werden, um durch ein WFMS handhabbar zu werden. Workflow-Instanzen enthalten Informationen zur Steuerung und Überwachung des Arbeitsvorgangs, dazu kommen Verwaltungsdaten für die Handhabung durch das WFMS. Handhabung bedeutet in erster Linie die Ausführung von Workflows (für eine weiterreichende Interpretation von „Handhabung“ siehe [Kasc97, S. 41ff.]). Unter „Ausführung“ ist die Abarbeitung von Anweisungen im Workflow-Schema zu verstehen, die ihrerseits Funktionen des WFMS in einer aufeinander abgestimmten Weise aufrufen, so daß eine Interaktion und Kommunikation mit den Anwendern entsteht.

Die Entscheidung über die Ausführung eines Workflows wird je nach WFMS von unterschiedlichen Faktoren abhängig gemacht (siehe dazu Abschnitt 2.2.2). Erfüllt eine Workflow-Instanz die Voraussetzungen zur Ausführung bzw. der Ausführung ihrer Subworkflows, signalisiert ein spezieller Zustand (startable) der Workflow-Instanz dem WFMS diese Eigenschaft. Welche weiteren Zustände und Zustandsübergänge für eine Workflow-Instanz möglich sind, legt das *Ausführungsmodell der Workflow-Instanz* (vgl. [Jabl96a, S. 193ff.]) fest. Die jeweilige Reaktion des WFMS auf die einzelnen Zustände einer Workflow-Instanz gibt das *Ausführungsmodell des WFMS* an. Ausführlich dargestellt sind diese Zusammenhänge in [Jabl96a] für das WFMS MOBILE. An Zustände und Zustandsübergänge der Ausführungsmodelle von Workflow-Instanzen sind Operationen gebunden, die das WFMS auf die Workflow-Instanz anwendet und die Workflow-Instanz in einen neuen Zustand überführt. Diese Funktionsweise des WFMS wird in [Jabl96a] bis zu einem Implementierungsmodell präzisiert und motiviert auch in diesem Technischen Bericht die konzeptionelle Sicht auf die Workflow-Ausführung. Für die Entwicklung von Ausführungsanweisungen wird die Modellvorstellung in den folgenden Abschnitten konkretisiert, und zwar hinsichtlich

- Ausführungsmodelle von Workflows,
- Einflußgrößen zur Bestimmung der Ausführbarkeit von Workflows und der
- kontrollflußzentrierte Workflow-Ausführung.

2.2.1 Ausführungsmodelle von Workflow-Instanzen

Ein hervorstechendes Merkmal aller Vorgänge ist ihre Veränderlichkeit (vgl. [Kasc97, S. 34ff.]), das gilt auch für Arbeitsvorgänge und somit für Workflows. Die Repräsentation von Workflows durch Workflow-Instanzen im WFMS muß dieser Dynamik Rechnung tragen, selbstverständlich nur in eingeschränktem Umfang und auch nur zu bestimmten Zeitpunkten. Mehrere Abstraktionsschritte führen von Sachverhalten in der Realwelt zu einer endlichen Menge von Zuständen einer Workflow-Instanz, die mit dem (unendlichen) Zustandsraum des Workflows nur bedingt korrespondieren: Zum einen haben nicht alle Zustände des Workflows eine für das WFMS sinnvoll interpretierbare Entsprechung, zum anderen verlangt gerade die Handhabung des Workflows durch das WFMS andere bzw. zusätzliche Zustände. Bei ihrer Ausführung durchläuft die Workflow-Instanz diese Menge von Zuständen und spiegelt durch Zustandsübergänge gemäß dem Ausführungsmodell Veränderungen im Workflow oder für die

Handhabung im WFMS notwendige Eigenschaften wieder. Aus dem grundverschiedenen Verständnis der WFMS-Entwickler über das, was ein Workflow ist (vgl. [Böhm97a, S. 77ff.]), erklärt sich die große Divergenz der in der Literatur vorgeschlagenen oder in WFMS-Prototypen bzw. Produkten implementierten Ausführungsmodelle. Ihr ausführlicher Vergleich ist kein Gegenstand dieses Berichts, könnte sich aber an folgenden Kriterien orientieren:

- Eigenschaften:
  - Topologie des Zustandsübergangs-Diagramms als Darstellung des Ausführungsmodells
  - Anzahl und Sichtbarkeit der unterscheidbaren Zustände im Ausführungsmodell
  - Bedeutung und Herkunft der Zustände (das Spektrum reicht von implementierungstechnisch notwendigen Zuständen bis zu jenen, die das Verhalten ausführender Mitarbeiter zeigen)

- Entstehung, Gültigkeit:
  - Zeitpunkt der Festlegung des Ausführungsmodells (bei der WFMS-Implementierung, bei der WFMA-Entwicklung, zur Ausführung der Workflow-Instanz ...)
  - Gültigkeit des Ausführungsmodells (spezifisch für Workflow-Instanzen, Workflow-Schemata oder allgemeingültig im WFMS)

Diese Kriterien werden im folgenden aufgegriffen, um das Spektrum der Zustände in Ausführungsmodellen zu verdeutlichen. Die Darstellung gliedert sich in zwei Teile: Abschnitt 2.2.1.1 beschreibt Zustände in Ausführungsmodellen, die als Folge des Modellierungsansatzes für Workflows festgelegt sind, sofern es sich nicht um ein erweiterbares WFMS handelt. Dort kann die Festlegung des Ausführungsmodells anwendungsspezifisch geprägt sein. Dieser Teil ist für die Entwicklung von Ausführungsanweisungen relevant, weil das anschließend eingeführte Funktionsmodell für Ausführungsanweisungen auf diesen Zuständen aufbaut. Abschnitt 2.2.1.2 vervollständigt das Thema „Ausführungsmodelle“ für eine Klasse von WFMS, in denen Workflows mit Ausdrucksmitteln für technische Systeme beschrieben werden und in denen es keine Ausführungsanweisungen der hier eingeführten Art gibt.

2.2.1.1 Beschreibung von Workflows durch Elemente des Workflow-Metaschemas

WFMS, in denen Arbeitsvorgänge nicht wie ein technisches System modelliert werden, haben in ihrem Metaschema *workflow-spezifische* Ausdrucksmittel und rechtfertigen damit die Bezeichnung *Workflow-Metaschema* für diese Beschreibungsebene. Für das Ergebnis der funktionalen Dekomposition von Arbeitsvorgängen (in Teilaufgaben, Arbeitsschritte, Handlungsanweisungen etc.) bieten Workflow-Metaschemata unterschiedlich benannte Elemente an, sogenannte *Strukturbausteine*; gängige Bezeichnungen für sie sind „Activity“, „Task“, „Step“ und natürlich „Workflow“. Die unterschiedlichen Konzepte dahinter und die Konsequenzen beim Aufbau zusammengesetzter Strukturen werden hier nicht diskutiert, eine ausführliche Behandlung findet sich in [Böhm97a, S. 107ff.]. Wichtig für die Entwicklung von Ausführungsanweisungen sind die Ausführungsmodelle dieser Strukturbausteine, die anhand folgender Kriterien unterschieden werden:

- **Topologie**. Ausführungsmodelle lassen sich durch gerichtete Graphen darstellen, in denen Knoten je nach Position Anfangs-, End- und Zwischenzustände bedeuten. Die Topologie des Graphen gibt Hinweise auf den Grad der Differenzierung von Zuständen im Ausführungsmodell; zwei Endzustände deuten beispielsweise auf unterscheidbare Ergebnisse der Ausführung hin (Beispiele dazu im Punkt „Bedeutung“ weiter unten).

- **Sichtbarkeit**. Die Sichtbarkeit eines Zustands ermöglicht, sein Eintreten oder Verlassen zu erkennen, um darauf zu reagieren. Sichtbarkeit ist ein relatives Kriterium und kann einem Zustand des Ausführungsmodells nur in Bezug auf einen „Beobachter“ zugesprochen werden. Für die Ausführungskomponente eines WFMS sind alle Zustände der eigenen Workflow-Instanzen sichtbar; bei Ausführung eines Workflows durch mehrere autonome Komponenten ist die Sichtbarkeit von Zuständen nicht-lokaler Workflow-Instanzen unter Umständen eingeschränkt. Entscheidend für die Möglichkeiten zur Festlegung der Workflow-Ausführung sind die Zustände, die der Entwickler unterscheiden und zur Definition von Abhängigkeiten nutzen kann. Von der Sichtbarkeit ist die Möglichkeit der Einflußnahme zu unterscheiden (siehe Abschnitt 3.2.3.2).

- **Bedeutung**. Workflows müssen im WFMS durch komplizierte Datenstrukturen repräsentiert werden, entsprechend groß ist der Zustandsraum aus den möglichen Werten dieser Daten. Um die Anwendbarkeit einer Operation auf der Workflow-Instanz zu entscheiden, muß der gesamte Zustandsraum auf eine kleinere Zahl von Zuständen projeziert werden. Anzahl und Bedeutung der Zustände bestimmen die Entwickler des WFMS, entweder einmalig bei der Implementierung (und damit unveränderbar) oder später bei der Konfigurierung eines WFMS bzw. bei der Definition des Ausführungsmodells für ein Workflow-Schema. Abgesehen von generischen Operationen wie dem Start eines Workflows gibt es keine allgemeingültige Festlegung, welche Operationen für Workflow-Instanzen notwendig sind; im Gegenteil, die Möglichkeit zur Einführung neuer Operationen und spezifischer Zustände zeichnen ein WFMS aus. Daher sind die folgenden Zustände nur Beispiele und keineswegs Bestandteil jedes Ausführungsmodells, zumal die Bezeichnungen in jedem WFMS andere Bedeutungen haben. Erschwerend für den Vergleich ist die oft ungenaue Positionierung von Ausführungsmodellen in der Literatur; meist fehlt eine klare Aussage, auf welcher Ebene (Konzept oder Implementierung) das vorgestellte Ausführungsmodell angesiedelt ist. Häufig anzutreffen sind folgende Zustände:

  - Mit ready (ausführbereit) oder startable (startbar) bezeichnete Zustände signalisieren, daß die Voraussetzungen für die Ausführung eines Workflows erfüllt sind. Die Ausführungsbereitschaft setzt sich aus mehreren Faktoren zusammen, begonnen mit der Bereitstellung von Betriebsmitteln auf Betriebssystem-Ebene bis zur Konsensbildung über alle ausführungsrelevanten Aspekte im WFMS (vgl. [Bußl97, S. 55ff.] und auch Abschnitt 2.2.2). In diesem Technischen Bericht beziehen sich ready und startable auf letzteres. Auf der gleichen Bedeutungsebene wie ready liegen Zustände wie running (laufend, aktiv) und done (beendet) oder terminated (abgeschlossen), wobei die durch running angezeigte Ausführung eines Workflows differenziert werden muß, wenn es sich um einen kompositen Workflow handelt (vgl. [Jabl96a, S. 248ff.]). Unter Hinzunahme von paused (vorübergehend ausgesetzt) ist das ein Satz von Zuständen, der respektive ande-

rer Bezeichner in vielen WFMS für Workflows vorkommt. Alle genannten Zustände machen Aussagen über den *aktuell gültigen* Zustand der Workflow-Instanz, mit Ausnahme von done reicht die Geltungsdauer nicht in die Zukunft hinein, was Auswirkungen auf die Kontrollflußdefinition hat und in Abschnitt 3.2.3.1 beschrieben wird.

- Ein als blocked oder not_executing (garantiert nicht mehr aktiv) bezeichneter Zustand drückt anders als beispielsweise running eine Eigenschaft des Workflows aus, die bis zum Ende der zugehörigen Workflow-Instanz gilt. Umfaßt das Ausführungsmodell einer Workflow-Instanz den Zustand blocked, erlaubt die darin zugesicherte Nichtausführung zusätzliche Möglichkeiten für die Kontrollflußdefinition (siehe Abschnitt 3.2.3.1).

- Zustände wie ok oder failed sind nicht nur Verfeinerungen des Zustands done, sondern beinhalten eine zusätzliche Information über die Workflow-Ausführung, in dem sie die Ausführung hinsichtlich eines Erfolgsbegriffs qualifizieren. Der Erfolgsbegriff kann unterschiedlich definiert sein, daher gilt sinngemäß das gleiche wie beim Zustand startable: Ausführungserfolg ist nur relativ zu einer konkreten Ausführungsumgebung bzw. Abstraktionsebene entscheidbar. Besonders deutlich wird die Relativität von „erfolgreicher“ bzw. „fehlgeschlagener“ Ausführung, wenn Ausführungserfolg einmal das Ergebnis eines Applikationsaufrufs ist und einmal als Aussage über den Verlauf einer Handlung in der Anwendungswelt zu verstehen ist. Danach sind Zustände denkbar, die den zeitlichen Verlauf der Workflow-Ausführung als Kriterium heranziehen und den Erfolg anhand der tatsächlichen Ausführungsdauer entscheiden.

- Mögliche Zustände eines Ausführungsmodells für Workflow-Instanzen sind aborted und committed in Anlehnung an Transaktionen bzw. als Folge der Gleichsetzung von Transaktionen und Workflows (z.B. in [Rusi94, S. 6], [Lieb95, S. 5]). Wie ok und failed können committed und aborted mögliche Unterzustände eines Endzustands wie done sein, allerdings reicht die Aussage über die von ok und failed hinaus: Ein aborted im Sinne des Transaktionsbegriffs bedeutet eine vollständige Wiederherstellung des vorherigen Zustands, was in seiner Bedeutung für Workflows bislang ungeklärt ist; für ein committed auf der Workflow-Ebene (!) gilt das gleiche. Noch weitergehende Bedeutung hätte ein Zustand compensated im Kontext der Workflow-Ausführung, auch hierfür gibt es bislang keine tragfähige Interpretation.

- Eine je nach Implementierung verschiedene Anzahl von internen Zuständen beschreibt z.B., ob eine Workflow-Instanz bereits mit den zur Ausführung notwendigen Daten versorgt ist (z.B. in [Schw95, S. 6], [Krad97, S. 24]). Auch gibt es Ausführungsmodelle, in denen der Bearbeitungsstand einer dem Workflow zugeordneten Aufgabe unterschieden wird (z.B. in [Schw93, S. 91]) oder die Zuordnung eines Workflows zur einem Aktivitätsträger als eigener Zustand vorkommt ([Krad97, S. 24]).

Die Beispiele zeigen, welche Bedeutung im Zustand einer Workflow-Instanz codiert sein kann. Aus der vorgenommenen Gliederung darf nicht geschlossen werden, daß hier ein gegenseitiger Ausschluß herrschen muß, also die Zustände nicht gemeinsam in einem Ausführungsmodell auftreten dürfen. Zum Teil wird ein Ausführungsmodell hierarchisch gestaltet, so

daß ein Zustand auf einer tieferen Ebene verfeinerte Unterzustände aufweist. Eine Vermischung von anwendungsbezogenen und workflow-spezifischen Zuständen ist nicht zu verhindern, wo Workflows nicht durch Elemente des Workflow-Metaschemas beschrieben werden, sondern unspezifische Beschreibungsmittel eingesetzt werden.

2.2.1.2 Beschreibung von Workflows durch unspezifische Ausdrucksmittel

In einigen WFMS werden Beschreibungsmittel für Workflows verwendet, die zur Modellierung technischer Systeme oder Prozesse entwickelt wurden. Als Beispiele sind hier insbesondere State-Activity-Charts und Petri-Netze bzw. Petri-Netz-Derivate zu nennen. Charakteristisch für diese WFMS ist der unmittelbare Einsatz dieser Formalismen für die Beschreibung vieler Aspekte eines Workflows, weshalb ein Workflow-Schema fast vollständig durch ein State-Activity-Chart bzw. Petri-Netz dargestellt wird (z.B. FUNSOFT-Netze [Gruh91] im WFMS LEU [Dink94]). Es liegt in der Verantwortung des Anwendungsentwicklers, mit Stellen und Transitionen bzw. Aktivitäten und Zuständen ein Workflow-Schema zu erstellen. Daher kann auch nicht von einem Ausführungsmodell der Workflow-Instanz gesprochen werden, da es kein ausgezeichnetes Element der Beschreibungssprache für Workflows gibt. Workflow-Instanzen sind Markenbelegungen von Stellen im Petri-Netz bzw. eine Menge gerade gültiger Zustände im State-Activity-Chart. Daraus folgt, daß der Zustand einer Workflow-Instanz sich indirekt aus dem Zustand der Stellen bzw. Aktivitäten ergibt, aus denen sich die Beschreibung des Workflow-Schemas konstituiert. Ausführungsanweisungen kommen bei der Beschreibung der Workflows nicht vor, da alle Beziehungen zwischen Aktivitäten durch die Elemente der Beschreibungssprache festgelegt werden, also Transitionen beim Einsatz von Petri-Netzen oder Übergangsbedingungen bei State-Activity-Charts. Aus diesem Grund werden derartige WFMS hier nicht weiter betrachtet.

2.2.2 Faktoren zur Bestimmung ausführbereiter Workflows

Jedem Zustand im Ausführungsmodell einer Workflow-Instanz ist eine Bedeutung zugeordnet, Beispiele dafür wurden in Abschnitt 2.2.1.1 genannt. Eine besondere Stellung hat der Zustand startable, weil er dem WFMS die Ausführungsbereitschaft des Workflows signalisiert. Faktoren, die die Ausführungsbereitschaft eines Workflows bestimmen, können sein (siehe auch [Jasp97, S. 331]):

- Eintritt eines Zeitpunkts oder Ablauf einer relativ/absolut definierten Frist
- Verfügbarkeit zur Ausführung benötigter Daten
- Fertigstellung oder Ergebnis eines anderen Workflows
- alle weiteren vom WFMS erkennbaren Ereignisse

Generell gibt es keine Einschränkung auf bestimmte Parameter, von denen die Entscheidung über die Ausführbarkeit einer Workflow-Instanz abhängig gemacht werden kann. Dies wäre auch nicht sinnvoll, weil die unterschiedlichen Eigenschaften der Arbeitsvorgänge eine flexible Zusammenstellung von Entscheidungskriterien verlangen. Im WFMS MOBILE hängt es

von der Konfiguration ab, welche Aspekte eines Workflows für die Entscheidung über seine Ausführbarkeit herangezogen werden (vgl. [Bußl97, S. 75]). In einer gegebenen Konfiguration erfolgt eine Konsensbildung über alle Aspekte, die im Workflow-Schema der Workflow-Instanz spezifiziert wurden. Sofern der Entscheidungsalgorithmus nicht konfigurierbar ist, muß über die Gestaltung des Workflow-Schemas die Möglichkeit zur Einflußnahme bestehen, beispielsweise durch Weglassen einzelner Aspekte in der Spezifikation des Workflows. Das kann dazu führen, die Ausführungsbereitschaft eines Workflows über eine Teilmenge der Aspekte zu definieren. Beispielsweise bedeutet die *datenflußgesteuerte* Ausführung von Workflows, die Ausführungsbereitschaft eines Workflows an das Vorliegen eines Datums zu binden. Dies kann mit Hilfe des im Workflow-Schemas explizit definierten Datenflusses erfolgen oder durch Zugriff auf einen extern verwalteten Datenbestand (z.B. eines Dokument-Management-Systems) realisiert werden, dessen Mechanismen zur Synchronisation von Lese- und Schreiboperationen auch die synchronisierte Ausführung der Workflows zur Folge haben. Im letztgenannten Fall muß das Ausführungsmodell der Workflow-Instanz um weitere Zustände erweitert werden, weil derart kontrollierte Workflows verschiedene Unterzustände von running benötigen und ein modifizierter Scheduling-Mechanismus erforderlich ist. Die datenflußgesteuerte Ausführung von Workflows wird hier nicht weiter behandelt (siehe dazu z.B. [Rusi95]), sondern die Ausführungsbereitschaft in Abhängigkeit vom Zustand anderer Workflow-Instanzen, die als *kontrollflußzentrierte* Ausführung charakterisiert wird und Gegenstand des nächsten Abschnitts ist. Zu beachten ist dabei, daß nicht für alle Anwendungsbereiche eines WFMS die Beschränkung auf den Kontrollfluß als einzigen Koordinationsmechanismus möglich ist. Die weitere Betrachtung setzt daher ein Anwendungsgebiet voraus, wo Subworkflows und Applikationsprogramme in zuvor bekannter Reihenfolge ausgeführt werden, alle Abhängigkeiten durch Konsumenten/Produzentenbeziehungen von Daten bekannt sind und bei der Workflow-Schema-Erstellung berücksichtigt werden können.

2.2.3 Kontrollflußzentrierte Ausführung von Workflows

Wird die Ausführung eines Workflows durch den Zustand oder das Ausführungsergebnis eines anderen Workflows kontrolliert, steuern also Informationen über *aktive* Objekte die Workflow-Ausführung, wird die Weiterleitung dieser Information als *Kontrollfluß* bezeichnet. Die *Kontrollflußdefinition* legt fest, von welchen Objekten kontrollflußrelevante Daten stammen, welche Verarbeitung diese Daten erfahren, welche Maßnahmen bei bestimmten Werten folgen und auf welche Objekte diese Maßnahmen wirken. Von Ausnahmen abgesehen erfolgt die Kontrollflußdefinition im Workflow-Schema und damit gültig für alle Workflow-Instanzen. Die Kontrollflußdefinition legt den *potentiellen* Kontrollfluß fest, der *tatsächliche* Kontrollfluß im Workflow ergibt sich aus laufzeitbestimmten Werten. Nur unter der idealisierenden Annahme einer fehlerfreien Workflow-Ausführung ist der tatsächliche Kontrollfluß genau eine Ausprägung des potentiellen Kontrollflusses. Ansätze zur Erkennung, Behandlung und Vermeidung von Fehlern bei der Workflow-Ausführung finden sich in z.B. [Lieb97].

2.3 Kontrollflußdefinition in Workflow-Schemata

Trotz verschiedener Interpretationen von Workflow-Management besteht Konsens über die Notwendigkeit, die *Ausführungsfolge* von Workflows festzulegen. „Ausführungs*folge*“ umfaßt mehr als Sequenzen; zur Ausführungsfolge gehören Auslassung, Auswahl, Varianten der nebenläufigen, parallelen und anderweitig zeitlich verschränkten Ausführung von Workflows. Im folgenden wird durchgehend von „Workflow“ gesprochen; dahinter steht das Konzept eines einheitlichen Ausdrucksmittels für Aufgaben auf jeder Ebene der funktionalen Dekomposition (andere Ansätze favorisieren eine explizite Zweiteilung, z.B. bei FLOWMARK [Leym92, Leym94] in „Activity“/„Process“, bei METEOR [Kris94, Shet96] in „Task“/„Workflow“). Dem Ansatz aus [Jabl96a, S. 126ff.] folgend werden hier Workflows unterstellt, die je nach *Verwendung* Sub- oder Superworkflows bzw. je nach innerer *Struktur* elementare oder komposite Workflows sind. Da Ausführungsfolgen von Subworkflows sichtbare Zeichen des Verhaltens des Workflows sind, wird die in [Curt92] für Prozesse vorgeschlagene und in [Jabl94] auf Workflows angewendete Bezeichnung „Verhaltensaspekt“ übernommen. Auch wenn das *Verhalten* eines Workflows (z.B. sein Start) von mehreren Faktoren abhängt (vgl. Abschnitt 2.2.2), stellt der vorliegende Bericht die kontrollflußzentrierte Ausführung von Workflows und damit die Kontrollflußdefinition in den Mittelpunkt.

Zur Definition des Kontrollflusses gibt es unterschiedliche Ansätze, die zudem noch verschiedenartig klassifiziert werden. In [Mühl97, S. 162] wird zwischen *diagramm-sprachlich* und *script-basiert* unterschieden; dies ist zum einen unscharf (z.B. bietet das WFMS FlowMark beides an), zum anderen wird die Präsentation dem Entwickler gegenüber in den Vordergrund gestellt, nicht die Ausdrucksstärke, Flexibilität usw. [Ritt97, S. 162] gliedert in Skriptsprachen, netzorientierte Methoden, logikbasierte Methoden, algebraische Methoden und ECA-Regeln. Das deckt viele Ansätze ab, zieht aber zur Charakterisierung unterschiedliche Betrachtungsebenen heran und ist daher ebenfalls ungeeignet (z.B. implementieren ECA-Regeln in [Kapp95b] eine netzorientierte Spezifikation, in [Tomb97] z.B. werden ECA-Regeln auf Regeln der Ausführungsumgebung EvE [Gepp96] abgebildet). Die Vorschläge werden nicht übernommen, statt dessen wird eine Einteilung in

- *Kontrollflußprimitive (KFP)*
- *Kontrollflußkonstrukte (KFK)* und
- *Ausführungsanweisungen (AA)*

eingeführt. Diese Einteilung fokussiert darauf, wie Zustände des Ausführungsmodells der Workflow-Instanzen in Beziehung gesetzt werden. Das ist ein syntaktisches Merkmal, es hat aber Auswirkungen auf die erreichbaren Ausdrucksmöglichkeiten. Die Festlegung des Verhaltensaspekts erfolgt auf der Ebene des Workflow-Schema, gilt aber für alle Workflow-Instanzen dieses Workflow-Schemas. Die Ausführungen der folgenden Abschnitte beziehen sich immer auf Workflow-Instanzen, weil auf dieser Ebene die unterschiedliche Funktionsweise der Kontrollflußprimitive, Kontrollflußkonstrukte und Ausführungsanweisungen deutlich wird. Die Verdeutlichung der Unterschiede unterstützt zudem eine grafische Notation: Abgerundete Rechtecke stellen Workflow-Instanzen dar, die beim Einsatz von Kontrollfluß-

primitiven und Kontrollflußkonstrukten über die Zustände des Ausführungsmodells (Kreise an Rand des Workflow-Symbols mit s für startable, d für done, r für running, ne für not_executing) in Beziehung stehen. Die Einbettung der Fragmente in den Kontext einer weiteren Kontrollflußdefinition ist an besonders hervorgehobenen Stellen (grau hinterlegten Kreise) möglich (siehe Abbildung 2.5 und Abbildung 2.6).

2.3.1 Kontrollflußprimitive

Kontrollflußprimitive setzen Zustände des Ausführungsmodells von zwei oder mehr Workflow-Instanzen in Beziehung. Die Bezeichnung „Kontrollfluß*primitv*“ weist auf einen einfachen bzw. elementaren Mechanismus hin. Basierend auf der im WFMS definierten Semantik des Kontrollflußprimitivs wird zur Laufzeit über die Ausführung der assoziierten Workflows entschieden. Beispielsweise führt das Kontrollflußprimitiv, das den Zustand done von Workflow-Instanz 1 und den Zustand startable von Workflow-Instanz 2 in Beziehung setzt und beide Zustände als Ursache-Wirkung-Zusammenhang interpretiert, dazu, daß nach Ende von Workflow 1 der Workflow 2 gestartet wird. Abwandlungen dieses Grundprinzips sind möglich, etwa wenn der Zusammenhang an Zusatzbedingungen (z.B. Werte laufzeitabhängiger Variablen) geknüpft ist. Als grafische Darstellung von zweistelligen Kontrollflußprimitiven werden Kanten in gerichteten Graphen verwendet, in denen Subworkflows die Knoten bilden. Man beachte, daß es sich bei diesen sogenannten Ablaufgraphen um die *Veranschaulichung* des funktionalen und verhaltensbezogenen Aspekts eines Workflow-Schemas handelt; Ablaufgraphen und Workflow-Schema sind nicht das gleiche. In einigen Publikationen wird der Ablaufgraph auch als konzeptionelles Schema eines Workflows bezeichnet (z.B. in [Reic98]), hier gilt der gleiche Einwand. Als Beispiel für die Verwendung von Kontrollflußprimitiven dient ein kommerzielles WFMS:

- Im WFMS FLOWMARK [Leym94] können jeweils zwei Aktivitäten A_1 und A_2 durch einen *Kontrollflußkonnektor* zu einem Ablaufgraphen verbunden werden. Der Kontrollflußkonnektor läßt sich noch durch eine zur Laufzeit auswertbare Transitionsbedingung qualifizieren. Kontrollflußkonnektoren sind ein Beispiel für Kontrollflußprimitive, die den Zustand done von A_1 mit dem Zustand startable von A_2 assoziieren. Die in Beziehung gesetzten Zustände (done, startable) und die Interpretation des Kontrollflußprimitivs können in FLOWMARK nicht verändert werden. Der Entwickler eines Workflow-Schemas erkennt nicht, daß er immer genau diese beiden Zustände in Beziehung setzt, sondern er hat lediglich zwei „Anknüpfungspunkte“ an jeder Aktivität, an denen der Editor die Verbindung mittels Kontrollflußkonnektoren erlaubt, bzw. eine bestimmte Anweisung (CONTROL <Activity$_i$> TO <Activity$_j$>), wenn er die FDL (FlowMark Definition Language) verwendet.

Kontrollflußkonnektoren sind ein Vertreter zweistelliger Kontrollflußprimitive. Die Zweistelligkeit ist nicht zwingend; wie das nächste Beispiel zeigt, können auch Zustände von mehr als zwei Workflows bzw. Aktivitäten in Beziehung gesetzt werden:

- ECA-Regeln können verwendet werden, um die Fertigstellung von Aktivitäten zu verknüpfen und unter Einbeziehung von Zusatzbedingungen weitere Aktivitäten zu starten. Kon-

trollflußprimitive auf der Grundlage von ECA-Regeln können auf alle sichtbaren Zustände unterschiedlich vieler Aktivitäten Bezug nehmen, von denen ein detektierbares Ereignis erzeugt wird (z.B. not_executing), und sind daher flexibler als die in FLOWMARK angebotenen Kontrollflußprimitive, die nur vom Zustand done ausgehen. Allerdings besteht beim Einsatz von ECA-Regeln die Gefahr, Zyklen und indeterministische Ausführungsfolgen zu definieren; ein Problem, das unabhängig vom Workflow-Management ist.

Abbildung 2.5 zeigt gemäß obiger Notation links vier Workflow-Instanzen, die durch vier Kontrollflußprimitive verbunden sind. Jedes einzelne der zweistelligen Kontrollflußprimitive assoziiert jeweils den Zustand d (done) einer Workflow-Instanz mit dem Zustand s (startable) einer zweiten Workflow-Instanz. Auf der rechten Seite ist ein vierstelliges Kontrollflußprimitiv, das die Zustände d (done) und ne (not_executing) zweier unterschiedlicher Workflow-Instanzen mit den Zuständen s (startable) zwei weiterer Workflow-Instanzen verknüpft.

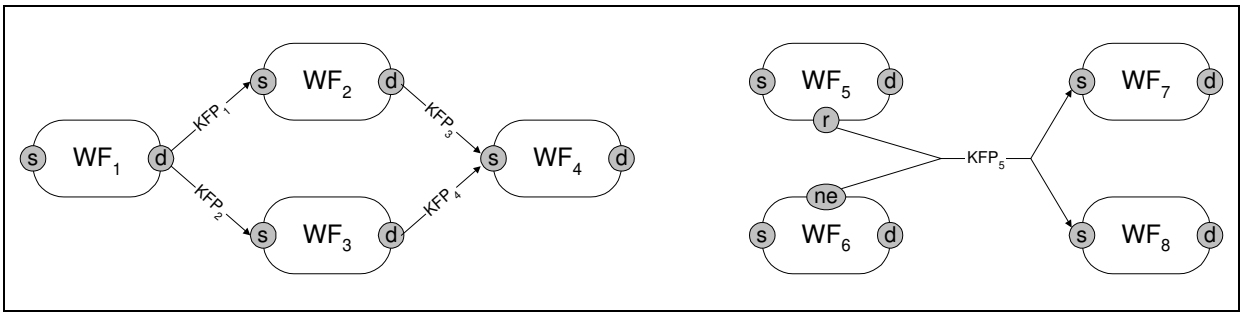


Abbildung 2.5: Beispiele für die Verwendung von Kontrollflußprimitiven

Als Beispiel für das Kontrollflußprimitiv KFP_1 in (Abbildung 2.5, links) kommt die FDL-Anweisung CONTROL FROM WF_1 TO WF_2 in Frage; KFP_5 (Abbildung 2.5, rechts) kann durch eine ECA-Regel realisiert werden, die aus den Ereignissen WF_5_running und WF_6_not_executing ohne Bedingung die Aktionen Start_WF_7 und Start_WF_8 ableitet (unter der Annahme, die genannten Zustände sind Bestandteil der Ausführungsmodelle und für den Entwickler sichtbar).

Die gemeinsame Behandlung von Kontrollflußkonnektoren und ECA-Regeln erfolgt unter dem Gesichtspunkt, einzelne Zustände einzelner Workflow-Instanzen zu assoziieren; ansonsten bestehen natürlich Unterschiede: Z.B. bilden die Kontrollflußkonnektoren in FLOWMARK einen zusammenhängenden und zyklenfreien Ablaufgraphen, in dem alle vorkommenden Aktivitäten eindeutig einem Workflow-Schema (in FLOWMARK Process genannt) zugeordnet sind. ECA-Regeln verbinden ebenfalls Aktivitäten, deren Zugehörigkeit zu einem Prozeß ist jedoch nicht mehr so leicht zu erkennen. Weitere Unterschiede bestehen in der Ausführung durch das WFMS bzw. die aktive Datenbank, weil durch die Ablaufgraphen in FLOWMARK die Menge der auswertbaren Kanten eindeutig bestimmt ist, bei ECA-Regeln zusätzliche Mechanismen notwendig sind, um die Menge der für ein Workflow-Schema relevanten Regeln zu identifizieren.

2.3.2 Kontrollflußkonstrukte

Der Einsatz von Kontrollflußprimitiven verlangt vom Entwickler, die gewünschte Ausführung der Workflows auf eine dafür notwendige Kombination der Kontrollflußprimitive abzubilden. Dieser Schritt entfällt bei einer Workflow-Sprache mit *Kontrollflußkonstrukten*. Kontrollflußkonstrukte bestehen aus einer Zusammenfassung von mehreren Kontrollflußprimitiven und einer Funktion, die das Verhalten dieser Gruppe aus Kontrollflußprimitiven bestimmt. Durch unterschiedliche Funktionen entstehen verschiedene Typen von Kontrollflußkonstrukten, die unterschiedliche Formen der Workflow-Ausführung beschreiben. Es gibt Beispiele für den Einsatz von Kontrollflußkonstrukten:

- Das WFMS WORKPARTY [Work95] bietet die Kontrollflußkonstrukte SPLIT (Abzweig), ALTERNATIV (Alternative), PARALLEL (Nebenläufigkeit), LOOP (Schleife) und SELECT (Wahlfreiheit), dazu noch ein Kontrollflußkonstrukt für die sequentielle Ausführung, das in der textuellen Repräsentation der Ablaufbeschreibung entfällt.

- Die Vorschläge der Workflow-Management-Coalition (WfMC) zur Kontrollflußdefinition lassen sich mangels präziser Beschreibung nicht zuordnen. Allerdings deuten die Darstellungen in [Work94] und [Work96] auf Kontrollflußkonstrukte hin, die als AND-Split, OR-Split, AND-Join, OR-Join, Iteration bezeichnet werden. Eine Realisierung dieser Kontrollflußkonstrukte mittels ECA-Regeln findet sich z.B. [Jasp97, S. 325ff.] oder in [Zuku96, S. 21].

Beim Einsatz von Kontrollflußkonstrukten muß der Entwickler Regeln beachten, so ist etwa die Zusammenführung der Äste eines XOR-Splits mit einem AND-Join aus naheliegenden Gründen unzulässig. Eine Tabelle zulässiger Kombinationen von OR-, XOR-, AND-Splits mit OR-, XOR-, AND-Joins findet sich in [Rose96, S. 116], zwar im Zusammenhang mit der Modellierung von Geschäftsprozessen durch Ereignisgesteuerte Prozeßketten (EPK) [Kell92, Sche94], aber in der Aussage übertragbar. Zum Teil wird die Einhaltung solcher Regeln durch Editoren sichergestellt (wie z.B. beim WFMS WORKPARTY oder in der Modellierungskomponente FLUX des Forschungsprototyps WATERLOO [Zuku96], der ECA-Regeln einsetzt).

2.3.3 Ausführungsanweisungen

Ausführungsanweisungen legen die Ausführung von Workflows fest, ohne explizit auf Zustände des Ausführungsmodells der Workflow-Instanzen Bezug zu nehmen. Im Unterschied zu Kontrollflußprimitiven und Kontrollflußkonstrukten definieren Ausführungsanweisungen keinen Kontrollfluß, sondern geben die Workflow-Ausführung direkt an. Um etwa die Hintereinanderausführung zweier Workflows festzulegen, wird die entsprechende Ausführungsanweisung der Workflow-Sprache verwendet (z.B. sequence(Workflow-1, Workflow-2)). Das ist etwas anderes als eine Kontrollflußdefinition der Form CONTROL FROM Workflow-1 TO Workflow-2. Obwohl beides zur sequentiellen Ausführung der Workflows führt, ist das erste eine am *Ergebnis* der Ausführung orientierte Anweisung und das zweite eine am Ausführungsmodell von WFMS und Workflow-Instanz orientierte *Durchführungsvorschrift*. Ausführungsanweisungen und Kontrollflußprimitive bzw. Kontrollflußkonstrukte dienen dazu, den

Verhaltensaspekt eines Workflows im Workflow-Schema zu definieren, der Unterschied in den Konzepten darf jedoch nicht übersehen werden.

- Das WFMS MOBILE bietet z.B. Ausführungsanweisungen für elementare Formen der Workflow-Ausführung (sequence, parallel, cond_branch), Schleifenkonstrukte (z.B. loop_until_false) und Auswahlentscheidungen (choice, n_out_of_m) [Jabl96a, S. 145ff.] an. In [Jabl94] gibt es zusätzlich noch delay, limit und deadline, die kompliziertere Beziehungen zwischen zwei Workflows ausdrücken. Die letztgenannten Ausführungsanweisungen werden anschließend genauer beschrieben und dienen hier nur als Beispiele für das Angebot.

Der nächste Abschnitt stellt die Konzepte zur Beschreibung des Verhaltensaspekts gegenüber.

2.3.4 Gegenüberstellung am Beispiel

Das Beispiel der alternativen Ausführung zweier Workflows verdeutlicht die Unterschiede der Konzepte. Abbildung 2.6 zeigt links die Verwendung von Kontrollflußprimitiven (KFP_1 und KFP_2), in der Mitte ein Kontrollflußkonstrukt (Auswahl) und rechts eine Ausführungsanweisung (Alternative).

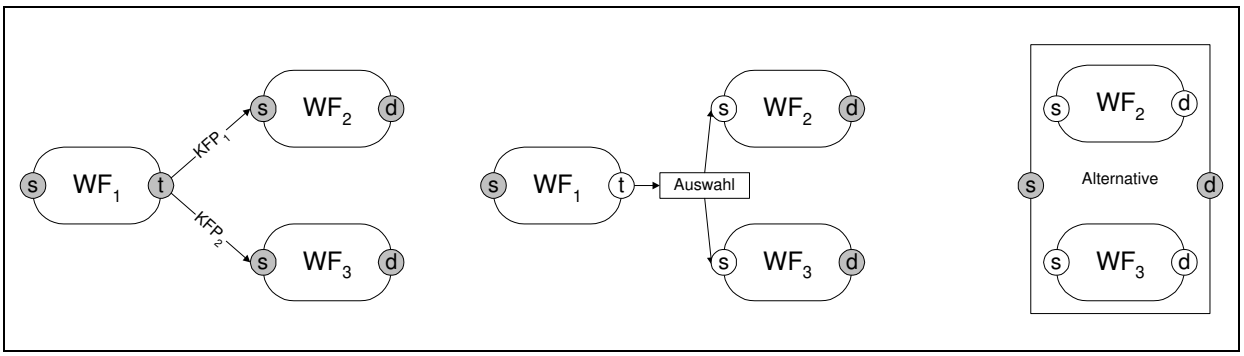


Abbildung 2.6: Möglichkeiten zur Beschreibung der alternativen Ausführung zweier Workflows

Bei den Kontrollflußprimitiven (Abbildung 2.6, links) muß der Entwickler darauf achten, die beiden Kontrollflußprimitive so mit Bedingungen zu versehen, daß immer genau eine von beiden erfüllt ist. Anderenfalls führt das Ende von Workflow-1 (WF_1) dazu, daß über KFP_1 und KFP_2 Workflow 2 (WF_2) und Workflow 3 (WF_3) in den Zustand startable übergehen können und damit beide ausführbar sind (zusätzliche Startbedingungen seien nicht definiert), was der intendierten Alternative widerspricht. Zwischen den beiden Kontrollflußprimitiven KFP_1 und KFP_2 besteht kein Zusammenhang, den das WFMS überprüfen könnte. Beim Kontrollflußprimitiv Auswahl (Abbildung 2.6, Mitte) legt die Syntax fest, daß drei Workflows zu verbinden sind; der Entwickler muß nur die richtige Fortsetzung der beiden Alternativen sicherstellen. Die Ausführungsanweisung Alternative kapselt die beiden alternativ auszuführenden Workflows und bildet damit einen Ausdruck der Form Alternative (Bedingung, Workflow-2, Workflow-3). Die Fehlerquelle, alternativ ausgeführte Workflows an einer anderen Stelle in der Kontrollflußdefinition durch ein AND-Join (im Fall des Kontrollflußkonstrukts) oder eine gleichbedeutende Formulierung mit Kontrollflußprimitiven zusammenzuführen, ist durch die Ausführungsanweisung ausgeschlossen.

Anzahl und Verteilung der Anknüpfungsstellen (grau hinterlegt) zeigen, daß bei Kontrollflußprimitiven bereits vorhandene Kontrollflußdefinitionen an jeder Stelle erweitert werden können. Allerdings kann eine Veränderung die Bedeutung der vorhandenen Kontrollflußdefinition verändern (unbemerkt und unbeabsichtigt vom Entwickler), weshalb diese Flexibilität fast als Nachteil anzusehen ist. Bei Kontrollflußkonstrukten sind die Erweiterungsmöglichkeiten etwas eingeschränkter; nur an den „Rändern" des Fragments sind neue Kontrollflußkonstrukte möglich, was allerdings nicht verhindern kann, daß im Beispiel von Abbildung 2.6 eine Zusammenführung beider Alternativ-Äste durch ein AND-JOIN erfolgt. Erst bei Ausführungsanweisungen sind die Workflows nicht mehr direkt erreichbar; eine weitere Ausführungsanweisung kann das vorhandene Fragment „umschließen", aber die darin festgelegte Workflow-Ausführung nicht mehr verändern. Eine abschließende Bewertung dieser und weiterer Eigenschaften nimmt der nächste Abschnitt vor.

2.3.5 Vergleichende Bewertung

Der Grund für die Kontrollflußdefinition im Workflow-Schema ist eine für die Anwendung notwendige Abfolge von Subworkflows. Ist die Anwendung durch *kausale* Abhängigkeiten der Subworkflows geprägt (*wenn ... dann ...*), geben Kontrollflußprimitive dies unmittelbar wieder und lassen die Modellierung intuitiver erscheinen. Die Ablaufmodellierung mit Kontrollflußprimitiven erfordert Kenntnisse über das Workflow-Ausführungsmodell im WFMS, weil erst aus dem Zusammenspiel von richtig gesetzten Kontrollflußprimitiven *und* Ausführungsmodell die gewünschte Ausführung mehrerer Workflows folgt. Da eine Workflow-Instanz mit mehreren Kontrollflußprimitiven in Verbindung stehen kann, muß der Entwickler die gegenseitige Beeinflussung berücksichtigen, um Widersprüche oder unerfüllbare Bedingungen zu vermeiden. Erfahrungen mit aktiven Datenbanken und ECA-Regeln zeigen, daß dies bei größeren Anwendungen problematisch ist. Von Vorteil ist die Möglichkeit, Kontrollflußprimitive auf Kanten und Workflows auf Knoten abzubilden und die Ablaufbeschreibung z.B. durch einen gerichteten Graphen zu veranschaulichen.

Kontrollflußdefinitionen mit Ausführungsanweisungen drücken die Ausführungsfolge der Workflows direkt aus und sind ohne Kenntnis des Ausführungsmodells der Workflows verständlich. Besonders deutlich wird dies, wenn mit einer Ausführungsanweisung explizit festgelegt wird, daß ein Workflow mehrfach ausgeführt oder ein ganzer Teilausdruck mehrfach abgearbeitet werden soll. Beim Einsatz von Kontrollflußprimitiven ist nur durch Untersuchung der gesamten Kontrollflußdefinition und dem „Verfolgen" aller möglichen Verzweigungen erkenntlich, daß ein Zyklus vorliegt, über den die Wiederholung realisiert wird. Bei umfangreicheren Kontrollflußdefinitionen ist die Erkennung von Zyklen aufwendig, das Problem ist beim Einsatz von ECA-Regeln in aktiven Datenbanken bekannt und nur für Spezialfälle zufriedenstellend gelöst (vgl. [Vadu97]).

Ausführungsfolgen mit vorwiegend *temporalen* Abhängigkeiten (*vor, nach* etc.) der Subworkflows profitieren von Ausführungsanweisungen. Beziehungen zwischen Workflows, die nicht Argumente einer Ausführungsanweisung sind, müssen auf anderem Weg hergestellt werden. Die Schachtelung von Ausführungsanweisungen erzeugt geschlossene Ausdrücke,

deren Visualisierung als Syntaxbaum allerdings unanschaulicher als ein Ablaufgraph ist. Vorteilhaft sind Ausführungsanweisungen bei komplizierten Ausführungsfolgen, die nur durch viele Kontrollflußprimitive auszudrücken wären. In einer einzigen Ausführungsanweisung befindet sich für den Entwickler die gesamte Funktionalität, weniger fehleranfällig und besser wiederverwendbar als durch ein Geflecht aus Kontrollflußprimitiven. Die Ausführung eines Workflows als Argument einer Ausführungsanweisung ergibt sich vollständig aus der Semantik der Anweisung und muß nicht wie bei Kontrollflußprimitiven erst herausgefunden werden, indem alle Kontrollflußprimitive, die mit dem Workflow in Verbindung stehen, interpretiert werden. Hinsichtlich der Portabilität ist die kompakte Semantik der Ausführungsanweisung ein großer Vorteil. Bei Kontrollflußdefinitionen mit Kontrollflußprimitiven muß neben dieser Beschreibung auch die Interpretationsregel für Kontrollflußprimitive und das Ausführungsmodell der Workflows vermittelt werden, um im Zielsystem die gleiche Ausführung zu garantieren.

Die Ausdrucksmächtigkeit von Kontrollflußprimitiven, Kontrollflußkonstrukten und Ausführungsanweisungen läßt sich nicht direkt vergleichen, weil die jeweilige Implementierung des einzelnen Konzepts darüber entscheidet. Setzen Kontrollflußprimitive nur die Fertigstellung und den Start von Workflows in Beziehung, erscheint die Workflow-Ausführung als atomares Ereignis bzw. kann nur so behandelt werden, obgleich dies natürlich nicht der Fall ist. Kausale Abhängigkeiten der Art „*Wenn Workflow A aktiv geworden ist, starte Workflow B*“ können nicht ausgedrückt werden; jede Reaktion auf die Ausführung eines Workflows wird bis zu dessen Fertigstellung verzögert[1]. Dies ist kein Defizit des Konzepts von Kontrollflußprimitiven, sondern liegt an der Implementierung, die nur die Zustände done und startable des Ausführungsmodells der Workflows zugänglich macht, d.h. als „Anknüpfungspunkt“ für Beziehungen zur Verfügung stellt. Allein mit den Zuständen done und startable können die unterschiedlichen Formen nebenläufiger Ausführung mehrerer Workflows nicht beschrieben werden; eine differenzierte Abstimmung der Ausführung ist nicht möglich, weil die dafür signifikanten Zwischenzustände nicht erkennbar sind. Diese Einschränkung gilt gleichermaßen für Ausführungsanweisungen. Werden nur Ausführungsanweisungen für die sequentielle, alternative oder parallele Ausführung von Workflows angeboten, ist die Ausführung eines Workflows aus Sicht des Entwicklers ein atomares Ereignis, auf dessen Verlauf er keinen Einfluß hat. Die Kombination von Ausführungsanweisungen zu zusammengesetzten Ausdrücken ändert nichts an der „Auflösung“, mit der die Workflow-Ausführung beschrieben wird. Zusammenfassend ist festzustellen:

- Die Ausdrucksmöglichkeiten in der Kontrollflußdefinition hängen primär von den sichtbaren Zuständen im Ausführungsmodell der Workflow-Instanz ab. Ob die Zustände vom

[1] Der Fall, daß es zu dieser Fertigstellung nicht kommt, weil z.B. ein technisches Problem die korrekte Ausführung eines Workflows verhindert, wird hier nicht diskutiert. Die Behandlung solcher Fehlerfälle ist zudem keine Aufgabe der Workflow-Schema-Entwicklung, sondern muß durch dedizierte Fehlerbehandlungsmechnismen des WFMS erfolgen, die dann für Instanzen aller Workflow-Schemata zuständig sind.

Entwickler mit Kontrollflußprimitiven einzeln in Beziehung gesetzt werden oder ob dies in der Implementierung von Ausführungsanweisungen geschieht, ist sekundär.

- Kausalbeziehungen werden durch Kontrollflußprimitive direkt ausgedrückt. Da nur einzelne Ereignisse der Workflow-Ausführung und nicht die Workflow-Ausführung als Ganzes in Beziehung gesetzt werden, ist mehr Sorgfalt bei der Kontrollflußdefinition notwendig. Es besteht anderenfalls die Gefahr, widersprüchliche Anforderungen an die Ausführung eines Workflows zu stellen, weil Kontrollflußprimitive immer nur einzelne Abhängigkeiten zum Ausdruck bringen. Ausführungsanweisungen setzen die Ausführung von Workflows als Einheit in Beziehung, was die Formulierung temporaler Abhängigkeiten begünstigt, aber in puncto Präzision im Ausdruck hinter den Kontrollflußprimitiven zurückbleibt.

- Beschreiben Ausführungsanweisungen den Verhaltensaspekt, verlangt die spezielle Funktion jeder einzelnen Ausführungsanweisung ein größeres Repertoire dieser Ausdrucksmittel. Bei Kontrollflußprimitiven wird die Auswirkung durch den Kontext der Verwendung bestimmt; die Funktion einer Ausführungsanweisung ist kontextinvariant.

- Die Vorteile beider Ansätze stehen zur Verfügung, wenn Ausführungsanweisungen auf der Basis von Kontrollflußprimitiven definiert werden und danach als Erweiterung der Workflow-Sprache angeboten werden. Komplizierte Ausführungsbeziehungen mehrerer Workflows sind nach einmaliger Definition handhabbar und ohne Kenntnis von Implementierungsdetails einzusetzen, können aber unter Rückgriff auf die Definition jederzeit erklärt werden.

Die genannten Gründe legen die Verwendung von Ausführungsanweisungen nahe, wenn die Workflow-Sprache erweiterbar ist. Zudem sind Ausführungsanweisungen ein „Container“, der weitere Funktionen realisieren und kapseln kann. Der vorliegende Bericht geht auf diese Möglichkeiten im weiteren ein.

3 Entwicklung neuer Ausführungsanweisungen

Abschnitt 3.1 motiviert die Entwicklung neuer Ausführungsanweisungen durch ein Beispiel: Beim Entwurf eines Workflow-Schemas ist die Ausführungsfolge der Subworkflows festzulegen; die Reihenfolge gibt der Arbeitsvorgang vor. Die Bewertung der verschiedenen Möglichkeiten zeigt, welche Vorteile erweiterbare Workflow-Sprachen und die Einführung neuer Ausführungsanweisungen haben. Für die Entwicklung von Ausführungsanweisungen führt Abschnitt 3.2 ein *Funktionsmodell* ein. Das Funktionsmodell dient der Erklärung; es beschreibt die Funktionsweise einer Ausführungsanweisung, ohne Details einer Implementierung vorwegzunehmen oder vorauszusetzen. Ein weiteres Hilfsmittel der Entwicklung sind Verteilte Transitionssysteme in Abschnitt 3.3. Verteilte Transitionssysteme spezifizieren das Verhalten von Systemen aus miteinander agierenden Komponenten. Ausführungsanweisungen und die von ihnen kontrollierten Workflows sind — auf der Abstraktionsebene des Funktionsmodells — interagierenden Komponenten, was die Verwendung der Verteilten Transitionssysteme motiviert. Die an sprechenden Symbolen orientierte Notation der Verteilten Transitionssysteme liefert gut lesbare Beschreibungen dieser „Kontrollfluß-Workflow-Systeme“; zudem erlaubt die formale Darstellung, durch Auswertungs- und Analysealgorithmen Eigenschaften der entwickelten Ausführungsanweisungen zu überprüfen. Abschnitt 3.4 zeigt den Einsatz der Verteilten Transitionssysteme anhand eines schrittweise erstellten Funktionsmodells einer Ausführungsanweisung.

3.1 Motivation für die Entwicklung einer Ausführungsanweisung

Bei der Erstellung einer WFMA analysiert der Anwendungsentwickler die Arbeitsvorgänge und konzipiert danach die Workflow-Schemata, die auch eine Reihenfolge festlegen, in der das WFMS die Subworkflows ausführt.

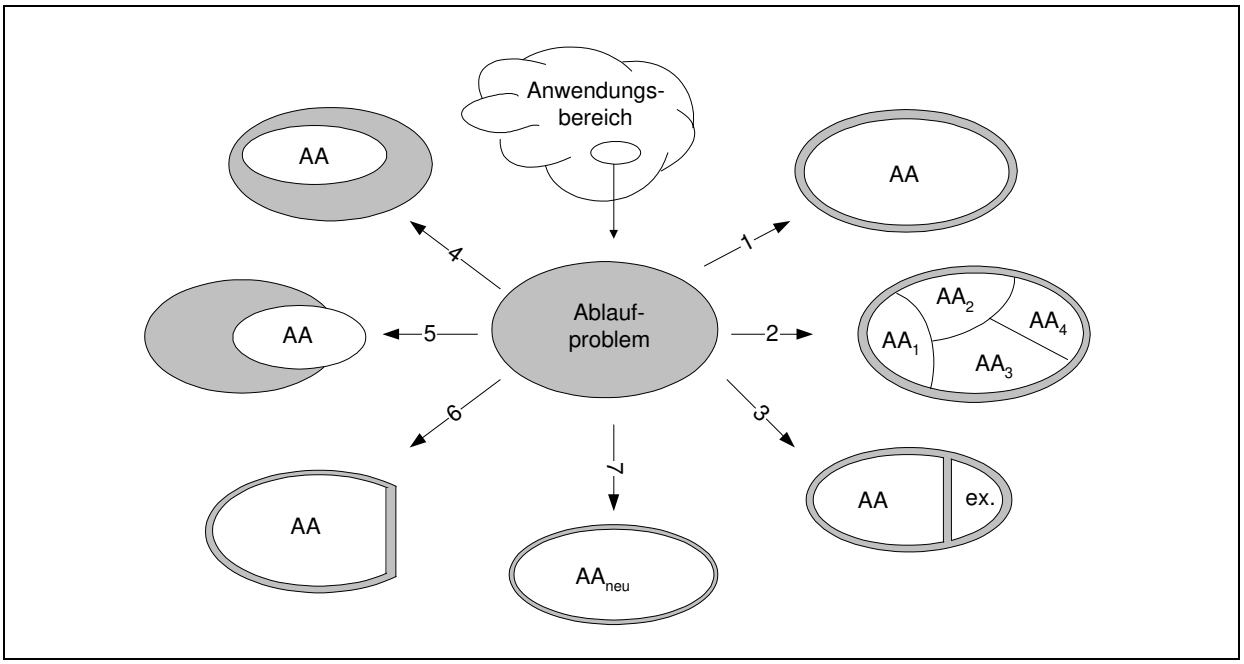


Abbildung 3.1: Realisierungsvarianten einer Workflow-Ausführung

Abbildung 3.1 stellt Möglichkeiten dar, ein Ablaufproblem (grau) durch Ausführungsanweisungen (weiß) zu beschreiben. Die Grafik symbolisiert die Annäherung von Problem und Lösung durch den Grad an Überdeckung beider Flächen. Folgende Möglichkeiten bestehen:

1. Die Workflow-Sprache bietet eine passende Ausführungsanweisung an; ihre Verwendung legt die Workflow-Ausführung in der geforderten Form fest.

2. Die Komposition mehrerer Ausführungsanweisungen liefert einen Ausdruck, der die gewünschte Workflow-Ausführung festlegt. Verschiedene Ausdrücke können die gleiche Workflow-Ausführung definieren; sie unterscheiden sich aber in den nichtfunktionalen Eigenschaften. Variationen bei der Bildung kompositer Ausführungsanweisungen sind ein zusätzlicher Freiheitsgrad für den Entwickler (vgl. [Böhm96]).

3. Reichen die Ausführungsanweisungen nicht aus, ist eine Realisierung der Ausführungslogik außerhalb des WFMS notwendig. In einem eigens dafür erstellten Steuerungsprogramm oder Betriebssystem-Script wird der gewünschte Ablauf explizit ausprogrammiert. Die Variante entzieht dem WFMS die Kontrolle und damit jede Einflußmöglichkeit.

4. Die eingesetzte Ausführungsanweisung garantiert die korrekte Ausführung der Workflows, bringt aber z.B. dem Anwender Nachteile. Nachteile können in der Begrenzung auf eine Auswahl von Abläufen bestehen, obwohl aus Sicht der Anwendung weitere Abläufe möglich sind.

5. Das Ablaufproblem wird partiell realisiert. Die verwandte Ausführungsanweisung bietet nur eine Teilmenge der erforderlichen Ausführungsfolgen an, läßt aber auch Ausführungsfolgen zu, die im Arbeitsvorgang bisher nicht vorgesehen waren.

6. Veränderungen des Arbeitsvorgangs führen dazu, daß sich die Menge der zulässigen Workflow-Ausführungen ändert, bis sie durch eine vorhandene Ausführungsanweisung beschrieben werden kann.

Eine detaillierte Gegenüberstellung dieser Realisierungsmöglichkeiten erfolgt in [Böhm97b] und wird hier nicht vertieft. Diskutiert wird hier die verbleibende Realisierungsvariante 7 durch die Entwicklung einer neuen Ausführungsanweisung, die auf die Anforderungen des Ablaufproblems zugeschnitten ist. Als Vorbereitung der Entwicklung stellt der nächste Abschnitt ein konzeptionelles Funktionsmodell für Ausführungsanweisungen vor.

3.2 Funktionsmodell für Ausführungsanweisungen

Abschnitt 2.2 führte die koordinierte Ausführung mehrerer Workflows auf den kontrollierten Wechsel von Zuständen der beteiligten Workflow-Instanzen zurück und charakterisierte die Ausführungsanweisung als kompaktes Ausdrucksmittel für eine derart gesteuerte Workflow-Ausführung. Demnach besteht die *Funktion* einer Ausführungsanweisung darin, die Zustände der von ihr kontrollierten Workflow-Instanzen zu überwachen. Innerhalb eines WFMS sind Ausführungsanweisungen keine aktiven Objekte, daher ist „überwachen“ im übertragenen

Sinn zu verstehen und bedeutet: Durch die Ausführungsanweisung stehen dem WFMS die notwendigen Informationen zur Verfügung, wann ein Zustandswechsel von Workflow-Instanzen entweder aktiv herbeizuführen, sein Auftreten zu verhindern, zu verzögern oder darauf zu reagieren ist. In der Entwurfsphase ist es zulässig, als Modell für die Funktionsweise einer Ausführungsanweisung die aktive Einflußnahme auf Zustandswechsel der Workflow-Instanzen anzunehmen. Das Funktionsmodell führt Ausführungsanweisungen weiterhin als Objekte ein, die zusammen mit den von ihnen kontrollierten Workflow-Instanzen ein reaktives System bilden. Ausführungsanweisungen haben zudem einen eigenen Zustand, der Zustandsänderungen der Workflow-Instanzen widerspiegelt. Indem das Verhalten dieses Systems (insbesondere die Reaktion auf Ereignisse) festgelegt wird, wird die Funktion der Ausführungsanweisung definiert und darüber die Ausführung der Workflows. Eine Beschreibung der Ausführungsanweisung umfaßt die Identifikation zulässiger Zustandskonstellationen von kontrollierten Workflow-Instanzen, die Reihenfolge, in der die Zustandskonstellationen auftreten können, sowie die Festlegung der Modalität von Übergängen zwischen einzelnen Zustandskonstellationen.

3.2.1 Ausdrucksmöglichkeiten im Funktionsmodell

Das Funktionsmodell erklärt Ausführungsanweisungen über die Beziehungen zwischen Zuständen der Ausführungsmodelle beteiligter Workflow-Instanzen. Abschnitt 3.2.1.1 stellt die unterschiedlichen Typen von Beziehungen vor, aus denen sich Klassen von Ausführungsanweisungen ableiten lassen. Die Klassifikation beginnt in Abschnitt 3.2.1.2, wo Beziehungstypen auf ihre spezielle Bedeutung für die Workflow-Ausführung untersucht werden.

3.2.1.1 Beziehungstypen im Funktionsmodell

Die Zustände von Ausführungsmodellen mehrerer Workflow-Instanzen können auf unterschiedliche Weise in Beziehung stehen; dieser Abschnitt stellt die möglichen Beziehungen vor und typisiert sie. Diese Typisierung ist der erste Schritt zur Klassifikation von Ausführungsanweisungen und basiert auf den verschiedenen Kombinationen, in denen Zustand und Zustandsübergang von Workflow-Instanzen in die Beziehung eingehen. Es wird absichtlich von „Beziehung“ und nicht nur von „Abhängigkeit“ gesprochen, weil Abhängigkeiten einen Ursache-Wirkung-Zusammenhang assoziieren, der nur eine von mehreren Formen der Beziehung ist. Zur Veranschaulichung möglicher Beziehungstypen zeigt Abbildung 3.2 die als Zustandsübergangs-Diagramm dargestellten Ausführungsmodelle zweier Workflow-Instanzen. Die Ausführungsmodelle sind auf vier Zustände (z_1-z_4 und z'_1-z'_4) reduziert und müssen auch nicht für beide Workflow-Instanzen identisch sein, zur Erläuterung der möglichen Beziehungstypen reicht diese Darstellung aus.

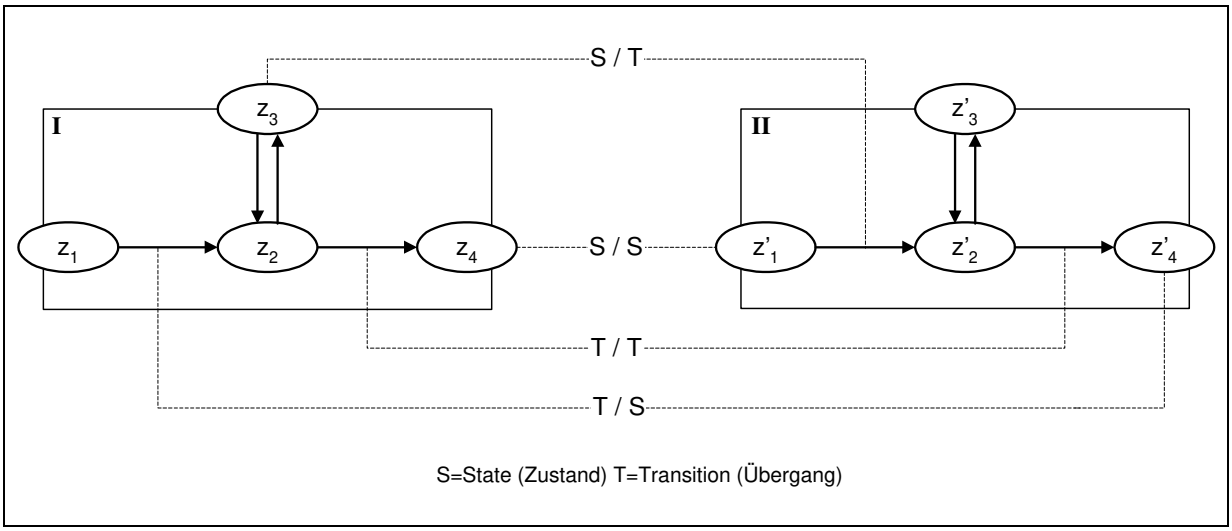


Abbildung 3.2: Beziehungen zwischen Zuständen und Zustandsübergängen

Jede gestrichelte Linie charakterisiert einen Typ von Beziehung: Zwischen z_3 und dem Übergang z'_1 nach z'_2 besteht eine *S/T-Beziehung* (State/Transition), weil ein Zustand (S) der Workflow-Instanz I (links) mit dem Übergang (T) in einer Workflow-Instanz II (rechts) in Beziehung steht (als was die Beziehung interpretiert werden kann, beschreibt der nächste Abschnitt). Zwischen dem Zustand z_4 der Workflow-Instanz I und dem Zustand z'_1 der Workflow-Instanz II ist eine *S/S-Beziehung* (State/State) angedeutet und eine *T/T-Beziehung* (Transition/Transition) ist zwischen dem Übergang von z_2 nach z_4 in Workflow-Instanz I und zwischen z'_1 und z'_3 in Workflow-Instanz II gezeigt, schließlich noch die *T/S-Beziehung* (Transition/State) zwischen dem Zustandsübergang von Workflow-Instanz I und dem Zustand in Workflow-Instanz II. Weitere Typen von Beziehungen gibt es zwischen zwei Zustandsübergangs-Diagrammen nicht.

3.2.1.2 Bedeutungen von Beziehungen im Funktionsmodell

Nach der Unterscheidung von Beziehungen in vier verschiedene Typen muß für die weitere Verwendung im Funktionsmodell präzisiert werden, welche Bedeutung jeder Typ von Beziehung hat. Mögliche Bedeutungen werden in den folgenden Abschnitten vorgestellt; ihre Konsequenzen für die Ausführungsanweisungen und die Workflow-Ausführung werden diskutiert.

Beziehungen als Ursache-Wirkung-Zusammenhänge in verschiedenen Modalitäten

Naheliegend ist die Interpretation einer Beziehung als Ursache-Wirkung-Zusammenhang, am Beispiel eines S/T-Typs heißt das: Befindet sich eine Workflow-Instanz im Zustand s, *muß* die andere Workflow-Instanz den Zustandsübergang t vornehmen bzw. das WFMS muß dies sicherstellen. Ist dies nicht möglich, und technische Probleme können leicht dazu führen, den geforderten Zustandsübergang t zu verhindern, muß eine Ausnahmebehandlung stattfinden. Sie kann darin bestehen, die Workflow-Ausführung an dieser Stelle zu unterbrechen und einen manuellen Eingriff von außen zu gestatten, der durch geeignete Workflow-Operationen die Situation auflöst. Statt eine zwingend erforderliche Reaktion zu unterstellen („muß“ über-

gehen), ist auch die Option möglich: Befindet sich eine Workflow-Instanz im Zustand s, *kann* die Workflow-Instanz den Zustandsübergang t vornehmen (eine Unterscheidung bezüglich der Zeitspanne, in der die Möglichkeit besteht, ist Gegenstand weiterer Forschung, gleiches gilt für die Behandlung der oben skizzierten Ausnahmesituation, vgl. dazu [Stro95, Hage98]).

„Können" und „Müssen" sind *Modalitäten* (vgl. [Lore85]), weiterhin gibt es noch das „geboten sein" und das „verboten sein". Ohne die in die Philosophie führenden Konsequenzen dieser Unterscheidung näher zu betrachten, ist die Einteilung in „kann", „muß", „darf" und „freigestellt" so grundlegend, daß sie der angestrebten Allgemeingültigkeit des Funktionsmodells gerecht wird. Damit entsteht neben der Typisierung der Beziehungen eine zweite Klassifikationslinie für Ausführungsanweisungen nach den Modalitäten, denn es ist ein Unterschied, ob das WFMS nach Maßgabe der Ausführungsanweisung Zustandsübergänge in Workflow-Instanzen erzwingt, gestattet, verhindert oder freistellt. In den Modalitäten liegt der Unterschied zu Ansätzen, Abhängigkeiten z.B. zwischen Transaktionen zu beschreiben (z.B. ACTA [Chry92] oder TANGO [Band94]). Eine detaillierte Gegenüberstellung ist in Arbeit.

Die Modalitäten von Kontrollflußprimitiven, die sich ansonsten auf das Funktionsmodell zurückführen lassen, sind nicht immer eindeutig, wie das folgende Beispiel zeigt: Im WFMS FLOWMARK verwendete Kontrollflußkonnektoren sind Beziehungen des Typs S/T, denn sie setzen den Zustand done einer Aktivität mit dem Zustandsübergang von instanciated in startable in Beziehung. Die Modalität dieser Beziehung ist kontextabhängig, weil das Ausführungsmodell des WFMS entweder alle einlaufenden Kontrollflußkonnektoren als Voraussetzung für die Ausführungsbereitschaft berücksichtigt oder nur eine, je nach Definition der betroffenen Aktivität. Im ersten Fall hat die Beziehung die Modalität „kann", im zweiten Fall die Modalität „muß"; die Modalität ist in diesem Fall also nicht eindeutig.

Eine Interpretation als Ursache-Wirkung-Zusammenhang ist nur für S/T- und T/T-Typen sinnvoll, weil hier die „Wirkung" im Zustandsübergang besteht, respektive seiner Verhinderung bei anderen Modalitäten. Für Beziehungen des S/S- und T/S-Typs ist die Interpretation als Ursache-Wirkung-Zusammenhang nicht stimmig, da die „Wirkung" in einem Zustand (S) bestehen würde, dessen Vorliegen bestenfalls überprüft werden kann. Dieser Sachverhalt motiviert eine weitere Interpretation der Beziehung, die im nächsten Abschnitt vorgestellt wird.

Beziehungen als aussagenlogische Prädikate

Die Interpretation einer Beziehung als Ursache-Wirkung-Zusammenhang ist von der Vorstellung geprägt, die Ausführung von Workflows primär durch das WFMS zu steuern. In vielen Anwendungssituationen sind diese Vorstellung und in ihrem Verständnis entwickelte Ausführungsanweisungen sinnvoll. Für eine vollständige Abdeckung des Spektrums möglicher Ausführungsanweisungen muß allerdings noch eine weitere Art der Workflow-Ausführung betrachtet werden: Durch Ausführungsanweisungen erhält das WFMS keine ablauforientierte Vorschrift zur Ausführung der Workflows, sondern eine deklarativ formulierte Vorgabe, welche Formen der Workflow-Ausführung zulässig oder verboten sind. Die Festlegung kann wie bisher auf Zustandskonstellationen oder Zustandsänderungen zurückgreifen, nur die Bedeu-

tung dieser Angaben ändert sich. Der Anwender wird die Lage versetzt, selber über die Ausführungsfolge von Workflows zu entscheiden — das WFMS „beobachtet“ bei der Workflow-Ausführung auftretende Zustände der Workflow-Instanzen. Die Reaktionen auf die beobachteten Zustände kann von Akzeptieren über Protokollieren bis zu Verhindern reichen, je nach Formulierung in der Ausführungsanweisung. Nach dieser Auffassung werden die Beziehungen zwischen Zuständen und Zustandsübergängen der Workflow-Instanzen zu Aussageformen, die durch Werte der gerade ablaufenden Workflow-Ausführung einen Wahrheitswert bekommen, der auch den Zustand der Ausführungsanweisung bestimmen kann. Die Implementierung einer solchen Form der Workflow-Ausführung birgt Probleme, von denen einige in [Tomb97] im Zusammenhang mit der Erkennung von Ereignissen in verteilten Ausführungsumgebungen beschrieben werden. Ausführungsformen, die auf ein Funktionsmodell mit aussagenlogischer Interpretation der Beziehungen zurückgehen, werden in diesem Bericht nicht weiter behandelt.

3.2.1.3 Zustände der Ausführungsanweisung

Ausführungsanweisungen wurden als zustandstragende Objekte eingeführt, und der letzte Abschnitt lieferte ein Beispiel für einen Zustand, der je nach Funktionsweise des Konstrukts aus den Zuständen bzw. Zustandsübergängen der kontrollierten Workflow-Instanzen folgt. Die Zustände der Ausführungsanweisung vervollständigen das Funktionsmodell, weil in diesen Zuständen Aussagen über die Ausführungsanweisung selbst und die Workflow-Instanzen codiert sind. Beispielsweise kann die Ausführungsanweisung einen Zustand done einnehmen, wenn die von ihr kontrollierten Workflows beendet sind. Über die Zustände der Ausführungsanweisung lassen sich Beziehungen zu weiteren Workflow-Instanzen beschreiben, etwa das Zusammenspiel von mehreren Subworkflows und einem Superworkflow oder bei der Komposition mehrerer Ausführungsanweisungen zu zusammengesetzten Ausdrücken. Man beachte auch hier den Charakter der Funktionsmodells — es dient der Erklärung und Planung neuer Ausführungsanweisungen, und in dieser Rolle kann sich der Zustand einer Ausführungsanweisung durchaus aus den Zuständen der kontrollierten Subworkflows ableiten.

3.2.2 Beispiele für Beziehungstypen bei der Ausführung von Workflows

Die folgenden Abschnitte erläutern die vorgestellten Beziehungstypen anhand von Beispielen; Abbildung 3.3 am Ende des Abschnitts stellt die Beispiele (1) bis (4) nochmals dar.

Beziehungen zwischen Zuständen und Zustandsübergängen (S/T-Typ)

Beim S/T-Typ ist der Zustand einer Workflow-Instanz Anlaß oder Vorbedingung für eine Zustandsänderung der anderen Workflow-Instanz. „Anlaß“ bedeutet, die zweite Workflow-Instanz *muß* diesen Zustandsübergang vollziehen, damit wieder ein zulässiger Zustand aus

Sicht der Ausführungsanweisung erreicht wird[2]. Bei einer „Vorbedingung“ *kann* die zweite Workflow-Instanz den Zustandsübergang vornehmen; es bleibt ohne den Vollzug ein zulässiger Zustand für die Ausführungsanweisung. Folgende Beispiele illustrieren die Fälle:

(1) Darf Workflow-Instanz B nur gestartet werden, wenn Workflow-Instanz A aktiv ist, muß die Ausführungsanweisung den Zustand running der Workflow-Instanz A zur Zusatzbedingung machen, um Wechsel von ready in running der Workflow-Instanz B zu erlauben. Es handelt sich also um eine zusätzliche Restriktion für den Zustandsübergang von ready in running der Workflow-Instanz B.

(2) Muß Workflow-Instanz B gestartet werden, wenn Workflow-Instanz A beendet ist, muß die Ausführungsanweisung den Zustand done zum Anlaß nehmen, in Workflow-Instanz B den Zustandsübergang von ready nach running zu erzwingen.

Im Fall (2) tritt wieder das Problem auf, daß der erforderliche Zustandsübergang in Workflow-Instanz B nicht garantiert werden kann, auch wenn die Voraussetzung (das Vorliegen eines Zustands in Workflow-Instanz A) erfüllt ist. Solange der Zustandsübergang nicht vorliegt, befinden sich die beiden Workflow-Instanzen insgesamt in einem unzulässigen Gesamtzustand, der theoretisch bis zur Erfüllung der „Nachbedingung“ andauert. Da ein unendlich langes Warten in der Praxis natürlich nicht sinnvoll ist, muß die Situation anders behandelt werden: Es ist eine Zeitschranke notwendig, die das Ende einer Wartezeit markiert. Wird diese Zeitschranke überschritten, muß von außen in die Workflow-Ausführung eingegriffen werden, weil ein Rücksetzen auf den vorherigen Zustand im allgemeinen nicht möglich ist.

Weitere Typen von Beziehungen ergeben sich, wenn der neue Zustand nicht sofort eingenommen wird, sondern erst zu einem späteren Zeitpunkt. Das bedeutet für das zweite Beispiel, daß die Workflow-Instanz B nicht automatisch gestartet wird, sondern ihre Ausführung ein notwendiges Kriterium für das Ende des übergeordneten Workflows ist, das zu einem beliebigen Zeitpunkt erfüllt werden kann. Diese Form von Beziehung führt zu einer neuen Klasse von Ausführungsanweisungen und wird zu einem späteren Zeitpunkt behandelt.

Beziehungen zwischen zwei Zustandsübergängen (T/T-Typ)

Bei T/T-Typen werden Zustandsübergänge in zwei Workflow-Instanzen so verbunden, daß ein Übergang wieder als Anlaß oder als Vorbedingung für den anderen Übergang dient. Eine dritte Möglichkeit besteht darin, entweder beide Übergänge oder keinen von beiden zu fordern, wie das folgende Beispiel zeigt:

---

[2] Da über das Eintreten eines Zustands in der Zukunft keine sichere Annahme gemacht werden kann, müssen Maßnahmen vorgesehen werden, die beim Ausbleiben des erwarteten Ereignisses ergriffen werden. Das setzt weiterhin voraus, daß auch ein Zeitfenster definiert wird, in dem das Eintreffen erwartet wird. Nach Ablauf der Frist ohne den erwähnten Zustandsübergang ist demnach ein Ausnahme- oder Fehlerzustand erreicht, alternativ kann auch versucht werden, den ursprünglichen Zustand wieder herstzustellen.

(3) Dürfen zwei Workflows A und B nur gemeinsam ausgeführt werden, ist für die Ausführungsanweisung der Zustandsübergang nur zulässig, wenn *beide* Workflow-Instanzen ihren Zustand von ready in running ändern oder nichts geschieht. Jeder einseitige Vollzug des Zustandsübergangs ist demnach unzulässig und wird, je nach Festlegung in der Ausführungsanweisung, vom WFMS behandelt. Die Reaktion des WFMS kann z.B. darin bestehen, den erfolgreichen Zustandsübergang rückgängig zu machen, um den ursprünglichen Zustand zu erreichen[3].

Im Unterschied zu Beziehungen des S/T-Typs wird das Verlassen eines Zustands als Vorbedingung/Anlaß genommen und nicht das Vorliegen oder Eintreten des Zustands. In Anbetracht des Anwendungsgebiets wird klar, warum beide Beziehungstypen notwendig sind: Nicht alle Zustandsänderungen einer Workflow-Instanz können jederzeit vollzogen werden und sind nicht garantiert erfolgreich (siehe Beispiel (3)). Daher kann das Verlassen eines Zustands nicht immer durch Erreichen des Nachfolgezustands ersetzt werden, zumal der Nachfolgezustand nicht eindeutig sein muß. Für bestimmte Anwendungsfälle reicht bereits der Versuch auf Seiten des einen Workflows, den Zustand zu ändern, um mit einem zweiten Workflow darauf zu reagieren, unabhängig davon, ob die Zustandsänderung des ersten Workflows letztlich erfolgreich ist oder nicht.

Beziehungen zwischen zwei Zuständen (S/S-Typ)

Eine Beziehung des S/S-Typs kann auf S/T- oder T/T-Typen zurückgeführt werden, wenn die Beziehung die Modalität „erzwingen“ hat. Das geht nicht, wenn zwei Zustände nicht gemeinsam auftreten dürfen, also die Modalität „verboten“ gilt, wie das folgende Beispiel zeigt:

(4) Soll mit einer Ausführungsanweisung verhindert werden, daß zwei Workflow-Instanzen gleichzeitig aktiv sind, aber sonst in beliebigem Wechsel ausgeführt werden dürfen, ist eine Konstellation unzulässig, in dem beide Workflow-Instanzen im Zustand running sind. Die Ausführungsanweisung darf demnach alle Zustandsübergänge erlauben, die diese Situation nicht entstehen lassen. Abschnitt 4.2.5 stellt eine Anwendungsmöglichkeit vor.

Alle Beziehungstypen wurden nur für zwei Workflow-Instanzen formuliert. Die Erweiterung auf drei und mehr Workflows steht noch aus. Generell gilt jedoch die folgende Einschränkung: Nicht alle Kombinationen von Zuständen und Zustandsübergängen sind auch praktisch zu implementieren, weil die Einnahme eines Zustands zu Effekten führen kann, die auch durch Rücknahme des Zustands nicht ungeschehen zu machen sind. Die Rücksetzbarkeit von Auswirkungen der Workflow-Ausführung ist ein sehr komplexes Thema und wird derzeit

---

[3] Die Anwendung einer solcher Ausführungsanweisung setzt voraus, daß der Zustandsübergang und alle damit verbundenen Auswirkungen rückgängig gemacht werden können, was keineswegs immer der Fall ist. Hat der Zustand running einer Workflow-Instanz z.B. zur Zuweisung an einen Bearbeiter geführt, ist das Rücksetzen auf ready nicht mehr möglich, weitere Beispiele nicht reversibler Zustände können leicht gefunden werden.

noch erforscht, weil Transaktionskonzepte aus dem Bereich Datenbanken allein als Lösung dazu nicht ausreichen (siehe z.B. [Leym95, Lieb98]).

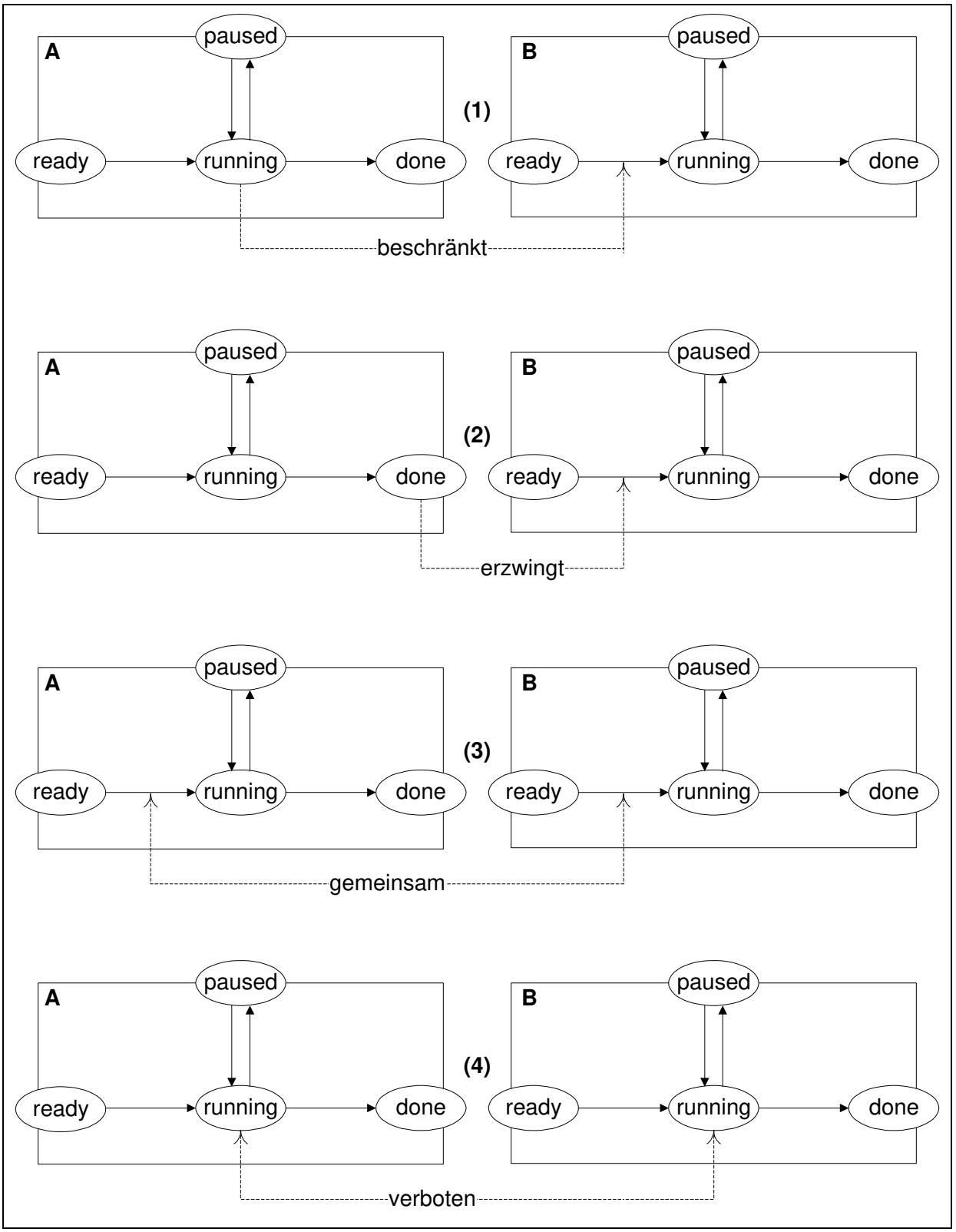


Abbildung 3.3: Beispiele (1) bis (4)

3.2.3 Funktionsmodell-orientierte Entwicklung von Ausführungsanweisungen

Das Funktionsmodell diente bisher als allgemeines Erklärungsmodell für die Funktionsweise von Ausführungsanweisungen. Für die Entwicklung einer speziellen Ausführungsanweisung sind die in Beziehung gesetzten Zustände und Zustandsübergänge zu benennen. Voraussetzung für die Implementierung einer neuen Ausführungsanweisung ist zudem ein WFMS, in dem Ausführungsanweisungen in den Auswahlvorgang für ausführbereite Workflows eingreifen und damit das „Votum“ des Verhaltensaspekts mitbestimmen. Das aspektorientierte Ausführungsmodell des WFMS MOBILE arbeitet nach diesem Prinzip (vgl. [Jabl96a, S. 194ff.], [Bußl97, S. 55ff.]). Aus dieser Ausführungsumgebung folgen Randbedingungen, an denen sich der Entwurf einer Ausführungsanweisung orientieren muß. Dazu legt Abschnitt 3.2.3.1 die Rolle des Ausführungsmodells der Workflow-Instanzen dar. Abschnitt 3.2.3.2 behandelt die in Beziehung gesetzten Zustände und Zustandsübergänge unter dem Gesichtspunkt der Durchführbarkeit durch das WFMS. Unter „Durchführbarkeit“ ist die Möglichkeit des WFMS zu verstehen, die im Funktionsmodell der Ausführungsanweisung spezifizierte Konstellation von Zuständen aktiv herbeizuführen bzw. unzulässige Zustandskonstellationen zu verhindern.

3.2.3.1 Einfluß des Ausführungsmodells auf die Gestaltung von Ausführungsanweisungen

Im Funktionsmodell für Ausführungsanweisungen nehmen die Ausführungsmodelle der Workflow-Instanzen eine zentrale Rolle ein, entsprechend stark ist der Einfluß auf die Möglichkeiten zur Gestaltung einer Ausführungsanweisung und diesbezügliche Entwurfsentscheidungen. Viele der im folgenden genannten Zustände sind aus implementierungstechnischen Gründen Bestandteil des Ausführungsmodells, allerdings nur für interne Verwaltungsfunktionen des WFMS gedacht und daher von außen nicht sichtbar. Von „außen“ kann bedeuten, dem Entwickler zugänglich, der z.B. zwei dieser Zustände durch ein Kontrollflußprimitiv in Beziehung setzt. Bei erweiterbaren WFMS, die Zugang zu tieferen Ebenen der Implementierung bieten, sind im allgemeinen alle Zustände des Ausführungsmodells sichtbar. Daß nur solche Zustände in Beziehung gesetzt werden können, die im Ausführungsmodell vorkommen, ist offensichtlich; weitere Zusammenhänge legt die folgende Aufzählung dar:

- Können nur Zustände wie startable und done im Ausführungsmodell (mit der Bedeutung wie in Abschnitt 2.2.1.1) angesprochen werden, stellt sich die Ausführung des Workflows als atomares Ereignis dar, eventuelle Zwischenzustände (z.B. eine Blockierung) werden nicht erkannt. Das führt zu Ausführungsanweisungen, unter deren Kontrolle die Workflows entweder strikt hintereinander oder alternativ ausgeführt werden können, weil dafür die Detektion von Start und Ende der Workflow-Ausführung ausreicht. Die garantierte Parallelausführung von Workflow-Instanzen ist nicht zu beschreiben, weil dazu der tatsächlich erreichte Zustand running notwendig, aber nach obiger Annahme nicht erkennbar ist. Nur die „abgeschwächte“ Form der Parallelausführung ist möglich; sie besteht in der aus Sicht des WFMS zeitgleichen *Bereitstellung* von Workflows zur Ausführung, was allerdings nur die Möglichkeit der Parallelausführung, nicht aber ihre garantierte Durchführung bedeutet.

- Unterscheidet das Ausführungsmodell die Zustände ready (ausführbereit) und running (aktiv), sind differenzierte Formen der nebenläufigen Ausführung oder der gegenseitigen Einflußnahme zwecks Synchronisation zwischen mehreren Workflow-Instanzen möglich. Beispielsweise wird der Start einer Workflow-Instanz davon abhängig gemacht, ob eine andere Workflow-Instanz tatsächlich aktiv ist, was ohne einen sichtbaren Zustand running nicht möglich ist. Sichtbare Zustände zwischen ready und done erlauben zudem, die Ausführung nachfolgender Workflows vorzuverlegen, d.h. bereits vor einem erfolgten Ende des Vorgänger-Workflows zu beginnen. Dieser „optimistische" Ansatz verlangt allerdings, auf Fehler in der Ausführung des ersten Workflows zu reagieren, falls der vorzeitig gestartete Workflow davon betroffen ist (für ein Beispiel aus der Software-Entwicklung siehe [Band94, S. 2]).

- Zustände wie paused (vorübergehend inaktiv) und not_executing (wird nicht mehr ausgeführt) erweitern die Möglichkeiten zur Beschreibung der Workflow-Ausführung zusätzlich. Ausführungsanweisungen, die von einer garantierten Nichtausführung eines Workflows ausgehen können, bringen kompliziertere Abhängigkeiten zum Ausdruck (z.B. mit deadline aus [Jabl96a, S. 152]).

Die genannten Zustände sind nur Beispiele (siehe Abschnitt 2.2.1.1), auch deshalb, weil sich in erweiterbaren WFMS neue Ausführungsmodelle für Workflow-Instanzen einführen und darin anwendungsspezifische Zustände definieren lassen. Allerdings spezialisieren anwendungsspezifische Zustände das Ausführungsmodell, so daß darauf abgestimmte Ausführungsanweisungen erforderlich werden. Ob die Konfigurierung eines WFMS so weit reichen soll, daß ein eigener Satz von Ausführungsanweisungen für eine Teilmenge der Workflow-Schemata eingeführt wird, ist nicht allgemeingültig entscheidbar. Der Verlust der universellen Verwendung einer Ausführungsanweisung für alle Workflow-Schemata eines WFMS ist vertretbar, wenn damit anwendungsnahe Zusammenhänge zwischen Workflows definiert werden. Abschnitt 4.1.2 behandelt diese Frage im Zusammenhang mit Designentscheidungen für Ausführungsanweisungen.

Für die Entwicklung von Ausführungsanweisungen läßt sich die Rolle des Ausführungsmodells der Workflow-Instanzen wie folgt zusammenfassen:

- Das Angebot an Zuständen im Ausführungsmodell bestimmt die beschreibbaren Workflow-Ausführungen. Sichtbare Zwischenzustände im Ausführungsmodell erlauben differenzierte Formen der koordinierten Workflow-Ausführung, führen aber zu Problemen, die aus der Aufhebung der Isolations-Eigenschaften bei Transaktionen bekannt sind.

- Verlangt eine bestimmte Form der Workflow-Ausführung eine Erweiterung oder Anpassung des Ausführungsmodells der kontrollierten Workflow-Instanzen, besteht die Gefahr, daß dafür entwickelte Ausführungsanweisungen nicht allgemein verwendbar bleiben.

- Erlaubt ein WFMS sowohl die Einführung neuer Ausführungsanweisungen als auch die Definition spezieller Ausführungsmodelle für Workflow-Instanzen, sichert das zum einen ein Höchstmaß an Flexibilität und Gestaltungsfreiheit für den Entwickler, erfordert aber

zum anderen mehr Abstimmungsaufwand, um die Kompatibilität zwischen Ausführungsanweisungen und Workflow-Schemata sicherzustellen.

Der nächste Abschnitt untersucht Zustandsübergänge in Hinblick auf die Durchführbarkeit.

3.2.3.2 Befolgung spezifizierter Zustandsübergänge im Funktionsmodell

Ausführungsanweisungen nehmen auf die Workflow-Ausführung Einfluß, indem sie die Behandlung der Workflow-Instanzen durch das WFMS regeln, bzw. das WFMS entnimmt der Implementierung der Ausführungsanweisung diesbezügliche Informationen; die Implementierung der Ausführungsanweisung geht wiederum auf das Funktionsmodell zurück, das Zustände und Zustandsübergänge in Beziehung setzt. Die Einflußnahme der Ausführungsanweisung kann auf zwei Arten erfolgen: Die Ausführungsanweisung legt fest,

(1) bis zu welchem Ereignis ein Zustandsübergang einer Workflow-Instanz zu *verhindern* ist (z.B. „Workflow B ist ausführbereit, wenn Workflow A beendet ist.“) oder

(2) welcher Zustandsübergang einer Workflow-Instanz zu *erzwingen* ist, wenn ein Ereignis eingetreten ist („Wenn Workflow A beendet ist, muß Workflow B starten.“);

wobei nach dem eingeführten Funktionsmodell nur Ereignisse zählen, die auf die Einnahme oder das Verlassen eines Zustands bzw. den Zustandsübergang einer anderen Workflow-Instanz zurückgehen (die Reaktion auf Zeitereignisse wird nicht betrachtet, vgl. dazu z.B. [Jasp97, S. 303] oder [Gepp97, S. 307]). Im weiteren wird das „Verhindern“ oder „Erzwingen“ als *Befolgung* bezeichnet, weil beides vom WFMS *befolgt* werden muß, um die im Funktionsmodell intendierte Workflow-Ausführung zu leisten. Für den Entwurf von Ausführungsanweisungen liefert die Fähigkeit zur Befolgung eines Zustandsübergangs ein wichtiges Kriterium für die Implementierbarkeit. Implementierbar sind Ausführungsanweisungen, deren Funktionsmodell Zustandsübergänge vorsieht, die mit dem Ausführungsmodell der Workflow-Instanzen konform sind. Dieses Kriterium ist anhand der Topologie des Zustandsübergangs-Diagramms überprüfbar. Weiterhin müssen die in Beziehung gesetzten Zustände und Zustandsübergänge untersucht werden, weil hier Abhängigkeiten zu weiteren Workflows, externen Applikationen, Handlungen des Anwenders etc. bestehen. Folgende Beispiele verdeutlichen die Situation:

1. Ob der Zustandsübergang von running in done möglich ist, hängt bei kompositen Workflows vom Zustand der Subworkflows ab, bei elementaren Workflows beispielsweise von einer synchron aufgerufenen externen Applikation. Eine analoge Abhängigkeit zu Subworkflows kann beim Übergang in den Zustand paused bestehen, wenn damit nicht ausschließlich der betroffene Workflow unterbrochen werden soll, sondern auch parallel ausgeführte Subworkflows.

2. Das Ende eines Workflows, z.B. ausgelöst durch den Anwender oder das Ende einer synchron aufgerufenen externen Applikation, kann vom WFMS nicht verhindert werden; zudem tritt der Zustandswechsel von running in done zu einem unbekannten Zeitpunkt ein.

Für die Entwicklung von Ausführungsanweisungen bedeutet das:

- Übergänge in Zustände, die von Zuständen weiterer Workflow-Instanzen abhängen, können nicht durch eine einzige Maßnahme des WFMS erzwungen werden. Vielmehr wird eine Kaskade von Workflow-Operationen notwendig, um den im Funktionsmodell der Ausführungsanweisung geforderten Zustand herbeizuführen.
- Vom Anwender oder externen Applikationen abhängige Zustände einer Workflow-Instanz können ebenfalls nicht erzwungen werden. Ist das Vorliegen dieser Zustände die Voraussetzung für Zustandsübergänge einer weiteren Workflow-Instanz, blockiert ein Ausbleiben die weitere Workflow-Ausführung.
- Enthält das Funktionsmodell der Ausführungsanweisung Beziehungen des S/T-Typs mit einer Ursache-Wirkung-Interpretation, bestimmt das WFMS durch Befolgung dieser Beziehung die Workflow-Ausführung, und der Anwender reagiert auf die ihm angebotenen Möglichkeiten. Stehen die Beziehungen zwischen Zuständen der Ausführungsmodelle für aussagenlogische Prädikate, kann das WFMS die Handlungen des Anwenders (die sich auf Zustände der Workflow-Instanzen auswirken) nur als zulässig oder unzulässig bewerten. Damit bestimmt der Anwender die Workflow-Ausführung, und das WFMS überwacht lediglich die Einhaltung der in der Ausführungsanweisung definierten Randbedingungen.

Aus den im Funktionsmodell enthaltenen Beziehungstypen leiten sich Hinweise für die Implementierung der Ausführungsanweisung ab. Insbesondere die Befolgung von Beziehungen des S/T- und T/T-Typs muß durch ein Protokoll geregelt werden, aus dem die Reaktion auf zur Zeit nicht mögliche Zustandsänderungen der Workflow-Instanzen hervorgeht. Nicht alle Zustandsänderungen können zu einem späteren Zeitpunkt erneut angestoßen werden, ohne damit die von der Ausführungsanweisung intendierte Form der Workflow-Ausführung zu verändern. Ebenfalls komplizierter ist die Behandlung von Beziehungen des T/T-Typs, wenn auf die Nichtausführbarkeit eines Zustandswechsels mit der Rücknahme der bereits erfolgten Zustandsänderung reagiert werden muß.

3.2.4 Zusammenfassung

In diesem Abschnitt wurde die koordinierte Ausführung von Workflows auf den kontrollierten Wechsel von Zuständen der beteiligten Workflow-Instanzen zurückgeführt. Die Zustandsänderungen werden einerseits durch die Ausführungsmodelle der Workflow-Instanzen festgelegt und andererseits durch Ausführungsanweisungen weiter eingeschränkt. Die Einschränkung erfolgt nicht durch die Ausführungsanweisung selbst, sondern diese stellt dem WFMS die Informationen zur Verfügung, auf deren Grundlage die Entscheidung über zulässige Zustandskonstellationen getroffen wird. Diese Sichtweise auf die Funktionsweise einer Ausführungsanweisung motiviert das eingeführte Funktionsmodell, anhand dessen Entwurfsentscheidungen bei der Entwicklung neuer Ausführungsanweisungen getroffen werden. Im Funktionsmodell werden die Zustände und Zustandsübergänge von Workflow-Instanzen in Beziehung gesetzt und diesen Beziehungen ein Typ und eine Interpretation zugeordnet. Der Typ ergibt sich aus der Kombination von Zuständen und Zustandsübergängen, als Interpretation ist ein Ursache-Wirkung-Zusammenhang und eine Interpretation als aussagenlogisches Prädikat möglich.

Ein Ursache-Wirkung-Zusammenhang wird durch das WFMS so umgesetzt, daß es das Eintreten eines Zustands oder den Vollzug eines Zustandsübergangs als Anlaß nimmt, gemäß den Vorgaben der Ausführungsanweisung darauf zu reagieren und damit eine neue Zustandskonstellation zu erzeugen, in der beispielsweise ein weiterer Workflow ausführbar wird. Die Interpretation der Beziehung als aussagenlogisches Prädikat erlaubt die Beurteilung einer Zustandskonstellation als zulässig oder unzulässig im Sinne der Ausführungsanweisung. Die Behandlung unzulässiger Zustandskonstellationen und weiterer Ausnahmesituationen erfolgt nicht.

3.3 Verteilte Transitionssysteme

Das Funktionsmodell aus Abschnitt 3.1 führt Ausführungsanweisungen als übergeordnete Einheit mehrerer Workflow-Instanzen ein, indem es ihre Zustände in Beziehung setzt. Aufgabe dieser Einheit ist die Koordination der Ausführung mehrerer Workflows untereinander und in Abhängigkeit vom Zustand der Ausführungsumgebung. Eine Beschreibung dieser Aufgabe muß drei Dinge umfassen: das Verhalten einzelner Workflow-Instanzen gemäß dem Ausführungsmodell des Workflow-Schemas, die Menge der einzelnen Zustände mehrerer Workflow-Instanzen, die gemeinsam auftreten dürfen, sowie alle Veränderungen dieser Zustandsmenge.

Jede Workflow-Instanz kann durch ein Zustandsübergangs-Diagramm, einen endlichen Automaten oder vergleichbare Konzepte beschrieben werden. Deren Ausdrucksmächtigkeit reicht nicht aus, um über das Verhalten *mehrerer* Workflow-Instanzen Aussagen zu machen. Zustandsübergangs-Diagramme sind *entweder* Darstellungen eines Systems mit sequentiellem Verhalten und Nicht-Determinismus *oder* Darstellungen eines Systems mit parallelem Verhalten. Mehrere Workflow-Instanzen bilden aber ein System, in dem Parallelität und Nicht-Determinismus gemeinsam auftreten. Nicht-Determinismus folgt beispielsweise aus der Tatsache, daß zustandsändernde Handlungen des Anwenders nicht vorhersehbar sind, Parallelität[4] entsteht bei der gleichzeitigen Bearbeitung unterschiedlicher Workflow-Instanzen.

Ein Formalismus muß nicht-deterministisches sequentielles und paralleles Verhalten sowohl unterscheiden als auch unabhängig voneinander ausdrücken können. Diese Forderungen erfüllen beispielsweise *Verteilte Transitionssysteme*, die in [Cap94] zur Spezifikation von Prozessen dienen. Abschnitt 3.3.1 führt die Verteilten Transitionssysteme informell ein, eine formale Definition und weitere Beispiele für ihre Einsatzmöglichkeiten finden sich auch in [Cap94], [Reis87] und [Reis91]. Die Besonderheiten beim Einsatz Verteilter Transitionssysteme im Workflow-Management beschreibt Abschnitt 3.3.2.

[4] Eine gute Übersicht zu den Interpretationen und Bedeutungen von „Parallelität“ findet sich in [Cap94, S. 27ff].

3.3.1 Informelle Einführung

Verteilte Transitionssysteme sind eine Erweiterung einfacher Transitionssysteme, um ein System zu beschreiben, das sowohl paralleles als auch nicht-deterministisches sequentielles Verhalten zeigt. Verteilte Transitionssysteme stellen das Verhalten beliebig vieler Subsysteme dar, die ihre Zustände sowohl lokal ändern, aber auch miteinander interagieren, um dann neue Zustände einzunehmen. Verteilte Transitionssysteme bestehen aus einer Zustandsmenge und einer Menge von Transitionen; verschiedene Token repräsentieren darin unterschiedliche Zustände. Eine Transition hat eine linke und eine rechte Seite, auf jeder Seite können mehrere Token stehen. Eine Transition ist *möglich*, wenn alle links mittels $*$ konkatenierten Token (Repräsentanten für Zustände „vorher“) vorliegen, und führt dann ununterbrechbar zu den rechts aufgeführten Token (Repräsentanten für Zustände „nachher“). Man beachte, daß jedes Token einen lokalen Zustand eines Subsystems des verteilten Gesamtsystems repräsentiert, zu dem es keinen „globalen“ Gesamtzustand geben kann (vgl. [Cap94, S. 81]). Sind mehrere Transitionen möglich, findet jeweils nur eine Transition statt. Token haben den Charakter von Ressourcen; eine Transition konsumiert und produziert Ressourcen, die deshalb mehrfach vorkommen können. Tabelle 1 zeigt Strukturen von Transitionen. Die Anzahl der Token pro Seite ist nicht begrenzt, ebenso können beliebig viele Transitionen nach in Tabelle 1 gezeigten Strukturen gebildet werden. Die Reihenfolge der Token auf der linken oder rechten Seite der Transitionen hat keine Bedeutung. Token mit gleichem Bezeichner können mehrfach vorkommen, sind dann aber nicht unterscheidbar. Für weitere Eigenschaften der Verteilten Transitionssysteme wird auf [Cap94, S 82ff.] verwiesen.

Bezeichnung	**Transition**	**Bezeichnung**	**Transition**
Transformation	$U \multimap V$	Transformation	$U * R \multimap V * S$
Konjunktion links	$U * W \multimap V$	Konjunktion rechts	$U \multimap V * W$
Produktion	$U \multimap V * U$	Absorption	$U * V \multimap U$
Disjunktion („nicht deterministischer split“)	$U \multimap V$ $U \multimap W$	Zusammenführung („join“)	$U \multimap V$ $W \multimap V$

Tabelle 1: Beispiele für einfache Transitionen

3.3.2 Verwendung zum Entwurf neuer Ausführungsanweisungen

Bei der Entwicklung von Ausführungsanweisungen fallen Entscheidungen, die die Funktionsweise festlegen und die resultierende Workflow-Ausführung bestimmen. Diese Entscheidungen müssen dokumentiert werden, um unterschiedliche Varianten vergleichen und bewerten zu können. Präzision und Eindeutigkeit im Ausdruck verlangen eine formale Notation, die Auswertungen und Rückschlüsse auf Eigenschaften der Ausführungsanweisung erlaubt.

Ein Verteiltes Transitionssystem deckt ab, was für die Beschreibung der Ausführungsanweisung relevant ist, z.B. die Workflows, deren Ausführung die Ausführungsanweisung kontrolliert. Für diese Verwendung repräsentieren einzelne Subsysteme (bestehend aus Transitionen und Token) die Workflow-Instanzen, und Interaktionen treten dort auf, wo die Ausführungsanweisung koordinierend auf die Workflow-Instanzen einwirkt. Zustände einer Workflow-

Instanz werden durch Token dargestellt, deren An- und Abwesenheit die Vorliegen des Zustands repräsentiert. Veränderungen von Zuständen der Workflow-Instanzen sind Transitionen, bei denen Token konsumiert und produziert werden, je nach darzustellendem Wechsel des Zustands. Durch gezielte Einführung von Token und Transitionen entsteht schrittweise die Definition eines Verteilten Transitionssystems. Dessen Verhalten gibt wieder, was die Ausführung der modellierten Ausführungsanweisung kennzeichnet. Ausgehend von einem initialen Zustand ergeben sich je nach Gestalt der Transitionen eine oder mehrere Abfolgen, in denen Token konsumiert und neue Token produziert werden. Dabei sind zwei Klassen von Transitionen zu unterscheiden: Die eine Gruppe von Transitionen beschreibt Erscheinungen, die im WFMS stattfinden. Dazu gehören Transitionen, die eine Zustandsänderung einer Workflow-Instanz beschreiben, aber mit der Ausführungsanweisung nichts zu tun haben. Die zweite Gruppe beschreibt die Ausführungsanweisung und damit Veränderungen der Zustände von Workflow-Instanzen, von denen nicht alle durch das WFMS kontrolliert werden können. Das Transitionssystem vollzieht alle Veränderungen nach, um Vorgaben für die Fortsetzung der Workflow-Ausführung zu machen.

Für die Entwicklung eines Verteilten Transitionssystems gelten anwendungsspezifische Regeln, die syntaktisch korrekte Transitionen verbieten können, weil sie eine aus Anwendungssicht unzulässige oder nicht mögliche Situation beschreiben würden (z.B. eine Workflow-Instanz ohne eindeutigen Zustand). Die Prüfung kann auf der Ebene einzelner Transitionen beginnen und sich bis zu Untersuchungen des dynamischen Verhaltens erstrecken (siehe dazu Abschnitt 3.4.4). Die Auswertung soll Fehler im Funktionsmodell finden und Eigenschaften der Ausführungsanweisung aufzeigen. Das Funktionsmodell ist fehlerhaft, wenn Transitionen unzulässige Vor- oder Nachbedingungen haben oder zwischen zulässigen Vor- und Nachbedingungen einen unzulässigen Zustandsübergang beschreiben. Abschnitt 3.4.4 nennt weitere Kriterien.

Ein erster Schritt zur Verhinderung von Fehlern ist die systematische Vergabe von Bezeichnern für die Token für Zustände der Workflow-Instanzen und Zustände der Ausführungsanweisung. Da alle Token den gleichen Typ haben und nur anhand des Bezeichners Fehler bei der Modellierung erkannt werden, sind sprechende Bezeichner notwendig, um bereits aus der Syntax Aussagen über die Eigenschaften der Ausführungsanweisung zu machen. Im folgenden werden Workflow-Instanzen einzeln benannt und ihr Zustand in Punkt-Notation angefügt (das Token `A.running` repräsentiert eine aktive Workflow-Instanz `A`, `B.done` symbolisiert eine beendete Workflow-Instanz `B`). Die Bezeichner für die Zustände leiten sich aus dem Ausführungsmodell des zugehörigen Workflow-Schemas ab. Zustände der Ausführungsanweisungen ergeben sich aus dem Zustand der überwachten Workflow-Instanzen. Der Entwickler von Ausführungsanweisungen kann allerdings entscheiden, welche Zustände er darüber hinaus einführt, um interne Zustände der Ausführungsanweisung zu repräsentieren. Transitionen beschreiben Veränderungen, also Zustandsänderungen der Workflow-Instanzen und der Ausführungsanweisung, die ebenfalls ein Verhalten hat. Die Transitionen setzen sich aus folgenden Gruppen zusammen: Erstens aus denen, die das Verhalten der Workflows beschreiben, deren Ausführung von der Ausführungsanweisung kontrolliert werden soll. Zweitens die Transitio-

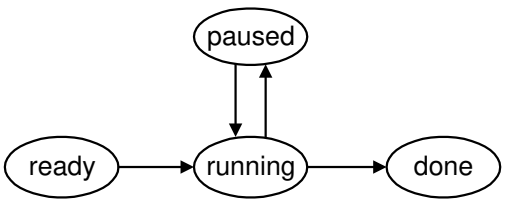
nen, die die Zusammenhänge dieser Workflow-Instanzen beschreiben. Der nächste Abschnitt zeigt, wie der Einsatz von Verteilten Transitionssystemen praktisch erfolgt.

3.4 Modellierung einer Ausführungsanweisung

Dieser Abschnitt beschreibt den Einsatz Verteilter Transitionssysteme zur Darstellung von Entwurfsentscheidungen. Abschnitt 3.4.1 beginnt mit der Repräsentation des Verhaltens von Workflows. Abschnitt 3.4.2 zeigt, mit welchen Transitionen und Transitionsgruppen Anforderungen an die Ausführung von Workflows präzisiert werden. Die auszugsweise dargestellten Transitionen sind immer im Kontext einer vollständigen Definition des Verteilten Transitionssystems zu sehen. Ausgehend von Basismechanismen werden typische Funktionen behandelt, die von einer Ausführungsanweisung übernommen werden.

3.4.1 Beschreibung des Verhaltens von Workflow-Instanzen

Aus den Zuständen im Ausführungsmodell leiten sich die Token ab, Zustandsübergänge werden zu Transitionen. Für eine Workflow-Instanz `A` mit einem Ausführungsmodell wie in [Jabl96a, S. 195] folgen die Transitionen [1] bis [6] in Abbildung 3.4. Die Token `init` und `exit` gehören nicht zur Beschreibung der Workflow-Instanz, sondern repräsentieren generische Zustände vor und nach der Lebensdauer der Workflow-Instanz.



[1]	init	—o	A.ready
[2]	A.ready	—o	A.running
[3]	A.running	—o	A.paused
[4]	A.paused	—o	A.running
[5]	A.running	—o	A.done
[6]	A.done	—o	exit

Abbildung 3.4: Transitionen zur Beschreibung des Verhaltens einer elementaren Workflow-Instanz

Für die Darstellung des Ausführungsmodells können auch weitere Transitionen und Token eingeführt werden; Abbildung 3.4 ist eher als einfachste Form der Beschreibung zu verstehen. Beispielsweise kann der Zustand unmittelbar nach der Instanziierung hinzugezogen und durch ein eigenes Token (z.B. `instanziiert`) dargestellt werden; gleiches gilt für Verfeinerungen der in Abbildung 3.4 gezeigten Zustände. Für die weitere Verwendung reichen Transitionen [1] bis [6] aus, daher wird zunächst keine Erweiterung der Beschreibung vorgenommen.

3.4.2 Formulierung von Anforderungen an die Workflow-Ausführung

Verhalten von Workflows wird auf koordinierte Zustandswechsel abgebildet. Damit ist festgelegt, wann welcher Zustand einer Workflow-Instanz eintreten kann und welche Nachfolgezustände aus Kontrollfluß-Sicht möglich sind. Als Notation dienen Transitionen, die Zustände bzw. Zustandswechsel einzelner Workflow-Instanzen darstellen. Workflows werden in Tran-

sitionen durch Token mit Großbuchstaben vom Beginn des Alphabets benannt, Großbuchstaben vom Ende des Alphabets sind Platzhalter für beliebige Token.

Die Beispiele der folgenden Abschnitte zeigen Anwendungen der Transitionsformen aus Tabelle 1 für die Darstellung verschiedener Details der Workflow-Ausführung. Es handelt sich noch nicht um Ausführungsanweisungen, sondern die Transitionen sind „Bausteine“, aus denen die Beschreibung der Ausführungsanweisungen erst entsteht. Abschnitt 3.4.2.1 stellt Transitionen vor, die Instanziierung und Deinstanziierung von Workflows repräsentieren. Durch die Erzeugung der Token werden Workflows „ins Leben gerufen“, deren Zustandswechsel anschließend beschrieben wird. Abschnitt 3.4.2.1 bietet „Basismechanismen“ an, um elementare Beziehungen zwischen Zuständen und Zustandsübergängen zu beschreiben, die im Funktionsmodell in Abschnitt 3.1 identifiziert wurden. Für alle Beispiele dieses Abschnitts gilt: Transitionen beschreiben *keine* Implementierung, sondern es werden Zustandswechsel festlegt, die das intendierte Verhalten der beteiligten Workflows garantieren. Es folgen daraus keine Vorgaben, wie das WFMS die aus Kontrollfluß-Sicht zulässigen Zustandswechsel herbeiführt.

3.4.2.1 Instanziierung und Deinstanziierung von Workflows

Per Vereinbarung wird festgelegt, die Instanziierung und Deinstanziierung von Workflows durch Erzeugen bzw. Konsumieren der Token <Workflow-Instanz-Name>.ready und <Workflow-Instanz-Name>.done zu beschreiben. Abbildung 3.5 zeigt, wie innerhalb einer Ausführungsanweisung ausgedrückt wird, eine neue Workflow-Instanz zu erzeugen oder eine beendete Workflow-Instanz zu entfernen. Selbstverständlich ist die Instanziierung und Deinstanziierung keine Aufgabe der Ausführungsanweisung, allerdings kann von ihr der Anstoß dazu kommen, weshalb eine Darstellung innerhalb der Transitionen notwendig ist.

[1]	$X_1 * X_2 * ... * X_n$	—o	A.ready
[2]	A.done	—o	$X'_1 * X'_2 * ... * X'_m$

Abbildung 3.5: Instanziierung und Deinstanziierung von Workflows

Die Benennung der Zustände ready und done hängt vom Ausführungsmodell der Workflow-Instanzen ab; zu verwenden ist der Zustand, den eine Workflow-Instanz unmittelbar nach ihrer Erzeugung durch den WFMS-Kernel einnimmt, und der Zustand, der dem WFMS-Kernel die Löschung der Instanz, Freigabe von Betriebsmitteln etc. erlaubt. Die Token X_i und X'_j repräsentieren Zustände, die vor der Instanziierung von Workflow A bzw. nach der Deinstanziierung vorliegen, sie haben keine weitere Bedeutung.

3.4.2.2 Basismechanismen zur Kontrolle von Zuständen und Zustandsänderungen

Für die Beschreibung des kontrollierten Wechsels von Zuständen der Workflow-Instanzen ist die Formulierung von Bedingungen und Alternativen notwendig. Dies geschieht unter Rückgriff auf die elementaren Ausdrucksmöglichkeiten in Verteilten Transitionssystemen (siehe Abschnitt 3.2.4, Tabelle 1). In den folgenden Beispiele fällt auf, daß neben den Token zur Re-

präsentation der Zustände von Workflow-Instanzen (z.B. A.ready) noch weitere Token vorkommen. Sie werden dort eingesetzt, wo das Konsumieren und Produzieren von Token für Workflow-Instanzen zu einer Verfälschung des darzustellenden Sachverhalts führen würde. Das Transitionssystem und seine aktuelle Token-Population beschreiben (unter Fokussierung auf ausgewählte Inhalte) den Zustand von Workflow-Instanzen während der kontrollierten Ausführung. So bedeutet die Anwesenheit des Token A.running daß die Workflow-Instanz A im Zustand running ist. Wird dieses Token nun für weitere „Aufgaben" verwendet, z.B. auf der linken Seite einer Transition, wird es beim Schalten dieser Transition konsumiert. Ein Blick auf Token-Population nach dem Schalten der Transition würde dann ein falsches Bild liefern, weil durch die Abwesenheit des Token für die Workflow-Instanz keine Aussage mehr zu deren Zustand möglich wäre. Aus diesem Grund werden eigene Token verwendet, deren Produzieren und Konsumieren die Sicht auf die Zustände der Workflow-Instanzen nicht verstellt, aber den gleichen Effekt (z.B. eine Synchronisation zwischen zwei Transitionen) bewirkt. Deutlich wird dies in den nächsten Beispielen, die zusätzliche Token bewußt einsetzten.

Entziehen einer Voraussetzung für Aktionen durch Verlassen eines Zustands

Zustandsübergänge einer Workflow-Instanz können Einfluß auf andere Workflow-Instanzen nehmen (siehe Abschnitt 3.2.2); Abbildung 3.6 zeigt ein Beispiel: Die Workflow-Instanzen A und B werden durch die Transitionen so in Beziehung gesetzt, daß der Beginn der Ausführung von Workflow A verhindert, daß Workflow B ausgeführt werden kann. Anders gesagt, durch Verlassen der Ausführungsbereitschaft von Workflow A wird die Voraussetzung für den Start von Workflow B entzogen. Die Umkehrung gilt nicht: Der Start von Workflow B beeinträchtigt die Ausführung von Workflow A nicht. Nach der Einteilung von Abschnitt 3.2.1.1 realisieren die Transitionen aus Abbildung 3.6 eine Beziehung des T/T-Typs mit Ursache-Wirkung-Interpretation und der Modalität „verhindern": Der Zustandsübergang ready / running von Workflow-Instanz A blockiert den Zustandsübergang ready / running von Workflow-Instanz B.

[1]	init	—o	A.ready ∗ B.ready ∗ **c**
[2]	A.ready ∗ **c**	—o	A.running
[3]	B.ready ∗ **c**	—o	B.running ∗ **c**

Abbildung 3.6: Entziehen einer Voraussetzung durch Verlassen eines Zustands

Das in Transition [1] erzeugte Token c stellt für die Workflow-Instanzen A und B eine potentielle Restriktion beim Übergang von ready nach running dar (nur mit c sind die notwendige Voraussetzungen für die Transition gegeben, wenn A.ready oder B.ready bereits vorliegen). Es wird zur tatsächlichen Restriktion, wenn einer von beiden Workflows ausgeführt wird, weil damit für den jeweils anderen Workflow der Start zunächst nicht möglich ist. Die Restriktion wird in Transition [3] wieder aufgehoben, da hier das Token c reproduziert wird.

Schaffen einer Voraussetzung für Aktionen durch Erreichen eines Zustands

So wie das Verlassen eines Zustands zum Anlaß genommen werden kann, eine Aktion auszulösen, so kann auch durch die Einnahme eines neuen Zustands einer Workflow-Instanz etwas ausgelöst werden. Abbildung 3.7 zeigt, wie nach dem Start des Workflows A durch Transition [2] die Voraussetzung entsteht, den Workflow B mit Transition [3] zu starten, in dem mittels zusätzlich eingeführtem Token c eine Beziehung hergestellt wird. Die Abhängigkeit ist einseitig, das heißt, die Ausführung der Workflow-Instanz A ist unabhängig von Workflow-Instanz B. Zu beachten ist, daß dieser Mechanismus nur eine Voraussetzung für eine Transition schafft, die unmittelbare Ausführung kann so nicht erreicht werden.

[1]	init	—o	A.ready ∗ B.ready
[2]	A.ready	—o	A.running ∗ **c**
[3]	B.ready ∗ **c**	—o	B.running

Abbildung 3.7: Schaffen einer Voraussetzung durch Erreichen eines Zustands

In der Typisierung von Abschnitt 3.2.1.1 realisieren die Transitionen aus Abbildung 3.7 eine Beziehung des T/T-Typs mit Ursache-Wirkung-Interpretation und der Modalität „erlauben“, denn der Zustandsübergang ready / running von Workflow-Instanz A erlaubt den Zustandsübergang ready / running von Workflow-Instanz B.

Doppelt restriktiver Zustandsübergang

Die Voraussetzungen für Zustandsübergänge von Workflow-Instanzen wurden bisher entweder einmalig geschaffen oder entzogen; mit den Transitionen in Abbildung 3.8 wird eine *vorübergehende* Voraussetzung für eine Zustandsänderung eingeführt. Transitionen [2] und [3] produzieren und konsumieren Token c dergestalt, daß Transition [4] „eingeschlossen“ wird.

[1]	init	—o	A.ready ∗ B.ready
[2]	A.ready	—o	A.running ∗ **c**
[3]	A.running ∗ **c**	—o	A.done
[4]	B.ready ∗ **c**	—o	B.running ∗ **c**

Abbildung 3.8: Doppelt restriktiver Zustandsübergang

Eine Anwendung besteht in der vorübergehenden Möglichkeit, die Workflow-Instanz B zu starten. Das ist nur zulässig, solange die Workflow-Instanz A ausgeführt ist, weder davor noch danach. Kann in dieser Phase die Workflow-Instanz gestartet werden, ist die anschließende Ausführung und das Ende von Workflow B unabhängig von Workflow A und auch umgekehrt.

Synchronisierte Zustandseinnahme

Da in Verteilten Transitionssystemen der verwendeten Form Transitionen „zeitlose“ Veränderungen der Tokenmenge bewirken, werden alle Token auf der rechten Seite der Transition und alle Token auf der linken Seite der Transition zeitgleich konsumiert und produziert.

Abbildung 3.9 zeigt, wie die Forderung zur gemeinsamen Instanziierung von Workflow A und Workflow B als Transition ausgedrückt wird.

[1]	X	—o	A.ready ∗ B.ready
[2]	A.done ∗ B.done	—o	Y

Abbildung 3.9: Synchronisierte Zustandseinnahme

Transition [1] und Transition [2] spielen für die Beschreibung von Ausführungsanweisungen eine wichtige Rolle, wenn die Ausführung auf die als Argumente in der Ausführungsanweisung genannten Workflows übergeht (näheres dazu in Abschnitt 3.4.3).

Synchronisierte Zustandsübergänge

Die Zustandsübergänge mehrerer Workflow-Instanzen können unmittelbar in Verbindung stehen (T/T-Typ, vgl. Abschnitt 3.2.1.1); Abbildung 3.10 zeigt zwei unabhängige Beispiele, in denen Zustandsübergänge von jeweils zwei Workflow-Instanzen in Beziehung stehen.

[1]	A.ready ∗ B.running	—o	A.running ∗ B.paused
[2]	C.paused ∗ D.running	—o	C.running ∗ D.done

Abbildung 3.10: Synchronisierte Zustandsübergänge

Transition [1] ermöglicht den Start der Workflow-Instanz A nur, wenn gleichzeitig sichergestellt ist, daß Workflow-Instanz B unterbrochen wird (Zustand paused). Ein anderes Beispiel gibt Transition [2], wo die Fortsetzung des zwischenzeitlich unterbrochenen Workflows C davon abhängig (noch im Zustand paused) ist, ob Workflow D beendet wird.

Die Beispiele zeigen einen Nachteil der Notation: Aus der Notation geht nicht hervor, ob z.B. in Transition [2] die Fortsetzung von Workflow C (von paused in running) die Ursache für das Ende von Workflow D ist (von running in done) oder ob die Beendigung von Workflow C die Unterbrechung von Workflow D aufhebt. Der Nachteil kommt allerdings nur zum Tragen, wenn man von der abstrakten Sicht der Workflow-Ausführung zur Implementierung übergeht.

3.4.2.3 Alternativen

Abbildung 3.11 zeigt, wie aus zwei Transitionen mit gleicher Vorbedingung X zwei unterschiedliche Zustände folgen können. Das erlaubt die Darstellung von Alternativen, hier am Beispiel der Instanziierung von Workflow A oder Workflow B gezeigt.

[1]	X	—o	A.ready
[2]	X	—o	B.ready

Abbildung 3.11: Alternative

3.4.3 Beschreibung einer Ausführungsanweisung als Transitionssystem

Komplizierte Folgen von Workflows mit gegenseitiger Einflußnahme lassen sich zwar auf der Ebene koordinierter Zustandswechsel formulieren, es gelten jedoch alle Nachteile wie beim Einsatz von Kontrollflußprimitiven (siehe Abschnitt 2.3.4). Ausführungsanweisungen entlasten den Workflow-Entwickler von der Wiederholung dieser Aufgabe, sofern die Abbildung der gewünschten Workflow-Ausführung auf koordinierte Zustandswechsel einmal vorgenommen wurde. Die notwendigen „Bausteine“ hat der letzte Abschnitt eingeführt, die nächsten Schritte zur Erstellung eines vollständigen Transitionssystems sind:

- Für die Ausführungsanweisung wird ein Token mit sprechender Bezeichnung eingeführt, das den ebenfalls einzuführenden internen Zuständen der Ausführungsanweisung voransteht und damit eine präfix-basierte Eindeutigkeit der Token garantiert. Aus der Betrachtung von Ausführungsanweisungen als zustandstragende Objekte folgen mindestens die Zustände ready, running und done respektive anderer Bezeichner für Zustände, in denen die Ausführung der kontrollierten Workflows beginnen kann (Zustand ready), gerade erfolgt (Zustand running) oder abgeschlossen ist (Zustand done). Alternative Zustandsräume werden in Abschnitt 4.2.1 vorgestellt, hier reichen zunächst die Zustände ready, running und done für die exemplarische Ausführungsanweisung SEQUENCE.

- Workflows, die von SEQUENCE strikt hintereinander ausgeführt werden sollen, sind der Einfachheit halber elementar und haben ein Ausführungsmodell wie in Abbildung 3.2. In den Transitionen werden sie durch A und B repräsentiert. Die angestrebte Sequentialisierung führt mittels der Basismechanismen aus Abschnitt 3.4.2.2 zur Transition A.done —o B.ready und reicht für die Kontrollflußdefinition zwischen den beiden Argument-Workflows aus.

- Die intendierte Ausführungsfolge von SEQUENCE(A,B) verlangt, zunächst den ersten Workflow zu starten, was die Transition SEQUENCE.ready —o A.ready ∗ SEQUENCE.running beschreibt. Die strikte Sequenz ist erst nach dem Ende des zweiten Workflows beendet, was durch SEQUENCE.running ∗ B.done —o SEQUENCE.done zum Ausdruck kommt.

Weitere Transitionen gehören nicht zur Beschreibung der Ausführungsanweisung SEQUENCE. Insbesondere keine Transitionen der Art A.ready —o A.running, weil dieser Zustandsübergang vom Workflow selbst oder durch Aktionen der Anwenders o.ä. vollzogen wird und nicht dem Einflußbereich der Ausführungsanweisung zuzurechnen ist; für A.running —o A.done gilt dies verstärkt, da die darstellte Terminierung eines Workflows im Normalfall nicht erzwungen werden kann, sondern vom Ausführungszustand eventueller Subworkflows, Aktivitäten des Anwenders etc. abhängt. Abbildung 3.12 zeigt die vollständige Beschreibung der Ausführungsanweisung SEQUENCE.

[1]	**SEQUENCE**.ready	—o	**SEQUENEC**.running ∗ A.ready
[2]	**SEQUENCE**.running ∗ A.done	—o	**SEQUENCE**.running ∗ B.ready
[3]	**SEQUENCE**.running ∗ B.done	—o	**SEQUENCE**.done

Abbildung 3.12: Ausführungsanweisung SEQUENCE

Da Ausführungsanweisungen der vorgestellten Art geschachtelt werden können, ist auf einheitliche Bezeichnung der Zustände zu achten. Sind die verwendeten Workflows komposit, enthalten sie weitere Ausführungsanweisungen, die dann in der oben gezeigten Weise den Übergang zu den Subworkflows herstellen (siehe dazu auch Kapitel 4.3).

3.4.4 Aussagen über die Ausführungsanweisung

Zu den syntaktischen Eigenschaften eines Verteilten Transitionssystems (vgl. [Cap94]) erfordert die spezielle Verwendung die Einhaltung weiterer Vorgaben: Abschnitt 3.4.4.1 formuliert Anforderungen an die Gestaltung von Verteilten Transitionssystemen, die Workflows und Ausführungsanweisungen beschreiben. Für den Gebrauch der Ausführungsanweisung relevante Eigenschaften nennen die Abschnitte 3.4.4.2 und 3.4.4.3. Abschnitt 3.4.4.4 geht auf die Möglichkeiten ein, sowohl die Einhaltung der Gestaltungshinweise und auch weitere Eigenschaften unter Ausnutzung der formalen Darstellung zu überprüfen. Die Einhaltung der Kriterien wird anhand des Modells überprüft, das in Form eines Verteilten Transitionssystems von der Ausführungsanweisung vorliegt. Demnach beziehen sich die Ergebnisse des Tests nur auf das Modell, für eine Implementierung der Ausführungsanweisung sind die Ergebnisse daher notwendig, aber natürlich nicht hinreichend; sie garantieren also nur eine fehlerfreie Ausgangsbasis für die Implementierung.

3.4.4.1 Gestaltungshinweise zu einzelnen Ausführungsanweisungen

Die Gestaltungshinweise gliedern sich in anwendungsunabhängige Empfehlungen, die ein syntaktisch korrektes und funktionsfähiges Transitionssystem garantieren. Anwendungsabhängige Empfehlungen berücksichtigen Eigenschaften der Anwendungsdomäne, also Workflows und Ausführungsanweisungen. Die Empfehlungen werden aus Gründen der leichteren Lesbarkeit nur umgangssprachlich formuliert; eine formale Darstellung mit Prädikaten-Logik erster Ordnung oder Ausdrücken in Temporaler Logik ist möglich, würde aber einen zusätzlichen Formalismus einführen und erfolgt daher in diesem Bericht nicht. Es sei ausdrücklich darauf hingewiesen, daß die genannten Empfehlungen Beispiele sind und daher keine Vollständigkeit in der Aufzählung existiert. Sie sind dennoch notwendig, um einen universell einsetzbaren Formalismus wie es Verteilte Transitionssysteme sind, für ein ganz konkretes Anwendungsgebiet nutzen zu können, ohne den Formalismus selbst anwendungsspezifisch zu verändern. Daher ist es notwendig, quasi von „außen“ sicherzustellen, daß im Sinne der Anwendung nur sinnvolle Formulierungen entstehen. Es ist prinzipiell nicht möglich, ein vollständiges Regelwerk anzugeben, zumal vieles auch intuitiv klar ist. Dennoch läßt sich leicht zeigen, daß die Mißachtung selbst der wenigen Regeln bereits zu Transitionen führt, die syntaktisch korrekt, aber unsinnig sind.

Anwendungsunabhängige Gestaltungshinweise

- **Transitions-Vorbedingung**. Transitionen sind unzulässig, wenn ein Token (ungleich dem Starttoken init) der linken Seite von keiner Transition erzeugt wird.

- **Wirkungslose Transitionen**. Transitionen, die bis auf Permutation links und rechts die gleichen Token aufweisen, sind wirkungslos und daher zu entfernen.

- **Rekursive Transitionen**. Transitionen, die auf der linken Seite Token enthalten, die eine echte Teilmenge der Token auf der rechten Seite bilden, könnten unendlich oft ausgeführt werden und sind daher zu entfernen.

Anwendungsspezifische Gestaltungshinweise

- **Konformität zum Ausführungsmodell**: Ausführungsanweisungen dürfen keine Zustandsübergänge von Workflow-Instanzen verlangen, die deren Ausführungsmodell widersprechen oder auf Zustände Bezug nehmen, die das Ausführungsmodell nicht vorsieht. Dazu gehört auch, keine Zwischenzustände des Ausführungsmodells unter Auslassung des Startzustands zu adressieren.

  Beispiel: Für das WFMS MOBILE wäre eine Ausführungsanweisung mit einer Transition unzulässig, die links das Token A.running und rechts das Token A.ready enthält, da Workflow-Instanzen diesen Zustandsübergang nur in der anderen Richtung vollziehen können (unter Annahme eines Ausführungsmodells wie in [Jabl96a, S. 249]).

  Prüfung: Test auf Vorkommen unzulässiger Zustandsübergänge (Suche nach Pfaden im Erreichbarkeitsgraphen, die unzulässigen Zustandsübergänge als Teilpfade haben).

- **Ausführung von Workflows**. Für jeden in der Ausführungsanweisung vorkommenden Workflow muß die *Möglichkeit* zum Start bestehen. Das Ende eines Workflows muß nicht berücksichtigt werden, um die Situation einer asynchronen Workflow-Ausführung beschreiben zu können.

  Prüfung: Erreichbarkeit des Startzustands jeder Workflow-Instanz durch eine Zustandsfolge vom Startzustand der Ausführungsanweisung ausgehend.

- **Zustände** der Ausführungsanweisung: Jede Ausführungsanweisung muß eindeutig bestimmte Zustände haben, die den Beginn und das Ende der von ihr ausgeübten Kontrolle über die Workflows kennzeichnen. Mehrere Zustände jeweils für Beginn und Ende sind zulässig, beispielsweise um die Funktion der Ausführungsanweisung zu parametrisieren oder um durch unterschiedliche Endzustände Informationen über den Verlauf der Ausführung zugänglich zu machen.

- Für die **Terminierung** gilt**:**

  (a) Einer der Endzustände der Ausführungsanweisung muß durch eine Folge der Transitionen erreichbar sein.

(b) Ist der Endzustand erreicht, dürfen keine weiteren Transitionen mehr möglich sein, die die Ausführungsanweisung betreffen.

Prüfung: Suche nach Zustandsfolgen vom Startzustand der Ausführungsanweisung zum Endzustand der Workflow-Instanzen im Erreichbarkeitsgraphen.

Weitere Gestaltungshinweise

- **Eindeutige Zustände**. Zu jedem Zeitpunkt muß die Ausführungsanweisung und jede Workflow-Instanz einen eindeutig bestimmten Zustand haben.

  Prüfung: Test auf gleichzeitige Anwesenheit von mehr als zwei Token für Zustände der gleichen Workflow-Instanz.

- Für den Start von Workflows gilt:

  (a) Erst nachdem die Ausführungsanweisung aktiv ist, darf ein von ihr kontrollierter Workflow gestartet werden.

  (b) Jeder Workflow darf höchstens einmal gestartet werden.

- Für das Ende von Workflows gilt:

  (c) Vor Ende der Ausführungsanweisung müssen alle von ihr synchron gestarteten Workflows beendet oder abgebrochen sein oder sicher nicht mehr ausgeführt werden.

  (d) Ein Workflow kann nicht mehrfach beendet werden.

  (e) Es können nur Workflows beendet werden, die zuvor gestartet wurden.

Die Einhaltung dieser Gestaltungshinweise während der Simulation des Verhaltens garantiert, daß keine unzulässigen Zustände in den Workflows oder der Ausführungsanweisungen nachgebildet werden. Bisher wurde nur der Fall betrachtet, daß eine Ausführungsanweisung als Argumente Workflows aufweist, deren Ausführung kontrolliert wird, z.B. SEQUENCE(A, B). Da statt der Workflows wieder Ausführungsanweisungen auftreten können (Abschnitt 4.3.1 beschreibt diese Situation genauer), gelten die Gestaltungshinweise genauso für Ausführungsanweisungen als Argumente, z.B. bei SEQUENCE(SEQUENCE(A, B), C).

3.4.4.2 Eigenschaften einzelner Ausführungsanweisungen

Den Entwickler eines Workflow-Schema interessieren folgende Eigenschaften einer Ausführungsanweisung:

E1: Alle möglichen Formen der Workflow-Ausführung unter einer Ausführungsanweisung.

Bedeutung: Bei der Definition komplizierter Ausführungsanweisungen können unbeabsichtigte Abfolgen der kontrollierten Workflows oder unzulässige Zustandskonstellationen der Workflow-Instanzen übersehen werden.

Beispiel: Alle Abfolgen der Workflows A und B, die DELAY (A, B) erzeugt.

Prüfung: Auswertung aller Zustandsfolgen zwischen Start und Ende der Ausführungsanweisung; Ableitung von Ausführungsintervallen aus dem Auftreten von Start- und Terminierungszustände der kontrollierten Workflows; Bestimmung und Test der Lagebeziehungen zwischen den Ausführungsintervallen.

E2: Einzelne Eigenschaften der Ausführungsanweisung; insbesondere die Sicherung oder Verhinderung bestimmter Zustandskonstellation der kontrollierten Workflows.

Bedeutung: Validierung der Eigenschaften als Nachweis zur Erfüllung der Entwurfsziele.

Beispiel: Kann in ENFORCE_OVERLAPS(A, B) die Situation auftreten, daß Workflow A erst nach Workflow B beendet wird? Gibt es eine solche Situation, liegt in der Spezifikation der Ausführungsanweisung ein Fehler vor.

Prüfung: Formulierung der zu prüfenden Eigenschaften als Ausdruck in temporaler Logik und Einsatz eines Model Checkers zur Validierung der Eigenschaft.

Neben Erfüllung der Kriterien und der Einhaltung dieser Hinweise steht der Nachweis aus, daß die Ausführungsanweisung die Eigenschaften aufweist, die als Ziel der Entwicklung definiert waren, etwa eine bestimmte Form der Ausführung mehrerer Workflows. Welche Eigenschaften im einzelnen zu validieren sind, ist bei jeder Ausführungsanweisung anders und ergibt sich aus den Anforderungen, die als Entwicklungsziel definiert waren.

3.4.4.3 Aussagen über mehrere Ausführungsanweisungen

Kriterien und Eigenschaften wurden bisher nur für eine einzige Ausführungsanweisung geprüft. In der Entwicklung von Ausführungsanweisungen ist jedoch genauso von Bedeutung, ob zwei Ausführungsanweisungen die gleiche Workflow-Ausführung erzeugen oder ob ein zusammengesetzter Ausdruck durch eine eigens dafür entwickelte Ausführungsanweisung ersetzt werden kann.

- **Äquivalenz**. Die Abbildung einer Ausführungsfolge der Workflows auf den koordinierten Zustandswechsel der Workflow-Instanzen ist nicht eindeutig. Demzufolge kann die gleiche Workflow-Ausführung durch unterschiedlich konzipierte Ausführungsanweisungen erreicht werden. Beide Ausführungsanweisungen sind dann hinsichtlich der von ihnen erzeugten Workflow-Ausführung äquivalent. Eine nachgewiesene Äquivalenz dient beispielsweise als Kriterium, eine Ausführungsanweisung gegen eine mit abweichender Funktionsweise austauschen zu dürfen. Das kann notwendig werden, um in zwei unterschiedlichen Ausführungsumgebungen Ausführungsanweisungen anzubieten, die dem Anwendungsentwickler die gleiche Workflow-Ausführung garantieren.

- **Ersetzbarkeit**. Ein Grund für die Entwicklung neuer Ausführungsanweisungen ist die Absicht, eine häufig benötigte Form der Workflow-Ausführung statt durch Kombination mehrerer Ausführungsanweisungen zu einem u.U. komplizierten Ausdruck in einer eigenen

Anweisung zu kapseln. Bevor der bisher verwendete Ausdruck in einem Workflow-Schema durch eine neue Ausführungsanweisung ersetzt werden kann, ist die dadurch unveränderte Ausführung der Workflows nachzuweisen.

Der nächste Abschnitt beschreibt, wie die Eigenschaften überprüft werden können.

3.4.4.4 Methoden zur Untersuchung und Auswertung Verteilter Transitionssysteme

Die Überprüfung der Kriterien und die Validierung von Eigenschaften erfordert Hilfsmittel. Bisher kam eine Simulationsumgebung für Verteilte Transitionssysteme zum Einsatz [Künz94]; sie erlaubt Rückschlüsse auf die Eigenschaften der Ausführungsanweisung, z.B.:

- Die Tokenmenge zeigt ausführbereite, aktive und beendete Workflows an. Die Ausführungsanweisung ist fehlerhaft spezifiziert, wenn Token Zustandskonstellationen anzeigen, die die geforderte Workflow-Ausführung ausschließt (z.B. A.running und B.running gleichzeitig bei SEQUENCE(A, B) aus Abschnitt 3.4.3).

- Ausführbereite Transitionen geben Veränderungen der aktuellen Tokenmenge an. Transitionen beschreiben entweder Aspekte der Workflow-Ausführung, die außerhalb der WFMS-Kontrolle stattfinden (z.B. das Beenden eines Workflows A durch A.running —o A.done), oder die Transition beschreibt eine Reaktion der Ausführungsanweisung in einer bestimmten Situation bei der Ausführung der kontrollierten Workflows (z.B. A.done —o B.ready).

- Durch die Auswahl einer von mehreren ausführbaren Transitionen simuliert der Entwickler Entscheidungen, auf die das WFMS keinen Einfluß hat oder die aus laufzeitabhängigen Werten folgen. Kein Einfluß besteht auf Handlungen des Anwenders und das Verhalten externer Applikationen, die mit simuliert werden müssen, wenn sie die Workflow-Ausführung betreffen.

Die Interaktion mit dem Simulator gibt einen ersten Eindruck der Funktionsweise einer Ausführungsanweisung und ihrer Auswirkung auf die Workflow-Ausführung, erlaubt aber keine systematische Auswertung. Diese ist allerdings erforderlich, um Eigenschaften sicher ausschließen oder bestätigen können. Die Abbildung Verteilter Transitionssysteme auf Stellen/Transitions-Netze ermöglicht z.B. Erreichbarkeitsanalysen und die Validierung von Eigenschaften mittels Model Checking; zur Formulierung von Anfragen über das Auftreten bestimmten Zustandskonstellationen dienen Ausdrücke in temporaler Logik. Dieser Punkt wird aus Platzgründen hier nicht weiter ausgeführt, ist aber in Bearbeitung.

4 Konzeption von Ausführungsanweisungen für eine Workflow-Sprache

In einem WFMS mit erweiterbarer Workflow-Sprache gibt es theoretisch einen initialen Zustand, an dem noch keine Sprachmittel definiert sind. Die Entwicklung einer WFMA beginnt dann mit der Planung, Spezifikation und Implementierung der benötigten Sprachmittel. Am Beispiel von Ausführungsanweisungen beschreibt dieses Kapitel die Vorgehensweise und diskutiert Möglichkeiten, Sprachmittel zur Beschreibung der Workflow-Ausführung bereitzustellen. Besonderer Wert wird auf die Erläuterung von Entscheidungen gelegt, die für diesen Aspekt der Workflow-Sprache erforderlich sind und Ausdrucksfähigkeit, Funktionsweise und Einsatzbreite der Ausführungsanweisungen bestimmen.

Abschnitt 4.1 erörtert Designfragen bei der Entwicklung der Ausführungsanweisungen. Diese Fragen betreffen sowohl die Gesamtheit der Ausführungsanweisungen als auch die Gestaltung der einzelnen Ausführungsanweisungen und ihres „Innenlebens“. Für die weitere Beschreibung in diesem Bericht ist es notwendig, die Fragen in einer bestimmten Art zu entscheiden und in den Ausführungsanweisungen umzusetzen. Unter anderen Bedingungen, beispielsweise bei anderen Einsatzgebieten, sind abweichende Entscheidungen möglich, die dann zu Ausführungsanweisungen mit anderen Merkmalen führen.

Abschnitt 4.2 stellt eine Auswahl von Ausführungsanweisungen vor. Verteilte Transitionssysteme dienen dazu, ihre Realisierung zu verdeutlichen. Die Implementierung im Sinne einer vollständig ausgeführten Integration in die Ausführungsumgebung kann sich aus der Realisierung ableiten, ist aber nicht mit ihr gleichzusetzen.

Abschnitt 4.3 rechtfertigt den erhöhten Aufwand der zweistufigen Vorgehensweise (Realisierung und Implementierung) durch zwei neuartige Möglichkeiten: erstens die Simulation der Funktionsweise von Ausführungsanweisungen im Kontext der Workflow-Ausführung vor seiner Implementierung und zweitens den Vergleich von unterschiedlichen Realisierungsvarianten für eine Ausführungsanweisung. In Anbetracht des hohen Aufwands einer vollständigen Implementierung einer Ausführungsanweisung im WFMS ist die Simulation im Vorfeld sinnvoll, weil sie Hinweise auf Fehler oder Unzulänglichkeiten im Design gibt, deren Entdekkung durch die Implementierung nur verzögert, aber nicht behoben wird.

4.1 Entscheidungen zum Design der Ausführungsanweisungen

Eine Reihe von Entscheidungen bestimmen die nichtfunktionalen Eigenschaften wie Handhabbarkeit, Wiederverwendung und Lesbarkeit der Ausführungsanweisungen. Dazu kommen Freiheitsgrade bei der Realisierung einer einzelnen Ausführungsanweisung, denn die beabsichtigte Ausführungsfolge der Workflows kann auf unterschiedliche Weise erreicht werden. Abschnitt 4.1.1 geht auf Entscheidungen ein, die die Gestaltung und das Erscheinungsbild insgesamt leiten. Abschnitt 4.1.2 behandelt Punkte auf der Ebene einer einzelnen Ausführungsanweisung und ihrer Funktionsweise.

4.1.1 Kontrollflußdefinition im Workflow-Schema

Die Workflow-Sprache soll eine ausgezeichnete Menge von Sprachelementen zur Festlegung der Workflow-Ausführung bekommen. Ein geschlossener Ausdruck im Workflow-Schema legt die Ausführung der Subworkflows fest. Dieser Ausdruck besteht aus Ausführungsanweisungen und Workflow-Variablen. Ausführungsanweisungen werden in Präfix-Schreibweise notiert und können als Argumente wieder Ausführungsanweisungen oder Workflow-Variablen aufweisen. Den wohlgeformten Klammerausdruck arbeitet das WFMS von links nach rechts ab. Jede Ausführungsanweisung gibt an, wie die Workflows bzw. Teilausdrücke zu behandeln sind. Sprunganweisungen kommen nicht vor, und die Abarbeitungsreihenfolge ist festgelegt. Sollen Workflows oder Teile der Kontrollflußdefinition mehrfach durchlaufen werden, müssen sie in eine Ausführungsanweisung eingekleidet werden, die die mehrfache Ausführung vorsieht. Da nicht wie beim Einsatz von Kontrollflußprimitiven die Möglichkeit besteht, Zyklen zu definieren, ist für die Wiederholung eine eigene Ausführungsanweisung notwendig. In Anlehnung an die Syntax der Workflow-Sprache MSL (Mobile Script Language) [Jabl96, S. 323ff.], [Bußl97, S. 303ff.] sieht die Verwendung der Ausführungsanweisungen wie folgt aus:

```
WORKLOW_TYPE SimpleWorkflow
        SUBWORKFLOWS
                A: Workflow_A;
                B: Workflow_B;
                C: Workflow_C;
                D: Workflow_D;
                E: Workflow_E;
        END_SUBWORKFLOW
        CONTROL_FLOW
        < Ausführungsanweisungen mit den Variablen für Subworkflow A bis E>
        END_CONTROL_FLOW
        ...
END_WORKFLOW_TYPE
```

Abbildung 4.1: Ausführungsanweisungen im Kontext eines Workflow-Schemas

In Abschnitt 4.2 werden dann Ausführungsanweisungen vorgestellt, die an der Stelle zwischen CONTROL_FLOW und CONTROL_FLOW_END in Abbildung 4.1 stehen können.

4.1.2 Entwurfsentscheidungen für Ausführungsanweisungen

Der Entwickler einer Ausführungsanweisung hat Freiheitsgrade beim Entwurf, die sich auf die nichtfunktionalen Eigenschaften auswirken. Die Entscheidungen betreffen

- Funktionsumfang („Welche Aufgaben übernimmt die Ausführungsanweisung?“),
- Spezialisierung („Wie universell soll die Ausführungsanweisung einsetzbar sein?“) und
- Argumenttypen („Workflow-Instanzen oder Workflow-Schemata als Argumente?“).

Die nächsten Abschnitte gehen auf diese Punkte ein.

Funktionsumfang

Das Funktionsmodell führt die Aufgabe einer Ausführungsanweisung auf die Kontrolle von Beziehungen zwischen den Zuständen und Zustandsübergängen der Workflow-Instanzen zurück. Diese Beziehungen verwendet das WFMS zur Laufzeit, um aus einer gegebenen Konstellation von Zuständen eine Nachfolgekonstellation zu bestimmen, und daran anschließend die Workflow-Instanzen auszuführen, die in der Nachfolgekonstellation den Zustand startable haben. Die Ausführungsanweisung transformiert demnach Zustandskonstellationen nach definierten Regeln, und indem der Entwickler der Ausführungsanweisung diese Regeln festlegt, bestimmt er die Funktionsweise der Ausführungsanweisung. Da jede Regel einen Teil der gewünschten Workflow-Ausführung beschreibt (siehe Abschnitt 3.4.2), ändert sich die Funktionsweise der Ausführungsanweisung mit der Anzahl der Regeln. Je mehr Zustandskonstellationen durch das Regelwerk abgedeckt werden, desto mehr anwendungsspezifische Logik ist in der Ausführungsanweisung enthalten und wird nicht im Workflow-Schema beschrieben.

> Beispiel: Im Regelwerk der Ausführungsanweisung kann festgelegt sein, den Zustand failed im Ausführungsmodell einer Workflow-Instanz in einen Zustand startable zu transformieren. Bei Anwendung dieser Regel erhält das WFMS die Anweisung, nach einem Scheitern der Workflow-Ausführung (erkennbar am Zustand failed) für die Workflow-Instanz den Zustand startable herzustellen, was gleichbedeutend mit einer erneuten Instanziierung des Workflows ist. In dem Regelwerk ist eine „Retry“-Logik untergebracht, im Workflow-Schema nicht mehr zu erkennen ist.

Andere Maßnahmen als die Wiederholung der Ausführung sind denkbar; sie bestimmen den Umfang der von der Ausführungsanweisung erbrachten Leistung, die von der erwähnten Wiederholung bis zur Behandlung von Ausnahmen allein durch die Ausführungsanweisung reichen kann.

Spezialisierung

Je mehr anwendungsspezifische Logik eine Ausführungsanweisung bereitstellt, desto weniger kann sie außerhalb dieser Anwendung eingesetzt werden:

> Beispiel: Eine Ausführungsanweisung VOTUM(wf_1, wf_2, wf_3, pro, contra) startet die Workflows wf_1 bis wf_3 gleichzeitig. Die Workflows können in zwei unterschiedlichen Zuständen terminieren. Wird nach Ende aller Workflows innerhalb der Ausführungsanweisung die Anzahl der aufgetretenen Zustände gezählt und von dieser Anzahl die Fortsetzung mit Workflow pro oder contra abhängig gemacht, realisiert diese Ausführungsanweisung ein Abstimmungsverfahren zwischen drei Beteiligten.

Der Entwickler von Ausführungsanweisungen muß entscheiden, in welchem Umfang er Ausführungsanweisungen für elementare Formen der Workflow-Ausführung anbietet und wo er diese Ausführungsformen zusammenfaßt und in einer einzigen Ausführungsanweisung kapselt. Ein Satz minimaler Ausführungsanweisungen begünstigt die Einsatzbreite der Workflow-Sprache, führt aber zu komplexen Ausdrücken, die zudem aufwendig in der Erstellung und

Pflege sind. Zur Frage der Spezialisierung gehört die Entscheidung, wieviel Argumente eine Ausführungsanweisung hat. Bei Ausführungsanweisungen für sequentielle, alternative oder parallele Ausführung von Workflows bleibt die Funktion erhalten, wenn die Ausführungsfolge zwischen mehr als zwei Workflows festgelegt wird.

> Beispiel: Für sequentielle Ausführung der Workflows a, b, c, d kann entweder eine spezielle Ausführungsanweisung SPECIAL_SEQ(a, b, c, d) eingeführt werden oder aus einer zweistelligen Ausführungsanweisungen SEQ wird ein Ausdruck der Form SEQ(a, SEQ (b, SEQ(c, d))) zusammengesetzt.

Hier gilt sinngemäß die Argumentation wie bei der Spezialisierung: Durch die Möglichkeit zur Schachtelung von Ausführungsanweisungen mit zwei Argumenten entstehen unübersichtliche Ausdrücke, auch wenn die Ausdrucksmöglichkeit zur Beschreibung der Workflow-Ausführung gleich bleibt.

Argumenttypen

Jeder Ausführungsanweisung sind eine bestimmte Anzahl von Argumenten zugeordnet, über deren Typ erst entschieden werden muß. Werden beispielsweise Workflow-Schemata als Argumente verwendet, bezieht sich die von der Ausführungsanweisung realisierte Funktion auf eine erst zur Laufzeit entstandene Workflow-Instanz dieses Workflow-Schemas (vgl. dazu auch [Jabl94, S. 3 ff.]). Werden statt der Workflow-Schemata zuvor deklarierte Variablen verwendet, die Workflow-Instanzen eines Workflow-Schemas bezeichnen, gelten zusätzliche Regeln.

> Beispiel: Der Ausdruck SEQUENCE(A, A) ist unzulässig, wenn A eine Variable ist, die eine Workflow-Instanz eines Workflow-Schemas bezeichnet. Um auszudrücken, daß zwei Instanzen des gleichen Workflow-Schemas hintereinander ausgeführt werden sollen, sind Variablen A und B zu deklarieren und als SEQUENCE(A, B) zu verwenden.

Im nächsten Abschnitt werden Ausführungsanweisungen vorgestellt, bei deren Entwurf sich unter anderen die eben genannten Fragen gestellt haben. Im Rahmen dieses Berichts werden Ausführungsanweisungen besprochen, die keine Fehlerbehandlung durchführen, sowohl elementare Workflow-Ausführungen als auch ausgewählte Spezialfunktionen realisieren und als Argumente Workflow-Variablen haben.

4.2 Ausführungsanweisungen für ausgewählte Formen der Workflow-Ausführung

Durch das breite Einsatzgebiet von WFMS ist es schwierig, einen universell einsetzbaren Satz von Ausführungsanweisungen anzugeben. Daher sind die Ausführungsanweisungen in diesem Abschnitt Beispiele, ohne Anspruch auf Vollständigkeit oder Ausschließlichkeit, in erster Linie dazu gedacht, den Einsatz von Verteilten Transitionssystemen zu demonstrieren.

Das angenommene Ausführungsmodell der Workflow-Instanzen beschreibt Abschnitt 4.2.1. Die anschließend vorgestellten Ausführungsanweisungen sind wie folgt gegliedert: Abschnitt 4.2.2 führt Ausführungsanweisungen ein, um die Auslassung oder Wiederholung der Workflow-Ausführung festzulegen. Die Ausführungsanweisungen in Abschnitt 4.2.3 stellen zwischen mehreren Workflows elementare Ausführungsbeziehungen (Sequenz, Alternative etc.) her. Abschnitt 4.2.4 behandelt Ausführungsanweisungen für verschiedene Formen der seriellen Ausführung; Abschnitt 4.2.5 geht auf unterschiedlichen Formen paralleler Ausführung mehrerer Workflows ein. Abschließend stellt Abschnitt 4.2.6 ein Klassifikationsschema vor und zeigt seine Verwendung am Beispiel der zuvor entwickelten Ausführungsanweisungen.

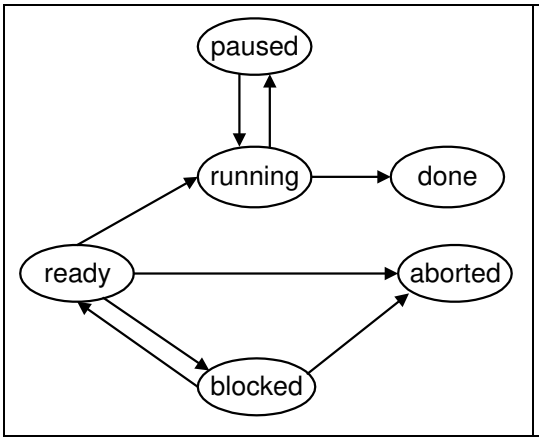
Um die Ausführungsanweisungen zu unterscheiden, erhalten sie einen sprechenden Bezeichner. SEQ bezeichnet die sequentielle Ausführung, OPT drückt eine optionale Ausführung aus usw.. Die Bezeichner dienen dazu, um im Transitionssystems Token zu benennen, die den Zustand der Ausführungsanweisung angegeben.

Weiterhin ist zu beachten, daß im Gegensatz zu den Beispielen in Abschnitt 3.4.2 die Zustände der Workflow-Instanzen nicht mehr direkt in Beziehung gesetzt werden, sondern indirekt über Transitionen, die zur Ausführungsanweisung gehören. Ansonsten kommen die Strukturen von Transitionen vor, die in Tabelle 1 vorgestellt wurden, unter anderem Transitionen mit der gleichen linken Seite, die Indeterminismus ausdrücken. Im Zusammenhang mit der Beschreibung von Ausführungsanweisungen bedeutet das: Alle Alternativen sind möglich, die Entscheidung darüber ist kein Bestandteil der Ausführungsanweisung.

Als Argumente einer Ausführungsanweisung sind weitere Ausführungsanweisungen, elementare oder komposite Workflows möglich. Für die Funktion der „äußeren“ Ausführungsanweisung ist die Art des Arguments unerheblich, und daher unterscheiden die folgenden Beschreibungen zu den Ausführungsanweisungen diese Fälle nicht.

4.2.1 Ausführungsmodell der Workflow-Instanzen

Das Ausführungsmodell für Workflow-Instanzen aus Abschnitt 3.4.1 schränkt den Gestaltungsspielraum für die zu entwickelnden Ausführungsanweisungen ein (Begründung siehe Abschnitt 3.2.3.1); Abbildung 4.2 zeigt daher die Erweiterung um die Zustände blocked und aborted. Eine Workflow-Instanz im Zustand blocked erfüllt alle Voraussetzungen zur Ausführung (siehe Abschnitt 2.2.2), wird aber z.B. aus Gründen der Synchronisation mit anderen Workflows noch von der Ausführung zurückgestellt. Von blocked und ready ist ein Übergang in aborted möglich, wo im Gegensatz zu blocked garantiert keine Ausführung mehr erfolgt.



[1]	init	—o	A.ready
[2]	A.ready	—o	A.running
[3]	A.running	—o	A.paused
[4]	A.paused	—o	A.running
[5]	A.running	—o	A.done
[6]	A.done	—o	exit
[7]	A.ready	—o	A.aborted
[8]	A.ready	—o	A.blocked
[9]	A.blocked	—o	A.ready
[10]	A.aborted	—o	exit

Abbildung 4.2: Ausführungsmodell für Workflow-Instanzen

Gemäß diesem Ausführungsmodell verhalten sich die angenommenen Workflow-Instanzen, für die im folgenden Ausführungsanweisungen vorgestellt werden. Ausführungsmodelle von Workflow-Instanzen unterscheiden sich zwischen einzelnen Workflow-Management-Systemen stark (siehe dazu auch Abschnitt 2.2.1), das gezeigte Ausführungsmodell ist daher nur als Beispiel anzusehen. Weiterhin ist zu beachten, daß die gezeigten Zustandsübergänge die sind, die vom Workflow-Management-System *kontrolliert* und ggf. auch selbst herbeigeführt werden können (vgl. [Jabl96a, S. 194ff., S. 210ff.]).

4.2.2 Elementare Ausführungsformen

Elementare Ausführungsformen beziehen sich auf die Ausführung eines Workflows in der einfachsten Erscheinungsform: Entweder findet die Ausführung statt oder nicht; wenn die Ausführung stattfindet, dann entweder nur einmal oder mehrfach. Die als OPT und MULT bezeichneten Ausführungsanweisungen repräsentieren diese Ausführungsformen eines Workflows explizit im Workflow-Schema.

OPT

Mit OPT(A) wird die Ausführung des Workflows A als Option angeboten und damit seine Auslassung gestattet. Die Option zur Ausführung des Workflows gilt nur an dieser Stelle im Kontrollfluß und ist von einem generellen Angebot der Ausführung zu einem beliebigen Zeitpunkt zu unterscheiden. Transition [1] in Abbildung 4.3 hält die Entscheidung für die Ausführung von Workflow A durch den internen Zustand OPT.running fest und kann den Zustand OPT.done durch Transition [3] demnach nur erreichen, wenn der Workflow A wieder beendet wurde. Das ist nicht erforderlich, wenn mit Transition [2] die Ausführung von Workflow A übergangen wird.

[1]	**OPT**.ready	—o	**OPT**.running ∗ A.ready
[2]	**OPT**.ready	—o	**OPT**.done
[3]	**OPT**.running ∗ A.done	—o	**OPT**.done

Abbildung 4.3: OPT(A), Variante II

Bei der zweiten Varianten der optionalen Ausführung in Abbildung 4.4 wird ein Workflow zunächst instanziiert und dem Bearbeiter zugewiesen, der dann seinerseits über die Ausführung oder die Auslassung entscheidet. Das spiegelt eine Situation wieder, in der die Entscheidung über die Workflow-Ausführung nicht formalisiert werden kann und dem Urteilsvermögen des Bearbeiters überlassen werden muß. Der Fall tritt bei Workflows auf, die zur Unterstützung einer Tätigkeit dienen, die für jeden Einzelfall anders beurteilt werden muß (für Beispiele siehe [Schl98]).

[1]	**OPT**.ready	—o	**OPT**.running ∗ A.ready
[2]	**OPT**.running ∗ A.ready	—o	**OPT**.done ∗ A.aborted
[3]	**OPT**.running ∗ A.done	—o	**OPT**.done

Abbildung 4.4: OPT(A), Variante II

Abbildung 4.5 stellt eine weitere Abwandlung dieser Ausführungsanweisung vor. Im Gegensatz zu OPT(A) arbeitet OPT_ASYNC_END(A) asynchron; das Ende von Workflow A wird nicht abgewartet, sondern bereits der Beginn der Ausführung von Workflow A reicht zur Fortsetzung aus. Zu beachten ist, daß mit OPT_ASYNC_END eine zeitlich überlappende Ausführung mit anderen Workflows entstehen kann, die nach dem Ende von OPT_ASYNC_END und vor dem Ende von Workflow A gestartet werden.

[1]	**OPT**.ready	—o	**OPT**.running ∗ A.ready
[2]	**OPT**.ready	—o	**OPT**.done
[3]	**OPT**.running ∗ A.running	—o	**OPT**.done ∗ A.running

Abbildung 4.5: OPT_ASYNC_END(A)

Ohne A.running in Transition [3] in Abbildung 4.5 wäre jede Bindung der Ausführung von Workflow A aufgehoben, noch ist das Ende von OPT vom erfolgreichen Start des Workflows A abhängig.

MULT

Neben der optionalen Ausführung eines Workflows ist die Mehrfachausführung eine weitere elementare Ausführungsform. Die MULT-Anweisung in Abbildung 4.6 realisiert eine aufnehmende Schleife wie ein REPEAT...UNTIL in prozeduralen Programmiersprachen.

[1]	**MULT**.ready	—o	**MULT**.running ∗ A.ready
[2]	**MULT**.running ∗ A.done	—o	**MULT**.running ∗ A.ready
[3]	**MULT**.running ∗ A.done	—o	**MULT**.done

Abbildung 4.6: MULT(A)

Für die Beschreibung der Funktion von MULT ist es unerheblich, ob das Ende der Schleife durch eine Bedingung, einen Zähler oder eine Entscheidung des Anwenders oder ein anderes Kriterium herbeigeführt wird. Die Transitionen [2] und [3] bringen zum Ausdruck, daß nach der einmaligen Ausführung das Ende der Ausführungsanweisung erreicht werden kann. Zu beachten ist, daß nicht die gleiche Workflow-Instanz ausgeführt wird, sondern jedesmal ein neuer Workflow instanziiert und deinstanziiert wird.

4.2.3 Elementare Ausführungsfolgen

Elementare Ausführungsformen legen die Ausführung *eines* Workflows fest; Ausführungsanweisungen wie ALT, SEQ und PAR setzen Beziehungen zwischen mehreren Workflows fest.

ALT

ALT drückt den gegenseitigen Ausschluß mehrerer Workflows aus; Abbildung 4.7 zeigt die Realisierung für zwei Workflows, die Erweiterung auf mehrere Workflows ist klar. Offen bleibt, wie der Indeterminismus zwischen den Transitionen [1] und [2] aufgelöst wird, sei es durch eine explizite Auswahl des Anwenders oder eine vom WFMS zur Laufzeit ausgewertete Bedingung; für die resultierende Workflow-Ausführung ist das unerheblich.

[1]	**ALT**.ready	—o	**ALT**.running ∗ A.ready
[2]	**ALT**.ready	—o	**ALT**.running ∗ B.ready
[3]	**ALT**.running ∗ A.done	—o	**ALT**.done
[4]	**ALT**.running ∗ B.done	—o	**ALT**.done

Abbildung 4.7: ALT(A, B)

Die ALT-Ausführungsanweisung macht deutlich, daß ein gegenseitiger Ausschluß gefordert ist. Beim Einsatz von Kontrollflußprimitiven kann dies nur durch entsprechende Kantenbewertungen erreicht werden und ist nicht unmittelbar ersichtlich.

SEQ

Eine Hintereinanderausführung zweier Workflows wird mit der Ausführungsanweisung SEQ erreicht (Abbildung 4.8). Gemäß der Klassifikation aus Abschnitt 3.2.1.1 realisiert SEQ eine Beziehung des S/S-Typs, da es die Zustände A.done und B.ready in Zusammenhang bringt.

[1]	**SEQ**.ready	—o	**SEQ**.running ∗ A.ready
[2]	**SEQ**.running ∗ A.done	—o	**SEQ**.running ∗ B.ready
[3]	**SEQ**.running ∗ B.done	—o	**SEQ**.done

Abbildung 4.8: SEQ(A, B)

Zu erkennen ist weiterhin, daß diese Realisierung der sequentiellen Ausführung den zweiten Workflow nur von der Beendigung des ersten Workflows abhängig macht und nicht bei-

spielsweise von dessen Ergebnis. Ist hingegen die Ausführung des zweiten Workflows nur sinnvoll, wenn der erste Workflow erfolgreich war, reicht diese Realisierung der Sequenz nicht aus. Da nicht allgemeingültig beantwortet werden kann, wann eine Workflow-Ausführung „erfolgreich“ war, muß dies individuell definiert werden und dem WFMS gegenüber dargestellt werden. Eine Möglichkeit dazu bietet ein Ausführungsmodell mit mehreren Endzuständen. Differenzierte Endzustände erlauben eine Realisierung der Sequenz wie in Abbildung 4.9.

[1]	**SEQ**.ready	—o	**SEQ**.running ∗ A.ready
[2]	**SEQ**.running ∗ A.aborted	—o	**SEQ**.running ∗ A.ready
[3]	**SEQ**.running ∗ A.committed	—o	**SEQ**.running ∗ B.ready
[4]	**SEQ**.running ∗ B.aborted	—o	**SEQ**.running ∗ B.ready
[5]	**SEQ**.running ∗ B.committed	—o	**SEQ**.done

Abbildung 4.9: SEQ_COMMIT(A, B)

Das Beispiel von SEQ_COMMIT zeigt, daß die Aufgabe einer Ausführungsanweisung weiter gefaßt werden kann: Die Entscheidung über eine mehrfache Ausführung der Workflows (in den Transitionen [2] und [4]) wird in der Ausführungsanweisung getroffen und macht es überflüssig, diese Situation im Workflow-Schema durch zusätzliche Anweisungen zu behandeln.

PAR

Die dritte Form einer elementaren Ausführungsfolge von Workflows ist die Zulassung einer zeitlich überlappenden Ausführung. In der einfachsten Form wird dies mit der Ausführungsanweisung PAR erreicht (Abbildung 4.10).

[1]	**PAR**.ready	—o	**PAR**.running ∗ A.ready ∗ B.ready
[2]	**PAR**.running ∗ A.done ∗ B.done	—o	**PAR**.done

Abbildung 4.10: PAR(A, B)

Die dargestellte Realisierungsvariante von PAR *erlaubt* die Parallelausführung, erzwingt bzw. garantiert sie jedoch nicht. Die gleichzeitige Bereitstellung ist sinnvoll, wenn die Workflows ohne gegenseitige Bezugnahme ausgeführt werden können und die nebenläufige Ausführung die Bearbeitungszeit verkürzen soll. *Verlangt* eine Anwendung hingegen die gleichzeitige Aktivität beider Workflows, ist die Nebenläufigkeit also nicht optional, erfüllt obige Realisierung von PAR diese Anforderungen nicht. Die mit Transition [1] beschriebene Bereitstellung beider Workflows bedeutet keineswegs, daß auch die Ausführung zum gleichen Zeitpunkt beginnt. Analog synchronisiert Transition [2] nicht das Ende der beiden Workflows A und B, sondern erlaubt die Fortsetzung erst nach Beendigung beider Workflows.

Die Formulierung der verlangten Parallelausführung setzt voraus, daß im Ausführungsmodell der Workflows mehr als nur die Zustände ready und done sichtbar sind, weil anderenfalls Ausführungsbereitschaft nicht von tatsächlicher Bearbeitung unterschieden werden kann. Allerdings reicht die Sichtbarkeit der internen Zustände nicht aus, sondern Zustandsübergänge

müssen auch genauer von der Ausführungsanweisung kontrolliert werden können, um eine präzise Synchronisation der Ausführung zu gewährleisten. Ausführungsanweisungen mit diesen Eigenschaften werden in Abschnitt 4.2.5 vorgestellt.

SER

SER bietet Workflows zur seriellen Ausführung an; im Gegensatz zu SEQ ist die Ausführung jedes einzelnen Workflows hinsichtlich der Anzahl nicht eingeschränkt, Mehrfachausführungen sind also zulässig. Abbildung 4.11 zeigt die Realisierung der SER für zwei Workflows.

[1]	**SER**.ready	—o	**SER**.running ∗ A.ready
[2]	**SER**.running ∗ A.done	—o	**SER**.running ∗ A.ready
[3]	**SER**.running ∗ A.done	—o	**SER**.running ∗ B.ready
[4]	**SER**.running ∗ B.done	—o	**SER**.running ∗ B.ready
[5]	**SER**.running ∗ B.done	—o	**SER**.done

Abbildung 4.11: SER(A, B)

SER beschreibt eine Workflow-Ausführung, die auch SEQ(MULT(A),MULT(B)) erreicht werden kann. Die Ersetzung von Ausführungsanweisungen oder ganzen Teilausdrücken unter Erhalt der Ausführungsfolge der Workflows wird in [Böhm96] behandelt.

4.2.4 Deskriptive Serialisierung

Erfordert eine Anwendungssituation, daß bestimmte Workflows ausgeführt werden müssen, besteht jedoch keine Vorgabe für Reihenfolge, kann dieser Freiheitsgrad auf zwei Wegen erreicht werden: Entweder werden sämtliche Möglichkeiten zulässiger Abfolgen explizit ausformuliert, oder die Ausführungsanweisung wird in die Lage versetzt, die dahinter stehende Anwendungslogik selbst zu realisieren. Beide Wege können aus Sicht der zulässigen Abfolgen von Workflow-Ausführungen gleich sein, ihre nichtfunktionalen Eigenschaften sind allerdings unterschiedlich (siehe [Böhm96]). Die folgenden Ausführungsanweisungen ALL und DELAY sind verschiedenartige Beispiele dafür, wie eine serielle Ausführung zweier Workflows nur in ihrem Ergebnis vorgeschrieben wird, die zur Erreichung zulässige Folge von Ausführungen aber offen bleibt.

ALL

ALL fordert die Ausführung von Workflows, gibt aber keine Reihenfolge vor. Die in Abbildung 4.12 gezeigte Realisierungsvariante erreicht durch das einmal vorhandene Token Sync, daß die gleichzeitige Ausführung beider Workflows verhindert wird (was auch durch ALL.running allein garantiert wäre, allerdings in mißbräuchlicher Verwendung des Tokens).

[1]	**ALL**.ready	—o	**ALL**.running ∗ A.ready ∗ B.ready ∗ Sync
[2]	**ALL**.running ∗ A.ready ∗ Sync	—o	**ALL**.running ∗ A.done ∗ Sync
[3]	**ALL**.running ∗ B.ready ∗ Sync	—o	**ALL**.running ∗ B.done ∗ Sync
[4]	**ALL**.running ∗ A.done ∗ B.done ∗ Sync	—o	**ALL**.done

Abbildung 4.12: ALL(A, B)

Durch diese Zusatzeigenschaft des gegenseitigen Ausschlusses unterscheidet sich ALL von PAR, weil im letzteren eine parallele Ausführung zulässig ist. Die unter ALL zulässige Ausführung von Workflows kann auch ohne expliziten Synchronisationsmechanismus erreicht werden. Das folgende Beispiel DELAY stammt aus [Jabl94] und unterscheidet sich von ALL in der Hinsicht, daß nicht alle Workflows wie bei ALL ausgeführt werden müssen.

DELAY

Im Gegensatz zu ALL sind die Ausführungen der Workflows bei DELAY voneinander abhängig. So kann zu Beginn jeder der beiden Workflows A und B ausgeführt werden, wenn jedoch die Entscheidung für Workflow A fällt, wird die Möglichkeit zur Ausführung von Workflow B blockiert und erst nach Ende von Workflow A wieder zugelassen.

[1]	**DELAY**.ready	—o	**DELAY**.running ∗ A.ready ∗ B.ready
[2]	**DELAY**.running ∗ A.ready ∗ B.ready	—o	**DELAY**.running ∗ A.running ∗ B.blocked
[3]	**DELAY**.running ∗ A.running ∗ B.blocked	—o	**DELAY**.running ∗ A.done ∗ B.ready
[4]	**DELAY**.running ∗ A.done ∗ B.ready	—o	**DELAY**.running ∗ A.done ∗ B.running
[5]	**DELAY**.running ∗ A.done ∗ B.running	—o	**DELAY**.running ∗ A.done ∗ B.done
[6]	**DELAY**.running ∗ A.done ∗ B.done	—o	**DELAY**.done
[7]	**DELAY**.running ∗ A.ready ∗ B.ready	—o	**DELAY**.running ∗ A.aborted ∗ B.running
[8]	**DELAY**.running ∗ A.aborted ∗ B.running	—o	**DELAY**.running ∗ A.aborted ∗ B.done
[9]	**DELAY**.running ∗ A.aborted ∗ B.done	—o	**DELAY**.done

Abbildung 4.13: DELAY(A, B)

4.2.5 Präskriptive Parallelausführung

PAR ließ offen, in welcher Form die Parallelausführung der Workflows stattfindet; jede Form zeitlicher Überlappung und sogar die Hintereinanderausführung ist zulässig. Nach der Unterscheidung in [Jabl94] liegt damit eine *deskriptive* Beschreibung vor, weil nur die Tatsache der Parallelausführung gefordert wird, aber keine Anweisung existiert, wie diese im einzelnen aussehen muß. Dieser Art der Parallelausführung werden hier *präskriptive* Varianten gegenübergestellt, bei denen die Form der Parallelausführung vorgegeben ist. Die Allen-Intervallbeziehungen [Alle83] illustrieren dabei unterschiedliche Formen der Parallelausführung, die schon in [Böhm96] zur Einführung neuer Ausführungsanweisungen dienten. Das Intervall repräsentiert die Dauer der Workflow-Ausführung, wobei es bei den Allen-Intervallbeziehungen nicht auf das Längenverhältnis ankommt, sondern auf die Lage der In-

tervalle zueinander. Des weiteren werden nur Beispiele behandelt, in denen das Ausführungsergebnis eines Workflows keinen Einfluß auf die Ausführung des anderen Workflows hat.

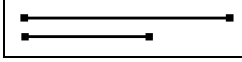
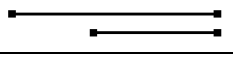
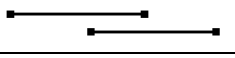
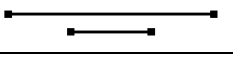
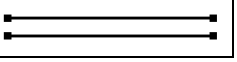
starts	*ends*	*overlaps*	*during*	*equal*
				

Tabelle 2: Allen-Intervallbeziehungen (Auswahl)

ENFORCE_OVERLAPS

Die Allen-Intervallbeziehung *overlaps* besagt, daß es zwischen zwei Intervallen eine Überschneidung gibt. Daraus folgt die neue Ausführungsanweisung ENFORCE_OVERLAPS(A, B), als EO abgekürzt, bei der Workflow A zunächst unabhängig von Workflow B zur Ausführung angeboten wird (Abbildung 4.14). Nach dem Start von A ist die Beendigung nur möglich, wenn bis B im Zustand running ist. Die Ausführung von B kann nur nach A beendet werden.

[1]	**EO**.ready	—o	**EO**.running ∗ A.ready
[2]	**EO**.running ∗ A.ready	—o	**EO**.running ∗ A.running ∗ B.ready
[3]	**EO**.running ∗ A.running ∗ B.ready	—o	**EO**.running ∗ A.running ∗ B.running
[4]	**EO**.running ∗ A.running ∗ B.running	—o	**EO**.running ∗ A.done ∗ B.running
[5]	**EO**.running ∗ A.done ∗ B.running	—o	**EO**.running ∗ A.done ∗ B.done
[6]	**EO**.running ∗ A.done ∗ B.done	—o	**EO**.done

Abbildung 4.14: ENFORCE_OVERLAPS(A, B)

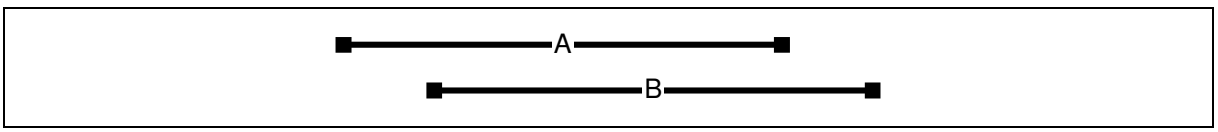


Abbildung 4.15: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_OVERLAPS(A, B)

Die von ENFORCE_OVERLAPS erbrachte Leistung kann auf zwei Arten interpretiert werden: Übernehmen die Workflows A und B beispielsweise Überwachungsaufgaben, so ist mittels ENFORCE_OVERLAPS garantiert, daß der Wechsel der Überwachung zwischen den Workflows „unterbrechungsfrei“ erfolgt. Die andere Interpretation stellt die Phase der gemeinsamen Ausführung in den Vordergrund und schafft damit die Voraussetzung, daß beide Workflows gemeinsam eine Operation ausführen, die eine Instanz allein nicht erbringen kann oder nicht ausführen darf.

OPTIONAL DURING

Die Intervallbeziehung *during* stellt eine Steigerung von *overlaps* dar, da hier die vollständige Inklusion eines Intervalls gefordert ist. Die Funktion von OPTIONAL_DURING(A, B), als OD abgekürzt, lehnt sich daran an, zeigt aber ein qualifizierteres Verhalten: Nur *wenn* B ausgeführt wird, *dann* muß es in dieser Form geschehen. Wird während der Ausführung A der Workflow B gar nicht gestartet, tritt diese Restriktion nicht in Kraft (Abbildung 4.16).

[1]	**OD**.ready	—o	**OD**.running ∗ A.ready
[2]	**OD**.running ∗ A.ready	—o	**OD**.running ∗ A.running ∗ B.ready
[3]	**OD**.running ∗ A.running ∗ B.ready	—o	**OD**.running ∗ A.running ∗ B.running
[4]	**OD**.running ∗ A.running ∗ B.ready	—o	**OD**.running ∗ A.done ∗ B.blocked
[5]	**OD**.running ∗ A.running ∗ B.running	—o	**OD**.running ∗ A.running ∗ B.done
[6]	**OD**.running ∗ A.running ∗ B.done	—o	**OD**.running ∗ A.done ∗ B.done
[7]	**OD**.running ∗ A.done ∗ B.done	—o	**OD**.done
[8]	**OD**.running ∗ A.done ∗ B.blocked	—o	**OD**.done

Abbildung 4.16: OPTIONAL_DURING(A, B)

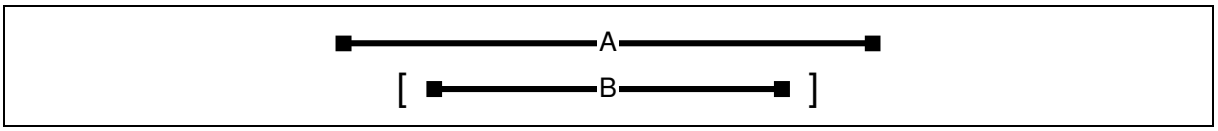


Abbildung 4.17: Intervalldarstellung der Workflow-Ausführung durch OPTIONAL_DURING(A, B)

ENFORCE_ENDS

ENFORCE_ENDS(A, B) erlaubt den beiden Workflows A und B einen voneinander unabhängigen Start, setzt jedoch das Ende der Ausführung in Beziehung. Das Ende eines jeden Workflows ist erst dann möglich, wenn auch der jeweils andere beendet werden kann. Die Auswirkung von ENFORCE_ENDS ist von einer üblichen Zusammenführung mehrerer Workflows zu unterscheiden, da bei solchen externen Synchronisationspunkten lediglich die Fortsetzung nach dem unabhängigen Ende beider Workflows erreicht wird. Allerdings ist für die Realisierung dieser Ausführungsanweisung eine Erweiterung der Ausführungsmodelle der Workflows um den Zustand term_ready notwendig (Abbildung 4.18).

[1]	**EE**.ready	—o	**EE**.running ∗ A.ready ∗ B.ready
[2]	**EE**.running ∗ A.ready	—o	**EE**.running ∗ A.running
[3]	**EE**.running ∗ B.ready	—o	**EE**.running ∗ B.running
[4]	**EE**.running ∗ A.running	—o	**EE**.running ∗ A.term_ready
[5]	**EE**.running ∗ B.running	—o	**EE**.running ∗ B.term_ready
[6]	**EE**.running ∗ A.term_ready ∗ B.term_ready	—o	**EE**.running ∗ A.done ∗ B.done
[7]	**EE**.running ∗ A.done ∗ B.done	—o	**EE**.done

Abbildung 4.18: ENFORCE_ENDS(A, B)

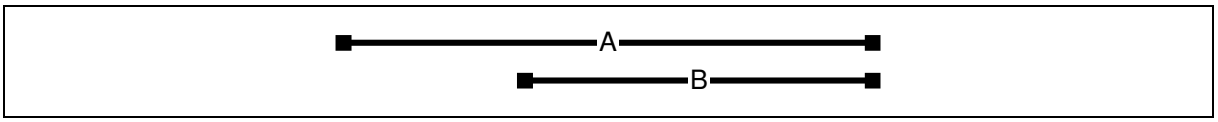


Abbildung 4.19: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_ENDS(A, B)

ENFORCE_SWITCH

ENFORCE_SWITCH(A, B) schreibt einen zweifachen Wechsel zwischen beiden Workflows vor. Sobald Workflow A durch Unterbrechung nicht weiter ausgeführt wird, kann Workflow B ausgeführt werden. Das Ende von Workflow B hebt die Unterbrechung von Workflow A auf und führt ihn weiter aus (Abbildung 4.20).

[1]	**ES**.ready	—o	**ES**.running ∗ A.ready
[2]	**ES**.running ∗ A.ready	—o	**ES**.running ∗ A.running ∗ B.ready
[3]	**ES**.running ∗ A.running ∗ B.ready	—o	**ES**.running ∗ A.paused ∗ B.running
[4]	**ES**.running ∗ A.paused ∗ B.running	—o	**ES**.running ∗ A.running ∗ B.done
[5]	**ES**.running ∗ A.running ∗ B.done	—o	**ES**.running ∗ A.done ∗ B.done
[6]	**ES**.running ∗ A.done ∗ B.done	—o	**ES**.done

Abbildung 4.20: ENFORCE_SWITCH(A, B)

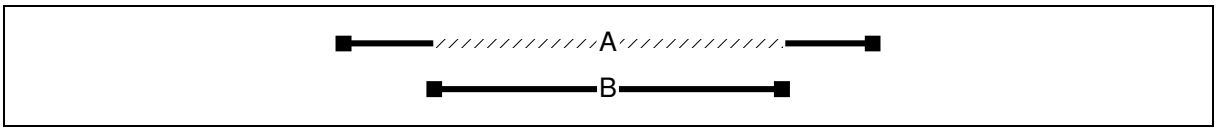


Abbildung 4.21: Intervalldarstellung der Workflow-Ausführung durch ENFORCE_SWITCH(A, B)

4.2.6 Klassifikationsmöglichkeiten für Ausführungsanweisungen

Die letzten Abschnitte haben Ausführungsanweisungen gezeigt, die über das Angebot der Ausdrucksmittel kommerzieller WFMS und verschiedener Forschungsprojekte hinausgehen, eine Vielzahl weiterer Ausführungsanweisungen sind möglich. Für die Klassifikation dieser Ausführungsanweisungen kommen verschiedene Ansätze in Frage; dieser Abschnitt zieht die resultierende Ausführung der kontrollierten Workflows heran. Eine darauf aufgebaute Klassifikation hilft dem Anwendungsentwickler bei der Wahl der geeigneten Ausführungsanweisung mehr, als wenn z.B. die Funktionsweise oder die Art der Implementierung als Klassifikationskriterium dient. Ausgangspunkt einer verwendungsorientierten Klassifikation sind Facetten der Workflow-Ausführung.

4.2.6.1 Facetten für die Klassifikation von Ausführungsanweisungen

Jede Facette charakterisiert nur einen Teilaspekt (!) bei der Ausführung von Workflows, beispielsweise das Verhältnis der Ausführungszeiten zweier Workflows zueinander. Facetten nehmen verschiedene Ausprägungen an, bei den Ausführungszeiten wären dies „hintereinander“, „überlappend“, „verschränkt“ usw. Die folgende Liste führt Facetten auf, die in der Klassifikation von Abschnitt 4.2.6.2 Verwendung finden:

F1: Die Stelligkeit der Ausführungsanweisung bestimmt die Anzahl *unmittelbar* kontrollierter Workflows; allerdings ist die Anzahl der *mittelbar* kontrollierten Workflows größer, wenn komposite Workflows als Argumente vorkommen.

Beispiel: OPT(A) ist einstellig, SEQ(A,B) ist mehrstellig.

F2: Die Modalität legt fest, ob eine die Ausführung betreffende Eigenschaft eingehalten werden muß oder nicht.

Beispiel: Bei SEQ(A, B) ist die einmalige Ausführung der Workflows zwingend, bei SER(A, B) nicht.

F3: Mehrfachausführung bedeutet die erneute Instanziierung des Workflow-Schemas.

Beispiel: MULT(A) erlaubt die mehrfache Ausführung von Workflow A, OPT(A) nicht.

F4: Gibt die Ausführungsanweisung eine Reihenfolge für die Ausführung der Workflows vor, kann diese Reihenfolge entweder fest vorgegeben sein oder erst zur Laufzeit bestimmt werden.

Beispiel: Bei SEQ(A, B) ist die Reihenfolge vorgegeben, bei ALL(A, B) ergibt sich die Reihenfolge aus der Entscheidung des Bearbeiters.

F5: Zwischen Workflows einer Ausführungsanweisung können Abhängigkeiten bestehen. Abhängigkeiten ergeben sich entweder aus der Reihenfolge der Ausführung (siehe F4) oder sind zur Synchronisation einer zeitlich überlappenden Ausführung (siehe F6) notwendig.

Beispiel: Bei SEQ(A, B) sind die Workflows bis auf die Ausführungsreihenfolge voneinander unabhängig, bei DELAY(A, B) kann die vorhandene Abhängigkeit dazu führen, einen von beiden Workflows auszulassen.

F6: Als Unterscheidung von Ausführungsformen kann die Zulassung der Parallelausführung dienen, ohne verschiedene Formen der Parallelausführung zu differenzieren (siehe F7) oder die Modalität der Parallelausführung (siehe F2) zu unterscheiden.

Beispiel: SEQ(A, B) verhindert die Parallelausführung bei der Workflows garantiert, PAR(A, B) ermöglicht sie zumindest.

F7: Erlaubt die Ausführungsanweisung die zeitlich überlappende Ausführung mehrerer Workflows, liefert die weitere Differenzierung der Parallelausführung ein charakteristisches Merkmal.

Beispiel: ENFORCE_ENDS(A, B) und ENFORCE_OVERLAPS(A, B) sind Spezialformen der Parallelausführung mittels PAR(A, B).

F8: Die Ausführung eines Workflows kann in der Ausführungsanweisung als atomares Ereignis dargestellt werden, oder die realisierte Funktion setzt eine kontrollierte Unterbrechung der Workflow-Ausführung voraus.

Beispiel: In SEQ(A, B) werden beide Workflows (von Seiten der Ausführungsanweisung) nicht unterbrochen, bei ENFORCE_SWITCH(A, B) ist die Unterbrechung eines Workflows Bestandteil der intendierten Ausführung.

F9: Innerhalb der Ausführungsanweisung ist eine Auswahl eines oder mehrerer Workflows möglich, die von der Ausführung ausgenommen, wiederholt oder anderweitig gesondert behandelt werden.

Beispiel: Bei ALT(A, B) bedeutet Auswahl die Entscheidung für einen von beiden Workflows, Auswahl kann aber auch wie z.B. bei SER(A, B) bedeuten, die Ausführung auszulassen.

Weitere Facetten sind möglich, wenn statt dem Ergebnis der Workflow-Ausführung z.B. die Details der Realisierung von Ausführungsanweisungen herangezogen werden.

4.2.6.2 Beispiel eines Klassifikationsschemas

Für die Charakterisierung einer Ausführungsanweisung sind mehrere Facetten notwendig, deren Ausprägungen ein Gesamtbild der Eigenschaften zeichnen. In diesem Abschnitt wird gezeigt, wie durch Kombination der Facetten ein Klassifikationsschema entsteht, das die Ausführungsanweisungen aus den Abschnitten 4.2.1 bis 4.2.5 einteilt. Das in Abbildung 4.22 vorgestellte Klassifikationsschema ist nur ein Beispiel, andere Kombinationen der Facetten liefern eine abweichende Einteilung der Ausführungsanweisungen.

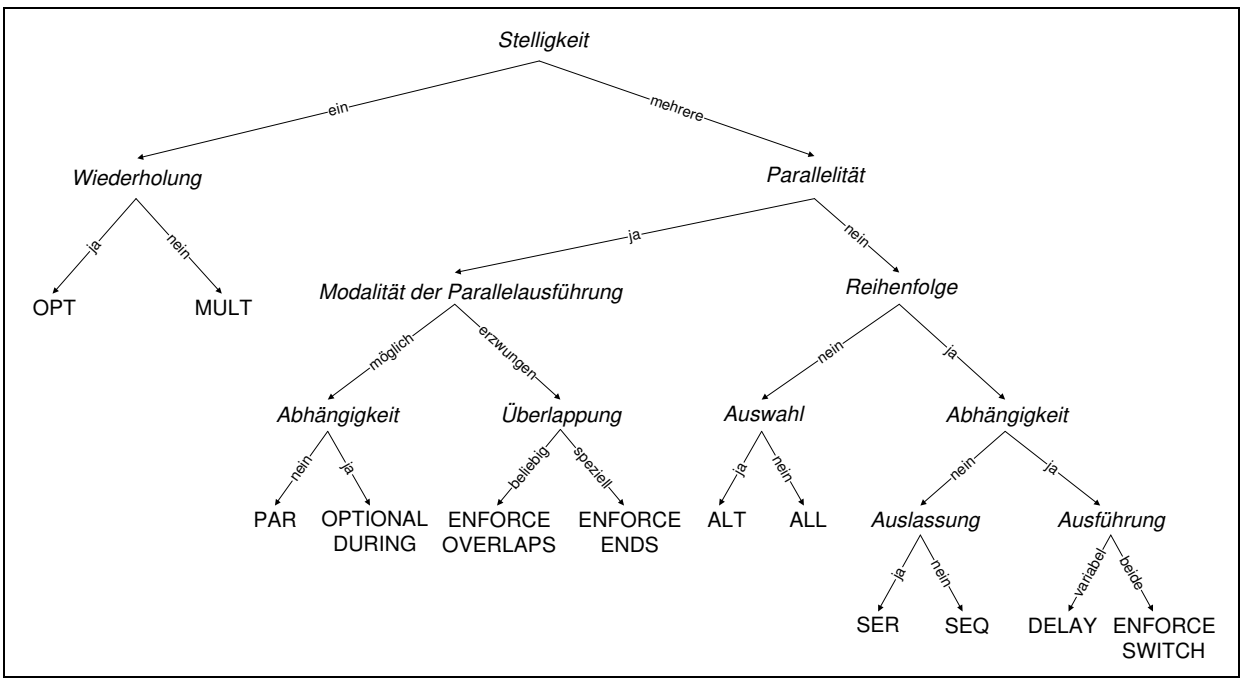


Abbildung 4.22: Beispiel für eine Klassifikation von Ausführungsanweisungen

Facette F1 unterteilt die Ausführungsanweisungen (ALL, ALT, DELAY, ENFORCE_ENDS, ENFORCE_OVERLAPS, ENFORCE_SWITCH, MULT, OPT, OPTIONAL_DURING, PAR, SEQ, SER) nach ihrer Stelligkeit. Die Stelligkeit ist mehr als eine syntaktische Eigenschaft: Mehrstellige Ausführungsanweisungen stellen zwischen den kontrollierten Workflows eine Beziehung her, die das Verständnis des Workflow-Schemas erleichtern und dem WFMS die Möglichkeit bieten, aus Informationen über mehrere Workflow-Instanzen Entscheidungen über die weitere Ausführung abzuleiten.

Die einstelligen Ausführungsanweisungen OPT und MULT unterscheiden sich hinsichtlich der Mehrfachausführung (Facette F3), die von MULT im Gegensatz zu OPT zugelassen ist. Beide Ausführungsanweisungen sind erforderlich, da anders als beim Einsatz von Kontrollflußprimitiven keine Auslassung oder Wiederholung durch die Art der Verknüpfung der Workflows erreicht werden kann.

Alle anderen Ausführungsanweisungen sind mehrstellig, die Beschränkung auf jeweils zwei Workflows ist keine generelle Einschränkung, sondern folgt aus der Wahl der Beispiele. Die verbleibenden Ausführungsanweisungen werden danach unterschieden, ob sie eine parallele Ausführung mehrerer Workflows zulassen. Weiter wird gefragt, ob die Parallelität der Ausführung *ermöglicht* wird, oder ob das WFMS über die Ausführungsanweisung die Ausführung der Workflows so steuert, daß eine Überlappung der Ausführungszeiten *erzwungen* wird.

Bei PAR und OPTIONAL_DURING, die beide die parallele Ausführung von Workflows erlauben, stellt PAR die Workflows gleichzeitig zur Ausführung bereit; während der Workflow-Ausführung besteht zwischen beiden Workflows keine Interaktion mehr. Bei OPTIONAL_DURING sind die Workflows voneinander abhängig, denn einer kann nur während der Ausführungszeit des anderen ausgeführt werden. Daher gibt das gewählte Klassifikationskriterium den Unterschied nicht vollständig wieder; es gibt weitere Eigenschaften der beiden Ausführungsanweisungen, die nicht durch das eine Kriterium erfaßt werden.

Durch ENFORCE_OVERLAPS und ENFORCE_ENDS steuert das WFMS die Ausführung der Workflows so, daß entweder während der Ausführungszeit eine Überlappung stattfindet oder das Ende beider Workflows synchronisiert wird; als Unterscheidungsmerkmal dient die Lagebeziehung der Intervalle (vgl. Abbildung 4.14 und Abbildung 4.18).

Die verbleibenden Ausführungsanweisungen unterbinden eine Parallelausführung der Workflows; die Hintereinanderausführung wird bezüglich der Existenz einer Reihenfolge unterschieden (Facette F4). Eine Reihenfolge fehlt in zwei Fällen: Bei der Auswahl eines von mehreren Workflows wie bei ALT oder durch die Zulassung aller möglichen Abfolgen wie bei ALL.

Ausführungsanweisungen, die Workflows in einer bestimmten Reihenfolge ausführen, können die Ausführung jedes einzelnen Workflows unabhängig von den jeweils anderen organisieren oder Abhängigkeiten zwischen den Workflows berücksichtigen. Bei SEQ und SER ist die Entscheidung, ob ein Workflow ausgeführt wird, unabhängig von den restlichen Workflows. Bei SEQ gibt es eine Reihenfolge, aber jeder Workflows muß ausgeführt werden, was bei SER nicht erforderlich ist, da die Auslassung akzeptiert wird.

DELAY und ENFORCE_SWITCH zeigen sowohl eine Reihenfolge für die Ausführung der Workflows als auch eine gegenseitige Abhängigkeit. Die Unterscheidung liefert die Tatsache, daß bei DELAY einer der beiden Workflows unter Umständen nicht zur Ausführung kommt, bei ENFORCE_SWITCH werden beide Workflows sicher ausgeführt (vgl. Abbildung 4.20).

Werden die Ausführungsanweisungen in einer anderen Reihenfolge mit den Kriterien verglichen, ergibt sich eine von Abbildung 4.22 abweichende Darstellung. Diese Einflußmöglichkeit erlaubt, die Ausführungsanweisungen für unterschiedliche Zwecke unterschiedlich zu

gliedern. Der Anwendungsentwickler benötigt eine andere „Sortierung“, wenn er nach einer Ausführungsanweisung für eine bestimmte Form der Workflow-Ausführung sucht, als der Entwickler von Ausführungsanweisungen.

4.3 Simulation der Funktionsweise von Ausführungsanweisungen

Die vollständige Implementierung einer Ausführungsanweisung ist aufwendig; ein Test der Funktionsweise im Vorfeld der Implementierung erscheint daher sinnvoll. Die Simulation gibt Aufschluß über Entwurfsfehler oder zeigt die Wirkung der Ausführungsanweisung im Kontext eines kompositen Workflows mit mehreren Subworkflows. Für Verteilte Transitionssysteme existiert eine Simulationsumgebung, die einen interaktiven Test des Schaltverhaltens erlaubt. Diese Simulationsumgebung wird verwendet, um die Ausführung von Subworkflows eines kompositen Workflows zu simulieren.

4.3.1 Aufbau zusammenhängender Ausführungsanweisungen

Geschachtelte Ausführungsanweisungen führen dazu, daß ein Workflow von mehreren Ausführungsanweisungen „eingekleidet“ wird (z.B. die bei `ALT( SEQ( A, B ), C )`). Diese Situation wird analog zur Situation behandelt, daß eine Workflow-Variable als Argument vorkommt. Die Nachbildung des kompositen Workflows erfordert zwei Schritte:

1. Jede Ausführungsanweisung hat eigene Transitionen und Token. Wird in einer Ausführungsanweisung ein Workflow als Argument genannt, enthält die Transition der Ausführungsanweisung bereits Token für die Zustände der Workflow-Instanz. Daher ist nur der Fall zu berücksichtigen, daß die Ausführungsanweisung als Argument in einem Ausdruck vorkommt. Hier müssen die Token für die Zustände der aufgerufenen Ausführungsanweisung die Token ersetzen, die in der aufrufenden Ausführungsanweisung die Workflow-Instanz repräsentierten. Token für gleichlautende Ausführungsanweisungen sind voneinander zu unterscheiden.

2. Es müssen die Transitionen für die Zustandsänderungen der elementaren Workflow-Instanzen eingefügt werden. Durch die neuen Transitionen dürfen keine neue Möglichkeiten geschaffen oder bestehende Transitionen umgangen werden.

Das Verteilte Transitionssystem entsteht durch Komposition aller Transitionen und Token unter entsprechenden Umbenennung von Token. Das Transitionssystem beschreibt das Verhalten des kompositen Workflows.

4.3.2 Beispiel einer automatisch erzeugten Kontrollflußbeschreibung

An einem Beispiel wird gezeigt, wie Transitionen eines Verteilten Transitionssystems erzeugt werden, um die Ausführung eines kompositen Workflows zu simulieren.

4.3.2.1 Schablonen für Transitionen einzelner Ausführungsanweisungen

Jeder Ausführungsanweisung ist eine Schablone, bestehend aus einer Menge von Transitionen, zugeordnet. Platzhalter in den Transitionen werden so ersetzt, daß sie die Aufrufstruktur der Workflows nachbilden. Exemplarisch für ALT, SEQ und PAR zeigen Abbildung 4.23 bis Abbildung 4.25 die Schablonen.

```
<SEQ,<1>,<2>> = {  SEQ.ready —o SEQ.running ∗ <1>.ready
                   SEQ.running ∗ <1>.done —o SEQ.running ∗ <2>.ready
                   SEQ.running ∗ <2>.done —o SEQ.done }
```

Abbildung 4.23: Schablone der SEQ-Ausführungsanweisung

```
<ALT,<1>,<2>> = {  ALT.ready —o ALT.running ∗ <1>.ready
                   ALT.ready —o ALT.running ∗ <2>.ready
                   ALT.running ∗ <1>.done —o ALT.done
                   ALT.running ∗ <2>.done —o ALT.done }
```

Abbildung 4.24: Schablone der ALT-Ausführungsanweisung

```
<PAR,<1>,<2>> = {  PAR.ready —o PAR.running ∗ <1>.ready ∗ <2>.ready
                   PAR.running ∗ <1>.done ∗ <2>.done —o PAR.done }
```

Abbildung 4.25: Schablone der PAR-Ausführungsanweisung

Platzhalter in spitzen Klammern werden bei der Erzeugung der Transitionen durch Token ersetzt. Schablonen für weitere Ausführungsanweisungen folgen aus den Realisierungen in Abschnitt 4.2.

4.3.2.2 Transitionen für komposite Ausführungsanweisungen

Die Transitionen sind entsprechend der Struktur der kompositen Ausführungsanweisung zusammenzufügen. Dazu reicht es aus, zu jeder Ausführungsanweisung im Ausdruck nur ihre unmittelbar folgenden Argumente, die selbst wieder Ausführungsanweisungen sein können, zu betrachten. In der Schablone für die Ausführungsanweisung werden die Token der jeweiligen Argumente eingesetzt und die produzierten Transitionen dem zu entwickelnden Transitionssystem zugefügt. Durch Traversieren des Ausdrucks von oben nach unten und außen nach innen werden alle Ausführungsanweisungen mit ihren Argumenten betrachtet. Das Verfahren wird an folgender kompositen Ausführungsanweisung demonstriert: SEQ(ALT(A, B), PAR(C, SEQ(D, E))). Abbildung 4.26 zeigt die vier Schritte beim Durchlauf des Ausdrucks, der in Baumstruktur dargestellt ist. Gleichlautende Ausführungsanweisungen werden durch Indizes unterschieden.

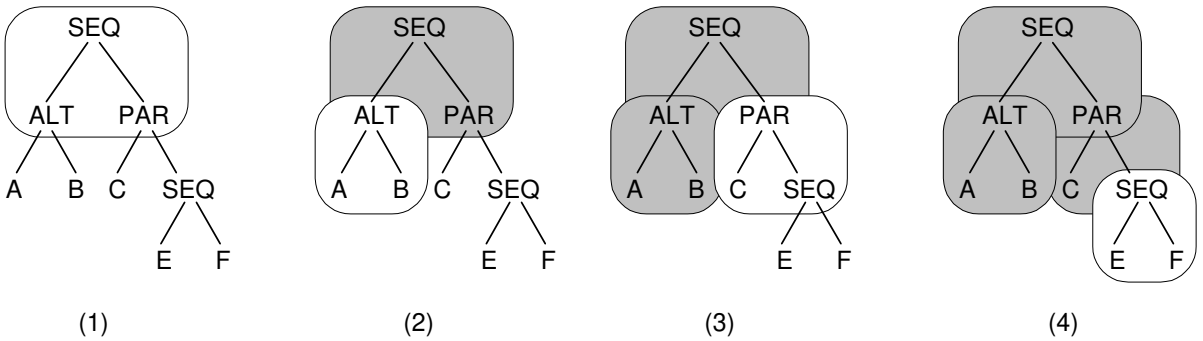


Abbildung 4.26: Traversieren einer kompositen Ausführungsanweisung

Schritt (1) liefert durch die SEQ-Schablone und den Argumenten ALT(A, B) und PAR(C, SEQ(E, F)) die Transitionen SEQ1.ready —o SEQ1.running ∗ ALT1.ready für den ersten Teilausdruck, SEQ1.running ∗ ALT1.done —o SEQ1.running ∗ PAR1.ready für den zweiten Teilausdruck und SEQ1.running ∗ PAR1.done —o SEQ1.done für das Ende gesamten Ausführung. Daß hinter ALT und SEQ wiederum neue Ausführungsanweisungen und Workflow stehen, ist für den Konstruktionsvorgang unerheblich. Die Schritte (2) bis (4) führen schließlich zu den Abbildung 4.27 gezeigten Transitionen.

SEQ1.ready	—o	SEQ1.running ∗ ALT1.ready
SEQ1.running ∗ ALT1.done	—o	SEQ1.running ∗ PAR1.ready
SEQ1.running ∗ PAR1.done	—o	SEQ1.done
ALT1.ready	—o	ALT1.running ∗ A.ready
ALT1.ready	—o	ALT1.running ∗ B.ready
ALT1.running ∗ A.done	—o	ALT1.done
ALT1.running ∗ B.done	—o	ALT1.done
PAR1.ready	—o	PAR1.running ∗ C.ready ∗ SEQ2.ready
PAR1.running ∗ C.done ∗ SEQ2.done	—o	PAR1.done
SEQ2.ready	—o	SEQ2.running ∗ E.ready
SEQ2.running ∗ E.done	—o	SEQ2.running ∗ F.ready
SEQ2.running ∗ F.done	—o	SEQ2.done

Abbildung 4.27: Transitionen für SEQ(ALT(A, B), PAR(C, SEQ(D, E)))

Im nächsten Schritt des Konstruktionsvorgangs werden Transitionen zugefügt, die das lokale Verhalten der Workflows beschreiben. Das geschieht hier durch einen Übergang von <Workflow-Name>.ready nach <Workflow-Name>.done, weil Zwischenzustände der Workflow-Instanzen wie beispielsweise running bei den verwendeten Ausführungsanweisungen keine Rolle spielen. Werden die Transitionen aus Abbildung 4.27 durch die Transition init —o SEQ1.ready vervollständigt, erfolgt die Übernahme in den Simulator.

4.4 Implementierung der Ausführungsanweisungen

Verteilte Transitionssysteme sind ein Hilfsmittel zum Entwurf und zum Test neuer Ausführungsanweisungen; die Simulationsumgebung dient dazu, das Verhalten und die resultierende Ausführung der Workflows zu überprüfen. Zur Implementierung der Ausführungsanweisungen in einem speziellen WFMS sind weitere Schritte notwendig, die allerdings durch die Unterschiedlichkeit der vorkommenden Modellierungsansätze und Methode zur Erstellung von Workflow-Schemata nicht durch ein allgemein gültiges Verfahren abzudecken sind.

Legt ein WFMS das Verhalten von Workflows durch Kontrollflußprimitive fest, lassen sich Transitionen auf diese Kontrollflußprimitive abbilden, sofern die an der Transition beteiligten Zustände mit denen übereinstimmen, die das Kontrollflußprimitiv in Beziehung setzt. Am Beispiel des WFMS FlowMark bedeutet dies, eine Transition A.done —o B.ready entspricht einem Kontrollflußkonnektor, der die Aktivität A mit der Aktivität B verbindet. Transitionen mit anderen Zuständen (z.B. running oder blocked) sind mangels vergleichbarer (und sichtbarer) Zustände in den Aktivitäten von FlowMark nicht abbildbar. Analog beim Einsatz von ECA-Regeln, wie in [Zuku96] und [Jasp97] beschrieben; hier sind durch die detektierbaren Ereignisse (z.B. über die gerade gestartete Ausführung einer Aktivität) mehr Transitionen abbildbar und damit mehr Ausführungsanweisungen implementierbar. Durch die Nähe der Verteilten Transitionssystemen zu Stellen/Transitions-Netzen besteht in WFMS, die solche Netze durch Beschreibung der Workflow-Ausführung einsetzen, die Möglichkeit der Implementierung. Alle drei Ansätze laufen eine mehr oder weniger vollständige Abbildung der Transitionen auf die im WFMS verwendete Notation heraus; eine weitergehende Nutzung der Verteilten Transitionssysteme besteht nicht, dafür sind keine Veränderung im Ziel-WFMS notwendig.

Ein erweiterbares WFMS wie MOBILE ermöglicht eine andere Form der Implementierung. MOBILE sieht für jeden Aspekt eines Workflows einen eigenen Server vor, der die zur Workflow-Ausführung notwendige Funktion erbringt. Es gibt einen eigenen Kontrollfluß-Server, der aus Sicht der anderen Komponenten kontrollfluß-spezifische Entscheidungen trifft und diese dem zentralen Kommunikationsmodul bereitstellt. Der Kontrollfluß-Server ist die Stelle in der MOBILE-Architektur, in der neue Kontrollflußkonstrukte implementiert werden.

Bedeutung des Verteilten Transitionssystems für die Implementierung

Ausführungsanweisungen werden durch Verteilte Transitionssysteme in der Form beschrieben, daß das Verhalten des Transitionssystems dem der Ausführungsanweisung entspricht und daher in einer Menge zulässiger Zustände der kontrollierten Workflow-Instanzen besteht. Die Transitionen beschreiben die Zustandsübergänge und zwar bedingt durch die Art der Entstehung in einer Weise, die als „Weg“ durch die Menge der zulässigen Zustände verstanden werden kann. Die Simulationsumgebung dient dazu, die Zustandsübergänge im Transitionssystem durch Token zu repräsentieren, deren Produktion und Konsumierung eine Funktion zugeordnet und aus der Anwesenheit bzw. Abwesenheit Schlüsse gezogen werden können. Können zwei oder mehr Transitionen schalten, erlaubt die Ausführungsanweisung unterschiedliche

Möglichkeiten zur Fortsetzung der Workflow-Ausführung. Das WFMS führt diese Entscheidung herbei und wählt dann die Transition aus, die dieser Entscheidung entspricht.

Kommunikation zwischen Kontrollfluß-Server und Koordinationsmodul

Das Koordinationsmodul stellt an den Kontrollfluß-Server eine Anfrage bezüglich der weiteren Möglichkeiten zur Workflow-Ausführung aus Sicht des Kontrollflusses. Aus den angebotenen Möglichkeiten wird eine Auswahl getroffen und anschließend dem Kontrollfluß-Server mitgeteilt, der durch die zugehörige Transition diesen neuen Zustand herstellt (vgl. [Bußl97, S. 55ff.]). Das Verteilten Transitionssystem als Subsystem des Kontrollfluß-Servers liefert zwei Informationen:

- **Menge aller möglichen Veränderungen**. Diese Frage zielt darauf ab, aus Kontrollfluß-Sicht zulässige Fortsetzungen der Workflow-Ausführung zu erfahren. Die Antwort besteht aus einer Liste mit Transitionen, die jede für sich eine mögliche Fortsetzung und das dann zu erwartende Ergebnis beinhalten. Liefert die Anfrage beispielsweise eine Transition, die auf der rechten Seite die Token A.running und B.running enthält, bedeutet das für das WFMS die Erlaubnis, aus Kontrollfluß-Sicht die Workflow-Instanz A und B zu starten.

- **Zulässigkeit einer bestimmten Veränderung**. Die Anfrage des WFMS kann so konkret sein, daß die Zulässigkeit einer ganz bestimmten Veränderung überprüft werden soll. Das WFMS teilt dem Kontrollfluß-Server dazu mit, welche Zustandsänderung geplant ist. Die Zustandsänderung ist zulässig, wenn es im Transitionssystem eine korrespondierende Transition gibt, die diese Zustandsänderung herbeiführt.

Die Entscheidung für eine der möglichen Zustandsübergänge oder die Durchführung der zuvor nachgefragten Zustandsänderung fällt dann innerhalb des Koordinationsmoduls unter Abstimmung mit den anderen Aspekten, die bei der Ausführung eines Workflows eine Rolle spielen. Wenn die Änderung in der Ausführungsumgebung vollzogen ist, muß diese dem Kontrollfluß-Server mitgeteilt werden, damit dieser den neuen Zustand abbilden kann.

Mitteilung über erfolgte Veränderungen

Sobald sich ein Zustand einer Workflow-Instanz geändert, bildet das Transitionssystems diesen Zustand (noch) nicht korrekt ab. Daher teilt das Kommunikationsmodul dem Kontrollfluß-Server diese Änderung mit und gibt damit die Erlaubnis, die zugeordnete Transition zu vollziehen. Damit nimmt das Transitionssystem einen neuen Zustand ein, der dann im nächsten Durchgang dazu führt, eine Anfrage nach den Möglichkeiten einer Fortsetzung der Workflow-Ausführung zu beantworten. Tritt der Fall auf, daß die Mitteilung über eine erfolgte Veränderung nicht mit einer Transition nachvollzogen werden kann, ist das ein Hinweis auf einen Fehler in der Ausführung und eine Abweichung vom Workflow-Schema. Die Behandlung dieser Situation wird an dieser Stelle nicht weiter diskutiert.

5 Abgrenzung von anderen Arbeiten

Die systematische Entwicklung von Ausführungsanweisungen für erweiterbare Workflow-Sprachen wurde bislang nicht behandelt. Allerdings gibt es zu Teilbereichen bereits Vorarbeiten und andere Ansätze in der inzwischen umfangreichen Literatur. Da eine Auswahl notwendig ist, werden ausschließlich Arbeiten neueren Datums vorgestellt. Die Diskussion erfolgt entlang der Themenbereiche des Technischen Berichts:

- Kontrollflußprimitive, Kontrollflußkonstrukte, Ausführungsanweisungen
- Rahmenwerke zur Spezifikation von Ausdrucksmitteln
- Spezifikation von Task-Dependencies
- Simulation der Workflow-Ausführung

Einige der referenzierten Arbeiten werden mehrfach genannt, da sie unterschiedliche Themenbereiche ansprechen. Die Auswahl beinhaltet keine Wertung.

5.1 Kontrollflußprimitive, Kontrollflußkonstrukte, Ausführungsanweisungen

[Jabl94] präsentiert eine umfassende Sammlung von Ausführungsanweisungen zur Beschreibung des Verhaltensaspekts von Workflows; basierend auf einem differenzierten Ausführungsmodell für Workflow-Instanzen. Angegeben wird sowohl eine Notation für die Ausführungsanweisungen als auch die Semantik unter Rückgriff auf Zustandswechsel der kontrollierten Workflow-Instanzen; letztgenannte Beschreibung verdeutlicht die *Funktionsweise* der Ausführungsanweisungen. Um die *Auswirkung* einer Ausführungsanweisung auf die Ausführung von Workflows zu beschreiben, gibt [Jabl96] Petri-Netze an. Darin treten Workflows als Transitionen auf, deren Ausführung von Markenbelegungen umgebender Stellen abhängig ist. Die Kombination von interner (in [Jabl94]) und externer Betrachtungsweise (in [Jabl96]) liefert ein recht genaues Bild der vorgestellten Ausführungsanweisungen. Die Komposition von Ausführungsanweisungen und die Simulation der resultierenden Workflow-Ausführung werden nicht angesprochen. [Sieg95] beschreibt eine prototypische Realisierung einer Ablaufsteuerung für die Ausführungsanweisungen in MOBILE.

[Krad97] schlägt vor, das Verhalten von Workflows durch individuell zu definierende, aus der jeweiligen Anwendung motivierte Bedingungen an Zustandsübergängen im Ausführungsmodell der Workflow-Instanzen festzulegen. Da als sichtbare Zustände im Ausführungsmodell nur initiated, executing und completed vorgesehen sind, gelten die Einschränkungen aus Abschnitt 3.2.3.1. Als Begründung wird die Autonomie der *Processing Entities* angeführt, die nach dem Start der Workflow-Ausführung keine Einflußnahme mehr zulassen würden. An anwendungsunabhängigen Beschreibungsmitteln werden Parallel Block und einen Iteration Block angeboten, die der bereits in [Jabl94] eingeführten Ausführungsanweisung repetition entsprechen. Zur Beschreibung der Funktion beider Konstrukte dient ein Zustandsübergangs-Diagramm in UML-Notation.

[Sadi97] empfiehlt gerichtete Graphen als konzeptuelle Modelle von Workflows; bestehend aus den Knotentypen *task*, *condition* und *synchronizer*, verbunden durch gerichtete Kanten zur Festlegung des Kontrollflusses. Struktur und Verhalten einer *task* werden nicht präzisiert, ein Ausführungsmodell fehlt. Aus der grafischen Darstellung geht hervor, daß von einer *task* beliebig viele Kanten aus- und eingehen können, gleiches gilt für die Elemente *condition* und *synchronizer*. Da jede *task* eine ununterbrechbar ausgeführte Aufgabe repräsentiert und Zwischenzustände nicht sichtbar sind, entstehen Ablaufgraphen, die nur eine sequentielle, alternative oder mögliche parallele Ausführung von Aufgaben zulassen. Fragmente aus Ablaufgraphen werden vorgestellt, die unter Annahme eines nicht dokumentierten Ausführungsmodells zu verschiedene Formen der *task*-Ausführung führen. Eine formale Definition der Semantik benutzter Konstrukte fehlt; es gibt jedoch eine Reihe informeller Anweisungen zum Aufbau von Ablaufgraphen. Gemäß der Einteilung aus Abschnitt 2.3 arbeitet der Ansatz mit Kontrollflußprimitiven, aus denen der Entwickler für jeden Ablaufgraphen einzeln unter Zuhilfenahme der *synchronizer* und *conditions* die gewünschte *task*-Ausführung erzeugt. Die weitere Verwendung der „konzeptuellen Modelle“ wird nicht diskutiert.

5.2 Rahmenwerke zur Spezifikation von Ausdrucksmitteln

ACTA ist ein in [Chry92] vorgestelltes Framework zur Beschreibung erweiterter Transaktionsmodelle. Mit dem Ziel, Eigenschaften und Verhalten von Transaktionen unterschiedlicher Transaktionsmodelle zu vergleichen, werden Abhängigkeiten zwischen Transaktionen (*inter-task dependencies*) und Auswirkung der Transaktionen auf Datenobjekte in einer deklarativen Notation beschrieben. ACTA liefert für jedes Transaktionsmodell einen Satz charakteristischer Axiome, die invariante Eigenschaften der Transaktionen dieses Modells ausdrücken. Auf dem gleichen Weg, nämlich über Constraints bezüglich des Auftretens von signifikanten Ereignissen in einer Historie, werden Korrektheitskriterien innerhalb der Transaktionsmodelle formuliert und bewiesen. Was als signifikantes Ereignis gilt, ist nicht festgelegt, sondern bereits Teil der Spezifikation und umfaßt bei klassischen ACID-Transaktionen start-, commit- und abort-Ereignisse, bei erweiterten Transaktionen entsprechend mehr, wie [Chry94] an ausführlich dokumentierte Beispielen zeigt.

Interessant an ACTA ist die axiomatische Spezifikation der transaktionsmodell-spezifischen Eigenschaften. Es ist zu untersuchen, ob dies auf Ausführungsanweisungen übertragen werden kann. Das gleiche Ziel wie ACTA hat auch die folgende Arbeit, nur daß TANGO statt der deklarativen Sichtweise eine operational orientierte Beschreibung von Transaktionen vorschlägt.

In [Band94] wird der Kontrollfluß zwischen verketteten Applikationen durch ein Petri-Netz (PN) modelliert. Dessen Auswertung liefert Aussagen über den Applikationsverbund; insbesondere über das Verhalten in Abhängigkeit von Zustand und Ausführungsergebnis einzelner Applikationen. Das Modell des Applikationsverbunds sieht für jede Applikation ein transitionsberandetes PN-Fragment vor; Stellen innerhalb des PN-Fragment sind mit applikationseigenen Zuständen oder Zuständen der Aufgabe beschriftet, die durch die Applikation unterstützt wird. Transitionen verbinden sowohl Stellen, die Zustände *einer* Applikation bezeichnen, als auch Stellen, die zu *unterschiedlichen* Applikationen gehören; Transitionen der letzt-

genannten Art bilden die Verkettung der Applikationen nach. Als primäre Verwendung für die TANGO genannte, semi-formale Notation wird die Beschreibung von Transaktionsfolgen in erweiterten Transaktionsmodellen vorgeschlagen. Die zuvor individuell modellierten Applikationen bekommen dazu transaktionsspezifische Zustände (startable, commit, abort) und sollen somit transaktionstypisches Verhalten wiedergeben. Als Nachweis für die Ausdrucksfähigkeit von TANGO werden in [Chry92] benannte Abhängigkeiten zweier Transaktionen durch die TANGO-eigene grafische Notation dargestellt.

Die Absicht hinter TANGO entspricht Zielen des eigenen Ansatzes (Hilfsmittel für Entwickler, ausführbare Spezifikation), die Herangehensweise (Modellbildung, Simulation/Test vor der Implementierung) ist vergleichbar, und die Unterschiede von PN und Verteilten Transitionssystemen sind bei der hier beschriebenen Verwendung gering. Allerdings dient TANGO nicht dazu, anwendungsunabhängige und wiederverwendbare Ausführungsanweisungen zu beschreiben, vielmehr wird durch Kontrollflußprimitive für jede Anwendungssituation der Kontrollfluß individuell festgelegt. Zudem abstrahiert TANGO so weit von Details der späteren Implementierung bzw. von den tatsächlichen Eigenschaften der Transaktionen, daß Teile der Spezifikation nicht umsetzbar sind. Beispielsweise wird das Commit einer Transaktion im Modell dadurch kontrolliert, daß die repräsentierende Transition eine zusätzliche Eingangsstelle bekommt. Nach der Semantik von PN verhindert dies tatsächlich den Schaltvorgang und spiegelt damit eine Verzögerung des Commits vor. In der Realität ist das Commit nicht kontrollierbar, sofern nicht Datenbank-interne Mechanismen verändert werden. Um derartige Fehler in der Modellierung zu vermeiden, wird im eigenen Ansatz zwischen dem Verlassen eines Zustands und dem Erreichen eines neues Zustands unterschieden (siehe Abschnitt 3.2.2). Diese Differenzierung gestattet, z.B. das Verlassen des Zustands running zu verhindern und nicht das Erreichen des Folgezustands committed. Trotz dieser Unterschiede ist TANGO ein interessanter Ansatz; eine Adaption der grafischen Notation wird erwogen.

5.3 Spezifikation von Task-Dependencies

[Sing97a] betrachtet Workflows als eine Gruppe von Aktivitäten, zwischen denen start/commit/abort-Abhängigkeiten bestehen. Ziel der Arbeit ist, die Abhängigkeiten für eine gegebene Menge von Aktivitäten zunächst formal zu beschreiben, um anschließend einen Scheduling-Algorithmus abzuleiten (siehe auch [Sing97b]). Eine Implementierung des Scheduling-Algorithmus würde von den Aktivitäten ausgehende Anforderungen für ein commit oder abort so akzeptieren oder zurückweisen, daß die resultierende Ausführung den zuvor spezifizierten Abhängigkeiten genügt. Durch Einführung eine Ereignis-Algebra und einer Menge von Residuations-Regeln wird ein zunächst komplexer Ausdruck aus allen Abhängigkeiten schrittweise vereinfacht. Als Ereignisse gelten alle Anforderung der kontrollierten Aktivitäten zu den oben genannten Operationen. Für den Kontext dieses Berichts ist der Ansatz dort interessant, wo aus den deklarierten Abhängigkeiten Schlüsse für die Zulassung oder Verhinderung von (Workflow-)Operationen gezogen werden. Dieser Punkt ist zur Zeit noch nicht ausreichend behandelt; eine Auseinandersetzung mit dem Ansatz in [Sing97a] scheint sinnvoll.

5.4 Simulation der Workflow-Ausführung

Eine Reihe von Arbeiten beschäftigt sich mit der *Simulation* der Workflow-Ausführung in einem fiktiven oder nicht benannten WFMS. In diese Kategorie fallen viele Ansätze, die z.B. PN und ihre Derivate einsetzen, um einzelne Aspekte von Workflows nachzubilden. Häufig wird der Verhaltensaspekt herausgegriffen; die Simulation findet in einer für PN entwickelten Ausführungsumgebung statt und liefert dementsprechend Aussagen über ein Modell, nicht über Workflows. Nicht alle Arbeiten stellen dies klar heraus, es wird vielmehr argumentiert, PN sind eine geeignete *Repräsentation* für Workflows bzw. Workflow-Instanzen in WFMS und nicht nur die Darstellung für ein *Modell* eines Workflows. Genaugenommen existiert unter diesen Umständen noch kein Workflow, sondern das PN-basierte Modell beschreibt ein Artefakt, das Teile eines zukünftigen Workflow-Schemas wiedergibt, und eine Workflow-Instanz stellt sich als Markenbelegung des PN dar. Wenn der notwendige Schritt vom PN-basierten Modell des „Workflow-Schemas“ zur Erstellung eines Workflow-Schemas mittels Workflow-Sprache unterbleibt, sind die Ergebnisse nur bedingt übertragbar und müssen erst in ihrer Bedeutung für Workflows (nicht ihrer Modelle) interpretiert werden.

Es gibt weiterhin Arbeiten, in denen die Simulationsumgebung so erweitert wurde, das z.B. externe Applikationen aufgerufen werden, Benutzer verwaltet und workflow-relevante Daten transportiert werden. Damit geht die Funktionalität über die Simulation hinaus. Zuvor erstellte Modelle von Workflow-Schemata werden mit Workflow-Schemata in dem Sinn gleichgesetzt, daß die Ausführungsumgebung die Realwelt betreffende Aktionen auf der Grundlage dieser Modelle von Workflow-Schemata auslöst, etwa ein externes Anwendungsprogramm startet und vom Benutzer die Erledigung einer zugewiesenen Aufgabe einfordert.

Ein Beispiel für diesen Ansatz findet sich in [Aals98]. Dort werden *tasks* (elementare Aufgaben) als transitionsberandetes PN-Fragment beschrieben und durch *routing constructs* verbunden, woraus ein PN mit jeweils genau einer Eingangs- und Ausgangsstelle entsteht, das als *WF-Net* bezeichnet wird und Teile eines Workflow-Schemas darstellt. Routing constructs sind ebenfalls PN-Fragmente, die den von der WfMC favorisierten Kontrollflußkonstrukten entsprechen. Zusätzlich eingefügte Transitionen zwischen Stellen aus unterschiedlichen Bereichen des PN werden als *trigger* bezeichnet und sollen die Reaktion auf externe Ereignisse erlauben. Im resultierenden WF-Net sind die verwendeten Konstrukte nicht mehr erkennbar, die Lesbarkeit ist gering. Interessant sind die in [Aals96] und [Aals97a] vorgestellten Methoden, aus der Struktur der WF-Nets Aussagen über ihr Verhalten abzuleiten, sowie die formal definierten Transformationsregeln zur Verfeinerung und Modifikation bestehender WF-Nets. Ergebnisse der letztgenannten Arbeiten sind laut [Aals98] in der Analysekomponente WOFLAN [Aals97b] implementiert.

Auch [Adam98] modelliert das Verhalten von *tasks* (elementare Workflows) und verschiedenen Abhängigkeiten zwischen den Talks, dies allerdings recht differenziert. Es werden *Control Flow*, *Value* und *Temporal Dependencies* unterschieden, und für jede Art von Beziehung ist ein Petri-Netz-Fragment angegeben. Die Beziehungen können zwischen allen Zuständen hergestellt werden, die für tasks vorgesehen sind (genannt werden *ready*, *running*, *failed* und *committed*).

Der wesentliche Unterschied zwischen den beiden Arbeiten besteht in der Behandlung des Verhaltensaspekts. Während [Aals98] sog. *Kontrollflußkonstrukte* nachbildet, die zwei oder mehr *tasks* verbinden, verwendet [Adam98] sog. *Kontrollflußprimitive*, die nur jeweils einzelne Zustände der *tasks* in Beziehung setzen. Das generelle Vorgehen deckt sich jedoch mit dem in [Aals98] und [Adam98], leichte Abweichungen betreffen lediglich die modellierten Aspekte. Weder in [Adam98] noch in [Aals98] wird der Einfluß externer Anwendungsprogramme auf die Ausführung von Workflows berücksichtigt, obgleich dies z.B. für Fragen der Terminierung durchaus von Bedeutung ist.

6 Offene Probleme und weitere Aktivitäten

Bei der Erstellung des Berichts zeigte sich an einigen Stellen noch Klärungsbedarf; einige Punkte werden hier genannt:

- Ein vollständiger Entwicklungsvorgang für Ausführungsanweisungen beginnt bei der Anforderungsdefinition und endet bei der Implementierung in einem WFMS. Der Bericht behandelt einen Teilabschnitt des Vorgangs: die Bildung eines Modells der Ausführungsanweisung und die Untersuchungen von Eigenschaften dieses Modells. Es ist daher notwendig, erstens die Modellbildung noch stärker an den Vorgaben eines konkreten Entwicklungsvorhabens zu orientieren und zweitens die für das Modell festgestellten Eigenschaften in ihrer Bedeutung für die Workflow-Management-Anwendung zu interpretieren.

- Die Beschreibung von Ausführungsanweisungen durch Verteilte Transitionssysteme fokussiert auf die Zustände der kontrollierten Workflow-Instanzen. Das gewählte Abstraktionsniveau erlaubt die Darstellung der Zustände als Token; den Eigenschaften der Zustandsübergänge in Workflow-Instanzen werden die Transitionen nicht gerecht. Beispielsweise kann ein Ereignis, das die Transition auslöst, nicht von einem Zustand, der bereits vorliegt, unterschieden werden (beides wird bisher durch Token gleichen Typs repräsentiert).

- Die bisher praktizierte Abbildung von Zuständen einer Workflow-Instanz auf Token und von Zustandsänderungen auf Transitionen ist naheliegend und genügt für eine erste Charakterisierung der Ausführungsanweisung. Für präzisere Aussagen zur Implementierbarkeit und dem Verhalten der Ausführungsanweisung kann eine „aufwendigere“ Modellierung von Workflow-Instanzen und Ausführungsanweisungen erforderlich werden.

- Die Beschreibung der Ausführungsformen spart bisher Situationen aus, die aus Fehlern in der Workflow-Ausführung folgen. Transitionen galten zudem als stets ausführbar, was einen jederzeitigen Vollzug der zugrunde liegenden Zustandsänderung im Workflow impliziert. Diese Annahme idealisiert die Verhältnisse: Zustandsänderungen können verzögert werden, und für Transitionen notwendige Ereignisse können ausbleiben. In Hinblick auf eine Implementierung von Ausführungsanweisungen wird die Notwendigkeit eines Protokolls deutlich, das die Erkennung und Behandlung dieser Situationen festlegt.

- Bei der Vorstellung des Funktionsmodells kam eine Form der Workflow-Ausführung zur Sprache, die dem WFMS eine neuartige — nicht mehr dominante Rolle — zuweist. Der Anwender entscheidet selbst über die Ausführungsfolge der angebotenen Workflows; das WFMS stellt nur sicher, daß zuvor festgelegte Randbedingungen nicht verletzt werden. Ausführungsformen für diese Form der Workflow-Ausführung wurden nicht weiter behandelt, obgleich sie die Unterstützung semi-strukturierte Arbeitsvorgänge ermöglichen. Es scheint sinnvoll, Modelle für derartige Ausführungsanweisungen näher zu untersuchen.

An die Klärung dieser Punkte schließen sich unter anderem folgende Arbeiten an:

- Externe Applikationen bestimmen das Verhalten der Workflows, aus denen sie aufgerufen werden. Die vorgestellte Simulation der Workflow-Ausführung muß daher auf externe Applikationen ausgedehnt werden, um die Auswirkungen eines nicht beendeten oder fehlerhaft ausgeführten Anwendungsprogramms auf die Workflows zu berücksichtigen.

- Die vorhandene Simulationsumgebung erlaubt keine Untersuchung der dynamischen Eigenschaften eines Verteilten Transitionssystems. Zur Validierung der Eigenschaften und Kriterien sind Erreichbarkeitsanalysen erforderlich. Da eine Erweiterung der Simulationsumgebung nicht vorgesehen ist, wird statt dessen der Weg über die Abbildung auf Petri-Netze gewählt, für die ausreichend Analysewerkzeuge zur Verfügung stehen.

Ausführungsanweisungen sind ein Bestandteil von Workflow-Schemata, ihre Entwicklung ist eine Teilaufgabe bei der Erstellung von Workflow-Management-Anwendungen. Zwischen beiden Entwicklungstätigkeiten bestehen Abhängigkeiten, die es herauszuarbeiten gilt. Dieser Punkt hat die größte Bedeutung, zum einen weil bisher keine Arbeiten auf diesem Gebiet existieren, zum anderen weil es für die Entwicklung Workflow-Management-Anwendungen einen signifikanten Beitrag bedeutet.

7 Zusammenfassung

Durch die Verfügbarkeit erweiterbarer Workflow-Sprachen entsteht eine neuartige Aufgabe bei der Entwicklung von Workflow-Management-Anwendung: die Einführung anwendungsspezifischer Sprachmittel. Am Beispiel von Ausführungsanweisungen zeigt dieser Bericht einen Ansatz, mit formalen Mitteln den Entwurf neuer Ausdrucksmittel zu begleiten.

Ausführungsanweisungen stellen anders als Kontrollflußprimitive und Kontrollflußkonstrukte eine ausdrucksstarke und komfortable Möglichkeit dar, den Verhaltensaspekt eines Workflows zu beschreiben. Innerhalb eines Workflow-Schemas wird die Ausführungsfolge der Subworkflows mit Ausführungsanweisungen festgelegt. Um neue Ausführungsanweisungen systematisch entwickeln zu können, präsentiert der Bericht zunächst ein konzeptionelles Funktionsmodell für Ausführungsanweisungen. Das Funktionsmodell erklärt die koordinierte Ausführung mehrerer Workflows durch kontrollierte Zustandsübergänge der zugehörigen Workflow-Instanzen. Umgekehrt bestimmt das Ausführungsmodell der Workflow-Instanzen maßgeblich, welche Ausführungsfolgen von Workflows überhaupt möglich sind und vom WFMS herbeigeführt werden können. Um diesen Zusammenhang herauszuarbeiten, beschreibt der Bericht ausführlich verschiedene Ausführungsmodelle und leitet aus den vorhandenen Zuständen Aussagen über die Möglichkeiten der Workflow-Ausführung ab.

Um Ausführungsanweisungen nicht nur präzise zu beschreiben, sondern auch auswerten, analysieren und simulieren zu können, kam nicht jeder Formalismus in Frage. Deklarative Ansätze zur Beschreibung der Semantik sind zwar ausdrucksstark und werden auch in anderen Arbeiten auf diesem Gebiet eingesetzt, decken aber die oben genannten Anforderungen nur bezüglich der Präzision, nicht aber hinsichtlich Auswertung, Analyse und Simulation ab. Als guter Kompromiß erwiesen sich Verteilte Transitionssysteme, die in ihrer absichtlich allgemein gehaltenen Konzeption workflow-spezifische Aspekte ausdrücken können, wenn ihre Verwendung an bestimmte Regeln geknüpft wird. Mit Hilfe dieser Regeln und weiteren Kriterien gelingt es, Verteilte Transitionssysteme so zu erstellen, daß ihr Verhalten den interessierenden Punkt bei Ausführungsanweisungen entspricht. Weite Teile des Berichts behandeln diese Verwendung und demonstrieren die Möglichkeiten an einer Reihe von Ausführungsanweisungen. Die vorgestellten Ausführungsanweisungen entsprechen zum Teil dem, was in den meisten WFMS angeboten wird, zum Teil gehen sie auch deutlich darüber hinaus.

Insgesamt stellt der Bericht einen ersten Schritt dar, um zumindest Ausführungsanweisungen so weit zu entwickeln, daß Voraussagen über die von ihnen erzeugte Workflow-Ausführung möglich werden. Die Ergebnisse richten sich vorwiegend an die Entwickler von Workflow-Management-Anwendungen, die im Vorfeld der Implementierung verschiedene Möglichkeiten zur Workflow-Ausführung untersuchen und vergleichen wollen. Der Übergang der modellierten Ausführungsformen in fertig implementierte Sprachmittel einer Workflow-Sprache steht noch aus, ist aber durch die Ergebnisse des Berichts vorbereitet worden.

Literatur

Aals96 van der Aalst W.M.P.: “Structural Characterizations of Sound Workflow Nets“, Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996.

Aals97a van der Aalst W.M.P.: “Verification of Workflow Nets“, in: Azema P., Balbo G. (eds.), *Application and Theory of Petri Nets 1997*, LNCS 1248, pp. 407–426. Springer-Verlag, Berlin, 1997.

Aals97b van der Aalst W.M.P., Hauschildt D., Verbeek H.M.W.: “A Petri-Net-based Tool to Analyse Workflows“, in: Farwer B., Moldt D, Stehr M.O. (eds.), *Proceedings of Petri Nets in System Engineering* (PNSE ’97), pp. 78–90, Universität Hamburg, September 1997.

Aals98 van der Aalst W.M.P.: “The Application of Petri Nets to Workflow Management“, to appear in: *The Journal of Circuits, Systems and Computers*, 1998.

Adam98 Adam N.R., Atluri V., Huang W.K.: „Modeling and Analysis of Workflows Using Petri Nets“, in: Journal of Intelligent Information Systems, Special Issue on Workflow and Process anagement, Volume 10, Number 2, March 1998.

Alle83 Allen J.F.: “Maintaining Knowledge about Temporal Intervals”, in: *Communications of the ACM*, Vol. 26, No. 11. November 1983.

Alon96 Alonso G., Agrawal D., El Abbadi A., Kamath M., Guenthoer R., Mohan C.: „Advanced Transaction Models in Workflow Contexts“, in: Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA, 1996.

Ambe95 Amberg M.: „Ableitung von Wf-Spezifikationen für Workflow-Managementsysteme aus Geschäftsprozeßmodellen“, Fachtagung der GI-Fachgruppe 5.2.1 MobIS, Bamberg 6.10.95.

Baet90 Baeten J., Weijland W.: „Process Algebra“, in: Cambrigde Tracts in Theoretical Computer Science, Number 18, Cambrige University Press, 1990.

Band94 Bandinelli S., Ceri S., Felder M.: “TANGO: A Notation for Describing Advanced Transaction Models”, Technical Report No. 19, ESPRIT-III Project GOODSTEP, March 1994.

Barb83 Barber G.: „Supporting Organizational Problem Solving with a Work Station“, in: *ACM Transactions on Office Information Systems*, Vol. 1, No. 1, pp. 45–67, January 1983.

Barb94 Barbara D., Mehrotra S., Rusinkiewicz M.: “INCAS: A Computation Model for Dynamic Workflows in Autonomous Distributed Environments”, Matsushita Information Technology Laboratory, Princeton, May 2, 1994.

Besl87 Beslmüller E.: “Office Modelling based on Petri Nets“, in: ESPRIT ‘88. Putting the Technology to Use. Proceedings of the 5th Annual ESPRIT Conference, p. 2 vol. xxiii+1759, pp. 977-987, 1987.

Böhm96 Böhm M., Meyer-Wegener K., Schulze W.: „Formale Beschreibung und Transformation von Realisierungsalternativen für Geschäftsprozesse“, in [Jabl96b].

Böhm97a Böhm M.: „Modellierung von Workflows“, Kapitel 8 in [Jabl97].

Böhm97b Böhm M.: „Entwurf und Bewertung von Varianten zur Geschäftsprozeß-unterstützung mit Workflow-Management-Systemen“, in: *Proceedings zum Workshop „Arbeitsplatzrechner-Integration zur Prozeßverbesserung“*, GI-Jahrestagung 23.9.97, Aachen, erschienen in: Softwaretechnik-Trends, Band 17, Heft 3, August 1997.

Brac84 Bracchi G., Pernici B.: “The Design Requirements of Office Systems“, in: *ACM Transactions on Office Information Systems*, Vol. 2, No. 2, pp. 151–170, April 1984.

Brei93 Breitbart Y., Deacon A., Schek H.-J., Sheth A., Weikum G.: „Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows“, in: *SIGMOD*

RECORD, Vol. 22, No. 3, September 1993.

Bußl97 Bußler C.: „Organisationsverwaltung in Workflow-Management-Systemen“, Dissertation, IMMD VI, Universität Erlangen-Nürnberg, 1997.

Butt91 Butterworth P., Otis A., Stein J.: “The GemStone Object Database Management System“, in: *Communications of the ACM*, Vol. 34, No. 10, pp. 64–77, October 1991.

Cap94 Cap C.H.: “A Calculus of Distributed and Parallel Processes — An Approach Using Linear Logic and Algebraic Specification”, Habilitationsschrift am Institut für Informatik der Universität Zürich, 1994, erscheint im Teubner-Verlag. Verfügbar auch unter: `http://www.tec.informatik.uni-rostock.de/IuK/publications/survey-for-www.ps`

Casa95 Casati F., Ceri S., Pernici B., Pozzi G.: “Deriving Active Rules for WorkFlow Enactment”, Dipartimento di Elettronica e Informazione, Politecnico di Milano, October 25, 1995.

Chry92 Chrysanthis P., Ramamritham, K.: “ACTA: The SAGA Continues“, in: *Database Transaction Models Advanced Applications*, Chapter 10, Elmagarmid A.K. (Ed.), Morgan Kaufmann, February 1992.

Chry94 Chrysanthis P., Ramamritham, K.: “Synthesis of Extended Transaction Models using ACTA“, Technical Report, University of Massachusetts, USA, 1994.

Crof84 Croft B., Lefkowitz L.: “Task Support In an Office System“ in: *ACM Transactions on Office Infortmation Systems*, Vol. 2, No. 3, pp. 197–212, July 1984.

Curt92 Curtis B., Kellner M.I., Over J.: “Process Modeling“, in: *Communications of the ACM*, Vol. 35, No. 9, pp. 75–90, 1992.

Deit96 Deiters W., Herrmann T., Löffeler T.: „Identifikation, Klassifikation und Unterstützung semistrukturierter Teilprozesse in prozeßorientierten Telekooperationssystemen“, in: Krcmar H., Lewe H., Schwabe, G. (Hrsg.), Herausforderung Telekooperation – Einsatzerfahrungen und Lösungsansätze für ökonomische und ökologische, technische und soziale Fragen unserer Gesellschaft, Fachtagung DCSCW ‘96, Stuttgart-Hohenheim, 30.9.96 – 2.10.96, Berlin u.a.: Springer, S. 261–274, 1996.

Dere76 Deremer F.L., Kron H.: “Programming-in-the-large versus programming-in-the-small“, in: *Informatik-Fachberichte Nr. 1*, Springer-Verlag, Heidelberg und IEEE Transactions of Software Engineering, Vol. SE-2, No. 2, pp. 80–86, June 1976.

Dink94 Dinkhoff G., Gruhn V., Saalmann A., Zielonka M.: “Business Process Modeling in the Workflow Management Environment LEU“, in: *Business Modelling and Reengineering, Proceedings of the 13th International Conference on the Entitiy-Relationship-Approach* , Manchester, UK, LNCS 881, Springer-Verlag, 1994.

Eder94 Eder J., Groiss H., Nekvasil H.: “A Workflow System Based on Active Databases”, in: *Proceedings Connectivity-94: Workflow Management - Challenges, Paradigms and Products* , Oldenbourg Verlag, Wien, München, 1994.

Geor94 Georgakopoulos D., Hornick M.F.: „A Framework for enforceable Specification of extended Transaction Models and Transactional Workflows“, in: *International Journal of Intelligent and Coopeartive Information Systems*, Vol. 3, No. 3, pp. 225–253, 1994.

Gepp96 Geppert A., Kradolfer M., Tombros D.: “EvE, an Event Engine for Workflow Enactment in Distributed Environments“, Technical Report 96.05, Department of Computer Science, University of Zurich, May 1996.

Gepp97 Geppert A., Tombros D.: „Ereignisgesteuerte Workflow-Ausführung in verteilten Umgebungen“, Kapitel 15.3.2 in [Jabl97].

Gruh91 Gruhn V.: “Validation and Verification of Software Process Models“, Dissertation, Universität Dortmund, erschienen als Technischer Bericht Nr. 394/91, Dortmund 1991.

Hage98 Hagen C., Alonso G.: „Flexible Exception Handling in the OPERA Process Support System“, in: 18th International Conference on Distributed Computing Systems (ICDCS 98), Amsterdam, The Netherlands, May 1998.

Hare87 Harel D., Pnueli A., Schmidt J.P., Sherman R.: “On the Formal Semantics of Statecharts“, in: *Proceedings of the Symposium on Logic in Computer Science*, June 22–25, pp. 54–64, 1987.

Hare88 Harel D.: “On Visual Formalisms“, in: *Communications of the ACM*, Vol. 31, No. 5, pp. 514–530, May 1988.

Hare90 Harel D. et al.: “STATEMATE: A Working Environment for the Development of Complex Reactive Systems“, in: *IEEE Transactions of Software Engineering*, Vol 16., No. 4, April 1990.

Hase87 Hasenkamp, U.: „Konzipierung eines Bürovorgangssystems“, Habilitationsschrift an der Wirtschafts- und Sozialwissenschaftlichen Fakultät der Universität zu Köln, 1987.

Herr98 Herrmann Th., Scheer A.-W., Weber H. (Hrsg.): „Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Management Systemen, Band I: Von der Erhebung zum Sollkonzept“, Physika-Verlag, 1998.

Hoar85 Hoare C.: „Communication Sequential Processes“, Prentice-Hall International, 1985.

Jabl94 Jablonski S.: “MOBILE: A Modular Workflow Model and Architecture”, in: *Proceedings of the 4th International Conference on Dynamic Modeling and Information Systems*, Noordwijkerhout, Netherlands, September 1994.

Jabl96a Jablonski S., Bussler C.: “Workflow-Management: Modeling, Concepts, Architecture and Implementation”, International Thomson Computer Press, 1996.

Jabl96b Jablonski S., Groiss R., Kaschek R., Liebhart W. (Hrsg.): „Geschäftsprozeßmodellierung und Workflow-Management-Systeme“, Proceedings-Reihe der Informatik ’96, Band 2, Klagenfurt, 1996.

Jabl97 Jablonski S., Böhm M., Schulze W. (Hrsg.): „Workflow-Management: Entwicklung von Anwendungen und Systemen“, dpunkt-Verlag, Heidelberg, 1997.

Jasp97 Jasper H.: „Workflow-Management-Systeme auf der Basis aktiver Mechanismen“, Kapitel 15.3 in [Jabl97].

Kapp95a Kappel G., Pröll B., Rausch-Schott S., Retschitzegger W.: “Active Object-Oriented Workflow-Management”, in: *28th Hawaii International Conference on System Science*, January 1995.

Kapp95b Kappel G., Lang P., Rausch-Schott S., Retschitzegger W.: „Workflow Management Based on Objects, Rules and Roles“ in: *Bulletin of the Technical Commitee on Data Engineering*, IEEE Computer Society, Vol. 18, No. 1, pp. 11–18, 1995.

Kara96 Karagiannis D., Junginger S., Strobl R.: “Introduction to Business Process Management Systems Concepts“, in: Scholz-Reiter B., Stickel E. (Hrsg.): „Business Process Modelling“, Springer-Verlag, S. 81–108, 1996.

Kasc97 Kaschek R., Paech B.: „Grundlagen der Modellierung von Vorgängen“, Kapitel 6 in [Jabl97].

Kell92 Keller G., Nüttgens M., Scheer A.-W.: „Semantische Prozeßmodellierung auf der Basis Ereignisgesteuerter Prozeßketten (EPK)“, in: Veröffentlichungen des Instituts für Wirtschaftsinformatik Nr, 89, Saarbrücken, 1992.

Kotz97 Kotz-Dittrich A., Muth P., Weikum G., Weißenfels J., Wodtke D.: „Partitionierungs- und Synchronisationstechniken für verteilte Workflow-Management-Systeme“, Kapitel 15.1 in [Jabl97].

Krad97 Kradolfer M., Geppert A.: “Modeling Concepts for Workflow Specification“, Technical Report of the University of Zurich, ifi-97.05, 1997.

Kris94 Krishnakumar N., Sheth A.: “Specifying Multi-system Workflow Applications in METEOR”, Bellcore Technical Memorandum TM-24198, May 1994.

Künz94 Künzli M., Künzli R.: „Konzeption und Implementierung der Sprache Petril“, Semesterarbeit im Fach Informatik, Institut für Informatik, Universität Zürich, 1994.

Laus95 Lausen G. (Hrsg.): Datenbanksysteme in Büro, Technik und Wissenschaft, Springer-Verlag,1995.

Leym92 Leymann F.: “A Meta Model to Support Modelling and Execution of Processes”, in: *Proceedings of the 11th European Meeting on Cybernetics and System Reseach* , Vienna, Austria, April 21–24, pp. 287–294, 1992.

Leym94 Leymann F., Roller D.: “Business Process Management With FlowMark“, in: *Proceedings COMPCON Spring 1994*, San Francisco, CA, February 28-March 4, 1994.

Leym95 Leymann F.: „Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems“ in: G.Lausen (Hrsg.), *Datenbanksysteme in Büro, Technik und Wissenschaft*, S. 51-70, Springer-Verlag, 1995.

Lieb95 Liebhart W.: “The Workflow Activity Description Language WADL“, Technischer Bericht der Universität Klagenfurt, 1995.

Lieb97 Liebhart W.: „Fehler und Ausnahmebehandlung im Workflow-Management“, Kapitel 8.3.4 in [Jabl97].

Lieb98 Liebhart W.: „Fehler- und Ausnahmebehandlung im Workflow Management“, Dissertation, Universität Klagenfurt, 1998.

Loch88 Lochochsky, F.H., Hogg J.S., Weiser, S.P., Mendelzon, A.O.: “Specifying Office Tasks“, in: SIGOIS (Special Interest Group on Office Information Systems) Bulletin, Vol. 9, No. 2&3, April&July 1988 (Proceedings Conference on Office Information Systems), pp. 46-54.

Lore85 Lorenzen P.: „Praktische und theoretische Modalitäten“, in: Grundbegriffe technischer und politischer Natur, 1. Aufl., Suhrkamp, 1985.

Lutz88 Lutze R.: “Customizing Cooperative Office Procedure by Planning“, in: Allen R. (ed.), Proceedings of the Conference on Office Information Systems (Palo Alto, CA), SIGOIS Bulletin 9, pp. 63–77, 1998.

Malo94 Malone T.W., Crowston K.: “The Interdisciplinary Study of Coordination“, in: *ACM Computing Surveys*, Vol. 26, No. 1, pp. 87–119, 1994.

Maze84 Mazer, M., Lochovsky, F.: “Logical Routing Specification in Office Information Systems“, in: *ACM Transactions on Office Information Systems*, Vol. 2, No. 4, pp. 303–330, October 1984.

Mühl97 zur Mühlen M., Rosemann M.: „Diagrammsprachliche Methoden“, Kapitel 10.3.2 in [Jabl97].

Nieh85 Niehuis S., Victor F.: „Modellierung von Büroprozeduren mit Pr/T-Netzen und Prolog“, in: Wedekind, H., Kratzer, H. (Hrsg.): „Büroautomation ´85“, German Chapter of the ACM, Verlag B.G. Teubner, Stuttgart, S. 88–107, 1985.

Nier89 Nierstrasz O.M., Tsichritzis D.C.: “Integrated Office Systems“, in: *Object-Oriented Concepts and Databases*, Kim, Lochovsky (eds.), Addison-Wesley, 1989.

Ober96 Oberweis A.: „Modellierung und Ausführung von Workflows mit Petri-Netzen“, Teubner-Reihe Wirtschaftsinformatik, B.G. Teubner-Verlagsgesellschaft, Stuttgart, 1996.

Ortn97 Ortner E.: Glossar in [Jabl97].

Öste96 Österle H., Vogler P. (Hrsg.): „Praxis des Workflow-Managements - Grundlagen, Vorgehen, Beispiele“, Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1996.

Puus97 Puustjärvi J., Tirry H., Veijalainen J.: Reusability and modularity in transactional workflows“ in: Information Systems, Vol. 22, No 2/3, Elseview Science Ltd., pp101-120, 1997.

Reic98 Reichert M., Dadam P.: “ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control“, in: Journal of Intelligent Information Systems (JIIS), Special Issue on Workflow and Process Management, Vol. 10, No. 2, 1998.

Rein93 Reinwald B.: „Workflow-Management in verteilten Systemen“, B.G. Teubner-Verlagsgesellschaft, Stuttgart, 1993.

Reis82 Reisig W.: „Petri-Netze — Eine Einführung“, Springer-Verlag, 1982.

Reis87 Reisig W.: „Das Verhalten verteilter Systeme“, in: GMD Berichte, Nummer 170, Oldenburg, 1987.

Reis91 Reisig W.: „Petri Nets and Algebraic Specification“, in: Jensen K., Rozenberg G. (eds.): High level Petri Nets, pp. 137–170, Springer-Verlag, 1991.

Ritt97 Ritter N.: „DB-gestützte Kooperationsdienste für technische Entwurfsanwendungen“, Dissertation zu Datenbanken und Informationssystemen, Band 33, infix-Verlag, St. Augustin, 1997.

Ronz85 Ronzani, S., Tisato, F., Zicari, R.: “An Office Specification Language Based on Path Expressions“, in: ACM SIGSMALL Symposium on Small Systems 1985, p. xiii+255, 128, pp. 108–127.

Rose96 Rosemann M.: „Komplexitätsmanagement in Prozeßmodellen“, Gabler-Verlag, 1996.

Rusi94 Rusinkiewicz M., Sheth A.: “Specification and Execution of Transactional Workflows“, in: Kim W. (ed.) *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Addison-Wesley, 1994.

Rusi95 Rusinkiewicz M., Klas W., Tesch T., Wäsch J., Muth P.: “Towards a Cooperative Transaction Model - The Cooperative Activity Model“, in: *21th International Conference on Very Large Databases*, (VLDB ‘95) in Zurich.

Sadi97 Sadiq W., Orlowska M.E.: „On Correctness Issues in Conceptual Modeling of Workflows“, in: Proceedings of the 5th European Conference on Information Systems (ECIS '97), Cork, Ireland , June 19-21, 1997.

Sche94 Scheer A.-W.: „Wirtschaftsinformatik - Referenzmodelle für industrielle Geschäftsprozesse“, 5. Auflage, Springer-Verlag, 1994.

Schl98 Schlund M., Schamburger R., Neeb J., Jablonski S., Böhm M.: „Workflow-Operationen als Bereicherung funktionaler Dekomposition in Workflow-Schemata“, MODELLIERUNG ‘98

Schu97 Schuster H.: „Middlewarebasierte Architektur von verteilten Workflow-Management-Systemen“, Dissertation, Institut für Mathematische Maschinen und Datenverarbeitung VI, Universität Erlangen-Nürnberg, 1997.

Schw93 Schwab K.: „Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystemes zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen”, Dissertation der Fakultät Sozial- und Wirtschaftswissenschaften, Universität Bamberg, 1993.

Schw95 Schwab K., Ludwig H.: „Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit“, Bamberger Beiträge zur Wirtschaftsinformatik, Nr. 30/1995, in: Tagungsband des Workshops „Koordinationsmechanismen bei der computergestützten kooperativen Arbeit“ der GI-Fachgruppe 5.5.1 am 7.7.95 in Bamberg.

Shet96 Sheth A., Kochut K., Miller J., Worah D., Das S., Lin C., Palaniswami D., Lynch J., Shevchenko I.: “Supporting State-wide Immunization Tracking using Multi-Paradigm Workflow Technology", in: *Proceedings of the 22nd International Conference on Very Large Databases* (VLDB ‘96), September 1996.

Sieg95 Siegmund M.: „Konfigurierbare Ablaufsteuerung in Workflow-Management-Systemen“, Studienarbeit im Fach Informatik, IMMD VI, Universität Erlangen-Nürnberg, 1995.

Sing97a Singh M.P.: “Formal Aspects of Workflow Management Part1: Semantics“, North Carolina State University TR-97-04, May 11, 1997.

Sing97b Singh M.P.: “Formal Aspects of Workflow Management Part2: Distributed Scheduling“, North Carolina State University TR-97-05, May 11, 1997.

Stei97a Stein K.: „Aspektorientierte Workflow-Modellierung: Ein Erfahrungsbericht“, in: Becker J., Rosemann M. (Hrsg.): „Organisatorische und technische Aspekte beim Einsatz von Workflowmanagementsystemen“, Workshop vom 10.4.97, Institut für Wirtschaftsinformatik, Universität Münster, 1997.

Stei97b Stein K.: „Überführung von Arbeitsabläufen in Workflows“, Kapitel 11 in [Jabl97]

Stro95 Strong D.M., Miller S.M.: „Exceptions and Exception Handling in Computerized Information Processes“, in: *ACM Transaction on Information Systems*, Vol. 13, No. 2, pp. 206–233, April 1995.

Tomb97 Tombros D., Geppert A., Dittrich K.R.: “Semantics of Reactive Components in Event-Driven Workflow Execution”, in: Proceedings of the 9th International Conference on Advanced Information Systems Engineering, Barcelona, Spain, June 1997.

Vadu97 Vaduva A., Gatziu S., Dittrich K.R.: “Investigating Termination in Active Database Systems with Expressive Rule Languages“, Nachdruck des RIDE-Workshops, Schweden, 1997.

Voss96a Vossen G., Becker J. (Hrsg.): „Geschäftsprozeßmodellierung und Workflow-Management“, International Thomson Publishing, Bonn, 1996.

Wede96 Wedekind H.: „Zu den Grundlagen von Workflow-Management-Systemen“, in: ITG Fachbericht, S. 7–18, *STAK ’96 - Softwaretechnik in Automation und Kommunikation*, München, März, 1996.

Wodt95 Wodtke D., Weissenfels J., Weikum G., Muth P., Sinnwell M.: „MENTOR: Entwurf einer Workflow-Management-Umgebung basierend auf State- und Activity-Charts”, in [Laus95].

Woet87 Woetzel, G., Kreifelts, Th.: „Die Vorgangssprache CoPlan Version 2“, Forschungsbericht FB-GMD-87-34, Institut für Angewandte Informationstechnik, GMD, St. Augustin, 1987.

Work94 Workflow Management Coalition: “Glossary“, November 1994.

Work95 Workparty, Benutzerhandbuch Version 2.0, Siemens-Nixdorf AG, Paderborn, 1995.

Work96 Workflow Management Coalition: “Interface 1: Process Definition Interchange“, Document Number WFMC-TC-0020, Draft 5.0, 5. February 1996.

Zuku96 Zukunft O., Rump F.: “From Business Process Modelling to Workflow Management: An Integrated Approach“. in: Scholz-Reiter B., Stickel E. (Hrsg.): „Business Process Modelling“, Springer-Verlag, S. 3–22, 1996.