Universität Klagenfurt

Institut für Informatik

Behandlung von Zeit in Workflow-Managementsystemen

–

Modellierung und Integration

Diplomarbeit

Betreuer: O. Univ. Prof. Dipl.-Ing. Dr. Johann Eder

Verfasser: Heinz Pozewaunig

Datum: 22. September 1996

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, daß ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Klagenfurt, am 22. September 1ßß6

Abstract

Workflows are the computational representation of complex business processes. The aim of workflow management systems is to support the design and execution of these processes. Although workflow management systems are gaining success and popularity in the world of business optimisation, current architectures suffer from the lack of some important concepts.

In this work a special aspect of modelling and executing workflow processes is discussed: the integration of time into workflow management. The notion of time is very important in the area of business process engineering. For instance, in many event chains it is sometimes necessary to specify the duration of a process, the date of resubmission of a document or the deadline for a task. A workflow management system must be able to handle such situations.

First the usability of some approaches is analysed and then two relevant classes of time constraints are identified within the area of workflow management. (1) Inherent time constraints originate from the structure of workflow processes, whereas (2) external time restrictions reflect the effects of business rules or habits. When an analyst defines an order of events he must take into consideration both of them.

In order to handle with inherent time restrictions the net optimisation algorithm PERT has been adapted and extended to add the capability of dealing with more general types of processes. Now, with the help of this new technique called ePERT you can also calculate alternative execution paths, which was not allowed in the original method. ePERT makes it possible to compute time limits for all workflow relevant control and execution structures like sequence, alternative, concurrency, iteration and choice constructions. Furthermore, a rule based mechanism is presented to deal with external time constraints. In this way you can specify and analyse date related or relative time relations between activities.

To demonstrate the benefits of this approach these conceptions are integrated into the prototypical workflow management system Panta Rhei, *which is a development of the Institute of Informatics at the University of Klagenfurt.*

Inhaltsangabe

Workflows sind die computerunterstützte Abbildung komplexer Geschäftsvorgänge. Workflow-Managementsysteme haben die Aufgabe, den Entwurf und die Ausführung dieser Prozesse in Unternehmen zu unterstützen. Trotz des weltweiten Erfolgs und der unwidersprochenen Vorteile, die der Einsatz dieser Vorgangssteuerungssysteme mit sich bringt, gibt es noch immer einige Aspekte, die zwar gewünscht, aber nicht unterstützt werden.

Diese Arbeit beschäftigt sich mit einem dieser Bereiche, der Integration von Zeit in Workflow-Managementsysteme. Zeit spielt eine wichtige Rolle im Geschäftsleben. Gesamtlaufzeiten, Wiedervorlagetermine und Endtermine sind nur einige wenige Beispiele für die Notwendigkeit von Zeitmodellierung und deren Einbindung in ein Workflow-Managementsystem.

Es werden verschiedene Ansätze zur Modellierung von Zeit untersucht. Im Zusammenhang mit dem hier vorliegenden Aufgabengebiet kann man zwei Klassen von Zeitabhängigkeiten identifizieren: (1) inhärente Zeitvorgaben, die sich aus der Ablaufstruktur der Workflow-Prozesse errechnen lassen und (2) externe Zeiteinschränkungen, die durch Geschäftsregeln und Usancen entstehen und im Prozeß berücksichtigt werden müssen.

Zur Behandlung der inhärenten Zeiteinschränkungen wird die Netzplantechnik PERT an die Erfordernisse der Prozeßmodellierung angepaßt und erweitert. Durch die in dieser Arbeit vorgestellten ePERT-Technik ist es nun auch möglich, alternative Wege für allgemeinere Prozesses korrekt abzubilden und die für sie notwendigen Zeitschranken zu errechnen. Mit Hilfe dieser Methode können alle für Workflows relevanten Abläufe, wie Sequenzen, Alternativen, Parallelitäten, Iterationen und Auswahlkonstrukte, zeitbezogen modelliert und analysiert werden.

Um externe Zeitvorgaben in die Prozessabläufe integrieren zu können, wird eine einfache Regelsprache vorgestellt, die für jeden Prozeß die Festlegung von kalenderbezogenen und relativen Zeitabhängigkeiten zwischen Aktivitäten ermöglicht.

Diese Konzepte wurden in das am Institut für Informatik der Universität Klagenfurt entwickelte prototypische Workflow-Managementsystem Panta Rhei *eingebracht, um die Vorteile der Behandlung von Zeitabhängigkeiten zu zeigen.*

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Der Konkurrenzdruck und die wirtschaftliche Notwendigkeit lassen viele Unternehmen nach neuen Wegen zur Verwaltung ihrer internen Abläufe suchen. Diese Wege sollen zu einer flexiblen Gestaltung der Prozeßstrukturen und zu geringeren Kosten führen. In Folge dieser Bemühungen versucht man, Prozesse auf ihren Sinn hin zu überprüfen und durch Vereinfachung zu optimieren. Dieses Vorgehen führt sofort zur Frage, wie diese neu entworfenen Prozesse durch den Einsatz rechnergestützter Methoden umgesetzt und stabilisiert werden können.

Eine Antwort darauf ist im Konzept *Workflow-Managementsystem* zu finden. Diese Vorgangssteuerungssysteme [BS95a] zielen darauf ab, die Ablauflogik der Geschäftsprozesse im Rechner abzubilden und die Verwaltung der Arbeitsflüsse zu erleichtern, beziehungsweise zu automatisieren.

Diese Aufgabe ist nicht einfach. Auf der einen Seite müssen die bestehenden Anwendungen weiterverwendet werden, da es sich kein Unternehmen leisten kann, alle alten funktionierenden Anwendungen wegzuwerfen und neue erstellen zu lassen, um die Abläufe zu optimieren. Die Investitionskosten, der Schulungsaufwand und die Umstellungszeit würden das Unternehmen zu stark belasten.

Auf der anderen Seite sind die optimierten oder neu entwickelten Geschäftsprozesse korrekt im Rechner abzubilden [ASW96]. Die Modellierung umfaßt u. a. folgende Aufgaben:

- Die Erfassung der Zusammenhänge der einzelnen Arbeitsschritte.
- Die Beziehung der Arbeit zur Aufbauorganisation herstellen, d. h. wer ist verantwortlich, wer darf prüfen, wer darf Bearbeitungen freigeben, ...
- Welche Mitarbeiter sind für die Bearbeitung zuständig, welche Mitarbeiter sind in der Lage, die Bearbeitung durchzuführen.

Beim Entwurf einer Modellierungskomponente wird sehr viel Wert darauf gelegt, alle organisatorischen Aspekte einzubinden, damit man verlustfrei Abbildungen von Geschäfsprozessen in ein Workflow-System überführt.

Doch alle diese Modellierungswerkzeuge, die entweder ein integrierter Bestandteil eines Workflow-Managementsystems sind, oder ein Programmpaket eines Zweitherstellers, haben eine Schwäche: die Ressource Zeit, die in vielen Prozessen eine zentrale Rolle spielt, wird kaum oder gar nicht behandelt. Dabei spielt Zeit eine wichtige Rolle im Geschäftsleben. Liefertermine, Bearbeitungsfristen, gesetzliche Vorgaben oder Qualitätsstandards schreiben die Einhaltung von zeitlichen Rahmenbedingungen vor. Auch innerbetrieblich ist man daran interessiert, die Bearbeitungsdauer eines Prozesses so kurz wie möglich zu halten, um Kosten zu senken. Doch wie kann man bei komplexen Workflows diese minimale Durchlaufzeit feststellen? Wie kann man diejenige Aktivität identifizieren, die für eine Verzögerung verantwortlich ist? Ist eine überdurchschnittliche Bearbeitungszeit ein Indiz für die Verzögerung der Gesamtbearbeitungsdauer, zum Beispiel eines Bestellvorgangs?

Die meisten Workflow-Managementsysteme ignorieren die Notwendigkeit der Modellierung von Zeitaspekten und liefern keine Unterstützung zur Beantwortung dieser Fragen.

Diese Arbeit beschäftigt sich aus diesem Grund mit dem Bereich der Integration von zeitlichen Rahmenbedingungen für Geschäftsprozesse in ein Workflow-System. Es werden vor allem zwei Fragen untersucht:

1. Wie kann Zeit für Workflows modelliert werden?

2. Wie können Zeitinformationen zur Steuerung und Überwachung der Integrität eines Workflow-Managementsystems verwendet werden?

Die Gliederung dieser Arbeit sieht folgend aus: Im zweiten Kapitel werden die Grundlagen von Workflow-Managementsystemen erklärt und auf die Notwendigkeit der Zeitmodellierung näher eingegangen. Außerdem wird untersucht, welche Typen von Zeitvorschriften für diesen Einsatzbereich relevant sind. Im dritten und vierten Kapitel werden Ansätze zur Zeitmodellierung dieser Typen vorgestellt.

Der größte Teil des Kapitels 7 beschäftigt sich mit der Integration und Adaption der gewonnenen Erkenntnisse in Workflowsysteme. Vor allem wird dabei der Workflow-Prototyp *Panta Rhei* (Kapitel 6) betrachtet, der am Institut für Informatik der Universität Klagenfurt entwickelt wurde.

Den Abschluß bilden ein Resümee über diese Arbeit und ein Ausblick auf mögliche zukünftige Weiterentwicklungen auf diesem Gebiet.

Kapitel 2

Grundlagen

2.1 Motivation und Begriffsbildung

2.1.1 Entwicklung und Ursprünge

Das Workflow-Konzept stellt eine direkte Analogie zum Begriff „Prozeß“ aus der Welt der Produktionsindustrie [GHS95, S 119] dar. Da im Bereich der Güterherstellung kaum mehr Potential zur Reduktion von Kosten vorhanden ist, versucht man nun mit großem Aufwand Kosten zu senken, indem man vor allem im Bereich der Administration routinemäßige Tätigkeiten automatisiert.

Workflow wird auf verschiedene Weise definiert. Eine zentrale Bedeutung kommt dabei dem Begriff „Geschäftsprozeß“ zu. Ein Geschäftsprozeß ist eine marktzentrierte Beschreibung der Tätigkeiten einer Organisation, der Kundennutzen nach sich zieht. Durch diese Charakteristik ist ein Geschäftsprozeß ein wertschöpfender Vorgang für das Unternehmen.

Ein Workflow ist laut [Mar94]:

> *„... the sequence of actions or steps used in business processes.“*

Workflows sind somit eine formale Repräsentation von Geschäftsprozessen und tragen direkt zur Wertschöpfung eines Unternehmens bei. Ein weiteres Merkmal sind (teil)automatisierte Übergänge zwischen den einzelnen Schritten. Das bedeutet, es gibt Regeln, die die Folgetätigkeit nach der Abarbeitung eines Schrittes des Geschäftsprozesses bestimmen [RzM96]. Diese Regeln bilden die Grundlage, die für die Umsetzung eines Workflows in automatische Abläufe notwendig ist.

Das Softwaresystem, das die Integrität dieser Abläufe garantiert, wird Workflow-Managementsystem (WfMS) genannt. Dabei müssen die einzelnen Teilschritte nicht notwendigerweise durch Computerprogramme repräsentiert sein. Auch Aktivitäten, die nicht rechnergestützt abgewickelt werden, kann man in die Ablaufsteuerung einbinden.

Ein Workflow-Managementsystem ist daher ein komplexes Softwaresystem, das Geschäftsprozesse, oder besser gesagt, deren Repräsentation durch Workflows, steuert, überwacht und koordiniert [BS95a]. Der Unterschied zu bisher bestehenden konventionellen Informationssystemen besteht darin, daß ein WfMS an sehr viele Geschäftsfelder angepaßt werden kann und deshalb als logische Schicht zwischen der eigentlichen Funktionalität der Anwendungen, wie zum Beispiel Buchhaltungsprogramme oder Textverarbeitungen, und einer Datenbank betrachtet wird (*middle ware*). Die Ablauflogik wird von der Anwendung und den Daten strikt getrennt. Damit kann durch eine Veränderung des Geschäftsprozesses eine mühevolle Anpassung der Applikationen entfallen, denn die Struktur des Ablaufes wird nur im WfMS verwaltet.

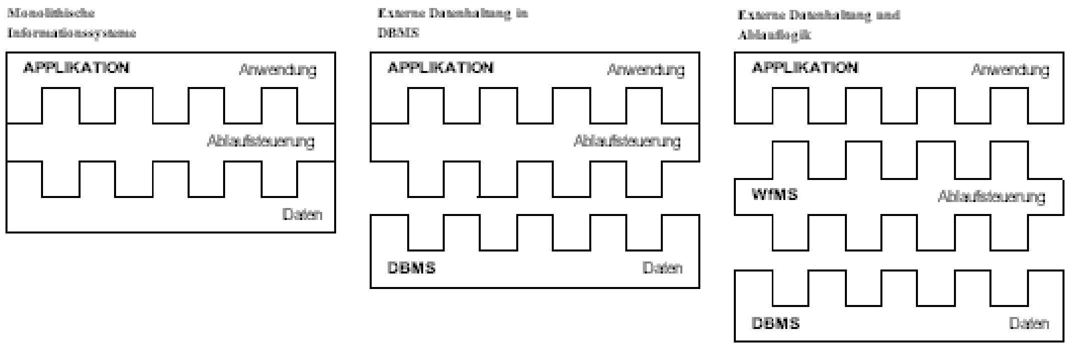

Abbildung 2.1: Aufbau von Informationssystemen, Quelle: [RzM96]

Die Aufgabe eines WfMS geht aber über die einer Steuerungsschicht hinaus. Der Benutzer hat Kontakt mit einzelnen Komponenten des Workflow-Systems und kann natürlich über seine Applikationen direkt auf die Datenbank zugreifen. Das WfMS steuert aktiv die Aufruffolge der einzelnen Applikationen.

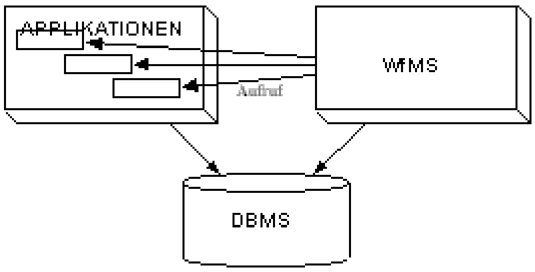

Abbildung 2.2: Beziehung zwischen Anwendung, WfMS und DBMS

Die Abbildung 2.2 stellt diese Beziehung zwischen Anwendungen, WfMS und Datenbankmanagementsystem aus dieser alternativen Sicht dar, die das Zusammenspiel dieser Komponenten verdeutlichen soll.

2.1.2 Vorteile durch Workflow-Managementsysteme

Die Vorteile, die durch den Einsatz eines WfMS entstehen, sind vor allem [BS95a]

Produktivitätssteigerung: Ein großer Teil der Prozeßzeit wird durch Warte- und Transportzeiten verbraucht. Durch die automatische Weiterleitung von Dokumenten können diese Zeiten stark verringert werden. Durch das Vermeiden physikalischer Kopien ist es auch möglich, dasselbe Formular von verschiedenen Sachbearbeitern gleichzeitig bearbeiten zu lassen und danach wieder zu synchronisieren. Diese Parallelisierung kann die Durchlaufzeit eines Prozesses enorm verringern.

Nachweisbarkeit: Die Nachweisbarkeit ist entweder gesetzlich vorgeschrieben oder durch interne Standards gegeben. Sie kann ohne Zusatzaufwand erreicht werden, da die Funktionalität der Datenbank die Archivierung automatisiert.

Qualitätssteigerung: Prozesse können qualitativ verbessert werden, da es nun nicht mehr möglich ist, Arbeiten einfach zu vergessen. Falls ein Schritt bewußt vermieden wird, wird dies zumindest aufgezeichnet.

Zustandsinformation: Der momentane Bearbeitungsstand eines Prozesses kann sofort ermittelt werden. Eine langwierige Suche nach Belegen entfällt und die gewonnene Arbeitszeit kann sinnvoller verwendet werden.

2.1.3 Aufbau von Workflow-Managementsystemen

Die Architektur eines WfMS besteht prinzipiell aus drei Teilen, dem Modellierungsteil (*build-time*), der Ausführungskomponente (*run-time*) und der Schnittstelle, die die Interaktion mit menschlichen Benutzern oder anderen IT-Anwendungen erlaubt.

Die Modellierung beschäftigt sich mit dem Entwurf der Ablauflogik der Workflows und der Einbindung der Applikationen. Auch die Aufbauorganisation (Rollen, Stellen, Kompetenzen und Mitarbeiter) ist festzulegen, damit ein Geschäftsprozeß korrekt abgebildet wird. Die Modellierungskomponente muß flexibel genug sein, ohne großen Aufwand sich ständig ändernde Geschäftsprozesse verarbeiten zu können. WfMS sind daher unmittelbar mit dem Begriff *Business Process Reengineering* verbunden [Jab95, HS95], da man sie als das Werkzeug ansieht, mit dem man neu entworfene Geschäftsprozesse leichter in die Realität umsetzen kann.

Die Laufzeitumgebung (*run-time*) beschäftigt sich mit der Instanzierung der Workflow-Prozesse, der Zuteilung von Akteuren zu den Arbeitsschritten, dem Lastausgleich, der Weiterleitung der Arbeit und der Auswertung und Beobachtung der Abläufe im WfMS.

Es gibt derzeit über 70 Workflow-Managementsysteme am Markt. Jedes System hat eine andere Auffassung von Kernelementen eines Geschäftsprozesses. Darum wurde die unabhängige Workflow-Management-Coalition, kurz WfMC, ins Leben gerufen. Der Zweck dieser Organisation ist die Standardisierung der Schnittstellen, die ein WfMS benötigt. Dadurch sollen Insellösungen vermieden und die Basis zur Interoperabilität zwischen verschiedenen Produkten geschaffen werden [WFM95].

WfMS entstanden durch den Versuch, bereits bestehende Technologien zur Automatisierung von administrativen Arbeitsabläufen zu erweitern und auf breiter Basis einzusetzen. Solche Technologien sind

- Bildverarbeitung,
- Dokumentenmanagement,
- Electronic-Mail-Anwendungen,
- Groupware-Lösungen,
- Transaktionsverarbeitende Systeme,
- Projektentwicklungs-Software und
- Business-Process-Reengineering-Werkzeuge.

Diese Vielzahl von Ausgangsprodukten war Anlaß für die WfMC, ein Workflow-Referenzmodell zu entwickeln. Es ist ein erster umfassender Versuch, die Schnittstellen eines WfMS zu standardisieren.

Ein WfMS besteht aus der zentralen Komponente des Workflow-Enactment-Dienstes. Dieser besteht aus einer oder mehreren Workflow-Engines, die für die Verwaltung der Workflow-Instanzen zuständig sind. Über die Programmierschnittstellen des *Workflow Application Interface WAPI* können Anwendungsprogrammen auf die Dienste des Workflow-Kerns zugegreifen.

Die WfMC befaßt sich auch mit der Definition und Abgrenzung der speziellen Begriffe des Workflow-Managements. Die wichtigsten im Zusammenhang mit dieser Arbeit sind

Prozeß: Eine koordinierte, parallele oder sequentiell bearbeitete Menge von Prozeßaktivitäten, deren Abfolge zur Entwurfszeit festgelegt wird. Aufgabe eines Prozesses ist die Erreichung eines Geschäftsziels. Prozesse werden auch als Prozeßtyp bezeichnet.

Prozeßinstanz: Ein Objekt, das zur Laufzeit erzeugt wird und nach dem Muster eines Prozeßtyps aufgebaut ist.

Prozeßaktivität: Ein Schritt oder eine Beschreibung einer Arbeit, die zu den Zielen eines Prozesses beiträgt. Eine Prozeßaktivität kann manuell oder automatisiert (Workflow-Aktivität) sein.

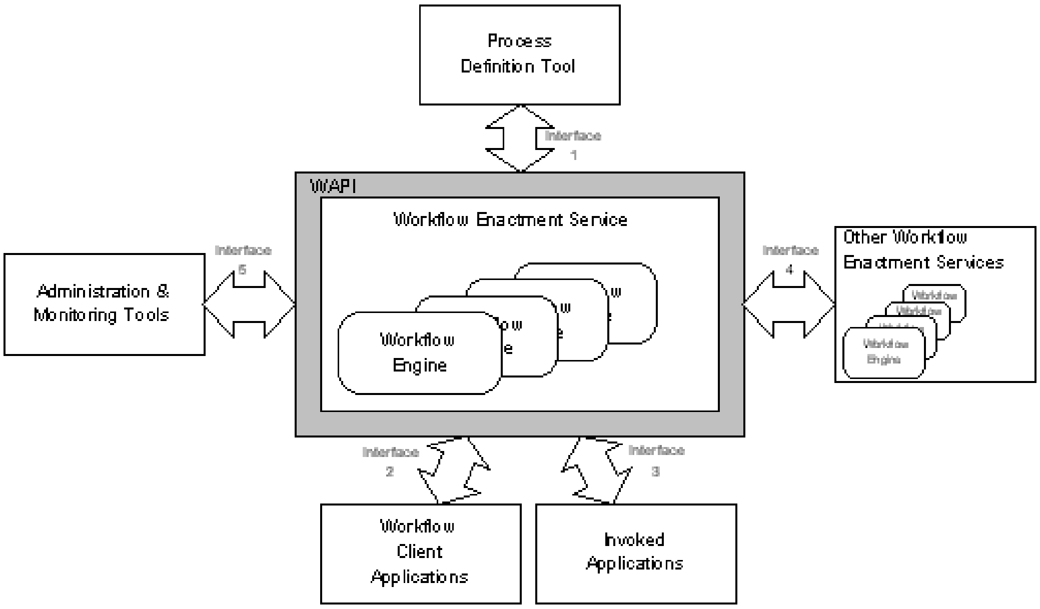

Abbildung 2.3: Referenzmodell der WfMC

Workflow-Aktivität: Ein im Computer abgebildeter Arbeitsschritt. Jeder dieser Arbeitsschritte trägt zum Erfolg eines Workflows bei.

Workflow: Die automatisierte Umsetzung eines (Teil)Prozesses. Diese technische Deutung des Begriffs steht im Gegensatz zur Definition der Seite 3, wo noch nicht von einer Abbildung in ein automatisiertes Modell gesprochen wird.

Workflow-Instanz: Das Laufzeitobjekt, das entsprechend der Struktur eines Workflows gebildet wird.

In dieser Arbeit werden für den Begriff *Workflow* auch noch Prozeß und Prozeßtyp als Synonym verwendet. Nur wo die Unterscheidung zu einem Laufzeitobjekt wichtig ist, wird dafür der Begriff *Prozeßinstanz* verwendet. Weiters wird die Bedeutung von Aktivität und Task der oben definierten Bedeutung von *Workflow-Aktivität* gleichgesetzt, mit der Ausnahme, daß in dieser Arbeit davon ausgegangen wird, daß eine Aktivität (ein Task) nicht mehr weiter unterteilt werden kann.

2.2 Warum ist Zeit wichtig?

Zeitvorgaben spielen beim Entwurf und der Ausführung von Workflowprozessen eine wichtige Rolle. Workflow-Prozesse sind oft an fixe Termine (Beispiel Termingeschäft: „Die Spielzeug-Lieferung für das Weihnachtsgeschäft muß am 30. November erfolgen, sonst ist das Geschäft hinfällig.") oder relative Zeitvorgaben, die sich auf die Gesamtdauer von Aktivitäten beziehen („Bauantragsbearbeitung muß innerhalb von sechs Monaten abgeschlossen sein.") gebunden [JZ96]. Diese Vorgaben müssen unbedingt eingehalten werden, sonst ergeben sich kostenintensive Konsequenzen. Auch zwischen den einzelnen Aktivitäten eines Prozesses sind zeitbezogene Abhängigkeiten häufig („Die Bestätigung einer Flugbuchung muß spätestens 24 Stunden nach der Buchung beim Kunden eintreffen").

Für solche Fälle ist es wichtig, daß der Prozeß-Designer ein Werkzeug zur Verfügung hat, mit dem er zeitliche Einschränkungen und Rahmenbedingungen formulieren kann. Weiters muß er die Systemreaktion festlegen können, die auf eine Übertretungen dieser Vorgaben folgt.

Auch mit Business Process Reengineering Tools (BPR-Tools) wie z. B. dem ARIS Toolset der IDG Prof. Scheer oder BONAPART von UBIS, werden Geschäftsprozesse modelliert. Sie sind durchaus in der Lage, mit der Ressource Zeit im Geschäftsprozeß umzugehen. Diese Werkzeuge können u. a. Rüstzeiten, Einschleuszeiten, periodische Sperrdauern und Liegezeiten für Geschäftsobjekte modellieren und die Auswirkungen untersuchen. Die Analyse, Simulation und Optimierung von Abläufen auch auf Grund von Zeitvorgaben ist ein wichtiges Einsatzgebiet dieser Programmpakete.

Sowohl Workflow-Managementsysteme als auch BPR-Werkzeuge haben Geschäftsprozesse zur Grundlage. Während aber WfMS vorwiegend die Steuerung und Verwaltung aktiver Prozesse als hauptsächliche Aufgabe haben, setzen sich BPR-Tools die Modellierung und anschließende Optimierung von Abläufen zum Ziel. Dadurch ergeben sich einige grundlegende Unterschiede. Betriebswirtschaftliche Geschäftsprozesse, die mit der Unterstützung von BPR-Werkzeugen entworfen werden, sind meistens technisch nicht detailliert genug, um sie für ein Workflow-System sofort nutzbar zu machen [Fü96]. Sie müsssen erst hinsichtlich ihrer Funktionen, Organisation, Programme, Daten- und Steuerungsflüsse verfeinert werden. Trotzdem ist es notwendig, die bereits modellierten und optimierten Geschäftsprozesse zu übernehmen. Andererseits liefern importierte Abläufe Daten, die für die Modellierung von Workflows nicht notwendig sind, bzw. die das WfMS nicht behandeln kann.

Viele BPR-Werkzeuge können die Informationen der Geschäftsmodelle über die von der WfMC definierten Schnittstelle 1 an ein WfMS weiterzugeben. Auch die bereits erfaßten Zeitinformationen können exportiert werden. Doch wegen der fehlenden Unterstützung dieser Konzepte auf Seite des WfMS gehen die Zeitinformationen bei dieser Einbindung verloren. Erst wenn auch auf dieser Seite die geforderte Funktionalität zur Verfügung steht, können BPR-Tools als Modellierungskomponente für Workflow-Prozesse ihre vollen Nutzen einbringen.

2.2.1 Zeitdesign und -planung

Durch die Einbeziehung von Zeit in eine Workflow-Umgebung muß der Designer sich neben den strukturellen Aspekten wie [GHS95]

- die erlaubte Parallelisierung von Teilprozessen,
- die mögliche Anzahl von Iterationen,
- Normal- und Ausnahmefälle,
- Organisations- und Rollenaspekte, ...

auch über die explizite Planung und damit auch Optimierung von zeitlichen Abläufen Gedanken machen. So kann die Kenntnis über Bearbeitungszeiten von Subaktivitäten und Gesamtdurchlaufzeit eines Prozesses der Anstoß sein, die Struktur neu zu entwerfen. Oft wird der Fall eintreten, daß idealisiert geplante Prozesse zu überlangen Bearbeitungszeiten führen, da die Tendenz besteht, parallele Strukturen im ersten Ansatz als Sequenz festzulegen.

Vor allem durch Analyse von Strukturen, die Nebenläufigkeit[1] erlauben, können große Optimierungspotentiale zu Tage treten. Häufig tritt der Fall ein, daß am Ende einer nebenläufigen Struktur unverhältnismäßig lange auf einen Zweig gewartet werden muß, obwohl alle anderen Abarbeitungszweige bereits weiter machen könnten. Durch die Analyse des zeitlichen Verhaltens werden solche Wartezeiten erkannt und können, wenn es die Semantik des Geschäftsprozesses zuläßt, vermieden werden.

Auch die Prüfung der Verträglichkeit unterschiedlicher Typen von Zeitvorgaben, muß durch die Modellierungskomponente ermöglicht werden. Zeiteinschränkungen, die sich nicht aus der Summe der einzelnen Bearbeitungszeiten ergeben und vom Workflow-Entwerfer „von außen“ eingebracht werden, können mit strukturellen Zeitvorgaben, die sich aus der Anordnung der einzelnen Aktivitäten ergeben, verglichen werden. Diese Prüfung zeigt einen Fehler an, wenn durch Strukturangaben Zeitvorschriften unmöglich einzuhalten sind.

Beispiel 2.1 *Konflikt zwischen struktureller und äußerer Zeitvorschrift*
Ein Prozeß P besteht aus der Abfolge der Teilschritte A und B. Die Aktivität A dauert 5 Tage, für B werden 3 Tage veranschlagt.

[1] In dieser Arbeit wird der Begriff „Nebenläufigkeit“ der Bezeichnung „Parallelität“ vorgezogen. Zwei Vorgänge heißen *nebenläufig*, wenn sie „unabhängig voneinander bearbeitet werden können“ [Dud88], was zeitlich verzahnt auch vom selben Akteur durchgeführt werden kann. Parallelität erfordert aber **gleichzeitige**, unabhängige Bearbeitung und folglich auch verschiedene Akteure. Nebenläufigkeit ist somit ein allgemeinerer Begriff als Parallelität.

Wenn nun eine Geschäftsregel R besagt, daß B spätestens 7 Tage nach dem Start des Prozesses beendet sein muß, so ergibt sich ein Konflikt aus der inneren zeitlichen Struktur des Prozesses P und der äußeren Geschäftsregel R, da der geforderte Zeitpunkt frühestens am 8 Tage erreicht wird.

□

Aufgetretene Konflikte zeugen von der fehlenden Deckungsgleichheit der Prozeßstruktur mit den Geschäftsregeln. Der Workflow-Entwickler hat nun die Wahl,

- die Struktur der Workflows anzupassen oder
- die Geschäftsregel zu modifizieren.

Vor allem bei umfangreichen und komplexen Workflow-Prozessen ist es ohne exakte zeitliche Planung schwer, inkonsistente Definitionen zu erkennen und zu lokalisieren. Zeitliche Modellierung gibt dem Prozeßdesigner eine weitere Dimension der Konsistenzprüfung vor, die die Stimmigkeit des Prozeßentwurfs steigern kann.

Folgende Zeitangaben müssen dem Workflow-Designer zur Verfügung stehen, damit er umfassende zeitlichen Regeln definieren kann.

2.2.2 Bezugspunkte von Zeitangaben

Relative Abhängigkeiten

Relative Zeitabhängigkeiten beziehen sich nicht auf Kalenderzeitpunkte, sondern treten zwischen zwei Zuständen von Aktivitäten auf. Als Zustand werden in dieser Betrachtung der Anfangs- und Endzustand einer korrekt abgearbeiteten Aktivität bezeichnet. Weitere Zustände, die sich auf transaktionale Definitionen wie *commit* oder *abort* beziehen, werden nicht betrachtet.

Absolute Zeitvorgaben

Kalenderdaten werden auch als absolute Zeitpunkte bezeichnet. Solche Vorgaben werden durch „äußere zeitliche Einschränkungen“ in den Prozeß eingebracht und können

- Stichtage (z. B. Bilanzstichtag, Revisionsdatum, Deadlines, Milestones) oder
- kritische Zeiträume (z. B. Inventurzeitraum) sein.

2.2.3 Fehlererkennung

Wenn für einen Workflow-Prozeß zeitliche Einschränkungen vorgegeben werden, ist es notwendig, Verletzungen der Einschränkungen festzustellen.

Ein „Zeitfehler“ tritt auf, wenn

- der letzte erlaubte Zeitpunkt für einen Zustand einer Aktivität nicht erreicht werden kann, oder
- ein für die Aktivierung eines Arbeitsschrittes wichtiger Kalenderzeitpunkt versäumt wird.

Das System muß die Laufzeitfunktionalität zur Verfügung stellen, um eine Übertretung einer spezifizierten Restriktion auch überprüfbar zu machen.

2.2.4 Fehlervermeidung durch dynamische Prioritäten

Zeitschranken werden definiert, um zur Laufzeit aufgetretene Fehlersituationen zu erkennen und um Heuristiken anzugeben, die mithelfen, solche Situationen zu vermeiden.

Eine weitere Aufgabe der Zeitplanung und -überwachung ist es, bei der Bestimmung des nächsten auszuführenden Arbeitsschrittes mitzuwirken, wenn

1. gleichwertige Alternativen derselben Instanz eines Workflows gewählt werden können, oder
2. bei unterschiedlichen Instanzen oder unterschiedlichen Prozessen der nächste auszuführende Arbeitsschritt gefunden werden muß.

Die aus dem Vergleich der Zeitvorgaben mit den aktuellen Zeitpunkten gewonnenen Informationen, müssen mit in die Prioritätenberechnung zur Auswahl des nächsten auszuführenden Tasks des Scheduler einbezogen werden [SST+96]. Je näher der aktuelle Zeitpunkt dem spätest möglichen Eintreten des entsprechenden Zustands kommt, desto dringlicher ist die Aktivierung und Zuordnung zu einem Akteur. Die Folge ist eine Anhebung der Priorität des Arbeitsschrittes.

2.2.5 Monitoring - Prozeßcontrolling

Eine der wichtigsten Aufgaben eines Workflow-Managementsystems ist die Überwachung der laufenden und die Auswertung der abgeschlossenen Vorgänge des Systems. Es wurde deshalb von der *Workflow Management Coalition* eine separate Schnittstelle vorgesehen, die zu diesem Zweck externen Applikationen den Zugang zum WfMS ermöglichen.

Zeitdaten werden unmittelbar zur Überwachung der Prozeßstati herangezogen. Durch die errechneten Daten können eingetretene oder sich abzeichnende Verzögerungen erkannt werden.

Die Monitoringkomponente des WfMS ist für die Erkennung solcher Zustände verantwortlich. Weiters ist eine Statistikfunktion wichtig, die die gewonnenen Informationen auswertet und so dem Benutzer sinnvoll zur Verfügung stellt.

2.2.6 Wahrscheinlichkeiten der Termineinhaltung

Durch die errechneten Zeitinformationen muß es auch möglich sein, Aussagen über

- die Überschreitung einer Frist,
- das Erreichen eines Zustands innerhalb eines gewissen Zeitraumes und
- den Eintritt einer Aktivität bzw. eines Prozesses in einen kritischen Zustand (Verspätung)

treffen zu können. Solche Bewertungen werden als Wahrscheinlichkeiten angegeben.

Diese Daten geben dem Prozeßverantwortlichen ein Instrument in die Hand, sofort auf negativ bewertete Veränderungen des Ablaufs zu reagieren, bevor die Auswirkungen zu spüren sind und die Prozesse nicht mehr steuerbar werden.

2.3 Transaktionsmanagement und Workflow

Trotz der Tatsache, daß in der Praxis eingesetzte Workflow-Managementsysteme die in sie gesetzten Anforderungen gut erfüllen, sind einige technische Probleme nur notdürftig gelöst worden [KR96]. Das liegt vor allem daran, daß die Entwickler die Prozeßsicht als hauptsächlichen Ansatz zur Lösungsfindung heranzogen.

Einige solche Schwierigkeiten bestehen darin,

- die Datenabhängigkeiten zwischen Workflows zu verfolgen,
- kooperierende Aktivitäten wirksam zu unterstützen und
- Recovery-Maßnahmen zur Verfügung zu stellen.

Diese Probleme werden von erweiterten Transaktionskonzepten vermieden [Moh95, GR93]. Deshalb kann die Robustheit und Flexibilität von heutigen Workflow-Managementsystemen durch die zusätzliche Funktionalität von verbesserten Transaktionsmodelle gesteigert werden [NDS96]. Zwar ist diese Vorgehensweise manchmal zu datenorientiert, doch die durchdachten Modelle zur sicheren Handhabung von Nebenläufigkeiten und Wiederherstellung des Datenbestands im Fehlerfall eignen sich gut für den Einsatz im Workflowbereich[2].

Transaktionsmodelle allein reichen aber noch nicht aus, um die geforderte Funktionalität zu erreichen. Die auffälligsten Unterscheidungen [AAA+96] ergeben sich aus dem

[2] Ein sehr interessanter Ansatz in die umgekehrte Richtung ist in der Arbeit von [AAA+96] beschrieben. Hier wird die Funktionalität eines existierenden Workflow-Managementsystems als Entwicklungsumgebung genutzt, um erweiterte Transaktionsmodelle zu implementieren. Es wurde der Versuch unternommen, das Verhalten *linearer Sagas* und *flexibler Transaktionen* mit Hilfe des IBM-Produkts *FlowMark* zu realisieren.

Einsatzgebiet von WfMS, also der Geschäftswelt. Es werden für diese Betrachtungsweise vier Objekttypen gesehen, **Benutzer**, **Aktivitäten**, **Programme** und **Daten**.

Die wesentlichsten Unterschiede sind deshalb

- menschlicher Bearbeiter als Akteure,
- das mögliche Bearbeiten von Arbeitsvorgängen **nur** durch eine Person ohne Einbeziehung eines Rechners,
- der Abkapselung von Tasks vor dem WfMS[3],
- die Lockerung von Ressourcenverwaltung aufgrund *semantischer* Serialisierbarkeit und
- der Abbruch einer Aktivität durch ein Mißlingen des Geschäftsvorfalls trotz technisch korrekter Ausführung [EL95a].

Um die Vorteile der erweiterten Transaktionsverfahren in den prozeßorientierten Workflow-Managementsystemen nutzen zu können, müssen diese Konzepte alle wünschenswerten Eigenschaften beider Sichten vereinigen. Einige Transaktionsmodelle wurden speziell für Workflows entwickelt [DHL91]. Solche Konzepte werden als *Workflowtransaktionen* bezeichnet [EGL96, EL96, EL95b, RS93].

Im weiteren Verlauf dieser Arbeit werden, wenn nicht anders erwähnt, die Begriffe Workflow-Aktivität, Task und Transaktion synonym verwendet.

2.4 Transaktionale Abhängigkeiten

Das Problem der Abhängigkeitsbeziehungen zwischen Transaktionen wird schon lange untersucht. Das Beispiel 2.2 zeigt eine mögliche Abhängigkeit der Transaktion B vom Endzustand der Transaktion A.

Beispiel 2.2 *Transaktionale Abhängigkeitsbeziehung*

Die Transaktion B kann **nur** dann starten, wenn die Transaktion A durch ein *abort* abgebrochen wurde (Kompensationstransaktion). □

In der Literatur findet man viele Klassifizierungen von Abhängigkeiten [CR92, RS93, Kle91, ASSR93] und Ansätze zur Lösung von Problemen bei der Spezifikation und Einhaltung von Integritätsbeziehungen zwischen Transaktionen. Grundsätzlich können Konflikte[4] und somit Abhängigkeiten durch

[3] Das WfMS kann nicht in den laufenden Arbeitsvorgang eingreifen oder dessen Status erfahren. Erst nach dem Abschluß des Tasks erhält das WfMS Informationen über Erfolg oder Mißerfolg einer Bearbeitung.

[4] Allgemein gesagt bedeutet es, daß zwei Operationen konfliktär und damit gegenseitig beeinflussend sind, wenn die Reihenfolge ihrer Abarbeitung eine unterschiedliche Auswirkung auf das Ergebnis hat.

1. die Struktur der einzelnen Transaktionen oder

2. durch das Verhalten der Transaktionen

entstehen [Elm92].

Um nicht-strukturelle Abhängigkeiten[5] zwischen Aktivitäten eines Workflows definieren zu können, müssen die eingesetzten Konzepte mächtig genug sein, um die notwendigen Beziehungen darstellen zu können [GH94]. Da in erweiterten Transaktionskonzepten, wie z. B. *nested transactions* [Moh95], Hierarchien erlaubt sind, kann nach dem Zusammenhang der Beziehung unterschieden werden:

Intra-transaktionelle Abhängigkeiten, die Beziehungen von hierarchisch zusammengehörenden Transaktionen zueinander beschreiben. Solche Beziehungen führen dann zu einem korrekten Ergebnis, wenn der Schedule der zusammengehörenden Transaktionen serialisierbar ist [VGH93].

Inter-transaktionelle Abhängigkeiten, die Beziehungen zwischen Transaktionen beschreiben, die nicht hierarchisch zusammengehören, sondern die voneinander unabhängigen Strukturen zugeordnet sind.

Als zusammengehörend werden die Transaktionen eines erweiterten Transaktionskonzeptes dann bezeichnet, wenn sie

- einen gemeinsamen Vorfahren in ihrem Ausführungsbaum haben, oder

- wenn eine Transaktion T_j ein Nachkomme der Transaktion T_i im Ausführungsbaum ist.

Zeitbezogene Restriktionen können in beide Klassen fallen und müssen deshalb in den meisten Fällen unabhängig vom Transaktionsmodell beachtet werden.

Eine Anwendung für die Spezifikation von Konflikten zwischen transaktionalen Zuständen ist in [GH94] angeführt und im Beispiel 2.3 wiedergegeben.

Beispiel 2.3 *Mögliche Zustandsabhängigkeiten zwischen Transaktionen*
Zwischen zwei Transaktionen T_i und T_j sind folgende Beziehungen möglich:

1. *backward-commit-begin (T_j, T_i)*: T_j darf nicht starten, bevor T_i mit *Commit* beendet wurde

2. *backward-abort-commit (T_j, T_i)*: T_j darf nicht *committen*, bevor T_i abortet.

3. *strong-commit-abort (T_j, T_i)*: T_j muß abbrechen, wenn T_i committet.

[5] Abhängigkeiten, die sich nicht aus der Einbindung von Aktivitäten in Kontrollstrukturen, wie *Sequenz*, *Iteration*, *Nebenläufigkeit*,... ergeben.

4. *forward-commit-begin*(T_j, T_i): T_j darf nicht starten, nachdem T_i mit *Commit* abgeschlossen hat. Das bedeutet, daß die abhängige Transaktion T_j vor dem positiven Ende von T_i gestartet werden muß, da sie andernfalls nie mehr gestartet werden kann. Es darf also T_j nur dann gestartet werden, wenn das Ereignis $Commit(T_i)$ (noch) nicht im Schedule aufgetreten ist.

□

Es hat sich gezeigt, daß noch weitere strukturelle Abhängigkeitstypen notwendig sind, um Transaktionsmodellierung in der Praxis einsetzen zu können. Einige sind z. B. in [CR92, S 357–358] angeführt.

2.5 Zeitliche Abhängigkeiten

2.5.1 Unterschied zu transaktionalen Abhängigkeiten

Abgesehen von den im vorigen Abschnitt angesprochenen transaktionalen Abhängigkeiten müssen noch weitere Restriktionen berücksichtigt werden, die auch für Workflow-Applikationen wichtig sind. Diese beziehen sich nicht nur auf den Zustand einer Transaktion, sondern auch auf den Zeitpunkt bzw. Zeitrahmen des Eintretens einer Zustandsänderung. Solche Bedingungen bewirken die Verzögerung oder das Beschleunigen des Eintretens eines bestimmten Zustands einer Transaktion [GRL94]. Die Ergebnisse sind abhängig vom

- **Zeitpunkt** des Eintritts eines Zustands, oder
- **aktuellen Zustand** einer Transaktion.

Einige Ansätze, „klassische" strukturelle Beziehungen[6] [ASSR93] zwischen Transaktionen zu behandeln, nehmen eine implizite zeitliche Ordnung und eine darauf aufbauende Relationsbeziehung zwischen den abhängigen Objekten an [Kle91].

Eine der Hauptursachen, warum klassische Korrektheitskriterien und darauf aufbauende Transaktionsmodelle nicht zur Behandlung von Zeitbedingungen geeignet sind, ist die eventuelle **Unabhängigkeit** des zeitlichen Konflikts von Datenwerten. Da ein klassischer Scheduler nur Datenkonflikte auflösen kann, wird eine Verletzung von zeitlichen Bedingungen nicht immer festgestellt [GRL94].

Ein weiterer Grund liegt auch in der **Konfliktlösungsstrategie**, die von Schedulern zur Erhaltung der Korrektheit angewendet wird. Es wird für jedes Paar konfliktärer Transaktionen T_i und T_j eine Abarbeitungsreihenfolge gesucht, bei der alle Operationen

[6] Als klassisch werden Abhängigkeiten wie *Commit-Dependencies, Abort-Dependencies* und *Conditional-Existence-Dependencies* bezeichnet.

der einen Transaktion vor allen *konfliktären* Operationen der anderen liegen. Für die Einhaltung zeitlicher Abhängigkeiten genügt das nicht, da hier **alle** Operationen von T_i vor **allen** anderen von T_j liegen müssen [GRL94].

Zeitliche Integritätsbedingungen, die auch in Workflow-Umgebungen Bedeutung haben, sind

1. absolute Zeitpunkte, zu denen eine Transaktion T_i in einen Zustand Y eintreten muß,
2. vorgeschriebene Laufzeiten für eine Transaktion oder Intervalle von Zeitpunkten, in denen eine Transaktion erlaubterweise ablaufen darf,
3. Beginnzeitpunkte von Aktivitäten, die abhängig vom Zustand (oder vom Eintritt in einen Zustand) einer Aktivität (Transaktion) sind,
4. Endzeitpunkte, die abhängig vom Zustand einer Aktivität (Transaktion) sind.
5. minimale Zeitabstände zwischen Aktivitätenzuständen,
6. maximale Zeitabstände zwischen Aktivitätenzuständen und
7. exakt einzuhaltende Zeitdauern zwischen Beginn oder Ende von abhängigen Aktivitäten oder Transaktionen.

Die Punkte 1 und 2 sind Realzeit-Abhängigkeiten, die anderen werden als Reihenfolgeabhängigkeiten bezeichnet [GH94].

2.5.2 Typen von Zeitrestriktionen

Bereits aus dem Beispiel 2.1 von Seite 9 und der o. a. Aufstellung sind unterschiedliche Klassen von zeitlichen Einschränkungen zu erkennen.

Zum einen werden Zeitschranken für den Gesamtprozeß aus den **vorgegebenen Dauern** für einzelne Aktivitäten errechnet. So ergeben sich erlaubte Intervalle, die für Beginn und Ende einer Aktivität vorgegeben sind. Wird ein solches Intervall überschritten, kommt es zu einer Verzögerung der Gesamtdauer. Zum anderen werden zeitliche Bedingungen durch **Geschäftsregeln** vorgegeben, die auf Grund von Usancen, der Unternehmenskultur oder gesetzlichen Regelungen beachtet werden müssen.

Inhärente Zeitbedingungen

Sie werden nicht explizit vom Workflow-Designer festgelegt, sondern ergeben sich unmittelbar aus der zeitlichen Struktur eines Workflows. Jeder Aktivität eines Prozesses wird eine Zeitdauer zugeordnet und die Gesamtlaufzeit des Prozesses ergibt sich aus der Summe der Zeiten des längsten Weges durch den Prozeßgraphen. Da in einem Workflow auch Alternativen zu berücksichtigen sind, werden für jeden Prozeßgraphen, der durch eine Workflow-Beschreibungssprache festgelegt wird, die Zeitdauern des längsten und kürzesten Weges protokolliert. Dadurch ergeben sich für jeden Anfangs- und Endzustand eines Arbeitsschrittes vier Zeitgrenzen.

1. E_s^{bc} (*earliest best case of state s*) ist der früheste Zeitpunkt für den Eintritt des Zustands s bei der Wahl des kürzesten Weges.
2. L_s^{bc} (*latest best case of state s*) ist der späteste erlaubte Zeitpunkt bei Wahl des kürzesten Weges.
3. E_s^{wc} (*earliest worst case of state s*) ist der früheste Eintritt in den Zustand s auf dem längsten Weg.
4. L_s^{wc} (*latest worst case of state s*) ist der späteste Eintritt des Zustands s. Wenn dieser Zeitpunkt überschritten wird, kommt es zu einer Verzögerung des Workflows.

Der Zweck dieser Restriktionen ist

1. die Machbarkeit mehrerer Zeitregeln schon zur Entwurfszeit zu prüfen,
2. die Vorgabe von Heuristiken für den Scheduler zur Prioritätenbestimmung der nächsten auszuführenden Aktivitäten, und
3. die Definition von Korrektheitskriterien, die die Legalität eines Schedules mitbestimmen.

Externe Zeitbedingungen

Externe Zeitbedingungen werden explizit durch ein Konstrukt der Workflow-Beschreibungssprache vom Entwerfer festgelegt. Sie sind Regeln, die sich aus dem Geschäftsbereich ergeben und deshalb „von außen“ Einschränkungen vorgeben.

Beispiel 2.4 *Externe Bedingungen*
Folgende Punkte sind Beispiele für diesen Typus von zeitlichen Restriktionen.

1. Vorgabe einer Gesamtlaufzeit für einen Bearbeitung ($öDB$) im öffentlichen Dienst beträgt 6 Monate.
2. Zwischen dem Abschluß einer Flugbuchung (FB) und dem Absenden der entsprechenden Bestätigung zum Kunden (KB) dürfen nur 24 Stunden vergehen.

3. Bei der Abmeldung von Kraftfahrzeugen (AM) kann die Kfz-Haftpflichtversicherung erst vier Wochen später storniert werden (St), da die offiziellen behördlichen Daten der Versicherung nicht eher bekannt gemacht werden können.
4. Zwischen dem Schneiden (Sch) und dem Verarbeiten (V) von Schnittholz muß das Material genau 4 Monate ruhen.
5. Das Beratungsgespräch eines Sozialarbeiters ($SozB$) muß mindestens 15 Minuten betragen.
6. Die Inventur (Inv) muß am 31. August 1996 abgeschlossen sein.

□

Eine zeitliche Bedingung bezieht sich auf den **Anfang** oder das **Ende** eines Arbeitsschrittes. Diese Zustände von Aktivitäten sind durch folgende **Beziehungstypen** angeordnet:

min	Die Zeitdauer zwischen zwei Zuständen darf nicht unterschritten werden.
max	Die Zeitdauer darf nicht überschritten werden.
fix	Die angegebene Zeitdauer muß exakt eingehalten werden.
date	Die Angabe bezieht sich auf einen Kalenderzeitpunkt.
	Der Zustand einer Aktivität muß zu diesem Zeitpunkt erreicht sein.

Tabelle 2.1: Beziehungstypen

Diese Beziehungstypen gibt es auch im Beispiel 2.4. Die Beziehung zwischen den Aktivitäten der Punkte 1 und 2 ist *max*. Ein Minimalabstand zwischen zwei Zuständen wird in den Punkten 3 und 5 vorgeschrieben. Der Typ *fix* ist in einer Produktionsumgebung häufig zu finden. Das Beispiel aus Punkt 4 ist von diesem Beziehungstyp. Im letzten Punkt wird ein Kalenderzeitpunkt vorgeschrieben, folgerichtig ist diese Beziehung vom Typ *date*. Die Zusammenfassung des Beispiels ist in der Tabelle 2.2 zu finden.

Zwischen denselben Aktivitäten ist immer nur **eine** Abhängigkeit des **gleichen Typs** erlaubt. Dadurch können zwischen zwei Aktivitäten A und B vierzehn externe zeitliche Abhängigkeiten deklariert werden.

Zeitmaß

Um die vorgeschriebenen Laufzeiten zwischen Zeitpunkten bestimmen zu können, wird eine Metrik definiert, die die Distanz zwischen diesen Punkten mißt. Die Semantik wird durch eine Menge T_{dist} von *zeitlichen Distanzen*, einer Negation $-$, die eine Abbildung

Beispiel	Verursacher	abhäng. Akt.	Typ	Dauer
	A	A	max	
3	A *AM*	A *St*	min	4 weeks
	A	A	fix	
1	A *öDB*	E *öDB*	max	6 months
5	A *SozB*	E *SozB*	min	15 min
	A	E	fix	
	E	A	max	
	E	A	min	
4	E *Sch*	A *V*	fix	4 months
2	E *FB*	E *KB*	max	24 hours
	E	E	min	
	E	E	fix	
		A	date	
6		E *Inv*	date	31.8.1996

Tabelle 2.2: Mögliche Zeitrestriktionen zwischen Aktivitäten

$T_{dist} \rightarrow T_{dist}$ ist, und einer Funktion *dist*, die eine Abbildung von $T \times T \rightarrow T_{dist}$ ist, bestimmt [Sch91].

1. $\forall d \in T_{dist} : -(-(d)) = d$
2. $d_1 \leq d_2 \Rightarrow -(d_2) \leq -(d_1)$
3. $0 \in T_{dist}$
4. $dist(t,s) = 0 \Leftrightarrow t = s$
5. $0 \leq dist(t,s) \Leftrightarrow t \leq s$
6. $dist(t,s) = dist(s,t)$

Aufbauend auf diese *zeitliche Metrik* müssen weitere algebraische Operationen definiert werden, um diese Maßzahl auch einsetzbar zu machen.

$+/-$	$T_{dist} \times T_{dist} \rightarrow T_{dist}$	Summe und Differenz der Zeitdistanz
$+/-$	$T \times T_{dist} \rightarrow T$	Zeitpunkt $\pm$ Distanz
$\star/$:	$T_{dist} \times \mathbb{R} \rightarrow T_{dist}$	Multiplikation und Division einer Distanz
...		

An dieser Stelle sei noch erwähnt, daß die meisten formalen Ansätze, Zeitbedingungen für Transaktionen zu definieren, von einer abstrakten Folge von Zeitpunkten ausgehen. Die Benennung dieser Zeitpunkte erfolgt durch Numerierung mit natürlichen Zahlen oder Definition von Ordnungsrelationen auf der Menge der Zeitpunkte [BS95b, MP91, RU71]. Für theoretische Betrachtungen oder Beobachtung von Transaktionen mit einer „natürlich vorgegebenen“ Ordnung durch eindeutige Transaktionsidentifikatoren ist diese Handlungsweise durchaus adäquat.

Für die Modellierung von Realweltanwendungen, die sich auf Kalenderdaten oder Intervalle zwischen solchen beziehen, sind diese Modelle aber nur von eingeschränktem Wert [Sch91, S 9 ff], da diese abstrakten Zeitpunkte erst auf Kalenderdaten umgerechnet werden müssen.

Anwendungsbezogene Architekturen verzichten deshalb auf die diskreten Betrachtungsweise abstrakter Zeitpunkte und verwenden (kalender)zeitbezogene Datentypen [AL$^+$94, Sch91, JZ96]. Solche Datentypen haben zwar formale Schwächen, wie z. B. die Klärung der Semantik bei der Verwendung von Zeitwerten unterschiedlicher Granularität als Operanden[7]. Doch durch die Nähe zum Anwendungsbereich und die einfachere Handhabung sind sie in der Modellierung besser einsetzbar.

Konzepte der Abhängigkeitsmodellierung mit Zeitbezug sind aus dem Bereich der formalen Spezifikation [Cor94], Systemmodellierung durch Petri-Netze [Han93, CC88, HR88], Transaktionsforschung [Elm92, VGH93, EV95] oder auf dem Gebiet der erweiterten Datenbanken [SW95, BB94] bekannt. Es werden auch Anstrengungen unternommen, die Bearbeitung dieser Aufgabenstellung dem Designer durch rechnergestützte Methoden leichter zu machen [PHLR93]. Nun stellt sich die Frage, ob einige dieser Vorgehensweisen auch für die Spezifikation von Zeit im Workflow-Bereich Verwendung finden können.

Im folgenden sollen verschiedene Vorgehen zur Modellierung von Abhängigkeiten zwischen Aktivitäten bzw. Transaktionen mit Berücksichtigung von Zeitrestriktionen vorgestellt werden. Diese sind, wie z. B. das „Modell der Abhängigkeitsdeskriptoren“ auf ein erweitertes Regelsystem aufgebaut (siehe Abschnitt 3.1), oder sie basieren auf einer temporalen Logik [MP91] (siehe Abschnitt 3.2), bzw. im Fall der *computational tree logic* auf einer Untermenge einer temporalen Logik [ASSR93, SW95].

Auch im Bereich temporaler Datenbanken sind viele Methoden zur Zeitmodellierung und der Darstellung von resultierenden Abhängigkeiten zu finden [TCG$^+$93, BB94, Dit93].

[7] So gibt es kein allgemeines Modell das z. B. die Bedeutung der Operation „(32. Kalenderwoche 1996) + (1 Monat)“ festlegt. Das Ergebnis dieser Addition ist nur von der Implementierung des Additionsoperators abhängig.

Ein Ansatz zur umfassenden formalen Definition einer Workflow-Semantik findet sich in den Arbeiten von Singh[8] über eine Ereignis-Algebra zur Definition von Abhängigkeiten zwischen transaktionalen Workflows [Sin96].

Ein Querschnitt dieser Vorgehensweisen wird in den nächsten Kapiteln kurz erläutert und ihre Eignung für den praktischen Einsatz im Workflow-Bereich diskutiert.

[8] Leider werden zeitliche Abhängigkeiten, die über einfache Reihenfolgebeziehungen hinausgehen, im Rahmen dieser Arbeit nicht behandelt.

Kapitel 3

Spezifikation von externen Abhängigkeiten

3.1 Spezifikation durch Deskriptoren

Abhängigkeitsdeskriptoren sind ein Werkzeug zur Definition von erweiterten Transaktionsmodellen [GH94]. Sie bestehen aus einer kleinen Menge von Basiselementen (*primitives*) und einer definierten Bedeutung dieser Elemente. Die neu entworfene Semantik des Transaktionsmodells kann zur Laufzeit durch den Eingriff des Scheduler erzwungen werden[1]. Mit Hilfe von Abhängigkeitsdeskriporen werden zustandsabhängige und temporale Beziehungen zwischen Transaktionen definiert.

Abhängigkeitsdeskriptoren sind 6-tupel, welche die unterschiedlichen Beziehungen einer Transaktion zu einer Menge von anderen Transaktionen darstellen. Sie haben die Form $(T_i, \tau, O, t, En, Post)$ und können als Regelsystem gesehen werden, welches die vorher definierte Korrektheitskriterien des erweiterten Transaktionssystems überwacht.

Die einzelnen Elemente eines Deskriptors haben folgende Bedeutung:

- T_i ist eine abhängige Transaktion.
- τ ist die Menge der Transaktionen, von denen T_i abhängt.
- O ist die Menge der (Daten)Objekte, die von der Abhängigkeitsregel beachtet werden müssen.
- t ist ein Zeitintervall, von dem T_i beeinflußt. Wenn z. B. eine Transaktion T_i um 16:00 Uhr starten muß, so steht in t **16:00**.

[1] Diese Erzwingbarkeit von Eigenschaften wird von den Autoren als große Differenzierungsmöglichkeit zu anderen Spezifikationswerkzeugen wie z. B. ACTA [CR92, CR94] gesehen.

- *Post* ist die Nachbedingung, also der Ausführungsteil, der die Semantik der Abhängigkeit enthält.
- *En* ist die Bedingung, unter der *Post* aktiviert wird.

Die *En*-Bedingung entspricht dem EC-Teil einer ECA-Regel[2]. Sie wird benötigt, um den **Bewertungszeitpunkt** einer Abhängigkeit festzulegen. Nur wenn die *En*-Bedingung wahr ist, wird auch die Nachbedingung *Post* ausgeführt.

Beispiel 3.1 *Abhängigkeitsdeskriptor*

Die Transaktion T_i kann nicht gestartet werden, bevor die Transaktion T_j positiv terminiert (*commit*). Da in diesem Beispiel die Datenobjekte und die einschränkenden Zeitintervalle nicht von Bedeutung sind, wird der Deskriptor folgend formuliert:

$$(T_i, \{T_j\}, O, t, T_i.status = BEGIN, COMMIT(T_j) < BEGIN(T_i))$$

Die abhängige Transaktion T_i wird von der (einelementigen) Menge $\tau = \{T_j\}$ beeinflußt. Die Auswertung der Regel macht erst dann Sinn[3], wenn T_i bereits gestartet wurde. Erst wenn der Status von T_i als „BEGIN" erkannt wurde, kann mit der Relation $<$ verglichen werden, ob der Zeitpunkt des positiven Endes von T_j vor dem Beginn von T_i liegt.

Ist das *commit* von T_j bereits in der Historie vorhanden, so kann die Transaktion T_i gestartet werden. Ist das nicht der Fall, muß die Operation $COMMIT()$ versuchen, den positiven Abschluß zu erzwingen, oder so lange warten, bis er eintritt. Danach wird T_i gestartet.

Der Unterschied zu $T_i.status = BEGIN$ und $BEGIN(T_i)$ besteht darin, daß im Status der **aktuelle** Datenwert, der den Scheduler aktiviert, festgehalten wird. Im anderen Fall ist die Operation $BEGIN(T_i)$ dafür verantwortlich, die Transaktion in den Zustand zu versetzen und den Zeitpunkt der positiven Ausführung festzuhalten. Der Rückgabewert der Operation ist der Zeitpunkt des Eintretens in den geforderten Zustand und muß persistent gemacht werden. Es muß für jede Transaktion festgestellt werden können, zu welchem Zeitpunkt welcher Zustand eingetreten ist. □

Wenn die Bedingung des Deskriptors *verletzt* ist, wurde die Abhängigkeit der Transaktion T_i von der Transaktion T_j nicht beachtet. Die Folge einer solchen Regelverletzung ist die Rücksetzung der abhängigen Transaktion.

Die Tabelle 3.1 zeigt das Ergebnis der Auswertung des Abhängigkeitsdeskriptors. Erst nach der Erfüllung von *En* wird der Scheduler aktiv und prüft die Einhaltung der Regel.

[2]**E**vent **C**ondition **A**ction

[3]Die Überprüfung dieser Regel kann erst erfolgen, wenn der Startzeitpunkt von T_j bekannt ist.

$$(T_i, \tau, O, t, En, Post) = \begin{cases} \text{erfüllt}, & En \text{ ist } \textbf{wahr} \text{ und } Post \text{ ist } \textbf{wahr} \\ \text{verletzt}, & En \text{ ist } \textbf{wahr} \text{ und } Post \text{ ist } \textbf{falsch} \\ \text{unbekannt}, & En \text{ ist } \textbf{falsch} \end{cases}$$

Tabelle 3.1: Auswertung des Abhängigkeitsdeskriptors

3.1.1 Darstellung von einfachen Abhängigkeiten durch Deskriptoren

Georgakopoulus und Hornick [GH94] identifizieren drei Klassen[4] von Abhängigkeiten, die zwischen Transaktionen auftreten können:

1. *backward state dependencies*,
2. *forward state dependencies*,
3. *strong dependencies*.

Backward state dependency: Das Erzwingen eines Zustands X einer Transaktion T_i, abhängig vom Zustand Y einer Transaktion T_j nennt man *backward-state-dependency*.

Mit Hilfe eines Deskriptors kann diese Tatsache folgend dargestellt werden:

$$(T_i, \{T_j\}, O, t, T_i.status = X, Y(T_j) < X(T_i))$$

T_i ist die Transaktion, die von der (in diesem Fall einelementigen) Menge T_j abhängt.

Das Beispiel 3.1 ist ein Vertreter einer *backward state dependency*.

Forward state dependency: Solche Abhängigkeiten sind dadurch charakterisiert, daß eine Transaktion T_i nicht mehr in den Zustand X kommen darf, **nachdem** T_j im Zustand Y war. Der Deskriptor sieht folgendermaßen aus:

$$(T_i, \{T_j\}, O, t, T_i.status = X, \neg(Y(T_j) < X(T_i)))$$

Die Abhängigkeit ist dann erfüllt, wenn T_i in den Zustand X kommt, bevor T_j im Zustand Y ist oder T_j den Zustand Y überhaupt nie erreicht hat.

Strong state dependency: Falls eine Transaktion T_i in einen Zustand X kommen **muß**, wenn eine andere Transaktion T_j in einen Zustand Y übergeht, spricht man von einer **starken** Zustandsabhängigkeit. Diese Integritätsbedingung wird durch den Deskriptor

$$(T_i, T_j, O, t, T_j.status = Y, X(T_i))$$

festgelegt.

[4] In der Literatur finden sich noch andere Bezeichnungen. So benennen z. B. [ASSR93] Backward state dependency, Forward state dependency und strong state dependency mit *Commit Dependency, Abort Dependency* und *Conditional Existence Dependency*.

3.1.2 Darstellung von komplexen Abhängigkeiten

Komplexe Bedingungen, die sich auf mehrere Transaktionen beziehen und unterschiedliche Abhängigkeiten darstellen, werden aus einfachen *Post*-Prädikaten mit Hilfe der logischen Operatoren $\wedge$, $\vee$ und $\neg$ zusammengesetzt.

So wird z. B. der Deskriptor für die *backward state dependency* „T_3 beginnt nicht, bevor T_1 und T_2 positiv beendet wurden“ mit Hilfe der logischen Konjunktion $\wedge$ gebildet:

Beispiel 3.2 *Komplexer Abhängigkeitsdeskriptor*

$$(T_3, \{T_1, T_2\}, O, t, T_3.status = BEGIN,$$
$$[COMMIT(T_1) < BEGIN(T_3)] \wedge [COMMIT(T_2) < BEGIN(T_3)])$$

□

Die Vorgehensweise für die Oder-Verknüpfung ist analog.

3.1.3 Kritik an Abhängigkeitsdeskriptoren

Abhängigkeitsdeskriptoren eignen sich gut für die Festlegung von Regeln, die auf Zustände von Transaktionen reagieren und solche Konsistenzbedingungen überprüfen.

Doch trotz der Vorteile des einfachen Entwurfs des Verhaltens von Transaktionsmodellen und der Spezifikation von zeitlichen Einflüssen zeichnen sich Probleme ab, die vor allem für Workflow-Umgebungen wichtige Konsequenzen haben.

1. Die Semantik der Operationen der *Post*-Bedingung greift **direkt** in die Transaktionen ein und versucht eine Zustandsänderung herbeizuführen. Schon aus der Sicht einer reinen Datenbankapplikation kann ein Erzwingen eines *commit* nie garantiert werden [ASSR93], da jederzeit durch einen Transaktions-, System- oder Mediafehler die Transaktion durch ein *abort* abgebrochen und zurückgesetzt werden muß [Vos90].

   Wie schon im Abschnitt 2.3 gesagt wurde, ist eine wichtige Annahme eines Workflow-Managementsystems die Abkapselung der Aktivitäten vor dem Workflow-Server. Das bedeutet, daß auch ein *abort* in einer Workflow-Umgebung nicht in allen Fällen durchgesetzt werden kann. Die Bedeutung der *Post*-Bedingung wird dadurch stark reduziert.

2. Die Korrektheit einer Abarbeitungsfolge von Transaktionen kann erst erkannt werden, wenn die Überprüfungsbedingung **nach dem Eintreten** des Ereignisses ausgewertet wurde. Sollte die Bedingung durch die *Post*-Bedingung verletzt worden sein, muß der Fehlermechanismus der Datenbank die entsprechenden Transaktionen zurücksetzen. Damit diese Art der Fehlerbehandlung nicht eintrifft, muß der Scheduler die abhängige Transaktion solange vom Eintritt in den Zustand abhalten, bis

- entweder die abhängigkeitserzeugenden Transaktionen in den gewünschten Zustand übergegangen sind, oder
- der Toleranzspielraum für die Wartezeit überschritten wurde.

3. Weiters ist anzumerken, daß es einige Zeit dauern kann, bis der von der *Post*-Bedingung angestoßene gewünschte Zustand $X(T_i)$ eintritt. Zwar wird durch eine Zustandsveränderung der abhängigkeitserzeugenden Transaktion T_j die begonnene Operation $X(T_i)$ nicht unterbrochen, es muß jedoch entschieden werden, ab wann die Durchführung von *Post* negativ bewertet wird. Es kann auch der Fall eintreten, daß die Operation von *Post* nie erfolgreich ist!

   Für solche Zwecke müssen **Zeitgrenzen** festgesetzt werden, nach deren Überschreitung *Post* vom Scheduler unterbrochen wird und es zu einer Verletzung der Abhängigkeit kommt.

   Hier stellt sich das Problem der korrekten Bestimmung dieser Toleranzwerte, um

   (a) bei zu **kurzer** Frist nicht unnötige Abbrüche hervorzurufen, oder

   (b) bei zu **langer** Frist die Zeitschranken des Gesamtprozesses nicht zu gefährden, oder die Abarbeitungszeit unnötig zu verlängern.

4. Zeit spielt in der Definition von Abhängigkeiten eine untergeordnete Rolle. Es ist nicht möglich, transaktionale Zustände mit Zeitbedingungen zu kombinieren. Die Behandlung von Mindestabständen, maximalen Intervallen oder Kalenderzeitpunkten ist nicht geregelt. Wie der Scheduler die *En*-Bedingung mit Zeitabhängigkeiten zu überprüfen hat, ist nur von der Implementierung, nicht aber von der Semantik abhängig.

5. Die einzig mögliche Reaktion auf auftretende Fehler besteht in der Zurücksetzung der entsprechenden Transaktionen. In einer reinen Datenbank-Umgebung ist dieses Vorgehen völlig korrekt. Wenn aber im Workflow-Bereich jede Zeitüberschreitung mit einem Abbruch und der Zurücksetzung des entsprechenden Geschäftsprozesses geahndet wird, ist das in vielen Fällen nicht die gewünschte Auswirkung. Da im Ansatz der Abhängigkeitsdeskriptoren Zeitbedingungen und transaktionale Bedingungen als zu einer Klasse gehörig betrachtet werden, können auch keine unterschiedlichen Reaktionen formuliert werden[5].

Die hier angeführten Kritikpunkte zeigen auf, daß vor allem in Workflow-Umgebungen und bei extensivem Gebrauch der zeitlichen Modellierungsmöglichkeiten das Modell der

[5]Siehe auch die Überlegungen von Seite 15 zur Datenabhängigkeit der Transaktionsmodelle.

Abhängigkeitsdeskriptoren an seine Grenzen stößt. Vor allem degeneriert der aktive und zustandserzwingende Mechanismus der *Post*-Bedingungen durch die Einschränkungen eines Workflow-Managementsystems zu einem Überwachungsmodul, das erst im nachhinein aufgetretene Verletzungen der Integritätsbedingungen feststellt. Ein Vorgehen, das aktiv den Schedule steuert und Regelverletzungen verhindert, ist empfehlenswerter.

3.2 Beschreibung durch Temporale Logik

Prädikatenlogik erster Ordnung eignet sich zur Spezifikation von auswertbaren und exekutierbaren Bedingungen, da jede logische Aussage in Abhängigkeit von der Belegung ihrer Variablen auf ihren Wahrheitsgehalt geprüft werden kann. Die klassische Logik versucht, mit Hilfe von Aussagen, deren Belegung mit Wahrheitswerten und deren aussagenlogischen Verknüpfungen, Schlußfolgerungen zu ziehen.

Vielfach genügt es aber nicht, Aussagen wie z. B. „Rom ist Hauptstadt Italiens“ zu betrachten, da diese einfachen Ausdrücke in den meisten Anwendungsgebieten nicht zu finden sind. Deshalb wurden Erweiterungen zur klassischen Aussagenlogik erforscht. Solche sind z. B. die Modallogik, bei der Voraussetzungen und Wahrscheinlichkeiten in die Schlußfolgerungen mit einbezogen werden können. Neben den klassischen logischen Operationen sind die zwei wichtigsten Konstrukte der Modallogik der *Möglichkeits-* und *Wahrscheinlichkeits*operator (Tabelle 3.2).

$\Box$	*„Es ist notwendig, daß...“*
$\Diamond$	*„Es ist möglich, daß...“*

Tabelle 3.2: Wichtige Operatoren der Modallogik

Aus der Modallogik und der topologischen Logik [RU71] entwickelte sich die temporale Logik. Zeit wird in diesem Zusammenhang entweder als

1. Kontinuum oder als

2. Abfolge diskreter Zeitpunkte

gesehen, die bei der Interpretation der Wahrheitswerte Schritt für Schritt (zu jedem Zeitpunkt) untersucht werden. Im folgenden wird nur dieser zweite Punkt betrachtet[6], da dieser dem „natürlichen“ Empfinden und der Modellierungsgewohnheit am nächsten kommt.

Hier kann man in der Literatur [Sza95] zwei verschiedene Denkrichtungen feststellen, die sich hauptsächlich in der Definition von Wahrheit unterscheiden. Basis des ersten Ansatzes ist eine sogenannte **Initialsemantik**. Eine Formel ist gültig in einer Interpretation,

[6]Vergleiche die Überlegungen aus dem Abschnitt 2.5.2 auf Seite 20.

wenn der **Anfangszustand** einer Interpretation eine Formel erfüllt [MP91]. Die zweite Version prüft die Gültigkeit einer Formel unter **einigen** Zuständen der Interpretation und wird **Normalsemantik** genannt [Sza95].

Die folgenden Abschnitte befassen sich mit einer kurzen Einführung in die temporale Logik der Initialsemantik und einem Anwendungsgebiet zur Spezifikation von programmähnlichen Ausdrücken.

3.2.1 Temporäre Initialsemantik

Jede Sprache der formalen Logik wird aus Formeln und einer zugehörigen Menge von Operatoren gebildet. Eine temporäre Logik wird aus einer Zustandssprache gebildet, die zur Konstruktion von Zustandsformeln benötigt wird. Zum Bau von generellen Formeln werden logische und temporäre Operatoren gebraucht. Die Ausführungen der nächsten Kapitel folgen weitestgehend der Notation von Manna und Pnueli [MP91] und beschäftigen sich mit der Definition einer temporären Logik auf der Basis der Prädikatenlogik erster Ordnung.

Basis der Sprache ist das Vokabular $\mathcal{V}$, das aus **getypten** Variablen besteht. Diese Menge ist in *rigide* (feste) und *flexible* Variablen aufgeteilt. Eine feste Variable darf ihren Wert während der Zeit nicht ändern. Der Unterschied zur Konstanten besteht darin, daß diese in diesem Zusammenhang als nullstellige Funktionen angesehen werden. Variablen, deren Belegung sich im Laufe der Zustandsübergänge ändern, sind flexibel.

Wie bei jeder Sprache auf der Basis der Prädikatenlogik werden ausgehend vom Grundkonstrukt *Ausdruck* weitere logische Konstrukte durch Operatoren ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) und Quantoren ($\forall$, $\exists$) gebildet. Grundmenge ist das Vokabular V, das eine Teilmenge von $\mathcal{V}$ $V \subseteq \mathcal{V}$ ist. Aus dieser können durch axiomatische Bildungsgesetze *Ausdruck*, *atomare Formeln*, *boolsche Formeln* und *Zustandsformeln über* V gebildet werden.

Der große Unterschied zur Prädikatenlogik erster Stufe wird erst durch die Interpretation der Konstrukte ersichtlich. Ein *Zustand* s *über* V ist eine Interpretation, die jeder Variable $u \in V$ einen Wert aus ihrem Wertebereich zuweist. Symbolisch wird das durch $s[u]$ ausgedrückt. Als *Modell* σ *über* V bezeichnet man eine unendliche Sequenz der Form

$$\sigma : s_0, s_1, s_2, \ldots$$

Jedes s_i ist ein Zustand[7] über V zum Zeitpunkt i. Die Modellierung der Zeit wird demnach durch die Indizierung der Zustände, die von $0 \cdots \infty$ läuft, ermöglicht.

Zur Auswertung einer Formel müssen alle Belegungen der Variablen zu einem Zeitpunkt i im vorhinein bekannt sein.

[7] Als Zustand wird der Vektor der Belegungen aller Variablen zu einem bestimmten Zeitpunkt i bezeichnet.

Die Berechnung des Wertes einer temporallogischen Formel kann kurz zusammengefaßt werden:

> Zuerst werden die Variablen mit den Werten des in Frage kommenden Zustands s gebunden, dann wird die Formel ausgewertet.

Der an einer detaillierten Verfahrensanweisung interessierte Leser wird auf Manna und Pnuelli [MP91] verwiesen. Das folgende Beispiel 3.3 zeigt anschaulich die Vorgehensweise.

Beispiel 3.3 *Auswertung einer temporalen boolschen Formel*

Der Zustand s über dem Vokabular $V = x, y, z$ wird mit $s = \langle x : 2,\ y : 1,\ z : 3\rangle$ festgelegt. Die boolsche Formel $[(x + 2 \cdot z = 2 \cdot y) \rightarrow (x \geq z)]$ wird aufgrund dieser Belegung folgend ausgewertet:

$$\begin{aligned} s[(x + 2 \cdot z = 2 \cdot y) \rightarrow (x \geq z)] &= (s[x] + 2 \cdot s[z] = 2 \cdot s[y]) \rightarrow (s[x] \geq s[z]) \\ &= (2 + 2 \cdot 3 = 2 \cdot 1) \rightarrow (2 \geq 3) \\ &= F \rightarrow F \\ &= W. \end{aligned}$$

□

3.2.2 Zukunftsoperatoren

Ausgehend von der Basis, welche die temporalen Zustandsformeln liefern, werden zu den bereits bekannten Operatoren sogenannte *Zeitoperatoren* hinzugefügt. Diese Klasse läßt sich in *Zukunfts-* und *Vergangenheits*operatoren untergliedern.

Zusätzlich müssen noch die Quantoren für die Gültigkeit im Zusammenhang mit Zeitoperatoren entsprechend erweitert werden.

Zum besseren Verständnis der Semantik dieser Operatoren wird nun der Begriff der Erfülltheit eingeführt.

Definition 3.1 *Erfülltheit*

Eine Formel p ist an einer Position j, $j \geq 0$, in einer Sequenz $\sigma : s_0, s_1, \ldots, s_j, \ldots$ *erfüllt*, geschrieben

$$(\sigma, j) \models p,$$

wenn p im j-ten Zustand von σ wahr ist. Ω

Stellvertretend für alle Zukunftsoperatoren wird der Next-Operator $\bigcirc$ definiert. Dieser Operator bewertet nicht die Belegung der Variablen des aktuellen Zustands, sondern die nächst folgende (unmittelbar zukünftige).

Wenn p eine temporale Formel ist, dann ist es auch $\bigcirc p$, gelesen als das „nächste p“. Die Semantik ist „Wenn $\bigcirc p$ an der Stelle j wahr ist, muß p an der Stelle $j + 1$ wahr sein“.

Name	Symbol	Semantik	Erläuterung
next	$\bigcirc p$	$(\sigma, j) \models \bigcirc p \Rightarrow (\sigma, j+1) \models p$	
always	$\Box p$	$(\sigma, j) \models \Box p \Rightarrow (\sigma, k) \models$ p, $\forall k \geq j$	Ab der Stelle $k \geq j$ immer wahr
perhaps	$\Diamond p$	$(\sigma, j) \models \Diamond p \Rightarrow (\sigma, k) \models p, \exists k \geq j$	Umkehroperator zu $\Box$
until	$p\,\mathcal{U}\,q$	$(\sigma, j) \models p\,\mathcal{U}\,q \Rightarrow \exists k \geq j,\, (\sigma, k) \models p$	
		und $\forall i,\ j \leq i < k, (\sigma, i) \models p$	p ist wahr, bis q wahr wird
wait for	$p\,\mathcal{W}\,q$	$(\sigma, j) \models p\,\mathcal{W}\,q \Rightarrow (\sigma, j) \models p\,\mathcal{U}\,q$	
		oder $(\sigma, j) \models \Box p$	wahr, wenn p wahr, außer q wahr

Tabelle 3.3: Zukunftsoperatoren

$$(\sigma, j) \models \bigcirc p \quad \text{genau dann, wenn} \quad (\sigma, j+1) \models p.$$

Beispiel 3.4 *Auswertung des Next-Operators*

Das Beispiel zeigt die Auswirkung des $\bigcirc$-Operators auf die Auswertung der temporären Formel $(x \geq 1) \wedge \bigcirc(x = 1)$. Die Folge der Belegungen von x ist periodisch.

j	0	1	2	3	4	5	6	...
x	0	0	1	1	0	0	1	...
$x \geq 1$	F	F	T	T	F	F	T	...
$x < 1$	T	T	F	F	T	T	F	...
$\bigcirc(x < 1)$	T	F	F	T	T	F	F	...
$(x \geq 1) \wedge \bigcirc(x < 1)$	F	F	F	T	F	F	F	...

$\Box$

Die Tabelle 3.3 stellt alle Zukunftsoperatoren als Übersicht dar. Die Semantik der Operatoren lehnt sich sehr stark an Programmiersprachen an. Der hauptsächliche Einsatz der Konstrukte der temporalen Logik ist tatsächlich im Bereich der Spezifikation und Verifikation von Echtzeitsystemen zu finden.

Für jeden temporallogischen Operator der Zukunft gibt es ein symmetrisches Gegenstück als Vergangenheitsoperator. In der Tabelle 3.4 werden diese Definitionen kurz dargestellt.

3.2.3 Anwendung der Temporalen Logik - CTL

Die in den vorherigen Abschnitten vorgestellten Operatoren der temporalen Logik ermöglichen die vollständige Modellierung von strukturellen und temporalen Abhängigkeiten zwischen einzelnen Prädikaten. Doch die Sprache der Logik eignet sich nur bedingt für die Erfassung und Beschreibung eines Problemfeldes im praktischen Einsatz. Der Grund dafür liegt vor allem in der gewöhnungsbedürftigen Syntax und der komplizierten Handhabung.

Name	Symbol	Semantik	Erläuterung
before	$\ominus p$	$(\sigma, j) \models \ominus p \Rightarrow (\sigma, j-1) \models p$	Gegenstück zu $\bigcirc$
was always	$\boxminus p$	$(\sigma, j) \models p \boxminus q \Rightarrow (\sigma, k) \models p, \forall k, 0 \leq k \leq j$	Gegenstück zu $\Box$
once	$\diamondminus p$	$(\sigma, j) \models \diamondminus p \Rightarrow (\sigma, k) \models p, \exists k, 0 \leq k \leq j.$	Gegenstück zu $\Diamond$
since	$p\,\mathcal{S}\,q$	$(\sigma, j) \models p\,\mathcal{S}\,q \Rightarrow \exists k, 0 \leq k \leq j, (\sigma, k) \models q$	
		$\forall i, k < i \leq j, (\sigma, i) \models p$	p gilt ab q
back to	$p\,\mathcal{B}\,q$	$(\sigma, j) \models p\,\mathcal{B}\,q \Rightarrow (\sigma, j) \models p\,\mathcal{S}\,q$	
		oder $(\sigma, j) \models \boxminus p$	Gegenstück zu $\mathcal{W}$

Tabelle 3.4: Vergangenheitsoperatoren

Außerdem sind die Operatoren untereinander nicht unabhängig, denn sie lassen sich durch äquivalente Ausdrücke substituieren. So gilt z. B.

$$\Box\, p \equiv \neg \Diamond \neg p.$$

Deshalb genügt eine Basismenge temporaler Formeln, um jeden Sachverhalt auszudrücken. [MP91, S 199 ff] Die Autoren schlagen als Basismenge folgende Operatoren vor:

$$\neg, \vee, \bigcirc, \mathcal{W}, \tilde{\ominus}, \mathcal{B},$$

wobei $\tilde{\ominus} = \neg \ominus \neg p$ ist.

Um die Modellierung mit Hilfe von temporaler Logik zu vereinfachen, wurden Ansätze entwickelt, von denen *Computational Tree Logic (CTL)* als eine Erweiterung (bzw. Einschränkung) der temporalen Logik hier kurz vorgestellt wird.

CTL ist eine Sprache, die auf dem Gebiet der verteilten und parallelen Anwendungen bekannt ist und eignet sich gut für die Beschreibung von Abhängigkeiten [EMSS93].

Die Semantik und Syntax lehnt sich stark an die bereits definierte temporäre Logik an. Hier werden nur mehr die wesentlichsten Unterschiede hervorgehoben.

Die Grundlage einer logischen Formel bilden atomare Ausdrücke. Eine Formel $EX_j f$ drückt z. B. aus, daß es einen direkten Nachfolgezustand gibt, der durch die Ausführung des Prozesses P_j entsteht, wenn die Formel f gültig war.

Eine Formel $\forall f\,\mathcal{U}\,g$ bedeutet, daß es für jeden Berechnungspfad einen Zustand gibt, in dem g wahr ist. Die Formel f hält dann entlang des Pfades, bis g wahr ist. Entsprechend ist auch die Bedeutung der Formel $\exists f\,\mathcal{U}\,g$. Hier muß es zumindest **einen** Pfad geben, in dem g wahr ist. Dann ist f gültig, bis g wahr ist.

Die Semantik einer CTL-Formel hängt immer mit einem n-Tupel $M = (S, A_1, \ldots, A_k, L)$ zusammen. Dabei ist

S eine Menge von Zuständen,

$\mathbf{A_i}$ ist eine binäre Relation $A_i \subseteq S \times S$, die die möglichen Zustandsübergänge von einem Zustand S_j in einen Zustand S_k durch einen Prozeß i festlegt und

L eine Menge von Benennungen von atomaren Ausdrücken, die in dem Zustand s_i wahr sind.

Diese Metastruktur muß der Bedingung der Totalität genügen, d. h. die Menge A enthält alle Zustandsübergänge der spezifizierten Klauseln.

$$A = \bigcup_{i=1}^{k} A_i.$$

Zu jeder Struktur M und einem Anfangszustand $s_0 \in S$ von M existiert ein zugehöriger Zustandsübergangsbaum (*computation tree*), dessen Wurzel mit s_0 bezeichnet wird und dessen Knoten mit den Zuständen $s_i \in S$ benannt werden. Die Notation $s \xrightarrow{i} t$ bezeichnet einen Zweig im Baum, wenn $(s,t) \in A_i$. Daß eine Formel f im Zustand s_0 der Struktur M wahr ist, wird als $s_0 \models f$ geschrieben. Die Bezeichnung $\models f$ bedeutet, daß die Formel f in allen Zuständen von S wahr ist. Erfülltheit kann somit induktiv definiert werden, interessierte Leser werden auf [ASSR93, S 144 ff] verwiesen.

Zeitabhängigkeiten in CTL

Die Erweiterung $\geq$, die zu CTL hinzugefügt wird, ermöglicht die Definition von Realzeitabhängigkeiten. Dabei bedeutet $\exists_F \ {}^{\geq t}e_t$, daß das Ereignis e_t nach t oder mehr Zeiteinheiten während einer Berechnung auftreten kann. Es ist vorgesehen, zwei Arten von Beziehungen modellieren zu können:

1. *Reihenfolgeabhängigkeiten:* Wenn zwei Ereignisse e_1 und e_2 auftreten, dann liegt e_1 zeitlich vor e_2, und e_2 tritt spätestens t Zeiteinheiten nach dem Auftreten von e_1 auf.
$$\forall_G[(e_2 \Rightarrow \forall_G \neg e_1) \wedge (e_1 \Rightarrow \neg\exists_F \ {}^{\geq t}e_2)]$$

2. *Existenzabhängigkeiten:* Wenn Ereignis e_1 irgendwann eintritt, dann tritt e_2 auch irgendwann ein, wobei e_2 dann nicht später als t Zeiteinheiten nach e_1 eintritt.
$$\neg\exists[\neg e_2\, \mathcal{U}\ (e_1 \wedge \exists_G \neg e_2)] \wedge \neg\exists_F[e_1 \wedge \exists_F \ {}^{\geq t}e_2]$$

3.2.4 Kritik an Temporaler Logik

Es gibt noch weitere Entwicklungen auf dem Gebiet der temporalen Logik, von denen wohl eine der wichtigsten die Arbeit von [RMSM+93] ist. Der große Nachteil logischer Sprachen ist ihre syntaktische Unhandlichkeit. Dieser wird hier umgangen, indem eine visuelle Repräsentation für die temporallogischen Formeln gewählt wurde. Die Komposition der Formeln aus (Sub-)Formeln und das logische Schließen wird durch die Sprache *Real Time Future Interval Logic (RTFIL)* ermöglicht. Durch die Festlegung von Intervallgrenzen, in denen eine Formel erfüllt ist, können auftretende Unschärfen erfaßt werden und führen trotzdem zu korrekten Ergebnissen.

Doch alle bisherigen Anstrengungen können nicht verhindern, daß die für den Workflow-Bereich notwendigen Voraussetzungen, temporale Logik als Modellierungssprache für die Definition von externen zeitlichen Abhängigkeiten zu verwenden, noch nicht geschaffen wurden. Die größten Probleme treten durch die Intention, umfassend einsetzbar zu sein, zu Tage.

1. Das Kapitel 3.2 über die Anwendung der Initialsemantik hat gezeigt, daß von einer Sequenz von Variablenbelegungen ausgegangen wird. Die Werte der Variablen müssen schon von Anfang an bekannt sein. Diese Belegungen simulieren das Fortschreiten der Zeit durch eine kontinuierliche Numerierung der Zustände (ein Zustand ist die Menge der Variablenbelegung zu einem Zeitpunkt). Erst dadurch sind die Voraussetzungen geschaffen, Schlußfolgerungen anhand der Anwendung spezieller zeitlicher Operatoren zu ziehen.

   Diese vollständige Axiomatisierung ist in der Praxis nicht gegeben und für die Lösung der Anwendungsprobleme oftmals auch nicht notwendig.

2. Realzeitabhängigkeiten im Sinne von kalenderbezogenen Sachverhalten können nur mit Mühe durch logische Ausdrücke formuliert werden.

3. Die Verständlichkeit der Syntax und Semantik bei der Definition gewöhnlicher Abhängigkeiten, ist für den Praxiseinsatz zu gering [Sin96, S 8]. Da es derzeit noch keine Werkzeuge gibt, die die Komplexität der temporalen Logik vor dem Workflow-Designer versteckt, ist die Akzeptanz kaum gegeben.

Die Vorteile, die sich aus der Verwendung prädikatenlogischer Sprachen ergeben, sind u. a. Beweisbarkeit von Aussagen und Vollständigkeit der Konzepte. Doch diese wiegen die Nachteile der Komplexität und mühevollen Anwendung nicht auf. Gerade im Bereich der Geschäftsprozeßmodellierung und in weiterer Folge des Workflow-Prozeßdesigns ist es oft notwendig, Ideen auszuprobieren und prototypisch zu verwirklichen, bevor an die vollständige Umsetzung der Konzepte gedacht wird. Temporale Logik unterstützt die Prozeßgestaltung nur indirekt und ist deshalb zur Modellierung von Zeitaspekten nur bedingt geeignet.

3.3 Anforderung an zeitliche Modellierung

Alle hier vorgestellten Modelle zur Definition von Abhängigkeiten zwischen Arbeitsschritte, seien das nun Transaktionen im Datenbankbereich oder elementare Workflow-Aktivitäten, haben ihre Vor- und Nachteile. Der größte Nachteil der hier vorgestellten Modelle liegt darin, daß sie entweder die zeitliche Modellierung zu Gunsten der Modellierung transaktionaler Aspekte zurückstellen oder von Voraussetzungen ausgehen, die in der Praxis nicht gegeben sind. Auch ist das Verständnis des Begriffes „Zeit" dieser Modelle zu unterscheiden von dem, der im Workflow-Bereich wichtig ist. Vielfach werden nur strukturelle Aspekte wie Reihenfolgeabhängigkeiten, Überlappungen oder Parallelitäten betrachtet [BQRT95]. Sogenannte *hard deadlines* [GRL94, S 6], also absolute, kalenderbezogene Vorgaben werden nicht beachtet.

Um im Workflow-Bereich externe Zeitabhängigkeiten, die sich auf Zustände von Aktivitäten beziehen, festlegen zu können, müssen gewisse Forderungen erfüllt werden.

- Die sprachlichen Konstrukte zur Definition aller im Workflow-Bereich notwendigen externen zeitlichen Beziehungen[8] müssen in der Workflow-Beschreibungssprache vorhanden sein und sich auf deren Elemente beziehen können.
- Der Einsatz und die Verwendung der sprachlichen Konstrukte muß deklarativ und intuitiv sein. So schreibt Singh [Sin96, S 9]:

  > *„Specifications should be lazy in that they should describe the conditions that must hold over entire computations, without regard to how an acceptable schedule may be generated. This accords well with the spirit of declarative specifications."*

- Zeitliche Definitionen müssen sich auf Zeitpunkte, Zeitintervalle und Zeitdauern beziehen [JZ96]. Kalenderwerte müssen ebenso wie relative Angaben erlaubt werden.
- Die Definition der Zeitwerte kann entweder zur Entwurfszeit festgelegt oder aber erst zur Laufzeit erfragt, bzw. errechnet werden.

  Beispiel 3.5 *Festlegung der Zeitwerte zur Laufzeit*
  Die Frist für die Bestätigung der Flugbuchung hängt von der Einstufung des Kunden ab. Bucht der Kunde die **Touristenklasse**, so beträgt die Frist eine Woche. Bei der **Business-Klasse** wird innerhalb von drei Tagen bestätigt. Fliegt der Kunde in der **VIP-Klasse**, muß die Bestätigung innerhalb von 24 Stunden erfolgen.

[8] Vgl. die Ausführungen von Seite 17.

Die Frist kann also nicht zur Entwurfszeit festgelegt werden, sondern sie wird durch eine Funktion errechnet. Die Aufgabe des Workflow-Entwerfers ist es, diese Funktion zu definieren, die den Wert dann zur Laufzeit bestimmt. □

- Die zeitlichen Restriktionen müssen vom Scheduler ohne übermäßigen Aufwand geprüft werden können.

- Die Reaktion auf eine Verletzung einer zeitlichen Bedingung in Workflow-Umgebungen muß flexibler als jene im Bereich der Datenbanktransaktionen sein. Transaktionen werden immer zurückgesetzt. Diese Vorgehensweise ist für Workflow-Transaktionen zu restriktiv. Es sollte die Anwendungssemantik entscheiden, wie einschneidend sich eine übertretene Zeitregel auf den Ablauf auswirkt.

Im Abschnitt 7.2 wird ein Konzept vorgestellt, das diese Anforderungen erfüllt.

Kapitel 4

Erkennung inhärenter Zeitvorgaben

4.1 Motivation

Die vorherigen Kapitel beschäftigen sich mit der Spezifikation von externen Zeitrestriktionen, die von „außen“ durch entsprechende sprachliche Konstrukte dem Workflow-System vorgegeben werden. Im Abschnitt 2.5.2 auf Seite 16 wurde noch ein weiterer Typus von zeitlichen Einschränkungen identifiziert. Dieser wurde als *inhärent* bezeichnet. Inhärente Vorgaben ergeben sich aus der zeitlichen Struktur des Workflow-Prozesses und werden nicht explizit formuliert. Die folgenden Kapitel beschäftigt sich mit der Aufgabe, solche Strukturen zu erkennen und aus der Workflow-Beschreibungssprache zu berechnen. Weiters wird in den folgenden Abschnitten behandelt, wie diese Informationen zur Steuerung der Abläufe genutzt werden und welchen Zusatznutzen man aus den gewonnenen Informationen erhält.

Ziel dieser Berechnung ist es, zeitkritische Prozesse korrekt zu steuern und gefährliche Situationen, also Verspätungen, rechtzeitig zu erkennen. Außerdem soll der Prozeß-Designer bei der Optimierung der Abläufe so früh wie möglich unterstützt werden. Durch detaillierte Informationen über die zeitliche Struktur kann er Optimierungspotentiale leichter entdecken und nützen.

In weiterer Folge soll das Zeitmanagement des Workflow-Servers ein erster Schritt hin zur Entwicklung einer Steuerungskomponente werden, welche Zeit-, Kosten- und Risikoinformationen verarbeitet und dem Prozeßverantwortlichen einen exakten und aktuellen Blick auf den Zustand der Prozesse erlaubt. Diese Steuerungskomponente ist das Äquivalent zu einem Produktionsleitstand aus dem Bereich der CIM-Automation[1].

[1]**C**omputer **I**ntegrated **M**anufacturing

Aus der Produktionsinformatik [Gal93, DJOR90] und dem Gebiet des „Operations Research“ [GKP92, Phi86, EvF77, SR69], abgekürzt durch OR, sind Verfahren bekannt, mit denen Parameter der Prozeßressourcen berechnet und gesteuert werden können. Wichtige Größen in diesem Zusammenhang können die Durchlaufzeiten, Kosten, Betriebsmittel oder Engpässe sein.

Die graphische Repräsentation der im Prototypen *Panta Rhei* (siehe Kapitel 6) verwendeten Definitionssprache WDL ist in ihrer Struktur den Netzplanmethoden des OR ähnlich. Deshalb liegt der Schluß nahe, die graphentheoretischen Algorithmen der Netzplantechnik zur Berechnung der zeitlichen Struktur eines Prozesses zu verwenden[2].

Es können mehrere Vorteile dieses Vorgehens erkannt werden [GKP92].

- Die Analyse und der Entwurf eines Geschäftsprozesses können im weiteren Sinn auch als Entwurf eines Projekts gesehen werden. Bei beiden ist die Struktur der Abläufe am Anfang nicht genau bekannt und die budgetären und zeitlichen Rahmenbedingungen müssen erst erarbeitet werden.

  Auch die Kontrolle und Zuteilung der laufenden Arbeit ist dem Scheduling ähnlich. Netzplantechniken eignen sich gut zur Planung und Verwaltung von Projekten und haben sich schon bei außergewöhnlichen und langfristigen Projekten bewährt [DD90, Zim92].

  Der größte Unterschied besteht darin, daß ein Projekt meistens nur eine Extension besitzt, also ein Plan für genau ein Projekt erstellt wird. Bei Geschäftsprozessen aber kann es der Fall sein, daß mehrere hunderttausend laufende Prozesse eines Typs (eines Plans) existieren [KAGM96]. Für den Entwurf von Netzplänen macht das keinen Unterschied, die Überprüfung der laufenden Prozesse eines Workflow-Systems auf ihre Übereinstimmung mit dem Netzplan, ist aber wesentlich schwieriger zu bewältigen.

- Der Sachverhalt kann durch Diagramme **visualisiert** werden. Damit können zeitliche Vorgaben für jeden einzelnen Arbeitsschritt sofort erkannt werden und wichtige Zeitbeziehungen zwischen Teilaktivitäten faßbar gemacht werden. Durch alleinige Untersuchung der Ablaufstruktur ist das manchmal nicht möglich.

- Netzplanmodelle sind **flexibel** und **verständlich**. Anpassungen an neue Situationen sind einfach und nachvollziehbar.

- Die eingesetzte Workflow-Beschreibungssprache ist der Netzplandarstellung in mancher Hinsicht ähnlich. Dadurch ist eine **Transformation** der Konstrukte der WDL

[2] So wird z. B. in [GRL94] eine Netzplanmethode als Algorithmus zur topologischen Sortierung einer Transaktionsfolge vorgeschlagen. Dieses Schedulingproblem (Prioritätenbestimmung) ist dieser Aufgabenstellung ähnlich.

in ein Netz erleichtert. Auch die Interpretation der Zeitvorgaben und Zuordnung zu den entsprechenden Aktivitäten fällt dadurch leichter.

Die Berechnung von Zeitschranken für eine verzweigte Aktivitätenkette fällt in eine Problemklasse, für die auch andere Methoden zur Lösungsfindung herangezogen werden können. Als Beispiel sind hier die Lineare Programmierung, Dynamische Programmierung oder Heuristische Suchverfahren zu nennen. Doch diese Verfahren sind nicht intuitiv anwendbar und Zwischenergebnisse der Berechnung können im Gegensatz zu Netzplanmethoden nicht ohne weiteres interpretiert werden. Deshalb werden wegen der oben angeführten Vorteile hier nur mehr Netzplantechniken untersucht.

Diese Modelle wurden in den fünfziger Jahren aufgrund der Nachteile der bis dahin verwendeten GANTT-Balkendiagrammtechnik entwickelt und eignen sich wegen ihrer flexiblen theoretischen Basis zur Darstellung einer Vielzahl praktischer Anwendung. Die bekanntesten Vertreter solcher gerichteten Graphen sind

CPM (Critical Path Method),

PERT (Program Evaluation and Review Technique),

MPM (Metra-Potential-Method),

LESS (Least Cost Estimating),

PD (Precedence Programming) und

RAMPS (Resource Allocation and Multi-Project Scheduling).

Im folgenden Kapitel 4.2 werden die ersten beiden Methoden genauer besprochen und ihre Einsatzmöglichkeiten im Bereich der Workflow-Managementsysteme geprüft.

4.2 CPM/PERT-Netzplantechnik

Die beiden Netzplantechniken *Critical Path Method* und *Program Evaluation and Review Technique* wurden Mitte der fünfziger Jahre ungefähr zur selben Zeit von Beratungsfirmen entwickelt.

Sie erleichtern die Planung und Verwaltung großer zeitorientierter Projekte. Das Ziel beider Methoden ist die Minimierung der gesamten Projektzeit, indem die Reihenfolge der sequentiellen und nebenläufigen Arbeitsschritte in einem Graphen festlegt werden. Gesucht wird der **längste Weg** durch diesen Graphen. Der längste Weg oder **kritische Pfad** bestimmt die Mindestdauer für das untersuchte Projekt. Die Suche nach dem längsten Weg in einem Graphen kann durch den *Dijkstra*-Algorithmus erfolgen (vgl. z. B. [DK88]).

Beide Methoden unterscheiden sich grundsätzlich nur in der Bestimmung der Zeitdauern für die einzelnen Aktivitäten. Bei CPM wird die Dauer durch **einen einzigen**

Wert bestimmt. Das stochastische[3] Verfahren PERT benötigt zur korrekten Errechnung der Zeitgrenzen die Festlegung von **drei** Dauern für die Aktivität [EvF77]:

- eine Untergrenze, die nie unterschritten wird,
- eine Obergrenze,
- der häufigste Wert, der für die Bearbeitung der Aktivität benötigt wird.

Die Schätzung der Unter- bzw. Obergrenze der Dauer einer Aktivität ermöglicht auch die Planung von neuartigen Projekten, von denen nur wenige Daten bekannt sind. Gerade beim Design von Workflows ist es für viele Teilaktivitäten unmöglich oder auch unsinnig, eine exakte Dauer vorzugeben, da genau festgelegte Bearbeitungsdauern in der Praxis des Geschäftsalltags kaum eingehalten werden können. Durch diese Überlegung ist die Verwendung eines stochastischen gegenüber eines deterministischen Verfahrens gerechtfertigt.

Grundsätzlich sind aber der Aufbau und die Berechnungsvorschriften bei beiden Netzplanmethoden identisch. Auf die Besonderheiten des PERT-Verfahrens wird später im Abschnitt 4.2.2 eingegangen.

Mit Hilfe eines Netzplans können unter anderem folgende Fragen beantwortet werden, die auch für den Bereich Workflow von Interesse sind.

- Wann wird der Workflow-Prozeß beendet sein?
- Welche Aktivitäten dürfen sich nicht verzögern, damit die Gesamtlaufzeit nicht verlängert wird?
- Wie groß ist die Wahrscheinlichkeit, daß ein Prozeß in x Tagen beendet ist?
- Hat sich ein Projekt bereits überdurchschnittlich verzögert?
- Kann durch die Umschichtung von Ressourcen die Laufzeit verringert und eine Verspätung wieder aufgeholt werden?
- Wie sehr können Zeitvorgaben für andere Aktivitäten gelockert werden, wenn sich kritische Teile bereits verspätet haben?

Zusätzlich zur Zeitplanung ist es auch möglich, monetäre Kosten in den Plan mit einzubeziehen und dadurch betriebswirtschaftliche Gesamtbewertungen durchzuführen. Diese werden wie bei der reinen Zeitplanung einer Abweichungsanalyse unterworfen. Weiters ist es auch möglich, Zinsen zu den Kostenbewertungen hinzuzufügen [Zim92, 27ff].

[3]Die Stochastik beschäftigt sich mit der Analyse zufallsabhängiger Ereignisse und deren Wert für die statistische Untersuchung.

4.2.1 Aufbau eines CPM-Netzplans

Ein Netzplan ist ein gerichteter, azyklischer Graph mit einer Menge von Knoten und Kanten. Die Knoten repräsentieren Zustände der Basisaktivitäten, als Zustand wird hier der Beginn oder das Ende einer Aktivität bezeichnet. Zustände werden von 0 beginnend geordnet durchnumeriert. Konkret bedeutet es, daß die Nummer i eines Vorgängerzustands kleiner sein muß als die Nummer j eines Nachfolgerzustands. Ein Netzplan hat immer **genau einen** Anfangs- und **genau einen** Endzustand. Die Aktivitäten selbst werden als gewichtete Kanten dargestellt[4].

Beispiel 4.1 *Aufbau eines Netzplans*

Das Beispiel stammt aus [Phi86, S 189ff]. Die Reihenfolgeabhängigkeiten der Aktivitäten A, ..., L werden folgendermaßen festgelegt:

1. A und B sind die ersten Aktivitäten des Projekts.
2. C folgt A und B.
3. E und F folgen B.
4. G ist nach F und C.
5. E ist der Vorgänger von H.
6. H ist der Vorgänger von I und J.
7. K folgt D und J.
8. L folgt K.
9. G, I und L sind die letzten Aktivitäten des Projekts.

Die Dauer der einzelnen Arbeitsschritte ist durch die folgende Tabelle vorgegeben.

Aktivität	A	B	C	D	E	F	G	H	I	J	K	L
Dauer in Wochen	5	3	2	1	3	2	8	1	2	3	4	7

□

Die Abbildung 4.1 zeigt die Struktur des resultierenden Netzplans. Vor allem die durch strichlierte Pfeile dargestellten **Scheinaktivitäten** sind interessant, da sie strukturelle Abhängigkeiten zwischen Zuständen durch *Nullzeit-Arbeitsschritte* darstellen. Solche Synchronisationen sind immer dann notwendig, wenn Zustände in nebenläufigen Arbeitsschritten überprüft werden müssen. Erst wenn ein solcher Zustand erreicht wurde, ist die notwendige Voraussetzung für weitere Arbeitsschritte gegeben. Detaillierte Anweisungen, wie Netzpläne erstellt werden, findet man in [DJOR90, Phi86, EvF77, SR69].

[4] Aus diesem Grund wird der Typ der CPM- und PERT-Netze auch als *Vorgangspfeilnetzplan* bezeichnet. Im Gegensatz dazu ist die MPM-Darstellung vom Typ *Vorgangsknotennetz*, da hier die Knoten die Aktivitäten darstellen.

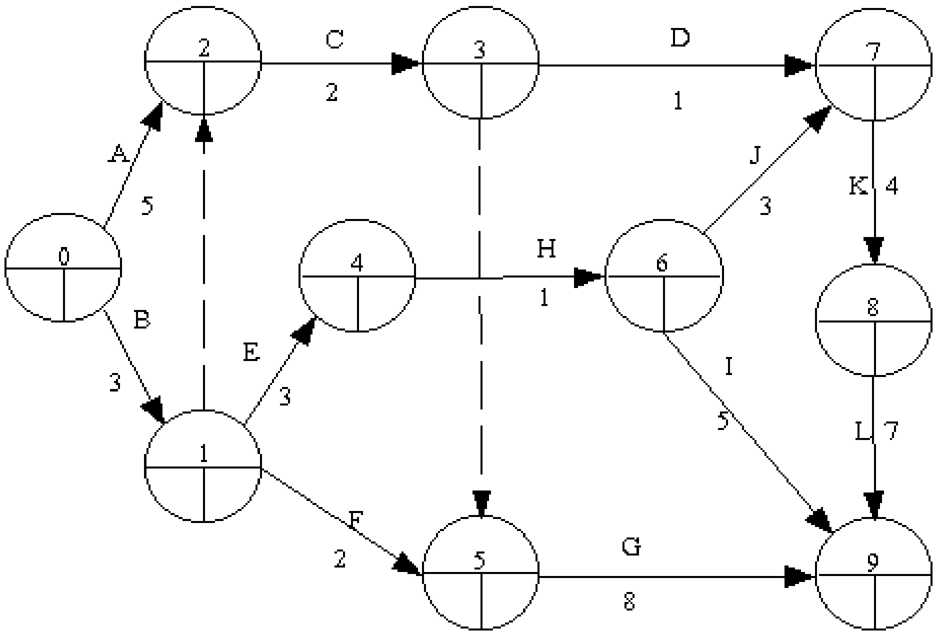

Abbildung 4.1: Netzplan

Da es eines der Ziele eines Netzplanes ist, die Gesamtdauer des Projektes zu bestimmen, werden nun der früheste Zeitpunkt E_j (*earliest*) und der späteste Zeitpunkt L_j (*latest*) für das Eintreten eines Zustands j berechnet.

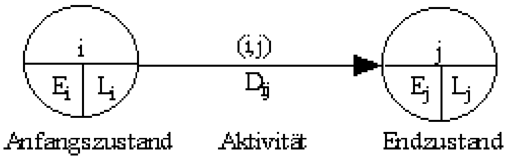

Abbildung 4.2: Darstellung einer Aktivität

Die Berechnung der Werte E und L wird in **zwei Phasen** durchgeführt. Bei der *Hinrechnung* wird vom Startwert E_α des ersten Zustands α des Netzes ausgegangen. Dieser wird immer relativ berechnet und ist daher 0. Alle folgenden Werte E_j werden aus den Informationen ermittelt, die vom direkten[5] Vorgänger i und der Dauer D_{ij} der Aktivität (i, j) zur Verfügung stehen, siehe dazu die Abbildung 4.2. Es ist nun darauf zu achten, daß in einer der Breitensuche ähnlichen Vorgehensweise alle direkten Nachfolger des aktuellen Knotens berechnet werden, bevor zum nächsten Zustand weitergegangen wird. Erst wenn alle Knoten einer Ebene aktualisiert wurden, kann ein Knoten der nächsten Ebene

[5]Ein Zustand i wird als **direkter** Vorgänger eines Zustands j bezeichnet, wenn i und j durch die Aktivität (i, j) verbunden sind. Das heißt, i ist der Anfangs- und j der Endzustand der Aktivität.

ausgewählt werden.

$$E_j = \max(E_{i_1} + D_{i_1 j}, \ldots, E_{i_n} + D_{i_n j});\ n = \text{Anzahl in } j \text{ endenden Aktivitäten}$$

Wenn alle Werte E der Zustände des Netzes berechnet und der letzte Zustand ω erreicht wurde, wird dessen spätest erlaubter Eintrittswert L_ω mit E_ω gleichgesetzt. Damit beginnt die zweite Berechnungsphase, die *Rückrechnung.*

$$L_i = \min(L_{j_1} - D_{ij_1}, \ldots, L_{j_m} - D_{ij_m});\ m = \text{Anzahl in } i \text{ startenden Aktivitäten}$$

Auch hier müssen alle Vorgängerknoten bereits aktualisiert sein, bevor der nächste Knoten zur Berechnung ausgesucht wird. Die Differenz zwischen E_i und L_i ist die **Schlupfzeit** der Aktivität (i, j), die die Toleranzdauer für den Start dieses Arbeitsschrittes angibt.

Die Anwendung der Berechnungsvorschriften auf das Beispiel 4.1 und die resultierenden Werte für den frühesten und spätest erlaubten Eintritt eines Zustands werden in der Abbildung 4.3 dargestellt.

Der maximale Spielraum eines Arbeitsschrittes wird *Total Float* TF genannt. Er wird für eine Aktivität (i, j) folgend berechnet:

$$\mathrm{TF}_{ij} = L_j - D_{ij} - E_i$$

Aktivität		E_i	D_{ij}	L_j	TF_{ij}
A	(0,2)	0	5	7	2
B	(0,1)	0	3	3	0
C	(2,3)	5	2	9	2
D	(3,7)	7	1	10	2
E	(1,4)	3	3	6	0
F	(1,5)	3	2	15	10
G	(5,9)	7	8	21	6
H	(4,6)	6	1	7	0
I	(6,9)	7	2	21	12
J	(6,7)	7	3	10	0
K	(7,8)	10	4	14	0
L	(8,9)	14	7	21	0

Tabelle 4.1: Spielraumberechnung TF aus Beispiel 4.1

Ist dieser Spielraum gleich null, dann bewirkt jede Verzögerung eines solchen Arbeitsschrittes gleichzeitig auch eine Verzögerung des gesamten Projekts. Deshalb werden solche Aktivitäten auch **kritische Jobs** genannt.

In der folgenden Abbildung 4.3, sind solche kritischen Jobs laut der Tabelle 4.1 die Aktivitäten B, E, H, J, K und L. Jede Verzögerung aller anderen Aktivitäten führt nicht zu einer Verschiebung des Endtermines von 21 Zeiteinheiten. Erst wenn ein spätest erlaubter Wert L_i eines nichtkritischen Zustands i überschritten wird, wirkt sich diese Verzögerung auch auf den Endtermin aus.

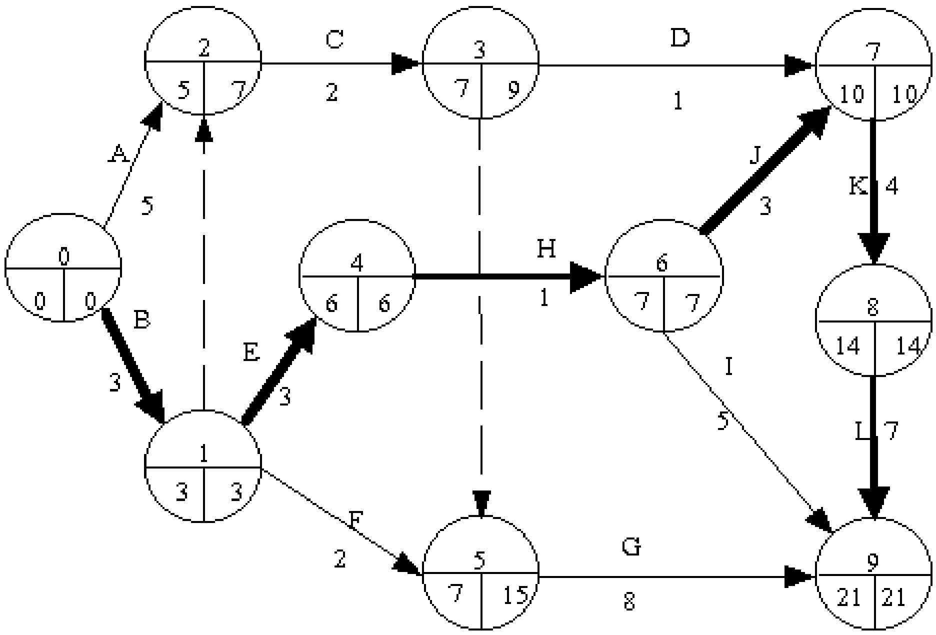

Abbildung 4.3: Kritischer Pfad

Der kritische Pfad im Graphen der Abbildung 4.3 berechnet sich, ausgehend vom Endknoten 9, indem aus der Menge der Vorgängerknoten derjenige Knoten zum kritischen Pfad hinzugefügt wird, der Anfangszustand eines kritischen Jobs ist. Da im Endknoten auf jeden Fall der kritische Pfad münden muß, existiert so ein Zustand. Im nächsten Schritt wird nun von den neu hinzugefügten Knoten[6] ausgegangen und es werden so lange weitere Knoten in die Menge eingefügt, bis der Anfangszustand erreicht wird.

Der längste Weg bzw. kritische Pfad wird in Abbildung 4.3 durch verstärkte Pfeile ausgedrückt.

4.2.2 Die PERT-Netzplantechnik

Die deterministische CP-Methode unterscheidet sich von der stochastischen PERT-Vorgehensweise vor allem durch die Art der Schätzung und der anschließenden Berechnung der Zeitdauer einer Aktivität. Wie schon auf Seite 39 angegeben wurde, müssen für die

[6]Es ist durchaus möglich, daß in einem Iterationsschritt mehrere Knoten hinzugefügt werden, da in einem Netzplan auch mehrere kritische Pfade existieren können.

Anwendung der PERT-Technik drei Schätzwerte für die Dauer einer Aktivität festgelegt sein:

1. Die optimistischste Schätzung a der Dauer für die betreffende Aufgabe. In der Praxis ist dies die Minimaldurchlaufzeit der Aktivität.
2. Der häufigste Wert[7] m und
3. die pessimistischste Schätzung b, die den Fall des *worst case* annimmt, auch wenn dieser Wert eher unwahrscheinlich ist.

Diese Schätzwerte werden benutzt, um den Mittelwert μ und die Varianz σ^2 zu berechnen. Für diese Berechnung wird die Annahme getroffen, daß die Zeitdauern β-verteilt sind. Eine β-Verteilung erleichtert im Gegensatz zu der sonst verwendeten Normalverteilung die Ermittlung der Zufallsparameter μ und σ^2 durch die Formeln

$$\mu = \frac{a + 4m + b}{6}$$

$$\sigma^2 = \left(\frac{b-a}{6}\right)^2.$$

Die Dichtefunktion einer β-verteilten Zufallsgröße ist in Abbildung 4.4 zu sehen.

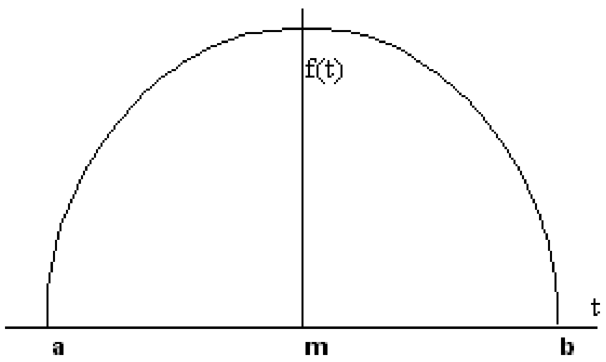

Abbildung 4.4: Dichtefunktion der β-Verteilung

Die Annahme der β-Verteilung gilt aber nur für die Berechnung der Zufallsparameter. Die Zeitdauern selbst werden als normalverteilt angesehen.

Im Netzplan selbst werden nun nicht mehr die Dauern, sondern der Mittelwert und die Varianz der Dauer eingetragen. Die Berechnung der frühesten Zeitpunkte E und spätesten Zeitpunkte L erfolgt durch die Hin- und Rückrechnungsvorschrift des CPM-Netzes, wobei nun der Mittelwert die Rolle der einfachen Zeitdauern des CPM-Netzes übernimmt.

[7] Modus

Beispiel 4.2 *PERT-Hinrechnung*

Gegeben ist ein Teilnetz, das die Beendigung der Aktivitäten A und B darstellt. Der Mittelwert des Anfangszustand 1 von A beträgt 1,0. Die Aktivität B startet durchschnittlich später, und zwar zum Zeitpunkt 2,1. Die Dauer von A ist 4 Zeiteinheiten, während B nur 2,5 Zeiteinheiten zur Verrichtung der geforderten Arbeit benötigt. Laut der Berechnungsvorschrift wird im gemeinsamen Endzustand 3 bei der Hinrechnung das Maximum der Summe von Anfangswert und Dauer eingetragen. In diesem Fall ergibt sich das aus dem Anfangswert $E_1 + D_{13} = 1,0 + 4,0 = 5,0$, weil die Summe der Aktivität B mit dem entsprechenden Anfangszustand E_2 mit 4,6 kleiner ist.

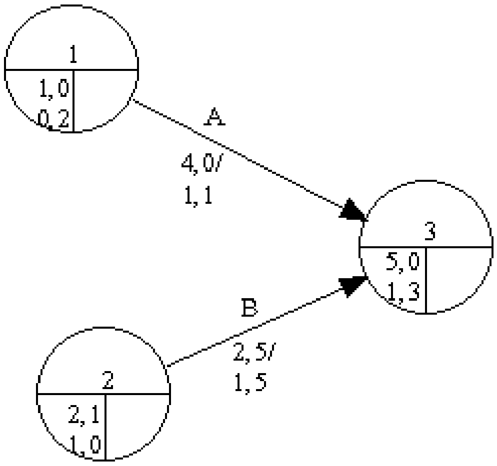

Abbildung 4.5: Hinrechnung in PERT

Obwohl die Varianz der Summe von A und E_1 kleiner ist als das Gegenstück von B muß die aktuelle Varianzensumme in E_3 eingetragen werden. Die Abbildung 4.5 veranschaulicht das Beispiel. Die Einträge der ersten Zeile eines jeden Feldes stellen den Erwartungswert dar, die zweite Zeile beinhaltet die Varianz des Zustands oder der Aktivität.

□

Da sich durch die Summenbildung von Mittelwerten und Varianzen der Verteilungstyp der Normalverteilung nicht ändert [DP88, S 396], kann die Berechnungsvorschrift für die Hin- und Rückrechnung iterativ ohne Einschränkungen angewendet werden.

Beispiel 4.3 *Weiterführung des Beispiels 4.1*

Für das Projekt aus Beispiel 4.1 werden für die einzelnen Aktivitäten A,..., K die optimistischen, pessimistischen und häufigsten Werte a, b und m der Tabelle 4.2 angenommen.

Prozeß	(i,j)		b	m	a	μ	σ^2
A	0	2	6,2	5	3,2	4,9	2,45444444
B	0	1	4	3	1,2	2,86666667	0,75111111
C	2	3	4,23	2	1,5	2,28833333	0,912025
D	3	7	1,4	1	0,8	1,03333333	0,13444444
E	1	4	5,8	3	2,2	3,33333333	1,77777778
F	1	5	2,5	2	1	1,91666667	0,34027778
G	5	9	12,8	8	6,6	8,56666667	10,4544444
H	4	6	1,5	1	0,5	1	0,11111111
I	6	9	3,02	2	1,1	2,02	0,47151111
J	6	7	4	3	2,5	3,08333333	1,17361111
K	7	8	5,6	4	3,6	4,2	2,35111111
L	8	9	9,9	7	6	7,31666667	7,0225

Tabelle 4.2: Erwartungswert und Varianz

Daraus ergeben sich die Zeitschranken für den Netzplan, der in der Abbildung 4.6 zu sehen ist.

Hier wurde der Endtermin des Projekts mit $21,8$ Tagen errechnet. Die Standardabweichung σ beträgt $\sqrt{13,187} = 3,6$ Tage. Aus dieser Planung kann man erkennen, daß mit mehr als 68 Prozent Wahrscheinlichkeit[8] alle Aktivitäten im Zeitraum vom 18. Tag bis zum 25. Tag abgeschlossen werden können. Der Projektverantwortliche kann durch Vergleich der aktuellen Zeit mit dem Erwartungswert jeder Aktivität sofort feststellen, ob sich eine Verzögerung abzeichnet und welche Aktivität hauptsächlich dafür verantwortlich ist.

□

4.2.3 Wahrscheinlichkeitsaussagen mit PERT

Die Methode PERT wird aus zwei Gründen dem einfacheren CPM-Verfahren vorgezogen.

1. Durch die Schätzung der Unter- und Obergrenzen der Prozeßdauer können auch Aktivitäten, für die keine Erfahrungwerte vorliegen, beurteilt werden. Das Gesamtergebnis wird durch eine ungenaue Einschätzung der Bearbeitungsdauer nicht wesentlich verfälscht.

2. Aufgrund der stochastischen Berechnung der Zeitgrenzen können Wahrscheinlichkeitsaussagen über interessierende Aspekte des Projekts getroffen werden [DJOR90].

[8] Die Wahrscheinlichkeit P, daß ein Wert X innerhalb der σ-Umgebung liegt, ist ungefähr $0,6827$, also $P(|X-\mu| < \sigma) \approx 0,6827$ [DP88, JB86]

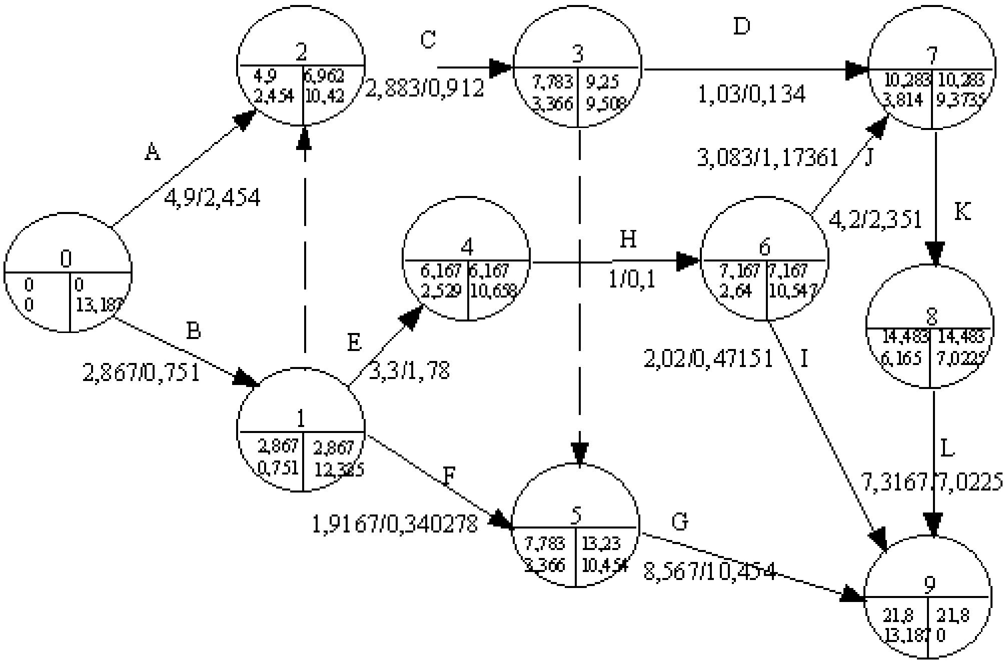

Abbildung 4.6: PERT mit Mittelwert und Varianz

Durch einfache Berechnungen ist es möglich, Fragen über die Gesamtdauer des Prozesses, verspätete Aktivitäten oder kritisch werdende Teile des Netzes fundiert zu beantworten.

Da die Zeitdauern der Aktivitäten für die Berechnung von Wahrscheinlichkeitsaussagen durch die Normalverteilung approximiert werden, kann mit der Dichtefunktion

$$f(x) = \varphi(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, -\infty < x < \infty$$

und deren Stammfunktion

$$F(x) = \phi(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{x} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} dt$$

gearbeitet werden, wobei hier x die Zufallsvariable, μ der Erwartungswert der Verteilung und σ die Standardabweichung ist (siehe z. B. [JB86]).

Durch die Zurückführung auf die *standardisierte Normalverteilung*[9] $N(0,1)$ kann jede normalverteilte Zufallszahl X durch die Gleichung $Y = \frac{X-\mu}{\sigma}$ in eine standardisierte Zufallsgrösse Y umgeformt werden.

[9]Eine standardisiert Normalverteilung hat den konstanten Erwartungswert $\mu = 0$ und die Standardabweichung $\sigma = 1$.

Für die Errechnung von Wahrscheinlichkeiten durch die Stammfunktion $\phi(Y)$ gibt es in der angeführten Literatur Wahrscheinlichkeitstabellen, die die entsprechenden angenäherten Werte für jedes $\phi(Y)$ beinhalten.

Somit berechnen sich Wahrscheinlichkeiten für die Variable x bei der Kenntnis von μ und σ wie folgt:

$$P(x < c) = \begin{cases} \phi\left(\frac{x-\mu}{\sigma}\right) & \text{wenn } c \geq \mu \\ 1 - \phi\left(\frac{|x-\mu|}{\sigma}\right) & \text{wenn } c < \mu \end{cases}$$

$$P(a \leq x \leq b) = \phi\left(\frac{b-\mu}{\sigma}\right) - \phi\left(\frac{a-\mu}{\sigma}\right)$$

Die praktische Anwendung der Formeln wird im folgenden Beispiel demonstriert.

Beispiel 4.4 *Aufgabenstellungen mit Wahrscheinlichkeiten*

Ausgehend von den Daten des Beispiels 4.3 kann man folgende Fragen über den Plan stellen.

1. *Wie groß ist die Wahrscheinlichkeit, daß das Projekt länger als 23 Tage dauert?*

   Gesucht ist $P(x > 23)$ eines Projekts mit $\mu = 21,8$ und $\sigma = 3,632$. Durch die Symmetrie der Dichtefunktion φ der Normalverteilung ergibt sich die Äquivalenz $\phi(x > c) = 1 - \phi(x \leq c)$. Werden die aktuellen Werte des Beispiels in die Gleichung eingesetzt, so führt das zu folgendem Ergebnis.

   $$P(x > 23) = 1 - \phi\left(\frac{23 - 21,8}{3,632}\right) = 1 - \phi(0,3304) \approx 1 - 0,6293 \approx 0,3707.$$

   Die Wahrscheinlichkeit, daß das Projekt länger als 23 Tage dauert, beträgt 37,7 Prozent.

2. *Wie groß ist die Wahrscheinlichkeit, daß das Projekt zwischen dem 17. und 25. Tag fertig wird?*

   Wie o. a. berechnet man die Wahrscheinlichkeit, daß ein Ereignis zwischen zwei Zeitpunkten liegt, durch die Bildung der Differenz der Wahrscheinlichkeiten beider Zeitpunkte.

   $$\begin{aligned} P(17 \leq x \leq 25) &= \phi\left(\tfrac{25-21,8}{3,632}\right) - \left(1 - \phi\left(\tfrac{|17-21,8|}{3,632}\right)\right) = \\ &= \phi(0,881) - (1 - \phi(1,321)) \approx 0,8106 + 0,9066 - 1 \approx 0,7172 \end{aligned}$$

   Die Wahrscheinlichkeit, daß sich der Endtermin des Prozesses im geforderten Intervall befindet, liegt ungefähr bei 72 Prozent.

3. *Gesucht ist die Wahrscheinlichkeit, daß der Zustand 3 kritisch wird?*

   Diese Fragestellung ist deshalb wichtig, da sie die sonst eher unbeachteten nichtkritischen Aktivitäten auch in eine Engpaßanalyse mit einbezieht.

Wie auf Seite 42 erklärt wurde, bewertet man eine Aktivität dann als kritisch, wenn deren Spielraum $TF = L_j - D_{ij} - E_i$ Null ist.

Als Wahrscheinlichkeit wird die Frage formuliert als $P(TF_3 \leq 0)$. Die Daten erhält man durch die Berechnung aus dem Netz. Der Mittelwert μ ergibt sich durch $9,25 - 7,783 = 1,467$, die Standardabweichung σ errechnet sich aus dem Wert $\sqrt{9,508 + 3,366} = 3,588$ für den Zustand 3.

$$P(TF_3 \leq 0) = \phi\left(\frac{0 - 1,467}{3,588}\right) = 1 - \phi\,(0,409) \approx 0,6591.$$

Die Wahrscheinlichkeit, daß der unkritische Zustand 3 doch kritisch wird, liegt in unserem Beispiel bei 65,91 Prozent!

□

Das Kapitel 4 beschäftigte sich mit der Fragestellung, wie strukturelle Zeitvorgaben, die hier als *inhärente Restriktionen* bezeichnet werden, erkannt, berechnet und ausgewertet werden können. Die Methode der Netzplantechnik wurde als geeigneter und erprobter Ansatz bewertet, mit dieser Problematik umzugehen.

Für den Einsatz im Workflow-Bereich genügen aber die Mittel der Netzplantechnik nicht. Vor allem das fehlende Konzept für den Umgang mit Alternativen stellt eine zu große Einschränkung dar. Deshalb wird im nächsten Kapitel die Netzplantechnik um die Behandlung von Alternativen erweitert.

Kapitel 5

ePERT – Erweiterung der Netzplantechnik

5.1 Alternativentransformation

Die Darstellung und Berechnung alternativer Prozeßausführungen ist in der Netzplantechnik nicht vorgesehen. Es ist nur möglich, Projekte zu untersuchen, die einen unveränderlichen Ablauf haben. Falls sich durch eine Entscheidung während des Projektablaufs einschneidende Änderungen der Prozeßstruktur ergeben, ist der Netzplan nicht mehr gültig. Eine Anpassung an eine neue Situation geschieht durch den Neuentwurf des Prozesses auf der Basis des aktuellen Wissens. Die Ergebnisse des vorher berechneten Plans sind kaum mehr wiederzuverwenden.

Da aber eine Möglichkeit gefunden werden muß, Alternativen zu verarbeiten, ergeben sich bei der Transformation mehrere Probleme.

5.1.1 Synchronisierende Scheinaktivitäten

Die erste Schwierigkeit ergibt sich durch die Intention, ePERT-Netze automatisch aus Prozeßbeschreibungen zu generieren. Durch die Aufspaltung der Prozesse in mehrere, vielleicht sogar hierarchisch gegliederte Abarbeitungszweige, müssen diese nach dem Ende einer verzweigenden Struktur (Alternative und Nebenläufigkeit) wieder synchronisiert werden. Im konventionellen Netzplan geschieht das durch Einfügen von Scheinaktivitäten, die Arbeitsschritte mit einer Ausführungszeit von Null darstellen. Diese Vorgehensweise ist für die automatische Konvertierung nicht geeignet, da diese in mehreren Phasen durchgeführt wird. Die erste Phase ist die Erzeugung der Netzstruktur, in weiteren Phasen werden dann erst die Zeitgrenzen errechnet. Zum Zeitpunkt der Strukturgenerierung liegt also noch keine Zeitinformationen vor. Diese Zeitinformationen werden beim manuellen Entwurf als Heuristiken benötigt, die das korrekte Einfügen der Scheinaktivitäten erlauben.

Aus diesem Grund wird bei der automatischen Transformation ein anderer Weg eingeschlagen. Für jeden Startzustand einer verzweigenden Struktur wird gleichzeitig auch ein entsprechender Endzustand erzeugt. Dieser wird nach dem Abschluß eines Zweiges über Scheinaktivitäten erreicht. Der Endzustand entspricht dem `end`-Statement prozeduraler Programmiersprachen[1]. Die Abbildung 5.1 zeigt diese Umformung.

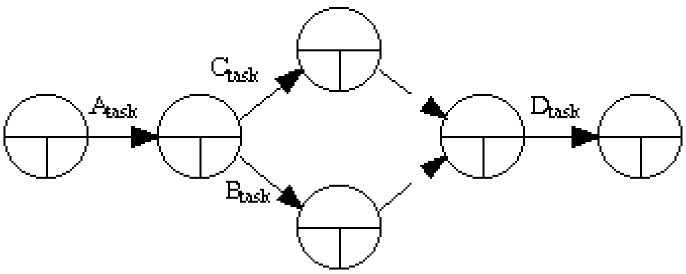

Abbildung 5.1: Transformation einer nebenläufigen Aktivität

Laut der einschlägigen Literatur [Zim92, Phi86, DJOR90] entspricht dieser Ansatz nicht den Regeln, die für die Konstruktion eines konventionellen Vorgangpfeilnetzplans gelten.

1. Ein Zustand muß Endpunkt zumindest einer realen Aktivität sein.

2. Die Anzahl der Scheinaktivitäten im Netzplan muß minimiert werden.

Die Einhaltung dieser Konstruktionsgrundsätze soll dazu beitragen, durch die Minimierung der Scheinaktivitäten das Optimierungspotential auszuschöpfen. Jede Scheinaktivität synchronisiert nebenläufige Zweige des Netzes. Es kann nun der Fall eintreten, daß eine Aktivität unnötigerweise auf das Ende einer anderen warten muß, obwohl durch eine alternative Netzplanstruktur mit gleichfalls korrekter Semantik keine Wartezeit nötig wäre. Durch die Minimierung der Anzahl der Scheinaktivitäten werden auch Wartezeiten minimiert.

Die Abbildung 5.2 zeigt eine den herkömmlichen Regeln folgende Netzstruktur für eine Nebenläufigkeit. Dadurch kann man auf einen Zustand und eine Scheinaktivität verzichten. Doch die Berechnung der **Richtung** der Scheinaktivität ist in diesem Falle schwieriger. Neben der vorgegebenen Reihenfolgebeziehung, die schon zur Entwurfszeit feststeht, müssen schon in dieser Phase Zeitwerte zur Verfügung stehen. Die Richtung der Scheinaktivität wird nämlich durch das Maximum der frühest möglichen Beginnzeitpunkte $E_{i,\ldots,k}$ aller nebenläufigen **Endknoten** $i, \ldots, k$ bestimmt. Als Endknoten werden die Endzustände der

[1]Durch die Einführung von expliziten Endzuständen von Verzweigungen entstehen im Netzgraphen *Artikulationen*. Eine Artikulation ist gegeben, wenn durch die Wegnahme des Knotens der Graph in Teilgraphen zerfällt [DP88].

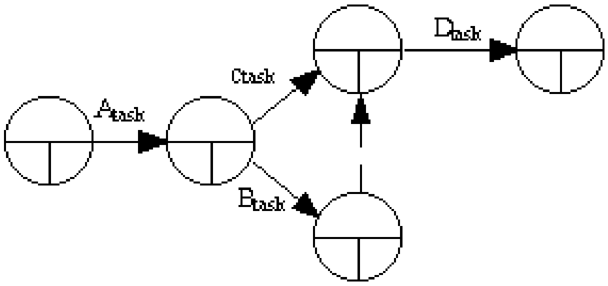

Abbildung 5.2: Konventionelle, korrekte Umwandlung

letzten Tasks der nebenläufigen Verzweigung bezeichnet. Alle Scheinaktivitäten müssen nun zu diesem „Maximumknoten“ führen. Der Grund für die Wahl des Maximumknotens als Endknoten ist, daß dadurch für alle anderen Endknoten Schlupfzeiten entstehen, die Spielräume für die Ressourceneinteilungen, Bearbeitungszeiten und Steuerungsmechanismen ermöglichen.

Da aber in dieser Phase der Transformation die Einbringung von Zeitinformation unnötig großen Aufwand mit sich bringt, wird die o. a. Umwandlung mit einem expliziten Endknoten bevorzugt. Das hat mehrere Vorteile.

- Einfachere und verständlichere Umwandlung der Prozeß-Logik.
- Automatische Synchronisation der Verzweigungen.
- Kein Reorganisationsaufwand während der Transformation bei aktualisierten Zeitwerten.

Trotz der dadurch erhöhten Anzahl von Scheinaktivitäten und Zuständen hat dieses Vorgehen sonst keine nennenswerten Nachteile, da das Endergebnis und die Interpretation darunter nicht leiden. Außerdem macht die später angeführte Erweiterung um alternative Konstrukte die ausdrückliche Darstellung von Endzuständen notwendig.

5.1.2 Vervielfältigung von Netzplänen

Der einfachste Weg, trotz des fehlenden Konzepts Alternativen in einen Netzplan zu übernehmen, wurde als Ansatz bereits oben angedeutet. Für jede abzubildende Alternative wird ein **eigener** Netzplan erstellt. Dadurch können alle möglichen Wege, die der Prozeß nehmen kann, strukturell dargestellt werden. Auch deren Zeitwerte sind in jedem einzelnen Plan einfach zu berechnen. Wenn beim Einlesen der Prozeß-Struktur eine Alternative, also ein `if`-Knoten, gefunden wird, müssen **alle** bisher generierten Netzpläne verdoppelt werden, um die weiteren Wege verwalten zu können. Das Prinzip dieser Berechnung ist in der Abbildung 5.3 zu sehen.

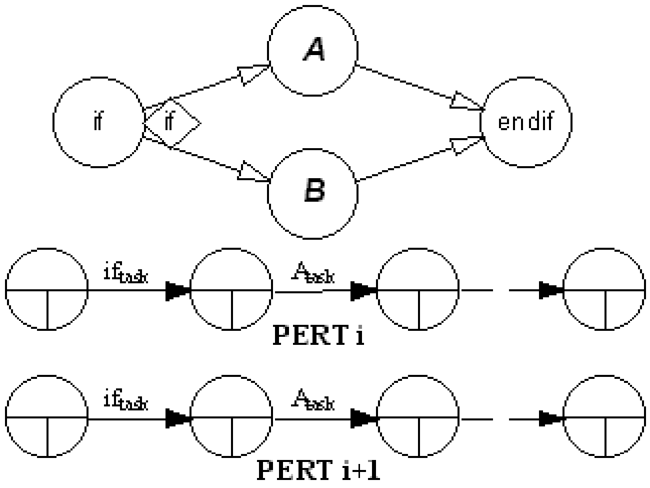

Abbildung 5.3: Alternative durch Vervielfachung

Diese Umformung ist korrekt, da sie die tatsächliche Auswirkung einer aufgetretenen Alternative widerspiegelt. Trotzdem ist dieser Ansatz nicht akzeptabel, da sofort einsichtig ist, daß der **Speicherbedarf und Rechenaufwand** auch bei kleinen Prozeßketten mit der entsprechenden Anzahl von Alternativen enorm wächst. Durch die Verdoppelung der Alternativen kommt es zu einem exponentiellen Wachstum der Anzahl der benötigten Netzpläne.

Dazu stellen wir einige Überlegungen an. Die Anzahl der Wege in einem Prozeß ist von dessen Struktur abhängig. Zur Schätzung dieser Zahl ist eine Umformung in einen Alternativenbaum notwendig. Ein Alternativenbaum stellt alle möglichen Verzweigungen des Prozeßablaufs übersichtlich dar. Die Umwandlung erfolgt durch eine Traversierung der Prozeßstruktur, wobei für jede Alternative ein Nachfolgeknoten im Alternativenbaum erzeugt wird.

Die Abbildung 5.4 zeigt an einem Beispiel die Umwandlung eines Workflow-Prozesses in einen Alternativen-Baum.

Die Anzahl x der möglichen Wege in einem Prozeß wird jetzt anhand des Alternativenbaums näherungsweise durch die Gleichung

$$x \approx b^t \tag{5.1}$$

berechnet. Der Exponent t ist die „durchschnittliche" Tiefe des Alternativenbaums, die Basis b entspricht dem Durchschnitt der Anzahl der Verzweigungen im Prozeß. Diese Basis ist auf jeden Fall $b \geq 2$, denn jede `if`-Struktur erzeugt zumindest zwei Wege, auch wenn **kein** `else` vorhanden ist, da auch das Überspringen einer Struktur einen neuen Weg darstellt.

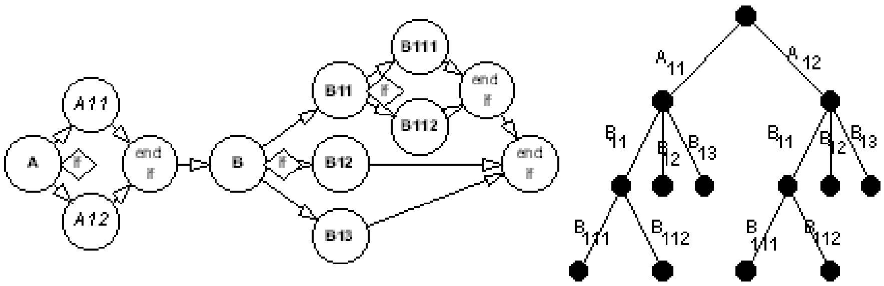

Abbildung 5.4: Workflow und zugehöriger Alternativenbaum

Die Höhe des Alternativenbaums wird durch die Anzahl der durchzulaufenden `if`s bestimmt. Da dieser Baum in den meisten Fällen nicht ausgeglichen ist, gelten folgende Grenzen, zwischen denen die Zahl der Wege liegt. Die geringste Höhe des Alternativenbaums wird durch jenen Ausführungspfad bestimmt, der durch die minimale Anzahl von Alternativen läuft. Im Beispiel der Abbildung 5.4 ist das der Weg von $\{A_{11} \mid A_{12}\}\{B_{12} \mid B_{13}\}$ mit nur zwei Alternativen. Das bedeutet die Anzahl der Wege ist auf **jeden Fall** $x \geq 2^2$. Da die maximale Anzahl der exekutierbaren Alternativen 3 ist, gilt sogar $x > 2^2$. Die Oberschranke kann durch die Verwendung des größten Grades[2] aller Knoten des Baumes als Basis bestimmt werden.

Wenn $\hat{t}$ die maximale Tiefe des Baumes und $\hat{b}$ der größte Grad des Baumes ist, dann ist die maximale Anzahl der Wege $x_{\max} = \hat{b}^{\hat{t}}$.

Die Obergrenze der Anzahl der Wege wird aber nur dann erreicht, wenn alle Verzweigungen vom gleichen Grad sind. In der Praxis wird die Anzahl der Wege weit unter dieser Grenze liegen.

Bei der Bezeichnung der minimalen Tiefe des Alternativenbaums mit $\check{t}$ liegt also die Anzahl der Wege zwischen der Ober- und der Untergrenze.

$$2^{\check{t}} \leq x \leq \hat{b}^{\hat{t}}$$

Für die Bewertung der Berechenbarkeit ist vor allem die unterste Schranke von Bedeutung. Durch das exponentielle Wachstum wird auch sie problematisch, vor allem, weil diese Schätzung in den meisten Fällen sehr optimistisch ist.

Beispiel 5.1 *Schätzung der Anzahl der Wege*

Ausgangspunkt ist der Alternativenbaum der Abbildung 5.4. Die minimale Höhe beträgt 2, die maximale Höhe ist 3. Der maximale Grad $\hat{b}$ der Knoten im Baum ist

[2]Der Grad $d(x)$ eines Knotens x in einem Graphen ist die Anzahl der mit x inzidenten Kanten, also die von x ausgehenden oder zu x führenden Kanten.

ebenfalls 3. Durch die Anwendung der o. a. Formel können die Unter- und Obergrenzen bestimmt werden.

$$2^{\check{t}} \leq x \leq \hat{b}^{\hat{t}} = 2^2 \leq x \leq 3^3$$

Als sehr optimistischer minimaler Wert ergibt sich eine Untergrenze von 4 Ausführungspfaden in diesem Prozeß. Konkret heißt das, es sind für diesen Prozeß mindestens vier Netzpläne zu erstellen und zu verwalten.

□

Nun stellt sich die Frage, wie tief ein Alternativenbaum werden darf (wie viele `if`-Konstrukte durchlaufen werden können), um noch eine vernünftige Anzahl von Netzplänen zu bewältigen. Durch die Umkehrfunktion der Gleichung 5.1 kann die durchschnittliche Tiefe $\bar{t}$ eines Alternativenbaums, die gleichbedeutend mit der Anzahl der zu optimierenden Netze ist, errechnet werden. Gegeben ist die untersuchte Anzahl der Netzpläne x und die durchschnittliche Verzweigungszahl $\bar{b}$.

$$\bar{t} = \frac{\log x}{\log \bar{b}}$$

Beispiel 5.2 *Berechnung der Tiefe*

Bei einer maximalen Anzahl von 1 Million zu bewältigender Netzpläne und einer durchschnittlichen Verzweigung von 3 ist die Tiefe des Alternativenbaums auf ca. $\bar{t} \approx \frac{\log x}{\log \bar{b}} = \frac{\log 1.000.000}{\log 3} = \frac{6}{0.47712} = 12,6$ beschränkt.

Bei einer durchschnittlich durchlaufenen Alternativenanzahl von 13 ist der Verwaltungsaufwand für die Erstellung und Berechnung der Netzpläne nicht mehr machbar.

□

Aus diesen Beispielen ist ersichtlich, daß die Verwaltung von Alternativen von Prozessen durch die Vervielfältigung von Netzplänen nicht sinnvoll ist. Schon durch einen durchschnittlich komplexen Prozeßablauf mit wenigen Alternativen wird der Speicherbedarf enorm groß. Für nur einen Prozeßtyp wäre es denkbar, diesen Aufwand auf sich zu nehmen. Doch in der Praxis sind viele verschiedene Prozeßtypen zu verwalten.

Auch bei der Zuordnung der einzelnen Pläne zu den Laufzeitinstanzen ergeben sich durch den verstärkten Verwaltungsaufwand erhebliche Probleme.

Aus diesem Grund müssen andere Lösungswege gefunden werden, die diese Nachteile vermeiden und trotzdem aussagekräftige Informationen liefern. Die Sprache der Netzplantechnik muß um einige Elemente erweitert und modifiziert werden, um die korrekte Umformung von auftretenden Alternativen zu erlauben.

5.1.3 Modifikation der Netzplantechnik

Alternativen stellen die unterschiedlichen Ablaufwege dar, die eine Prozeßinstanz während ihrer Laufzeit beschreiten kann. Jeder Weg führt zu einer anderen Ablaufstruktur, auch die Ergebnisse der Wege sind unterschiedlich. Eine Darstellung, die alternative Verzweigungen zuläßt, ist deshalb eine **Kumulation** aller möglichen Wege. Das ist auch der Ansatz, der bei der Adaption der Netzplantechnik verfolgt wird. Da es nicht möglich ist, die Informationen aller Wege zu verwalten, muß ein Kompromiß gefunden werden.

Im Zusammenhang mit Zeitschranken für Tasks, also elementaren Prozeßschritten, sind ab nun nur noch der Weg mit der **minimalen** und **maximalen** Länge von Interesse. Der minimale Weg durch das Netz beschreibt diejenige Abarbeitungsfolge, die zur kürzesten Bearbeitungszeit des Prozesses führt. Die maximale Zeit wird für jene Folge von Aktivitäten benötigt, die bei jeder Alternative den Zweig mit der längsten Bearbeitungsdauer ausführt.

Um die notwendigen Daten verwalten zu können, muß der Zustandstyp des Netzplans um genau diese Felder erweitert werden. Das Feld, das die minimalen Werte beinhaltet und deshalb den besten Fall der Prozeßbearbeitung darstellt, wird *best case* benannt. Analog dazu wird das Feld mit der maximalen Länge mit *worst case* bezeichnet. Für jede der beiden Eintrittszeiten eines Zustands, den frühesten Eintrittszeitpunkt E und den spätesten Eintrittszeitpunkt L muß jetzt der beste und schlechteste Fall protokolliert werden.

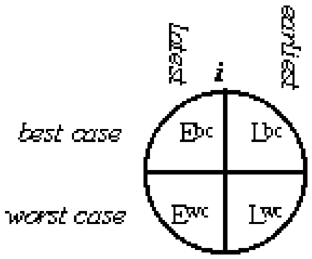

Abbildung 5.5: Erweiterter Zustandstyp

Die Abbildung 5.5 zeigt den erweiterten Zustandstypen. Die Abkürzungen E^{bc} und E^{wc} stehen für den frühesten (*earliest*) Eintrittszeitpunkt im besten, bzw. schlechtesten Fall, während L^{bc} und L^{wc} für die jeweiligen Werte der spätesten (*latest*) Zeiten stehen.

Als nächste Veränderung der Netzplantechnik zur Adaption an die Anforderungen ist die Verwendung von Scheinaktivitäten als Synchronisationsmechanismus zu nennen. Diese Technik wurde bereits auf der Seite 51 vorgestellt und ausführlich begründet.

Um die veränderte Struktur und unten angeführten Modifikationen der Berechnungsvorschriften von der konventionellen Netzplantechnik abzugrenzen, wird in der weiteren

Arbeit von einem *ePERT*-Netz (*extended Programm Evaluation and Review Technique*) gesprochen.

5.1.4 Strukturelle Umwandlung von Alternativen

Als erste einschneidende strukturelle Änderung werden im Gegensatz zu konventionellen Netzplänen Alternativen erlaubt. Sie stellen unterschiedliche Exekutionspfade dar, von denen pro Extension immer nur einer beschritten werden darf[3]. Jeder ausgeführte Alternativenzweig entspricht dabei einem eigenen herkömmlichen Netzplan.

Die Bedeutung des Netzplans ändert sich dadurch insofern, daß im *ePERT* ein Weg nicht mehr Repräsentation **einer** Ausführung, sondern die **Verdichtung aller Exekutionen** ist, die über diese Alternative laufen.

Die strukturelle Repräsentation ist die gleiche, die auch für Nebenläufigkeiten im konventionellen Netzplan gilt. Für jeden alternativen Abarbeitungszweig wird im ePERT eine Verzweigung eingefügt. Der einzige Unterschied besteht darin, die interne Bezeichnung als Alternative sowohl am Beginn und auch am Ende der Struktur zu protokollieren. Die graphische Repräsentation ist in der Abbildung 5.6 zu sehen. Die Kennzeichnung einer Alternative im Netz geschieht durch eine bogenförmige Verbindung, die über die Aktivitäten gelegt wird, welche direkt nach dem `if`-Zustand gereiht sind. Im Beispiel wird dieser Knoten mit „1" bezeichnet.

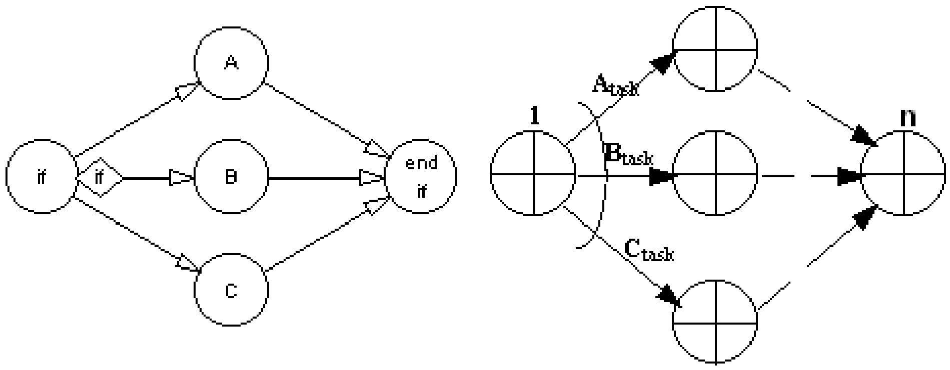

Abbildung 5.6: Alternative im erweiterten Netzplan

Die Interpretation des Endzustands der Struktur unterscheidet sich jedoch von jener der Nebenläufigkeit. Während dort die Endpunkte Synchronisationsstellen sind, an denen auf das Ende aller Zweige gewartet werden muß, bedeuten Endzustände von Alternativen **Kumulationen der Ergebnisse** der bisherigen Wege.

Das äußert sich vor allem in den Berechnungsvorschriften, die für die Ermittlung der frühesten und spätesten Zeitpunkte im besten und schlechtesten Fall gelten. Es werden

[3]Dies entspricht der Semantik eines *exclusive or* für den Verzweigungsknoten.

nur mehr die Ergebnisse des bis dahin kürzesten und längsten Weges weiterverwendet. Alle Zeitschranken für Exekutionspfade, die innerhalb dieser Intervallgrenzen liegen, sind für die weitere Berechnung nicht mehr von Bedeutung.

5.2 Berechnungsvorschriften im *ePERT*-Netzplan

Im vorigen Abschnitt wurde gezeigt, wie die Struktur eines konventionellen Vorgangspfeilnetzplans an die Anforderungen, die aus der Darstellung von allgemeinen Prozessen entstehen, angepaßt werden muß. Die nächste Änderung betrifft die Berechnungsvorschriften, die sich aus der geänderten Bedeutung eines ePERT-Netzplans ergeben. Da es sich hierbei nicht mehr um die Darstellung **aller** zu verrichtenden Aktivitäten, sondern um **mögliche** Aktivitäten handelt, die ein Prozeß ausführen kann, ändert sich auch die Interpretation und damit die Berechnung der frühesten und spätesten Eintrittszeitpunkte E und L für einen Zustand.

5.2.1 Hinrechnung

Sequenz

Die Berechnung der frühesten Werte E einer Sequenz unterscheidet sich nicht von der eines herkömmlichen Netzes[4]. Es muß nur bedacht werden, daß beide Werte E^{bc} und E^{wc} sich aus der Summe der jeweiligen Vorgänger mit den Werten der Aktivität ergeben.

Früheste Beginnzeitpunkte E_j	
best case	$E_j^{bc} = E_i^{bc} + D_{ij}$
worst case	$E_j^{wc} = E_i^{wc} + D_{ij}$

Tabelle 5.1: Früheste Beginnzeitpunkte nach Sequenz

Dabei ist der Zustand j der **direkte** Nachfolger des Zustands i, d. h. i ist der Anfangszustand und j der Endzustand der Aktivität (ij). Die Dauer der Aktivität wird durch D_{ij} bestimmt.

Alternative

Im Endzustand j einer alternativen Struktur müssen die Werte des bis zu diesem Punkt ermittelten längsten und kürzesten Weges festgehalten werden. Der Wert des kürzesten

[4]Die Bezeichnungen E und L stehen sowohl für den Mittelwert **und** die Varianz eines Zustands. In der weiteren Folge der Arbeit wird darauf hingewiesen, wenn sich die Berechnungen von μ und σ^2 unterscheiden.

Weges E_j^{bc} errechnet sich aus dem Minimum der Erwartungswerte der ankommenden *best case*-Wege, E_j^{wc} aus dem Maximum der Erwartungswerte der *worst case*-Wege.

best case	$E_j^{bc} = \min(\{E_\varsigma^{bc} + D_{\varsigma j}\})$	$\forall$ direkten Vorgänger ς von j
worst case	$E_j^{wc} = \max(\{E_\varsigma^{wc} + D_{\varsigma j}\})$	$\forall$ direkten Vorgänger ς von j

Tabelle 5.2: Früheste Beginnzeitpunkte nach Alternative

Die Variable ς steht für alle Zustände, die direkter Vorgänger des Zustands j sind.

Beispiel 5.3 *Kürzester und längster Weg nach Alternative*
In einem Netzplan wurden bis zu einem Knoten i 10 Zeiteinheiten als Erwartungswert des kürzesten Weges und 16 für den längsten Weg berechnet. Nun stellt sich die Wahl, die Prozesse A, B oder C zu starten, wobei die entsprechenden Zeitdauern dieser Aktivitäten mit 5, 8 und 2 festgelegt wurden.

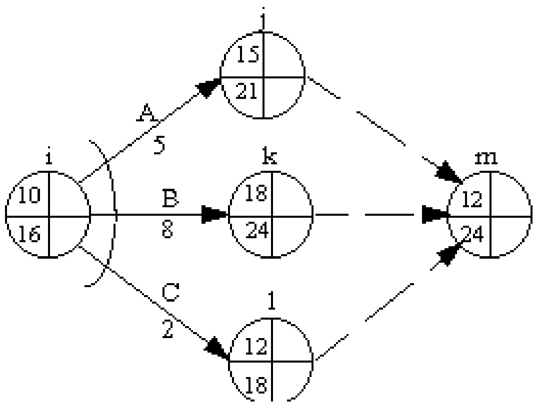

Abbildung 5.7: Früheste Startintervalle nach Alternative

Wie in der Abbildung 5.7 zu sehen ist, ergibt sich der Wert des besten Falls mit 12 und der schlechteste mit 24 Zeiteinheiten. Die Ergebnisse des Knotens j werden nicht mehr im erweiterten Netzplan berücksichtigt, da diese Werte innerhalb der Intervallgrenzen des *best* und *worst case* liegen. □

Nebenläufigkeit

Nebenläufige Kontrollstrukturen müssen ebenfalls über Scheinaktivitäten zusammengeführt werden. Hier gibt es, wie bei der Sequenz, bis auf die Berücksichtigung von zwei Werten für den *best* und *worst case* keine Änderung gegenüber dem Vorgehen bei herkömmlichen Netzplänen.

Auch in dieser Tabelle werden der Endzustand mit j und die direkten Vorgängerzustände mit ς bezeichnet.

best case	$\max(\{E_\varsigma^{bc} + D_{\varsigma j}\})$	$\forall$ direkten Vorgänger ς von j
worst case	$\max(\{E_\varsigma^{wc} + D_{\varsigma j}\})$	$\forall$ direkten Vorgänger ς von j

Tabelle 5.3: Früheste Beginnzeitpunkte nach Nebenläufigkeit

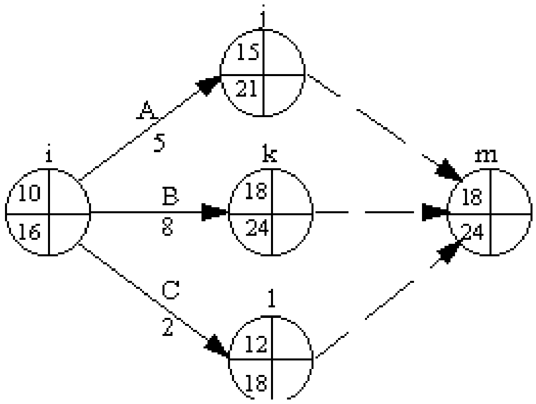

Abbildung 5.8: Früheste Startintervalle nach Nebenläufigkeit

Die Abbildung 5.8 zeigt die Auswirkung der Synchronisation einer nebenläufigen Struktur im ePERT.

5.2.2 Rückrechnung

Nach der Errechnung der Werte E für alle Zustände, muß der Plan rückgerechnet werden. Ziel der Rückrechnung in einem erweiterten Netzplan ist, neben der Beantwortung der auf Seite 39 angeführten Fragen,

- die Anzeige der noch wählbaren Alternativen, die abhängig vom aktuellen Zeitpunkt zeitlich noch möglich sind, und
- die Benennung des Zeitpunkts für einen Task, der noch alle zukünftigen Alternativen wählbar macht.

Die Werte $E_{\max}$ des letzten Zustandes max werden zur Vorgabe der Projektendzeit, also $L_{\max}^{bc} := E_{\max}^{bc}$ für den besten und $L_{\max}^{wc} := E_{\max}^{wc}$ für den schlechtesten Fall. Dabei ist zu beachten, daß die Varianz des Prozesses auf 0 und nicht auf die Varianzen von bc und wc $EVar_{\max}$ gesetzt wird, da der Endtermin keiner Streuung unterliegt und fixiert ist.

Ausgehend von diesen Werten werden nun alle spätest erlaubten Anfangszustände der Kontrollstrukturen berechnet.

Sequenz

Die Werte L einer Sequenz werden durch eine einfache Rückrechnung bestimmt. Dabei ist i der Anfangs- und j der Endzustand einer Aktivität (i, j) mit der Dauer D_{ij}. Eine einfache oder Sequenz-Rückrechnung liegt vor, wenn die Werte eines Zustands, der Anfang genau einer Aktivität ist, bestimmt werden.

***best case*:** Der Wert L_i^{bc} berechnet sich aus der Differenz $L_j^{bc} - D_{ij}$. Hier kann aber der Sonderfall eintreten, daß der Wert $E_i^{bc} > L_i^{bc}$, d. h. der frühest erlaubte Zeitpunkt ist größer als der spätest erlaubte Zeitpunkt für den Eintritt des Zustands! Diese Situation zeigt sich immer bei jenen Zuständen, die

1. die letzten Zustände **vor** dem Ende von Alternativen sind, und
2. die **nicht** am **minimalen** Weg liegen. Da bei der Rückrechnung der Werte L^{bc} vom Minimum ausgegangen wird, sind diejenigen Zustände, die nicht am minimalen Weg liegen, von diesem Phänomen betroffen.

In diesem Fall wird der Wert von L_i^{bc} auf E_i^{bc} gesetzt. Siehe dazu die Abbildung 5.9, bei der der Zustand α nicht am minimalen Weg liegt und deshalb koordiniert werden muß.

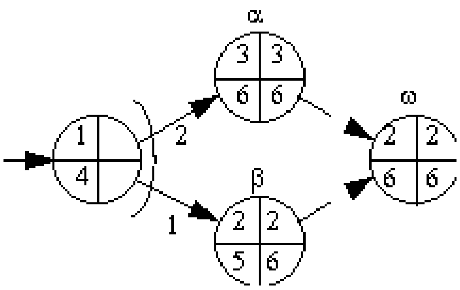

Abbildung 5.9: Koordination nach einfacher Rückrechnung

Dieser Schritt wird durch folgende Überlegungen gerechtfertigt:

1. Der früheste Beginn einer Aktivität kann niemals zeitlich nach dem spätesten Beginn der Aktivität liegen.
2. Der Fall einer *negativen* Schlupfzeit für einen Zustand kann nur dann eintreten, wenn der betreffende Knoten nicht am **minimalen** Weg liegt. Der minimale Weg ist der kritische Pfad des Prozesses mit der kürzesten Dauer.

   Dieser Weg liegt zwar nicht am minimalen, aber an einem **kritischen Pfad** eines anderen Prozesses, der im erweiterten Netzplan vorhanden ist! In weiterer

Folge bedeutet das, ein ePERT-Netz kann **mehrere** kritische Pfade **unterschiedlicher** Länge besitzen, da ein erweiterter Netzplan die Komprimierung vieler konventioneller Pläne darstellt.

Eine negative Differenz zwischen dem frühesten und spätest möglichen Starttermin einer Aktivität zeigt demnach einen Zustand an, der am kritischen Pfad eines einzelnen Netzes, nicht aber am minimalen kritischen Pfad des erweiterten Netzplans liegt!

worst case: Die Rückrechnung zur Ermittlung des spätesten Starttermines im *worst case* muß nicht mit dem Wert E^{wc} koordiniert werden, da der Ausgangspunkt für die Rückrechnung das Maximum aller Weglängen ist. Deshalb kann niemals ein negativer Wert durch eine Differenzenbildung auftreten. Der Wert L_i^{wc} ergibt sich als einfache Differenz von $L_i^{wc} := L_j^{wc} - D_{ij}$.

Dieser Wert L_i^{wc} kann für jeden Zustand i als spätest möglicher Starttermin einer Aktivität interpretiert werden, ab dem die Länge des folgenden Weges durch das Netz **minimal sein muß**, um den spätesten Endtermin (die Projekt*deadline*) halten zu können. Anders ausgedrückt heißt das, es gibt noch **zumindest eine** Abfolge von Arbeitsschritten, die den Projektendtermin einhalten läßt. Wenn ein Zustand zum Wert L^{wc} nicht erreicht wird, gibt es keine Abfolge von Arbeitsschritten mehr, die das Projekt nicht verzögert.

In der Abbildung 5.9 wird der Ausschnitt eines ePERT-Netzplans gezeigt, bei dem beide zum Endknoten ω führenden Wege kritische Pfade enthalten. Durch die Gleichheit der Werte E_β^{bc} und L_β^{bc} wird indiziert, daß dieser Zustand am minimalen kritischen Pfad liegt.

Die Varianzen in beiden Fällen bc und wc werden **addiert** und nicht subtrahiert, da ja die Streuung für jeden Schritt der Rückrechnung größer wird. In der Tabelle 5.4 werden die Berechnungsschritte zusammenfassend dargestellt.

$$\begin{aligned}
L_i^{bc} &= \begin{cases} L_j^{bc} - D_{ij}, & \text{wenn } L_j^{bc} - D_{ij} - E_i^{bc} \geq 0 \\ E_i^{bc}, & \text{sonst} \end{cases} \\
L_i^{wc} &= L_j^{wc} - D_{ij} \\
LVar_i &= LVar_j + DVar_{ij}
\end{aligned}$$

Tabelle 5.4: Rückrechnung nach Sequenz

Nebenläufigkeit

Die Rückrechnung für die Anfangsknoten von Nebenläufigkeiten in einem erweiterten Netzplan unterscheidet sich nicht vom herkömmlichen Netzplan. Die Erwartungswerte von L^{bc} und L^{wc} werden als späteste Startzeitpunkte für Zustände **separater** Netze betrachtet und können deshalb mit dem bereits bekannten Verfahren ermittelt werden:

$$\begin{aligned} L_i^{bc} &= \min(L_\sigma^{bc} - D_{i\sigma}) \\ L_i^{wc} &= \min(L_\sigma^{wc} - D_{i\sigma}) \end{aligned}$$

Tabelle 5.5: Nebenläufigkeit-Rückrechnung

Diese Zeitpunkte errechnen sich aus dem Minimum der Rückrechnung der ausgehenden Wege für den besten und den schlechtesten Fall.

Die Variable σ in der Tabelle 5.5 steht für alle Endzustände von nebenläufigen Aktivitäten, die den Startzustand i haben.

Alternativen

Für die Errechnung des spätest möglichen Startzeitpunktes von Aktivitäten einer Alternative müssen die beiden Fälle *worst case* und *best case* unterschiedlich behandelt werden.

Der späteste Startzeitpunkt für die Aktivitäten (i, σ) wird im besten Fall durch das Minimum der Menge $\{L_\sigma^{bc} - D_{i\sigma}\}$ bestimmt. Der Wert L^{bc} wird folgendermaßen interpretiert: Zu diesem Zeitpunkt können alle alternativen Aktivitäten aus der Menge $\{(i, \sigma)\}$ gestartet werden und **alle** zukünftigen Alternativen bleiben weiterhin wählbar, so daß der späteste Endtermin eingehalten wird.

$$\begin{aligned} L_i^{bc} &= \min\{(L_\sigma^{bc} - D_{i\sigma})\} \\ L_i^{wc} &= \max\{(L_\sigma^{wc} - D_{i\sigma})\} \end{aligned}$$

Tabelle 5.6: Alternativen-Rückrechnung

Für den Wert des *worst case* wird das Maximum von $\{L_\sigma^{wc} - D_{i\sigma}\}$ für den Zeitpunkt L_i^{wc} genommen. Dieser Zeitpunkt gibt an, daß beim Start der folgenden Aktivitäten es zumindest noch **einen** Weg durch das Netz gibt, der die Zeitvorgaben für das Projekt halten läßt.

5.3 Kritische Pfade im erweiterten Netzplan

Durch die Tatsache, daß ein erweiterter Netzplan die „Verdichtung“ aller möglichen Netzpläne darstellt, die sich aus der Wahlmöglichkeit bei Alternativen ergibt, kann er mehrere kritische Pfade haben.

- Der **minimale** kritische Pfad ist jener Weg durch das Netz, der die kürzeste Prozeßdauer hat. Die Länge ist der Wert des Feldes $L^{bc}_{\max}$ des letzten Zustands max des ePERT-Netzplans.
- Der **maximale** kritische Pfad ist der Weg, der entsteht, wenn bei jeder Alternative der Zweig mit der längsten Bearbeitungsdauer gewählt wird. Seine Länge steht im Feld $L^{wc}_{\max}$.
- Der **einfache** kritische Pfad ist ein Weg, der kritischer Pfad für ein Netz ist, daß nicht zur kürzesten oder längsten Bearbeitungsdauer führt.

Aus dem Graphen des Netzes kann nur der minimale kritische Pfad direkt abgelesen werden. Ausgehend vom letzten Zustand max wird jeder Knoten, dessen Werte E^{bc} und L^{bc} denselben Wert haben, zum minimalen kritischen Pfad hinzugefügt. Bei mehreren Knoten, die dieser Bedingung genügen, wird der Zustand mit dem kleinsten Wert E^{bc} bzw. L^{bc} genommen. Dieser gewählte Zustand ist dann der Ausgangspunkt für die Selektion des nächsten Zustands, bis der Anfang des ePERT-Netzplans erreicht wurde. Die Ergebnismenge enthält alle Zustände des kritischen Pfades.

Die anderen Wege entdeckt man, indem die alternativen Netze, die durch das ePERT repräsentiert sind, explizit berechnet und als separate Netzpläne dargestellt werden.

Wenn eine Aktivität ausgeführt wird, die am maximalen kritischen Pfad liegt, so bedeutet das nicht in jedem Fall, daß der Gesamtprozeß zur spätest möglichen Zeit beendet sein muß. Der Prozeßscheduler kann bei jeder alternativen Entscheidung einen Weg wählen, der nicht auf dem maximalen Weg liegt und dadurch die Gesamtlaufzeit verkürzen. Wenn aber ein Prozeß einen Zeitpunkt erreicht, der am maximalen Weg liegt, dann kann die kürzest mögliche Laufzeit nie mehr erreicht werden[5]!

Umgekehrt bedeutet es aber für den am minimalen Weg liegenden Prozeß nicht, daß er die maximale Laufzeit nie mehr erreichen kann, da Verzögerungen immer wieder auftreten können. Einem solchen Prozeß sind aber noch alle zukünftigen Alternativen offen. In weiterer Folge bedeutet das: eine Verzögerung schränkt die zukünftige Auswahl ein.

[5]Die Ausnahme ist der Spezialfall der Gleichheit des minimalen und maximalen Weges.

5.4 Einschränkung der Wahlmöglichkeiten

Der Wert L^{wc} eines Zustandsknotens zeigt jenen Zeitpunkt an, bei dem die Vorgabe der Ausführungszeit für den gesamten Prozeß noch zu halten ist, wenn ab nun der minimale Weg eingehalten wird.

Je näher die aktuelle Zeit an diese Obergrenze kommt, desto eingeschränkter wird die Wahl der Alternativen. Es kann eine Aktivität (i, j) zu einem Zeitpunkt t nur mehr dann gestartet werden, wenn die Ungleichung

$$L_j^{wc} \geq t + D_{ij}$$

gilt. Falls diese Bedingung nicht zutrifft, muß man eine andere Aktivität auswählen. Wenn $t \leq L_i^{wc}$ ist, gibt es noch zumindest **einen** nächsten Arbeitsschritt, der die Zeitvorgaben nicht verletzt.

Bei der Wahl einer Alternative wird die Einhaltung des erlaubten Weges vom Scheduler für jede Prozeßinstanz während ihrer Laufzeit überprüft.

Bevor im Abschnitt 7 auf die Anpassung und den Einsatz dieser Methode im Workflow-Bereich eingegangen wird, stellt das nächste Kapitel die Architektur des Workflow-Prototypen *Panta Rhei* vor. Dadurch sollen die Voraussetzungen geschaffen werden, die Integration der Netzplantechnik in ein Workflow-System nachvollziehen zu können.

Kapitel 6

Das Workflow-System *Panta Rhei*

Der Prototyp *Panta Rhei*[1] wurde an der Universität Klagenfurt entwickelt. Er stellt den Versuch dar, neue Konzepte umzusetzen, die durch die beginnende Akzeptanz des Internet zur Abwicklung von Geschäftsfällen [EGL96, GE95] und die Anforderungen an die Stabilität und transaktionale Sicherheit von WfMS [EL96, EL95a, EL95b, Lie95] wichtig wurden.

Der folgende Abschnitt beschreibt die Systemarchitektur von *Panta Rhei*. Die hervorstechendsten Merkmale dieser Architektur sind einerseits die Verwendung einer aktiven Datenbank zur Unterstützung der Realisierung der Workflow-Engine [Nek95], die Integration von Workflow-Transaktionen und weiters die Anbindung an das Internet durch Verwendung von HTML-Formularen als Datencontainer und WWW-Browsern[2] als Workflow-Clients [EGL96]. Das *World Wide Web* ist eine Entwicklung des Europäischen Forschungszentrums für Partikelphysik (CERN) und ermöglicht den einfachen Zugriff auf Dokumente, die über das Internet erreichbar sind.

6.1 Architektur

Die Abbildung 6.1 zeigt die Architektur des Workflow-Systems. Die drei Hauptkomponenten sind der HTTP-Server[3], die Workflow-Server und das Datenbankmanagementsystem.

HTTP-Server: Er ist für die Kommunikation mit den Workflow-Clients über die Verbindung eines Internet-Browsers zuständig. Seine Aufgabe ist außerdem, den korrekten Aufbau der versendeten und empfangenen Formulare, die dem HTML-Standard[4] ent-

[1] „Alles fließt ...“, ein Zitat, das Heraklit zugesprochen wird.

[2] Der englische Ausdruck *„... to browse“* bedeutet „umsehen, einsammeln“. Ein Browser ist demnach ein Software-Programm zur Sammlung und Darstellung von Informationen.

[3] Das *hyper text transfer protocoll* HTTP ermöglicht die einfache Kommunikation mit anderen Rechnern über das Internet.

[4] HTML *hyper text markup language* ist das Standardformat zur Darstellung von Text, Graphik und Multimedia-Daten im Internet [BLC95]

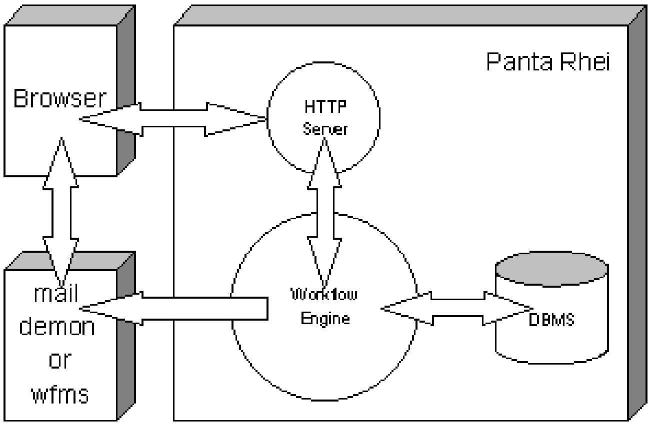

Abbildung 6.1: Basisarchitektur von Panta Rhei

spechen müssen, zu überprüfen. Jede Kommunikation mit den Clients erfolgt über diese Schnittstelle. Die Vorteile, die durch die Verwendung von Web-Browsern als Clients erwachsen, sind vielfältig:

- **Plattformunabhängigkeit:** Web-Browser sind für alle gängigen Plattformen erhältlich. Das Verhalten dieser Software sollte unter jedem Betriebssystem identisch sein. Dadurch entfällt der Programmieraufwand, der durch die Unterstützung verschiedener Plattformen entsteht.
- **Bedienbarkeit:** Die Interaktion mit dem World Wide Web ist für den Benutzer nach einer kurzen Einarbeitungszeit unproblematisch. Es entfällt die Gewöhnungsphase an die neue Oberfläche des Clients.
- **Mobilität:** Der Benutzer kann seine Arbeit im Büro oder zu Hause erledigen. Die Bearbeitung ist nicht an die Verbindung mit dem Workflow-Server gebunden. Erst nach dem Abschluß des Arbeitsschrittes muß die Verbindung zum Workflow-Server wieder aufgenommen werden.
- **Übertragungssicherheit:** Durch den eingebauten Verschlüsselungsmechanismus des Browsers ist die Übertragung geheimer Informationen gewährleistet.
- **Programmierunterstützung:** Durch die Verwendung standardisierter Protokolle ist auch die Implementierung der Server-Aspekte einfacher. Viele kommerzielle Datenbanken bieten Schnittstellen zum World Wide Web an, die den Zugriff auf Daten über das Internet erleichtern.

Workflow-Server: Ein Workflow-Server oder eine Workflow-Engine ist laut Definition der WfMC [WFM95] ein Software-Dienst, der

> *„...provides the run time execution environment for a workflow instance (individually, or in conjunction with other workflow engines).“*

Die zentrale Software-Komponente ist also verantwortlich für die Generierung von Prozeßinstanzen, die Verwaltung dieser Laufzeitobjekte und deren Daten, Zuteilung der Aktivitäten zu Akteuren, Auflösung von Rollen, die korrekte Weiterleitung an den nächsten Akteur und die Reaktion im Fehlerfall.

Datenbankmanagementsystem: Die Vorteile, die die Nutzung eines DBMS zur persistenten Verwaltung der Daten und Metadaten eines Workflowsystems bringen, sind sowohl für die *build time*-Komponente und die *run time*-Umgebung unübersehbar.

- Die Spezifikation der Prozeßtypen wird in einem Metaschema abgelegt. Auch weitere Metainformationen wie Aufbauorganisation, Rollenverwaltung, Stellenzuordnungen, werden in diesem Schema integriert verwaltet [Nek95].
- Die Dynamik-Information der Prozeßinstanzen wird als Ausprägung für jede Instanz in der Datenbank abgelegt. Dadurch kann festgestellt werden, welchen Status eine bestimmte Prozeßinstanz hat: Welche Aktivität wurde bereits absolviert, welche Aktivität muß als nächstes ausgeführt werden [EG96].

Die Eigenschaften des DBMS können ohne zusätzlichen Aufwand die Qualität des WfMS steigern.

Das Transaktionskonzept erlaubt den uneingeschränkten, gleichzeitigen Zugriff auf die Datenbank durch mehrere Benutzer, ohne das diese sich gegenseitig negativ beeinflussen. Durch das Authorisierungskonzept können Benutzergruppen verschiedene Rechte zugesprochen erhalten und das Recovery-System der Datenbank minimiert das Risiko eines Verlusts bereits geleisteter Arbeit [Dat90].

Der Trigger-Mechanismus der aktiven Datenbank, die in Panta Rhei Verwendung findet, steuert die dynamische Änderung, die durch die Bearbeitung und Weiterleitung der Formulare stattfindet. Jeder Zustandswechsel wird vom Monitoringmechanismus sofort erkannt und aktiviert eine Regel[5]. Diese Regel wurde aus der Prozeßbeschreibung generiert und leitet einen weiteren Bearbeitungsschritt ein. Dadurch fällt ein großer Teil der Funktionalität der Workflow-Engine in den Zuständigkeitsbereich der aktiven Datenbank. Die Intention der Architektur von Panta Rhei ist es, soviel Funktionalität wie möglich von der Datenbank übernehmen zu lassen, um den Implementierungsaufwand für den Server gering zu halten und trotzdem die komplexen Anforderungen zu erfüllen.

[5]Eine detaillierte Einführung in das Gebiet der aktiven Datenbanken ist in [Wid92, Cha92] zu finden.

6.2 Die Workflow-Beschreibungssprache WDL

Die für das Workflow-System entwickelte *workflow description language* WDL wurde nach folgenden Kriterien entwickelt:

- Einfache und deskriptive Handhabung der Sprache. Der Workflow-Designer muß nach einer Einarbeitungsphase in der Lage sein, zumindest einfache Geschäftsprozesse abzubilden.
- Die Sprache soll universell einsetzbar sein. Das Einsatzgebiet darf nicht auf wenige und spezielle Aufgabenbereiche eingeschränkt werden.
- Gute Abbildbarkeit auf eine Implementierung. Obwohl diese Eigenschaft für den Prozeßentwickler keine Rolle spielt, ist gerade im universitären Umfeld eine einfache Umstellung auf neue technische Gegebenheiten von Bedeutung.

Die Sprache WDL besteht aus fünf Hauptkomponenten [Tha96], die sich auch im Geschäftsprozeß widerspiegeln:

- Prozeßdefinition
- Taskdefinition
- Rollendefinition
- Organisationsstruktur
- Formulardefinition

Für die Definition von Zeitinformationen und Abhängigkeiten zwischen Prozessen und Tasks sind nur die Bereiche Prozeß- und Taskdefinition interessant.

6.2.1 Taskdefinition

Tasks sind die Basisaktivitäten eines Workflows und können nicht mehr weiter unterteilt werden. Der Begriff Task entspricht der *process activity* des Sprachgebrauchs der Workflow Management Coalition, vgl. dazu Abschnitt 2.1. Die Definition besteht aus einem Task-Kopf und dem Deklarationsteil. Parameter für diesen Arbeitsschritt sind Formulare. Wird sonst nichts spezifiziert, so werden die Formulare dem Benutzer angezeigt, dadurch können rein manuelle Tätigkeiten in die Kette der Arbeitsprozesse eingefügt werden. Für jeden Prozeß kann eine neben der eigentlichen Aktivität noch eine *post condition* angegeben werden, die den Status der Bearbeitung feststellt [Nek95]. Die maximale Bearbeitungszeit für den Task wird durch das Feld `maxtime` festgelegt. Der Wert wird entweder zur

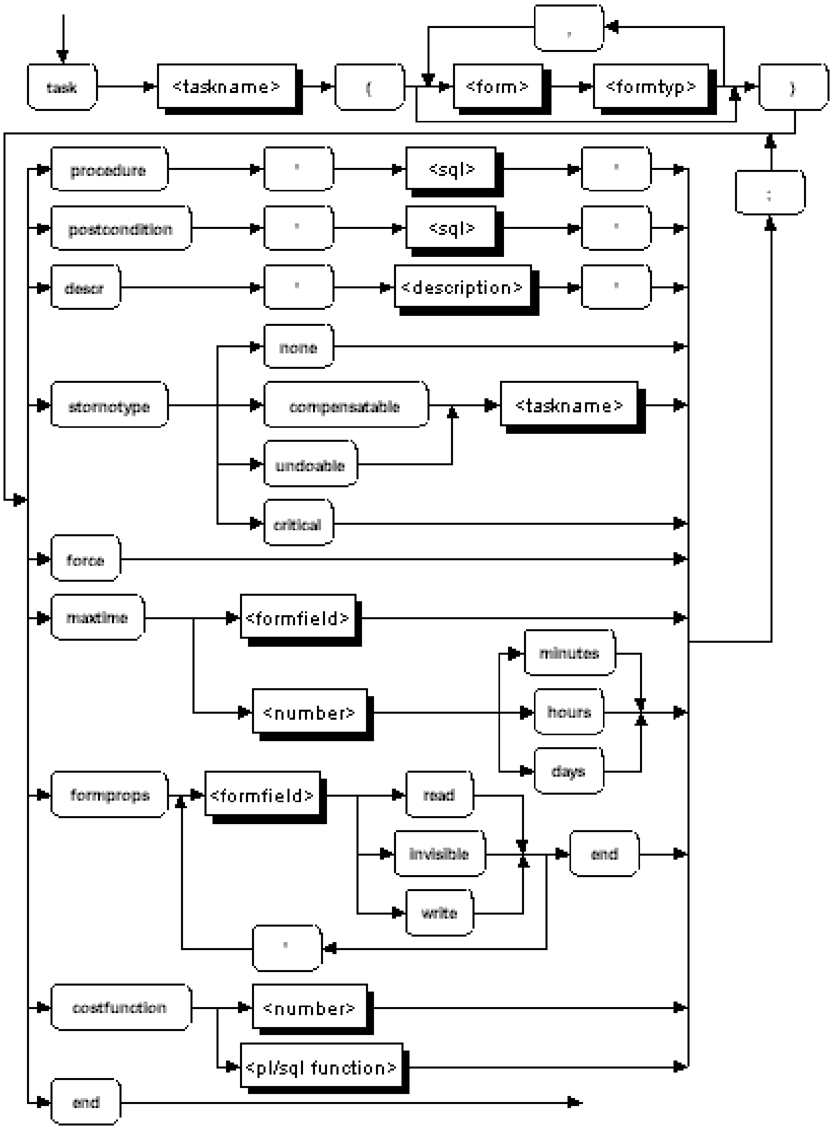

Abbildung 6.2: Syntax der Taskdefinition

Entwurfszeit vom Prozeßdesigner mit Minuten, Stunden oder Tagen festgelegt, oder zur Laufzeit durch das Lesen eines Formularfeldes erfragt.

Jedes syntaktische Element des Deklarationsteil kann jeweils nur einmal festgelegt werden. So ist es z. B. nicht möglich, die Durchlaufzeit mehrmals für einen Tasktyp festzulegen.

6.2.2 Prozeßdefinition

Prozesse bestehen aus einem Prozeßkopf und dem Deklarationsteil. Im Prozeßkopf werden der Name und die Parameter festgelegt. Der Typ der Parameter ist immer ein (HTML)-Formular.

Durch den Deklarationsteil können der Prozeßverantwortliche, ein Betreff, eine kurze

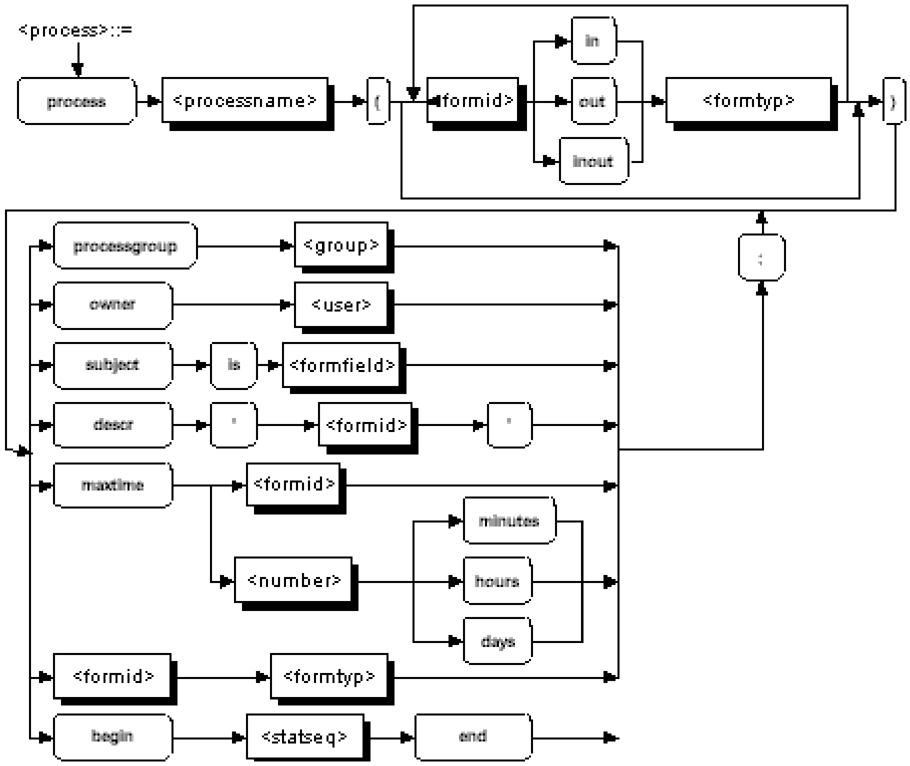

Abbildung 6.3: Syntax der Prozeßdefinition

Beschreibung oder die Durchlaufzeit festgelegt werden. Die Gesamtzeit, die dieser Prozeß zur Abarbeitung benötigen darf, wird im Feld `maxtime` definiert.

In der nach dem Schlüsselwort `begin` startenden Sequenz von Sprachelementen wird die Ablauflogik des Prozesses festgelegt. Die für die Untersuchung von Zeitabhängigkeit wichtigsten Elemente der Ablauflogik sind, neben der Sequenz, die Schleifenkonstrukte (`while`, `repeat until` und `foreach`), die Alternative `if`, zwei Nebenläufigkeits- (`andpar`, `orpar`) und zwei Auswahlstrukturen `ranked choice` und `free choice`.

Die genaue Syntax der Workflow-Beschreibungssprache WDL ist im Anhang A.1 zu finden. Neben der textuellen Repräsentation kann die Struktur eines Prozesses auch graphisch dargestellt werden. Eine Übersicht über die graphische Darstellung der wichtigsten Sprachkonstrukte ist in der Abbildung 6.4 zu sehen.

Schleifen

Schleifen sind bereits aus dem Bereich der Programmiersprachen bekannt. Nur die Foreach-Iteration nimmt eine Sonderstellung zu den konventionellen Programmiersprachen ein, da sie, ähnlich dem Cursor-Konstrukt datenbankgebundener Sprachen, Zeilen von Tabellen abarbeiten kann. Tabellen sind in diesem Fall Werte eines Formulars, auf das die Schleife zugreift.

Alternative

Das If-Statement entspricht seinem Gegenstück der konventionellen Programmiersprachen. Auch hier wird im Bedingungsteil des Ausdruckes entsprechend eines Wertes entschieden. Elsif-Abschnitte können beliebig oft wiederholt werden.

Nebenläufigkeit

Wenn Teile eines Prozesses nebenläufig bearbeit werden können, muß das entsprechende Konstrukt der WDL gewählt werden. Es sind zwei Typen denkbar. Im ersten Fall müssen alle Teilzweige erfolgreich abgeschlossen sein, bevor nach der Synchronisation der Teilbereiche weitergearbeitet werden kann. Das entsprechende Konstrukt der WDL ist das `andpar`.

Im zweiten Fall muß nur ein Zweig erfolgreich sein, um mit der Bearbeitung fortfahren zu können. Hier ist das `orpar` zu wählen. Die restlichen, noch nicht beendeten Abarbeitungszweige werden abgebrochen.

Auswahl

Eine Auswahl oder *c*hoice-Struktur stellt eine Besonderheit dar, die in klassischen Programmiersprachen nicht bekannt ist. Syntaktisch sind sie dem `andpar` bzw. `orpar` ähnlich. Der Workflow-Designer entwirft mehrere Wege mit entweder statischer (`ranked choice`) oder dynamischer (`cost choice`) Prioritätenbestimmung. Zur Laufzeit wählt der Scheduler den Weg mit der höchsten Priorität aus. Ist dieser erfolgreich, d. h. wird das `end`-Statement der Auswahl erreicht, werden die anderen Zweige nie mehr gestartet und die Ausführung setzt mit dem nächsten Statement nach dem Auswahlende fort. Nur wenn die Bearbeitung abbricht, wird der nächste Zweig der Auswahl gestartet. Erst wenn alle Zweige fehlschlagen, wird diese Struktur negativ bewertet.

Zur Reihung der Auswahl stehen zwei sprachliche Elemente zur Verfügung. Beim *ranked choice* gibt der Workflow-Designer die Priorität vor, nach der zur Laufzeit der Weg bestimmt wird. Im Gegensatz dazu wird beim *cost choice* durch eine Kostenfunktion zur Laufzeit entschieden, welcher Zweig der nächste auszuführende ist.

Die in der Abbildung 6.4 dargestellten Elemente dienen nicht zur graphischen Modellierung eines Workflow-Prozesses, sondern zur vereinfachten Veranschaulichung von WDL-Sachverhalten. Aktivitäten sind als Kreise, Übergänge als Pfeile dargestellt. Die rautenförmigen Zeichen dienen zur Markierung von Verzweigungen.

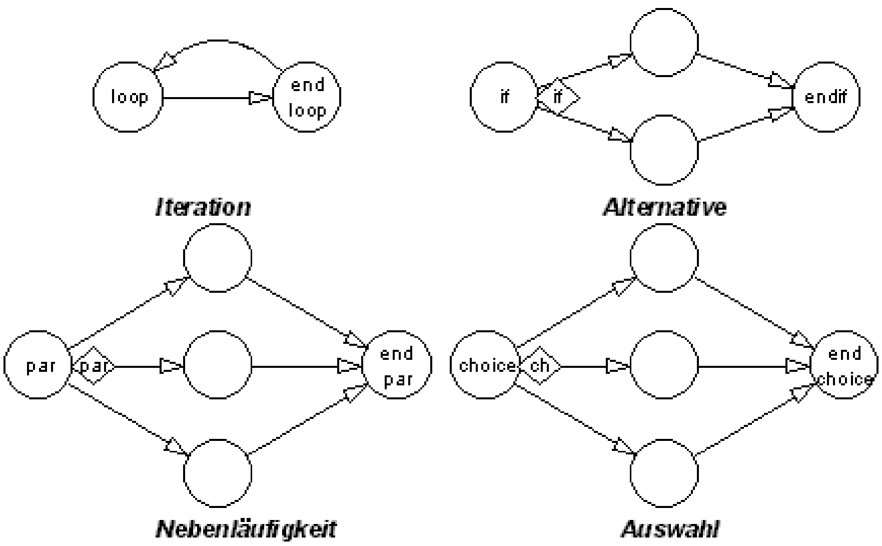

Abbildung 6.4: Graphische Repräsentation WDL

Kapitel 7

Integration von Zeitrestriktionen

In Kapitel 5 wurde mit der erweiterten Netzplantechnik ePERT ein Konzept vorgestellt, mit dem strukturelle Zeitrestriktionen für Projekte und Prozesse berechnet und kontrolliert werden können. Nun stellt sich die Frage, wie die gewonnenen Erkenntnisse in ein Workflow-Modell einfließen können, um

1. die Auswahl der nächsten Aktivität einer Prozeßinstanz zu ermöglichen, die die Einhaltung aller Zeitvorgaben garantiert,
2. die Folge von Tasks zu ermitteln, die die minimale Bearbeitungszeit für den Gesamtprozeß ergibt, oder
3. die Aktivität, die an einer Verzögerung schuld trägt, zu identifizieren.

Die nächsten Seiten befassen sich mit der Integration von Zeit in den Workflow-Prototypen Panta Rhei. Zunächst wird untersucht, ob und wie sich die Struktur eines Prozesses, die durch die WDL spezifiziert wird, in einen äquivalenten erweiterten Netzplan ePERT umformen läßt. Der nächste Schritt ist die Adaption der Vorschriften zur Berechnung der Werte für die Anfangs- und Endzustände und der kritischen Pfade.

7.1 Strukturtransformation

Der erste Schritt zur Einbettung von zeitlichen Regeln ist die Überführung zeitrelevanter Teile der Workflow-Beschreibung in eine äquivalente Struktur eines Netzes. Da vor allem die Struktur der Workflow-Prozesse die Zeitschranken bestimmt, muß für jede Kontrollstruktur der WDL eine Umformungsregel gefunden werden, die die Ablauflogik des Prozesses in eine Reihenfolgeabhängigkeit der Netzdarstellung transformiert.

Relevante Teile sind

- Sequenz,

- Alternative,
- Iteration,
- Auswahl und
- Nebenläufigkeit.

An dieser Stelle ist anzumerken, daß einige Sachverhalte der einen Darstellung in der anderen nicht modellierbar sind. So z. B. dürfen im Netzplan keine Zyklen auftreten. Damit ist die direkte Umformung von Schleifen der WDL nicht möglich und es müssen andere Lösungen gefunden werden. Andererseits können durch das Netz Sachverhalte ausgedrückt werden, die in der Workflow-Beschreibungssprache nicht möglich sind, siehe dazu das Beispiel 7.1.

Beispiel 7.1 *Intertask-Abhängigkeit im Netzplan*

Gegeben sei folgende Projektstruktur:

1. A und B sind die Startaktivitäten.
2. D folgt auf A.
3. C folgt auf A und B.
4. E folgt auf D.

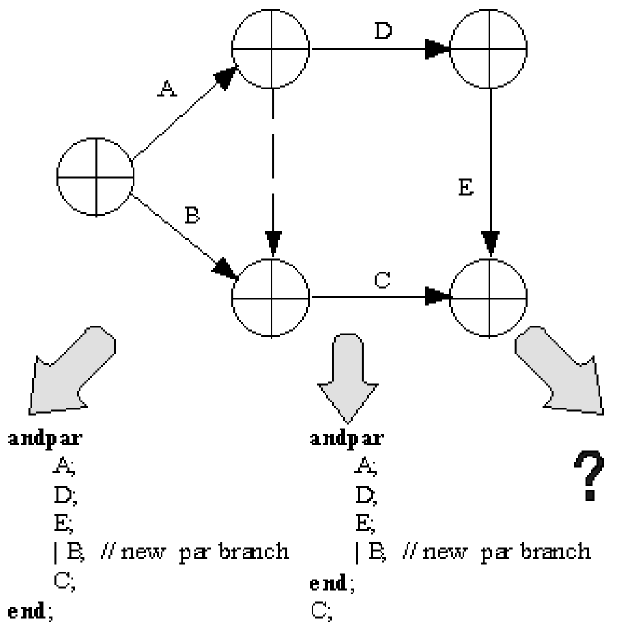

Abbildung 7.1: Diskrepanz zwischen PERT und WDL

Die Schwierigkeit besteht darin, daß die Arbeitsschritte A und B nebenläufig begonnen werden und dadurch unabhängig voneinander sind. Der Task D hängt nur von A und kann sofort nach dem Ende von A gestartet werden. Die Aktivität C aber muß auf das Ende dieser Aktivität warten und liegt deshalb eigentlich nach der nebenläufigen Struktur. Andererseits sind die Zustände von D und E für die Ausführung von C nicht wichtig.

Durch eine Programmiersprache kann der obige Sachverhalt nicht ausgedrückt werden, da es hier zu einer Kommunikation zwischen Zuständen kommt, die in nebenläufigen Zweigen abgearbeitet werden (Abbildung 7.1). Nur durch die Zuhilfenahme von Transaktionsregeln (siehe Abschnitt 3) kann diese Semantik korrekt formuliert werden.

□

7.1.1 Tasktransformation

Der einfachste Fall ist die Umwandlung eines Prozesses, der nur aus einem Task besteht[1].

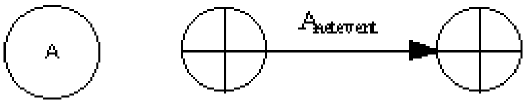

Abbildung 7.2: Tasktransformation

Da der rekursive Algorithmus zur Abarbeitung der Prozeßdefinition von einem existierenden Zustand ausgeht, müssen für einen Task nur mehr die ihm entsprechende Netzaktivität mit ihren aktuellen Daten und der jeweilige Endzustand erzeugt und in das ePERT-Netz eigefügt werden.

7.1.2 Transformation einer Sequenz

Eine Sequenz ist eine Abfolge von zeitlich nacheinander ausgeführten Aktivitäten. Nacheinanderausführung bedeutet, daß die Folgeaktivität erst dann gestartet werden darf, wenn der direkte Vorgänger beendet wurde. Diese Reihenfolgebeziehung wird Ende-Start-Abhängigkeit genannt. Eine Sequenz wird daran erkannt, daß der Vorgängertask nur einen Nachfolger in der Prozeßstruktur hat.

Da mit der Sprache WDL auch Hierarchien formuliert werden können, müssen diese für die Umformung aufgelöst werden. Eine Hierarchie ist dann gegeben, wenn ein Prozeß Teil eines anderen Prozesses ist. Durch die rekursive Bearbeitung der WDL-Prozeßdefinition werden die Ebenen eingeflacht.

[1]Genau genommen wird nicht ein Task umgewandelt, sondern die **Verwendung** des Task in einem Prozeß. In der Metastruktur von Panta Rhei wird nämlich zwischen dem Typ eines Tasks und seiner Verwendung (`wf_task_call`) für eine Prozeßdefinition unterschieden, um einen Typ wiederverwendbar zu machen.

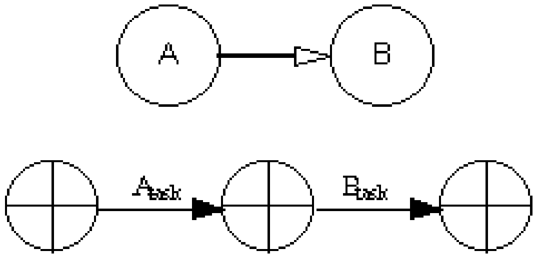

Abbildung 7.3: Sequenztransformation

Der Vorgang der Sequenztransformation wird durch die Abbildung 7.3 verdeutlicht. Ausgehend von einem existierenden Startzustand wird für jeden Task eine Netzaktivität mit ihrem Endzustand in das ePERT-Diagramm eingefügt.

7.1.3 Transformation von Nebenläufigkeit

Nebenläufige Tasks werden unabhängig voneinander ausgeführt. Diese Abarbeitung ist einfach in einen erweiterten Netzplan zu transformieren, da der vorwiegende Zweck eines Netzes die Berechnung und Synchronisation nebenläufiger Strukturen ist. Deshalb ist ein äquivalentes Konstrukt schon im konventionellen PERT vorhanden.

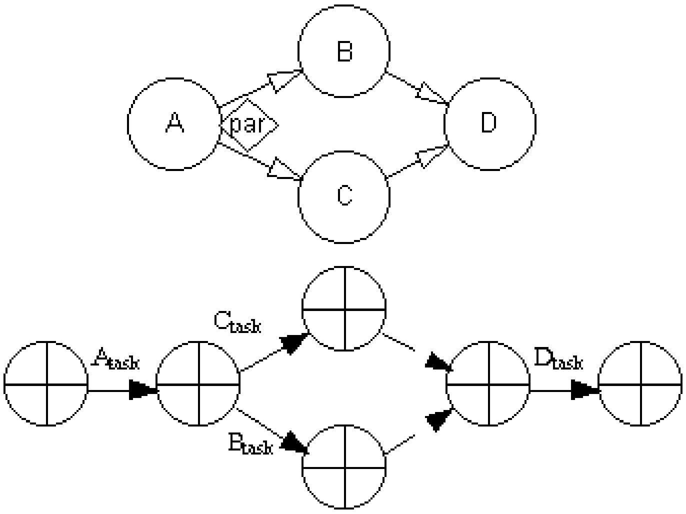

Abbildung 7.4: Transformation einer nebenläufigen Aktivität

Der Unterschied zur Darstellung im herkömmlichen Netzplan besteht darin, daß im ePERT eine Synchronisation durch Scheinaktivitäten durchgeführt wird.

7.1.4 Transformation einer Alternative

Die strukturelle Transformation eines alternativen Prozesses in ein ePERT unterscheidet sich nicht von der Vorgehensweise, die bereits bei der Nebenläufigkeitsumwandlung vorgenommen wurde. Die wichtigsten Punkte wurden bereits im Kapitel 5.1.4 angesprochen.

7.1.5 Transformation einer Auswahl

Auswahl-Kontrollstrukturen stellen ein Konzept dar, welches die Abarbeitung nach einer Prioritätenreihung und gleichzeitig die Ausnahmebehandlung berücksichtigt (Abschnitt 6.2.2).

Die Priorität, nach der die Ausführung gereiht wird, kann entweder

- zur Entwurfszeit durch eine Designentscheidungen (*ranked choice*) oder
- zur Laufzeit durch das Ergebnis einer Prioritätenfunktion (*cost choice*)

bestimmt werden.

Die Besonderheit besteht darin, daß beim Abbruch einer Aktivität die Abarbeitungsfolge der nächstniedrigeren Priorität ausgeführt wird. Das geschieht, bis entweder ein Zweig erfolgreich ist, oder alle Auswahlzweige versucht wurden und die gesamte Bearbeitung abbricht.

Deshalb kann ein `choice` auch als eine Folge von Alternativen gesehen werden, wobei die Entscheidung, welchen Schritt der Scheduler für die nächste Bearbeitung auswählt, nicht wertabhängig ist, sondern vom Zustand der Workflow-Transaktion bestimmt wird.

Behandlung einer gereihten Auswahl

Eine gereihte Auswahl oder `ranked choice` wird konzeptuell in eine Folge von Alternativen aufgebrochen und nach den Regeln, die für die Umwandlung von `if`s gelten, behandelt. Die strukturelle Äquivalenz zwischen dem `ranked choice` und `if` wird in der Abbildung 7.5 verdeutlicht.

Diese Deckungsgleichheit zwischen den beiden Konstrukten gilt aber nur für die Betrachtung von Zeitdauern, da folgende Vereinfachung angenommen wird: Die Bewertung aller Zeitdauern der Aktivitäten innerhalb einer Auswahl entspricht den Werten, die vom Workflow-Designer für ihre korrekte Bearbeitung festgesetzt wurden. Tatsächlich ist das aber nicht völlig richtig, da ein abgebrochener Task nicht in jedem Fall dieselbe Zeitspanne benötigt, die ein korrekt ausgeführter Arbeitschritt für sich beansprucht. Deshalb liegt die benötigte Zeit für den längsten Weg, der über mehrere abgebrochene Aktivitäten führt, unter dem vom Netz errechneten Wert, da dort jede Aktivität mit ihrer vollen Bearbeitungsdauer bewertet wurde.

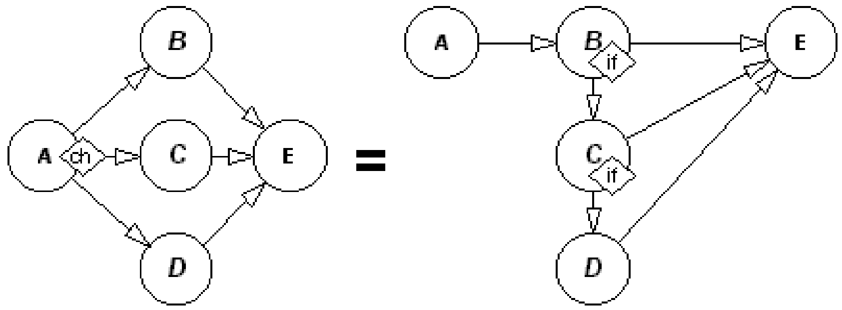

Abbildung 7.5: Strukturbeziehung zwischen Auswahl und Alternative

Beispiel 7.2 *Zeitbewertung einer* **`ranked choice`**-*Auswahl*

Für den Workflow, der in der Abbildung 7.5 gezeigt wird, werden folgende Bearbeitungszeiten für die einzelnen Task festgelegt.

Task	Dauer
B	2
C	4
D	4

Während der Abarbeitung muß B nach einer Zeiteinheit abgebrochen werden. Auch der Task C muß aufgrund knapper Ressourcen nach der selben Zeit eingestellt werden. D ist aber erfolgreich.

Die Zeitbewertung des längsten Weges durch die Auswahl wurde mit 10 angenommen. In den meisten Fällen liegt die tatsächliche Bearbeitungsdauer weit unter diesem pessimistischen Wert, im Falle des Beispiels bei 6 Zeiteinheiten. Für die Prozeßplanung ist diese Bewertung von Nachteil, da dadurch Optimierungspotential verloren geht.

□

Da aber während des Entwurfs keine Erfahrungswerte über das tatsächliche Abbruchrisiko des Workflows vorliegen und die Ermittlung des durchschnittlichen Abbruchszeitpunktes einer Aktivität schwierig ist, wird dieser Nachteil in Kauf genommen, um die Auswahl bewerten zu können.

Die Tatsache, daß die gesamte Auswahl fehlschlägt, muß im Netzplan nicht modelliert werden, da hier der Recovery–Mechanismus des Workflow-Systems die semantisch korrekte Kompensation vollzieht.

Behandlung einer Kostenauswahl

Die Umformung einer `cost choice`, also einer Auswahl, deren Prioritätenreihung erst zur Laufzeit bestimmt wird, ist ungleich schwieriger. Da nicht bekannt ist, in welcher Reihenfolge die Aktivitäten zur Laufzeit ausgeführt werden und da insbesondere die Prioritätenfunktion für jede Instanz andere Ergebnisse bringen kann, ist die Umformung in **eine** Alternativensequenz nicht möglich. Auch die Berechnung aller möglichen Exekutionsfolgen ist praktisch nicht durchführbar, da es sich um eine vollständige Permutation der Zweige der Auswahl handelt. Die Anzahl der verschiedenen Anordnungen einer Permutation ist $n!$[2]. Deshalb errechnet sich die Anzahl m der verschiedenen Kombinationen [DP88] als

$$m = \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

Der Parameter n ist die Anzahl der möglichen Aktivitäten, r ist die Anzahl der Verzweigungen. Diese Zahl wird sogar noch erhöht, wenn die Zweige der Auswahl aus mehr als einem Task bestehen, da nach jedem abgebrochenen Task der Zweig der nächstniedrigeren Priorität gestartet wird.

Beispiel 7.3 *Anzahl der Wege bei* `cost choice`

Die Workflow-Struktur aus der Abbildung 7.5 sei ein `cost choice`. Die Priorität kann erst zur Laufzeit bestimmt werden. Der Wert n für drei Aktivitäten ist 6, da auch die Nichtausführung als alternative Kombination betrachtet wird. Es sind laut der Formel $\binom{n}{r} = \binom{6}{4} = 15$ Exekutionspfade möglich.

1	A→B→E	9	A→D→C→E
2	A→C→E	10	A→B→C→D→E
3	A→D→E	11	A→B→D→C→E
4	A→B→C→E	12	A→C→B→D→E
5	A→B→D→E	13	A→C→D→B→E
6	A→C→B→E	14	A→D→B→C→E
7	A→C→D→E	15	A→D→C→B→E
8	A→D→B→E		

□

Durch die enorme Anzahl an Exekutionspfaden ist die Errechnung aller möglichen Wege nicht wünschenswert.

Die Umformung muß deshalb in zwei Schritten erfolgen.

[2]Die Anzahl $n!$ wird „n Faktorielle“ oder „n Fakultät“ genannt und berechnet sich als $n! = 1*2*3*\cdots*n$.

1. Schritt: Die Auswahl wird nach dem Muster einer Nebenläufigkeit transformiert. Dadurch ist es möglich, den kürzesten und längsten Zweig zu ermitteln.

2. Schritt: Nun müssen die Wege durch die Auswahl berechnet werden. Dabei wird der Zweig, der in der vorigen Phase als Minimum identifiziert wurde, als bester Fall angesehen und im Netz eingetragen. Der schlechteste Fall einer Auswahl ist der Abbruch aller Aktivitäten und der Versuch, mit dem nächsten Zweig erfolgreich zu sein. Deshalb ergibt sich die Bewertung des *worst case* durch die **Summe** der Bearbeitungszeit **aller Zweige**. Das ist auch der Lösungsansatz, der bei der Umformung einer `cost choice` verfolgt wird. Im Netzplan wird nur mehr die Struktur des kürzesten und längsten Weges dargestellt, wobei sich der längste Weg durch die Hintereinanderausführung aller Zweige ergibt. Die beiden Wege sind **Alternativen**. Das Beispiel 7.4 erläutert die Vorgehensweise an einem Beispiel.

Beispiel 7.4 *Zeitbewertung einer* `cost choice`*-Auswahl*
Die Bearbeitungszeiten, die im Beispiel 7.2 definiert wurden, gelten auch hier. Durch die Phase 1 wird der Zweig, der über den Task B führt, als Minimum berechnet (Dauer ist 2 Zeiteinheiten). Das Maximum ergibt sich durch die Hintereinanderausführung aller Zweige und beträgt 10 Zeiteinheiten. Dadurch ergibt sich eine Alternativenstruktur, wie sie in der Abbildung 7.6 zu sehen ist.

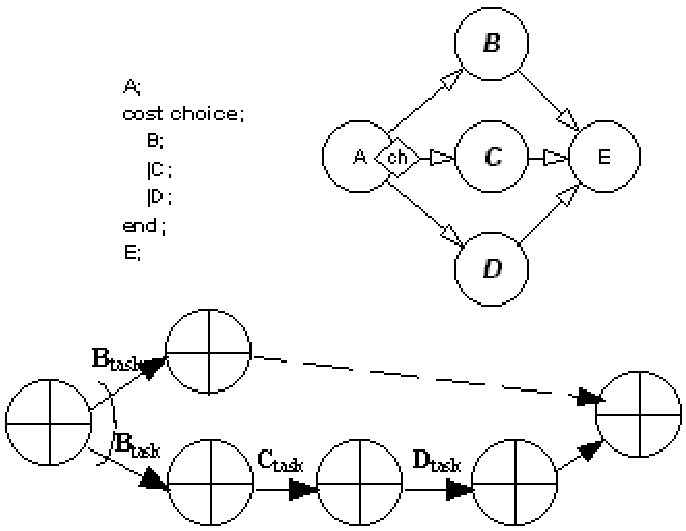

Abbildung 7.6: Umwandlung einer `cost choice`

Die Auswahl wird in ein `if`-Konstrukt umgewandelt, das den minimalen und maximalen Zeitverbrauch der Auswahl repräsentiert.

□

An dieser Stelle ist anzumerken, daß die Aktivität B zweimal im Netzplan vorkommt. Das ist die konkrete Ausprägung der gewandelten Bedeutung eines Netzplans. Steht im konventionellen Netzplan ein Projekt mit seinen Abläufen im Vordergrund, so bestimmt die Darstellung des *ePERT* den *möglichen* Verlauf, den ein Prozeß nehmen kann. Das drückt sich auch durch die geänderte **Interpretation** der Netzaktivitäten innerhalb einer Auswahl aus.

1. Wenn eine Struktur innerhalb einer Auswahl durchgeführt wird, bedeutet das, daß alle Vorgängerstrukturen negativ terminierten.

2. Wenn eine Struktur positiv terminiert, so heißt das, daß alle Nachfolgeaktivitäten nicht mehr ausgeführt werden.

Die Überprüfung der Zeitvorgaben, die durch diese Struktur errechnet werden, sind für Aktivitäten innerhalb der Auswahl nicht mehr in jedem Fall sinnvoll. Da die Eintrittsgrenzen für einen Zustand von der Struktur der Abläufe abhängen, gelten diese errechneten Grenzen für eine `cost choice` nur mehr unter folgenden Bedingungen:

1. Alle Aktivitäten, die im Netzplan vor einer zu überprüfenden Aktivitiät A liegen, wurden bereits ausgeführt[3].

2. Keine der nach A im Plan liegenden Aktivitäten wurde ausgeführt.

Nur so ist garantiert, daß die Summe der aktuell verbrauchten Zeiten mit jener der Sequenz des Netzplans übereinstimmt. Gilt diese Bedingung für jede Aktivität der Auswahl, so ist die Exekutionsfolge genau die, die auch im Plan berechnet wurde.

In den meisten Fällen wird das aber nicht zutreffen. Die zeitliche Korrektheit einer Auswahl kann demnach nur am Ende der Kontrollstruktur überprüft werden.

7.1.6 Transformation von Schleifen

Auch Schleifen oder Iterationen sind Konstrukte, die nicht in konventionellen Netzplänen existieren. Falls es bei einem Projekt zu Wiederholungen von Arbeitsabläufen kommt, müssen diese in einem Netzplan **ausdrücklich** modelliert werden. Wird z. B. ein Prüfungsverfahren bis zu dreimal wiederholt, werden alle drei Iterationen als Aktivitäten sequentiell im Netz modelliert. Dieser Ansatz ist für eine Modellierung von Workflows nicht tragbar. Einerseits sind in den wenigsten Fällen Erfahrungswerte über die genaue Anzahl der Iterationen vorhanden, andererseits kann die Wiederholungszahl für jede Instanz der Iteration stark schwanken.

[3]Die Reihenfolge der Ausführungen ist nicht von Bedeutung.

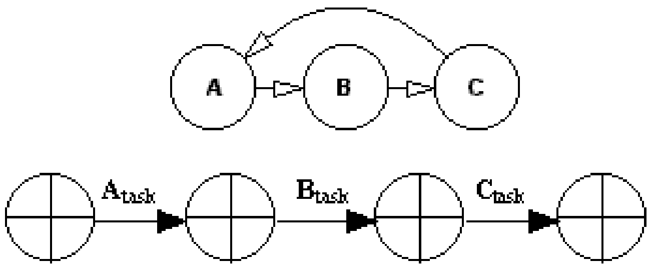

Abbildung 7.7: Umwandlung einer Schleife

Deshalb kommt der Schleife eine Sonderstellung zu. Sie wird nicht durch ein eigenes Strukturkonstrukt im Netzplan modelliert, sondern in eine Sequenz umgeformt. Diese Sequenz entspricht genau **einem** Schleifendurchlauf. Die Informationen, die benötigt werden, um die zeitliche Verhalten der Iteration zu wahren, werden durch die Berechnungsvorschrift gewonnen. Auch die abweisende Schleife `while` wird auf diese Weise umgewandelt, da es hier nur um die Strukturinformation geht und die zeitliche Verhaltensmodellierung durch die Errechnung der frühesten und spätesten Eintrittszeitpunkte erfolgt.

Die Informationen, wie sich Iterationen zur Laufzeit verhalten, können nur mehr durch die Zeitdaten erhalten bleiben. Diese müssen schon zur Designzeit festgelegt werden.

Zu jedem Schleifenkonstrukt *WHILE*, *UNTIL* und *FOR EACH* werden Daten festgelegt, die helfen, die Dauer der Schleifenabarbeitung zu bestimmen. Dem Workflow-Designer stehen drei Möglichkeiten zur Verfügung:

1. Die Anzahl der Durchläufe wird auf einen **konstanten Wert**, meistens 1 gesetzt. Dadurch werden die Aktivitäten der Schleife mit ihren festgelegten Zeitdauern in die Sequenz aufgenommen. Dieses Vorgehen sollte aber nur dann geschehen, wenn überhaupt keine Aussagen über das zeitliche Verhalten der Schleife getroffen werden können. Durch die simplifizierende Annahme, daß die Schleife konstant oft durchlaufen wird, kann die Aussagekraft des Gesamtergebnisses der Zeitberechnung erheblich gemindert werden.

2. Die **mögliche Anzahl der Durchläufe** wird durch die Ober-, Untergrenzen und den häufigsten Wert geschätzt.

   $\mathbf{a}_i$ für die minimale Anzahl der Iterationen, die niemals unterschritten werden kann.

   $\mathbf{b}_i$ für die maximale mögliche Anzahl der Wiederholung.

   $\mathbf{m}_i$ für die häufigste Anzahl der Durchläufe.

   Die Werte a_t, m_t und b_t aller Tasks der Schleife werden dann entsprechend der vorgegeben Schleifenwerte multipliziert. Die korrekte Dauer eines (iterierten) Tasks

t kann aufgrund dieses pragmatischen Ansatzes durch folgende Multiplikationen

$$\begin{aligned} a_t * a_i &= a_{tc} \\ b_t * b_i &= b_{tc} \\ m_t * m_i &= m_{tc} \end{aligned}$$

bestimmt werden. Diese Daten werden zur Errechnung der kumulierten Mittelwerte und Varianzen der Schleifenelemente benötigt.

Beispiel 7.5 *Schleifenberechnung durch Multiplikation*
Eine Schleife umfaßt die Aktivität B mit den Werten 1, 7 und 10 für die minimale, häufigste und maximale Dauer. Durch die Annahme der β-Verteilung zur Berechnung des Mittelwertes ergibt sich $\mu = \frac{a+4m+b}{6} = \frac{1+4*7+10}{6} = 6.5$.
Die Schleife selbst wird mindestens einmal, maximal achtmal und am häufigsten fünfmal durchlaufen (Abbildung 7.8).

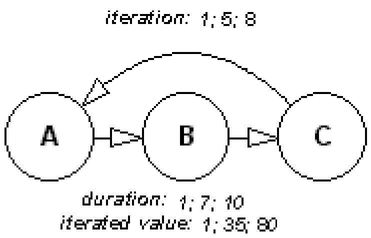

Abbildung 7.8: WDL-Schleife

Durch die Multiplikation der Werte ergibt das demnach die Ergebnisse 1, 35 und 80 für die gesamte Schleife. Der Mittelwert der Schleife ist $\mu = \frac{1+4*35+80}{6} = 36.8$. Das entsprechende *ePERT*-Netz ist in der Abbildung 7.9 zu sehen.

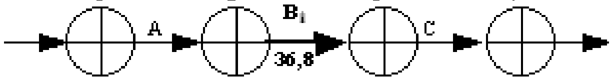

Abbildung 7.9: ePERT-Netz einer Schleife

Die akkumulierte Aktivität B_i und ihre Grenzwerte repräsentieren die wiederholte Ausführung von B. □

Deshalb stehen die Werte der Zustände der einzelnen Tasks für die **jeweils letzte** Wiederholung der Schleife. Zur Laufzeit kann nun während der Iteration für den einzelnen Task nicht festgestellt werden, ob sich eine Verspätung abzeichnet. Erst wenn die vorgeschriebene Dauer für die Schleife überschritten ist, wird dies erkannt. Welcher Task in welcher Wiederholung daran schuld trägt, ist jedoch nicht mehr festzustellen[4].

3. Es wird die **Dauer** der **gesamten** Schleife festgelegt. Die Abarbeitung des Schleifenkörpers wird als **eine** Aktivität gesehen und für sie werden die Werte a_{id}, b_{id} und m_{id} zur Definition der minimalen, maximalen und häufigsten Abarbeitungsdauer verwendet. Die gesamte Schleifenkonstruktion wird damit zeitlich durch den errechneten Mittelwert und die Varianz bewertet. Diese beiden Werte werden in den letzten Zustand der Schleife (Anfangszustand der Scheinaktivität, die für das `end` steht) eingetragen.

   Es ist schon zur Entwurfszeit leicht möglich die prinzipielle Integrität dieser externen Vorgabe zu prüfen. Wenn nämlich bei einer nicht-abweisenden Schleife die Gesamtsumme der Mittelwerte der Schleifenaktivitäten größer als der für die Schleife extern definierte Mittelwert ist, wurde dadurch ein **Entwurfsfehler** entdeckt: Die Schleife kann aufgrund der zu engen Zeitvorgabe kein einziges Mal durchlaufen werden[5]. Da es sich bei dem Schleifenkonstrukt um eine Durchlaufschleife handelt, liegt hier ein Widerspruch vor.

   Bei dieser Art der Zeitdefinition muß beachtet werden, daß die Überschreitung der Vorgaben für die einzelnen Tasks nicht zu einem Zeitfehler führt, wenn bis dahin die Gesamtzeit nicht überschritten wurde. Im Normalfall wird bei jeder Überschreitung der Zeitgrenzen einer Aktivität ein Zeitfehler generiert. Nun muß die für die Einhaltung der zeitlichen Integrität zuständige Komponente des Workflow-Servers wissen, daß die überprüfte Aktivität innerhalb einer Iteration liegt. Dann wird Überprüfung einer solchen Aktivitäten ausgesetzt. Erst bei Überschreitung der festgelegten Dauer der Schleife generiert sie einen Fehler. Dieses Verhalten entspricht genau der Behandlung von externen Regeln[6].

[4]Dies gilt nur für den Vergleich mit den Daten des Netzplans. Durch „manuelle“ Analyse der Ausführungszeiten der einzelnen Schleifentasks ist immer eine Überprüfung möglich.

[5]Das ist nicht völlig korrekt, da es bei Überlappungen der Varianzen der beiden Durchlaufzeiten sehr wohl zu Abarbeitungen der Schleife kommen kann. Doch handelt es sich hier um einen Sonderfall, der je nach Überlappungsbreite nur selten vorliegt.

[6]Tatsächlich fällt die Bestimmung einer maximalen Durchlaufzeit einer Schleife in die Klasse der externen Regeln, da ein externer Trigger generiert und überprüft wird, der einen Zeitfehler feststellt.

7.1.7 Berechnungsvorschriften für WDL-Konstrukte

Da alle Konstrukte der WDL auf Sequenz, Nebenläufigkeit oder Alternative zurückzuführen sind, können die Berechnungsvorschriften des ePERT-Netzplans ohne Änderung verwendet werden. Nur für die Überprüfung des Laufzeitverhaltens müssen einige Aspekte der Auswahl und Schleifen gesondert betrachtet werden. Der Vorteil dieser Vorgehensweise liegt auf der Hand: es können alle Kontrollstrukturen der WDL mit diesen drei Klassen von Berechnungsregeln ermittelt werden und die Behandlung aller Sonderfälle ist nicht mehr nötig.

7.1.8 Ein Beispiel

Beispiel 7.6 *Netzplan für Lebensversicherungsantrag*

Die Bearbeitung eines Antrags für eine Lebensversicherung eines bestimmten Tarifs wird auf drei Phasen aufgeteilt. Der Prozeß muß folgenden (stark vereinfachten) Ablauf haben, wobei jede Phase abgeschlossen sein muß, bevor die nächste beginnt.

1. **Antragsprüfung**. Es gibt zwei Antragstypen: den Normalantrag, gekennzeichnet mit „normal“ und den Antrag über die Verkaufsaktion „Sichere Jugend“ („aktion“), der eine vereinfachte Antragsprüfung erlaubt:
   - Nach der Dateneingabe der Antragsaufnahme für die Verkaufsaktion erfolgt eine einfache Bonitätsprüfung durch eine Gehaltsbestätigung.
   - Der normale Lebensversicherungsantrag muß nach der Antragsaufnahme die Schritte Bonitätsprüfung und Gesundheitsattest durchlaufen.
2. **Polizzierung**
   - Die Polizzierung verläuft teilautomatisch und muß nur durch eine Polizzenausstellungsroutine angestoßen werden. Gleichzeitig wird die Provision für den Vermittler errechnet und verbucht.
3. **Zustellung** der Polizze
   - Je nach Wunsch des Vermittlers kann die Polizze entweder direkt dem Versicherungsnehmer zugesandt werden, was durch eine vollautomatische Postversandstelle geschieht, oder aber sie wird manuell ihm selbst zugestellt.

Der Workflow-Designer legt den folgenden Prozeßablauf fest, wobei die Rollenbezeichnungen `clerkLI` für den Sachbearbeiter steht, der für den Lebensversicherungsantrag zuständig ist. Die Rolle `operator` ist die Bezeichnung für den Bearbeiter diverser Druckvorgänge, `calcClerk` ist der Sachbearbeiter für die Verprovisionierung und `shipRes` ist der Verantwortliche für die Versendung diverser Dokumente.

```
PROCESS lv (antrag INOUT LVantragT)
  DESCR 'Einfacher Lebensversicherungsantrag';
  SUBJ IS antrag.subj;
  ...

BEGIN
    IF antrag.type='normal' THEN
    clerkLI checkNorm;
    ANDPAR
      clerkLI extendCredWorth;
     |clerkLI checkHealth;
    END;
  ELSIF antrag.type='aktion' THEN
    clerkLI checkAction;
    clerkLI simpleCredWorth;
  END;
  ANDPAR
    operator makeInsPol;
   |calcClerk calcProv;
  END;
  IF antrag.shipping='versicherungsnehmer' THEN
    shipResp shipPolicyHolder;
  ELSIF antrag.shipping='vermittler' THEN
    shipResp shipInsBroker;
END;
```

Die folgende Abkürzungen werden für die Benennung der einzelnen Aktivitäten des Workflows verwendet:

A	checkAction	Antragseingabe „Aktion“
B	checkNorm	Normalantrag
C	checkHealth	normale Gesundheitsprüfung
D	extendCredWorth	Bonitätsprüfung
E	simpleCredWorth	Aktionsbonitätsprüfung
F	makeInsPol	Polizzierung
G	calcProv	Verprovisionierung
H	shipPolicyHolder	Direktversand
I	shipInsBroker	Vermittlerzusendung

Die Dauern der einzelnen Teilaktivitäten werden durch das Konstrukt der Taskspezifikation der WDL folgend festgelegt:

Aktivität	A	B	C	D	E	F	G	H	I
Dauer	5	4	7	5	3	2	1	5	6

Aus diesen Daten wird der Netzplan aus Abbildung 7.10 erzeugt[7].

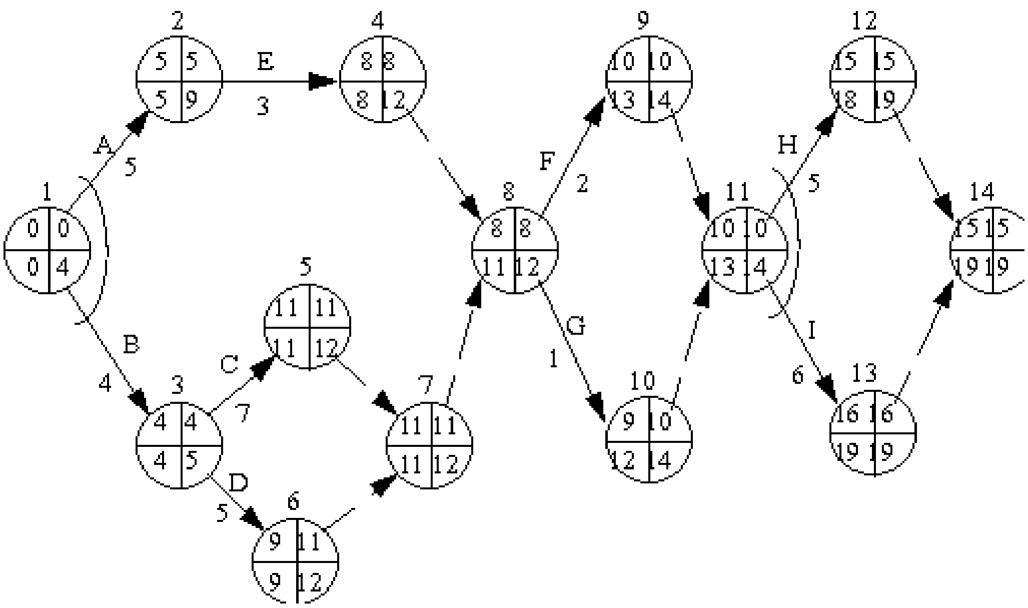

Abbildung 7.10: ePERT des Prozesses Lebensversicherungsantrag

Bei der einfache Rückrechnung für den Wert L_7^{bc} bzw. L_{13}^{bc} wird der späteste dem frühesten Startzeitpunkt gleichgesetzt, obwohl die Differenz vom Nachfolge-Knoten zum Vorgänger-Knoten negativ ist.

Auffällig ist auch der Wert $L_1^{wc} = 4$ des Anfangsknotens. Er besagt, daß der Prozeß „Lebensversicherungsantrag" auch zu einem späteren Zeitpunkt als 0 gestartet werden kann. Eine Verzögerung der vorgesehenen Gesamtdauer tritt erst dann auf, wenn der Start nach dem Zeitpunkt 4 erfolgt.

Je näher der Startzeitpunkt einer Aktivität dem spätest möglichen Termin rückt, desto eingeschränkter ist die Wahl der Alternativen. Wenn z. B. der Start des Projekts um 1 erfolgt, so sind noch beide Alternativen A und B erlaubt, da $L_2^{wc} \geq 1 + D_{12} = 9 \geq 1 + 5$ und $L_3^{wc} \geq 1 + D_{13} = 5 \geq 1 + 4$ gilt. Bei einem Startzeitpunkt von 2 ist aber nur noch der Weg über die Aktivität A erlaubt, wenn der Projektendtermin von spätestens 19 eingehalten werden soll!

In diesem Zusammenhang stellt sich die Frage, ob der kürzere Weg den Geschäftsregeln nach überhaupt zulässig ist. In diesem Beispiel müßte mit dem Vorgesetzten abgeklärt werden, ob für einen Normalantrag der Weg eines „Aktionsantrages" angewendet werden darf, um die Verspätung wieder einzuholen.

[7]Dieser Netzplan wurde dahingehend vereinfacht, daß am Ende von Verzweigungen fürs das entsprechende `end`-Konstrukt keine eigene Aktivität erzeugt wird. Dadurch wird die Darstellung ohne Semantikverlust übersichtlicher.

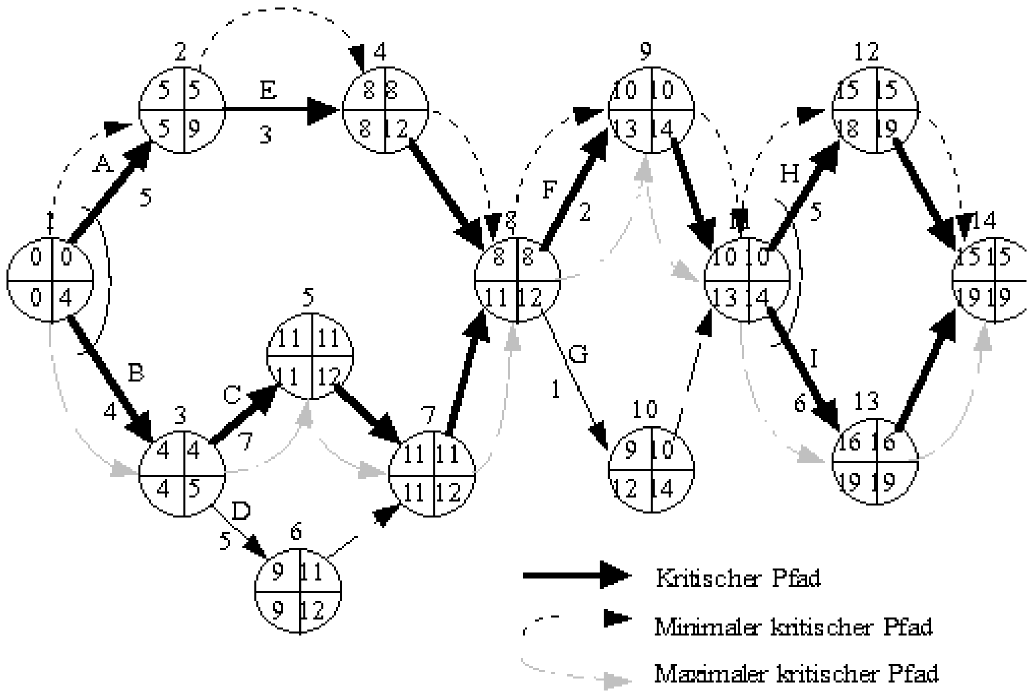

Abbildung 7.11: Kritische Pfade im Prozeß Lebensversicherungsantrag

Die Abbildung 7.11 zeigt die diversen kritischen Pfade der Antragsbearbeitung. □

7.2 Externe Zeitbedingungen

Im Kapitel 3 wurden Modellierungsansätze für transaktionale Zeitabhängigkeiten und temporallogische Beziehungen besprochen. Wie schon auf Seite 34 verdeutlicht wurde, erschweren manche Eigenschaften dieser Modellierungswerkzeuge ihren Einsatz im Workflow-Bereich. Da in dieser Arbeit nur die zeitlichen Beziehungen von Interesse sind, genügt es, die in Tabelle 7.1 festgelegten Beziehungstypen zwischen Aktivitäten zu unterscheiden.

Beziehungstyp	Abkürzung
Ende-Anfang	EA
Ende-Ende	EE
Anfang-Anfang	AA
Anfang-Ende	AE

Tabelle 7.1: Anordnungs-Beziehungstypen zwischen Aktivitäten

Die Struktur im Vorgangspfeilnetzplan ist immer vom Typ einer EA-Beziehung. **Externe** Zeitrestriktionen können aber zwischen allen Zuständen von Aktivitäten eines

Workflow-Prozesses auftreten und deshalb von jedem der o. a. Typen sein[8].

Zusätzlich zu den Anordnungsbeziehungen sind noch Vereinbarungen zu beachten, die sich auf die Zeitpunkte, Zeitintervalle bzw. absolute Kalenderwerte beziehen.

- Minimalabstände $\min d$ sind Zeiträume d zwischen zwei Zuständen von Aktivitäten, die **nicht unterschritten** werden dürfen.
- Maximalabstände $\max d$ legen Zeitdauern d fest, die zwischen zwei Zuständen von Aktivitäten **nicht überschritten** werden dürfen.
- Fixabstände fix d müssen zwischen zwei Aktivitäten **genau** eingehalten werden.
- Kalenderbezogene Einschränkungen *date D* erzeugen eine Abhängigkeit eines Zustands von einem Kalenderzeitpunkt.

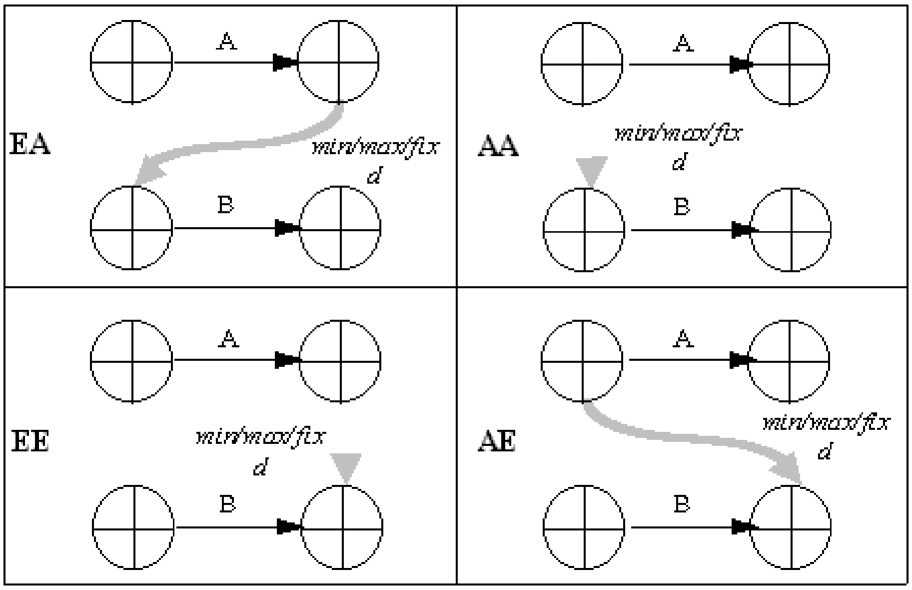

Abbildung 7.12: Externe Anordnungsbeziehungen

Die Dauer d, auf die sich die Regeln beziehen, kann auf zwei Arten festgelegt werden.

1. d wird zur Entwurfszeit festgelegt. Ein Beispiel ist die maximale Gesamtdauer eines Aktenlaufs einer Behörde mit 6 Monaten.
2. d wird erst zur Laufzeit berechnet. In Abhängigkeit von einem oder mehreren Werten wird durch eine Funktion bestimmt, wie groß der Zeitraum d ist. Siehe dazu das Beispiel 3.5 auf der Seite 34.

[8]Als extern werden zeitliche Beziehungen dann angesehen, wenn sie unabhängig von der Ablaufstruktur als vorgegebene Beziehungen, die von der Geschäftswelt bestimmt werden, vom Workflow-Designer in das Modell eingebracht werden, siehe Abschnitt 2.5.2.

Alle diese Informationen müssen zur Laufzeit bekannt sein, damit der Scheduler die Einhaltung der zeitlichen Integrität garantieren kann. Diese Integritätsbedingungen werden vom Workflow-Designer durch ein Konstrukt der WDL festgelegt.

Solche zeitlichen Zusatzbedingungen sind nicht immer einzuhalten. Manchmal verhindern Zeitschranken, die sich durch den Ablauf des Workflows (inhärent) ergeben, die Einhaltung der externen Bedingung. Eine solche Zeitrestriktion zwischen einer abhängigkeitserzeugenden Quelle s und einem abhängigen Arbeitsschritt d ist **unmöglich** einzuhalten, wenn bei den Vereinbarungen min, max und fix die Bedingungen der Tabelle 7.2 gelten.

Vereinbarung	Bedingung
fix	$(L_s^{wc} + D < E_d^{wc}) \vee (E_s^{bc} + D > L_d^{wc})$
min	$E_s^{bc} + D > L_d^{wc}$
max	$L_s^{wc} + D < E_d^{bc}$

Tabelle 7.2: Verletzung von externen Zeitrestriktionen

Diese Widersprüche sind zur Designzeit leicht zu entdecken und müssen dann vom Prozeßverantwortlichen überdacht und schließlich durch Veränderung der Struktur oder der externen Regel bereinigt werden. Zur Laufzeit wird der externen Regel folgend, der absolute Zeitpunkt für einen Prozeß berechnet und im korrekten Intervall überprüft.

Zur Spezifikation von externen Regeln noch eine Bemerkung. Es ist, ausgenommen die Datumsbeziehung, auch möglich, alle Kombinationen von Beziehungstypen und Vereinbarungen durch die Vorgangknotennetzplantechnik, insbesondere die *Metra Potential Method* MPM, darzustellen [Zim92]. Da aber

1. kein Vorteil während der Berechnung der Zeitgrenzen oder der Ausführung durch die Anwendung von MPM entsteht, und

2. die Methode der externen Regeln in späterer Folge auch für interprozeßabhängige Spezifikationen erweitert werden soll,

wird weiterhin die Vorgangpfeilnetztechnik ePERT und die Definition einer separaten Regelmenge vorgezogen.

7.3 Laufzeitverhalten

Die folgenden Abschnitte beschäftigen sich mit der Anforderung an die Laufzeitumgebung zur Bewältigung der Aufgaben, die durch die Integration von Zeit in ein Workflow-System entstehen. Insbesondere geht es um das Erkennen von Zeitüberschreitungen und die richtige Reaktion auf diese Situation. Auch die Behandlung von gewissen Ausnahmen, die durch Schleifen und Auswahl-Konstrukte notwendig sind, wird hier besprochen.

7.3.1 Verletzung inhärenter Zeitbedingungen

Eine inhärente Zeitbedingung ist verletzt, wenn die Obergrenze L_s^{wc} für den Eintritt eines Zustands s einer Aktivitäteninstanz durch eine Verzögerung des Ablaufs nicht erreicht wird. Dadurch kann die vorgegebene Gesamtlaufzeit des Prozesses nicht mehr eingehalten werden[9]. Diese Verletzung fällt laut [Gal95] in die Klasse der *semantischen Fehler*[10] und muß deshalb vom Workflow-Designer als Ausnahmefall zum allgemeinen Geschäftsfall behandelt werden.

Er muß für jeden **Prozeß** eine Voreinstellung festlegen, die die Reaktion im Falle eines Zeitfehlers bestimmt:

1. Es soll keine Reaktion erfolgen (`noreact`).
2. Der Benutzer oder Prozeßverantwortliche wird gewarnt (`warning`), einen Abbruch des Gesamtprozesses kann aber nur der Prozeßverantwortliche initiieren.
3. Es wird automatisch ein alternativer Abarbeitungsweg des Prozesses eingeschlagen, ein Zeitfehler führt demnach zu einer standardisierten Reaktion (`alternativ`).
4. Der Prozeß wird abgebrochen, da das Ergebnis nicht mehr verwertbar ist (`timeabort`). Es werden nun alle kompensatorischen Tätigkeiten durchgeführt, die für diesen Fall vorgesehen sind.
5. Der Benutzer oder Prozeßverantwortliche trifft die Entscheidung über das weitere Fortfahren (`user`). Er kann alle vorher angeführten Reaktionen veranlassen, also
   (a) fortfahren
   (b) fortfahren und alle zukünftigen Warnungen unterdrücken,
   (c) Alternativen als Ausnahmebehandlung anstoßen oder
   (d) die Aktivität abbrechen,

   Außerdem muß der Workflow-Designer die Frage der Rechteverwaltung klären, da die Reaktion auf einen Ausnahmefall eventuell stärkere Rechte erfordert.

[9] Praktisch besteht natürlich durch verstärkte Ressourcenzufuhr oder Notmaßnahmen die Möglichkeit, die Laufzeit der folgenden Aktivitäten unter den Mittelwert zu drücken und dadurch mit der Summe der Laufzeiten wieder in die erlaubten Grenzen zu kommen.

[10] Mißlingen einer Aktivität, die nicht auf Serverausfall, Kommunikationsstörung oder technisches Gebrechen einer Applikation zurückzuführen ist.

Diese möglichen Reaktionen auf einen Zeitfehler werden in einer Datenbankrelation abgelegt (siehe Abb. A.2 im Anhang A). Eine Reaktion auf einen Fehler, der durch Übertretung einer inhärenten Zeitrestriktion eintritt, gilt also für den gesamten Prozeßtyp.

Der Designer entscheidet dadurch, ob eine Zeitüberschreitung u. a. ein fataler Fehler oder eine mehr oder weniger alltägliche Situation darstellt und entwirft das Verhalten des Schedulers.

Das allgemeine Ausführungsmodell zur Abarbeitung einer Aktivität eines Workflows [Lie95] ist in der Abbildung 7.13 zu sehen. Nach der Überprüfung einer *precondition* wird der eigentliche Task ausgeführt. Tritt ein technischer Fehler auf, wird dies durch das Ereignis *abort* angezeigt und der Task abgebrochen. Ist das nicht der Fall, wird die Nachbedingung *postcondition* ausgeführt. Je nach dem Ergebnis des vorherigen Arbeitsschrittes wird nun entschieden, ob dieser erfolgreich war (*succ*) oder ob ein semantischer (*fail*) Fehler auftrat. Ein Beispiel für einen semantischen Fehler ist das Mißlingen einer Hotelreservierung wegen Überbuchung.

Auch bei der Überprüfung der *postcondition* kann durch technische Probleme ein Abbruch *abort* auftreten. Aufgrund des *clean room*-Ansatzes werden etwaige Kompensationstätigkeiten bereits in der *postcondition* durchgeführt.

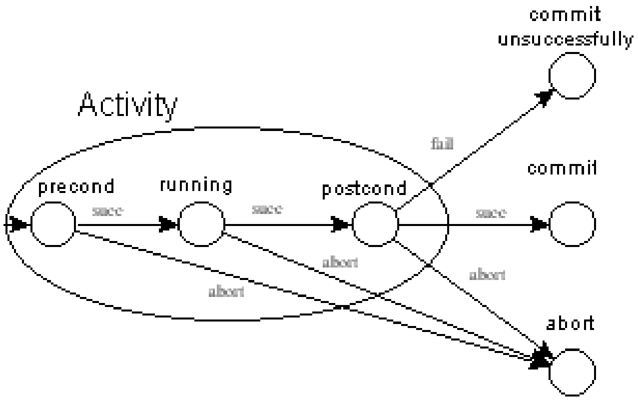

Abbildung 7.13: Zustandsdiagramm einer Aktivität

Zur Behandlung von Zeitfehlern wird dieses Ausführungsmodell, insbesondere die Funktionalität der *postcondition*, erweitert.

Konzeptionell kann ein Zeitfehler nie **vor** der Ausführung einer Aktivität auftreten, da schon bei der Vorgängeraktivität diese Bedingung überprüft wurde und Wartezeiten bereits der aktuellen Aktivität zugerechnet werden. Deshalb findet die Überprüfung von zeitlichen Bedingungen in der *post condition* eines Tasks statt.

Wird durch die Prüfung ein Zeitfehler gefunden, so muß entsprechend der für den Prozeßtyp voreingestellten Reaktion entschieden werden (Abbildung 7.14):

- Bei den Reaktionstypen `user` (der Benutzer wählt die Option *Abbruch*) und `timeabort` erzeugt das System ein *commit unsuccessfully*. Die Aktivität wird bei der Zurücksetzung entsprechend ihres Stornotyps[11] behandelt.

- Die Aktivität kann ordnungsgemäß abgeschlossen werden. Dies tritt dann ein, wenn `user` (ohne Abbruch), `noreact` oder `warning` als Reaktion definiert waren. Falls in so einem Fall der transaktionsspezifische Parameter der **folgenden** Aktivität *non vital* ist, besteht die Möglichkeit durch Nichtausführung Zeit gutzumachen.

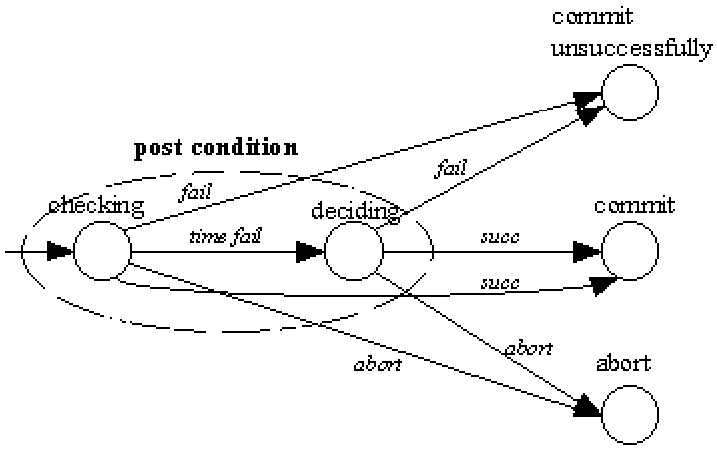

Abbildung 7.14: Detailliertes Zustandsdiagramm der *postcondition*

Der Vorteil, Zeitfehler auf die bestehenden Endzustände *commit*, *commit unsuccessfully* und *abort* des Exekutionsmodells abzubilden, besteht darin, daß die bisherige Architektur des Workflow-Servers von Panta Rhei nicht erweitert werden muß, um das neue Konzept aufzunehmen.

7.3.2 Verletzung externer Zeitbedingungen

Überwachung von Zeitregeln

Bei jeder Änderung eines Zustands wird die Menge aller externen Zeitregeln auf das Eintreffen einer Regelbedingung überprüft[12]. Wenn sich zum Beispiel eine Regel „Nach dem Ende von A darf maximal 1 Tag vergehen, bis B startet.“ im System befindet, geschieht nach dem Eintreten des Endzustands von A folgendes:

1. Die Zeit und das Datum für den Endzustand von B werden ermittelt. Das geschieht durch die Addition des in der Regel festgelegten Intervalls oder des Rückgabewerts der definierten Funktion mit dem Systemdatum.

[11] Der Stornotyp einer Aktivität bestimmt die Art der Zurücksetzung. Erlaubte Stornotypen einer Aktivität sind *none*, *undoable*, *compensatable* oder *critical* und werden bei der Taskdefinition festgelegt.

[12] Diese Regeln werden in der Datenbanktabelle `wf_time_dep` verwaltet, gezeigt in Anhang A.3

2. Regelbezeichnung, ermitteltes Datum und der Vereinbarungstyp `min`, `max` oder `fix` werden in die Datenbanktabelle `wf_act_active_timdep` eingetragen.

Die Workflow-Engine überprüft regelmäßig diese Tabelle und erkennt Übertretungen von aktivierten Regeln vom Typ `max`, `date` und `fix`, wenn das **Systemdatum** `sysDate` größer als das geforderte Eintrittsdatum `depDate` für den abhängigen Zustand ist.

Auch wenn der **Eintrittszeitpunkt** eines Zustands vor dem geforderten Zeitpunkt liegt (Typ `min`, `fix` und `date`), ist ein Zeitfehler aufgetreten. Falls es sich bei dem abhängigen Zustand um den Start einer Aktivität handelt, kann durch eine Verzögerung der Ausführung der Zeitfehler korrigiert werden, was aber bei einem Endzustand unmöglich ist.

Jedesmal wenn ein abhängiger Task ohne Regelübertretung ausgeführt wurde, wird das entsprechende Objekt aus `wf_act_active_timdep` gelöscht. Auch wenn bei einer `min`-Bedingung der definierte Zeitpunkt überschritten wurde, kann diese Bedingung entfernt werden.

Zusätzlich muß beim Abbruch eines Prozesses die Menge der aktiven Zeitabhängigkeiten bereinigt werden, d. h. jene aktivierten Objekte werden entfernt, deren Voraussetzungen nicht mehr zutreffen. Bei obigem Beispiel wird nach dem (kaskadierenden) Abbruch des Prozesses der Aktivierungszeitpunkt mit allen Informationen aus `wf_act_active_timdep` gelöscht.

Reaktion auf Verletzung

Für inhärente Zeitbedingungen genügt es, wenn für jeden Prozeßtyp eine Reaktion auf Verletzungen spezifiziert wird, da Zeitüberschreitungen sich vor allem auf die Laufzeit des Gesamtprozesses auswirken.

Externe Einschränkungen beziehen sich aber auf den Zustand von Aktivitäten und jede dieser Regeln hat eine andere Semantik. Es darf z. B. nicht sein, daß eine verspätet abgeschickte Flugbuchungsbestätigung die selben Auswirkungen wie eine versäumte Hotelreservierung hat, auch wenn beide Bedingungen sich auf den selben Prozeß beziehen. Das Vorgehen, eine Reaktion pro Prozeßtyp festzulegen, ist zu einschränkend und läßt dem Workflow-Designer keinen Spielraum.

Deshalb kann er für **jede externe** Zeitbedingung eine Reaktion festlegen. Nur wenn keine Reaktion spezifiziert ist, wird die Voreinstellung des Prozeßtyps übernommen, die für inhärente Regeln gilt.

Der Typ dieser Reaktion ist derselbe wie er bereits für inhärente Regeln in der Aufzählung von Seite 92 festgelegt wurde.

Durch die Möglichkeit, für jede Regel eine individuelle Reaktion festzulegen, steht dem Prozeßdesigner ein flexibles Mittel zur Verfügung, den Workflow dem Ablauf der Geschäftsprozesse anzugleichen und auf jede Verletzung einer Geschäftsregel adäquat zu reagieren.

7.3.3 Prioritäten

Eine wichtige Aufgabe für das Workflow-System ist die Entdeckung von aufgetretenen Zeitfehlern. Vorteilhafter ist aber die Strategie, Zeitfehler soweit wie möglich zu verhindern. Durch die Zeitinformationen (speziell die Standardabweichungen), die der Scheduler über die Abläufe der Task und Prozesse hat, kann die Gefahr einer Verletzung einer inhärenten Zeitbedingung rechtzeitig erkannt werden.

Der Scheduler überprüft regelmäßig alle Prozeßinstanzen. Ein Prozeß ist dann in Gefahr, sich zu verspäten, wenn der aktuelle Zeitpunkt den Erwartungswert eines Zustands überschreitet. Je weiter die Zeit fortschreitet, ohne daß der geforderte Zustand eintritt, desto gefährdeter ist eine pünktliche Beendigung des Prozesses. Bei Überschreitung der Zeitgrenze $sysTime > \mu + \sigma$ ist eine Verspätung eingetreten.

Durch diese Beobachtung ist eine kontinuierliche Bewertung des Status einer Aktivität möglich. Wird eine Verzögerung festgestellt, so hat der Scheduler die Möglichkeit, die Priorität des Task anzuheben (dynamische Prioritätenberechnung). Beispielsweise kann er veranlassen, den Task im Eingangskorb des Bearbeiters (*to-do-list*) neu einzureihen oder eine Warnung ausgeben, um dem Benutzer die Dringlichkeit der Bearbeitung anzuzeigen.

Auch die Monitoringtätigkeit, also die Beobachtung des Gesamtsystems, wird durch die vorhandene zeitliche Funktionalität direkt unterstützt, da alle Systeminformationen jederzeit zur Verfügung stehen.

7.4 Erweiterung der WDL zur Zeitverarbeitung

Zur Verwaltung der Zeitspezifikation werden zusätzliche syntaktische Elemente benötigt die in die Workflow-Beschreibungssprache WDL aufgenommen werden.

1. Für die **Prozeßdefinition** wird im Deklarationsteil ein weiteres syntaktisches Konstrukt `<reaction>` zur Verfügung gestellt. Damit ist es nun möglich, für jeden Prozeßtyp eine Reaktion auf einen Zeitfehler zu bestimmen. Die genaue Syntax ist in der Abbildung 7.15 zu sehen.

2. Zu jedem **Tasktyp** werden folgende Werte festgelegt:

   $\mathbf{a}_t$ ist die kürzest mögliche Zeitdauer für diese Basisaktivität,

   $\mathbf{b}_t$ ist die maximale Dauer und

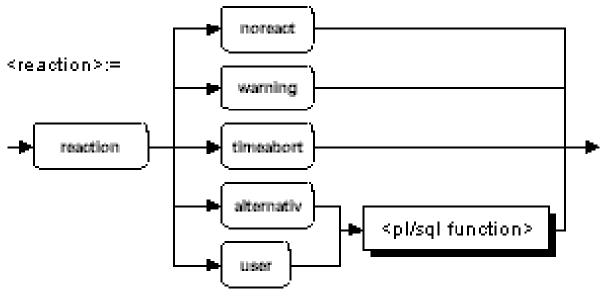

Abbildung 7.15: Syntax der Reaktionsfestlegung

$\mathbf{m}_t$ ist der häufigste Wert.

Aus diesen werden der Mittelwert und die Varianz der Dauer der Aktivität errechnet. Diese Werte werden bei der Tasktypdefinition durch die Elemente der Abbildung 7.16 festgelegt

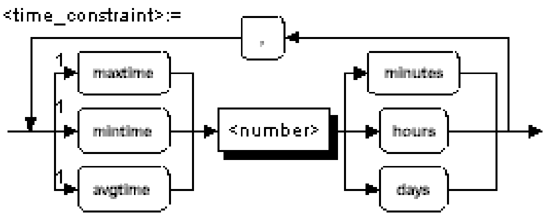

Abbildung 7.16: Syntax der Zeitfestlegung für Tasks

Pro Tasktyp kann nur jeweils einmal eine Mindest-, Höchst- und Durchschnittslaufzeit bestimmt werden. Durch die Einbeziehung von Zeitgrenzen, die zur Entwurfszeit errechnet werden, ist es nicht mehr möglich, die maximale Bearbeitungszeit durch den Wert eines Formularfeldes zu ermitteln[13], da diese Information erst zur Laufzeit eines Tasks zur Verfügung steht. Außerdem kann dieser Zeitwert für jede Instanz ein anderer sein. Eine generelle, für alle Instanzen geltende Planung wird damit unmöglich gemacht.

3. Da drei Alternativen zur Behandlung von **Schleifen** zur Verfügung stehen, gibt es ebenso viele Möglichkeiten zu deren Festlegung.

   1) Die Bestimmung der **Anzahl der Durchläufe** wird durch das Konstrukt der Abbildung 7.17 nach dem Schleifenkopf festgelegt.

      Beispiel 7.7 *Festlegung der Schleifeniteration*
      Für eine abweisende Schleife, die die Existenz einer Unterschrift eines Antrages prüft, werden die folgenden Iterationshäufigkeiten bestimmt.

[13] Vgl. die Änderung mit dem Syntaxdiagramm der Taskdefinition von Seite 70.

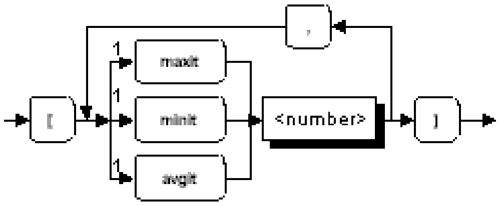

Abbildung 7.17: Festlegung der Schleifeniterationen

```
WHILE antrag.signature == '' DO //no signature entered
  [minit 0, maxit 2, avgit 1]
  officialInCharge requestSignature(antrag);
  ...
END;
```

□

2) Die Festlegung eines konstanten Wertes ist ein Spezialfall der Iterationsfestlegung. Deshalb muß kein neues Konstrukt eingeführt werden. Ein konstanter Wert wird durch `avgit <number>` definiert, wenn die beiden anderen Felder `maxit` und `minit` nicht verwendet werden.

3) Wenn die gesamte Schleife zeitlich bewertet werden soll, wird nach dem Schleifenkopf eine Zeiteinschränkung `'[' <time_constraint> ']'` definiert.

Beispiel 7.8 *Festlegung der Schleifendauer*

```
REPEAT
  [mintime 1 minutes, maxtime 3 minutes, avgtime 1 minutes]
  customer enterSignature(antrag);
  ...
UNTIL antrag.signature # '';
```

□

Falls kein Wert für die Schleifenabarbeitung bestimmt ist, wird die Konstante 1 für die Zeitbewertung angenommen.

4. **Externe** Zeitrestriktionen werden für jeden Prozeß innerhalb des Deklarationteils spezifiziert. Es können beliebig viele Regeln gebildet werden, die sich auf Zustände von Tasks des Prozesses beziehen. Es steht dem Prozeßdesigner ein Sprachelement (Abbildung 7.18) zur Verfügung, mit dem er relative und absolute Zeiten zwischen Zuständen von Aktivitäten formulieren kann.

Beispiel 7.9 *Externe Zeitrestriktion*

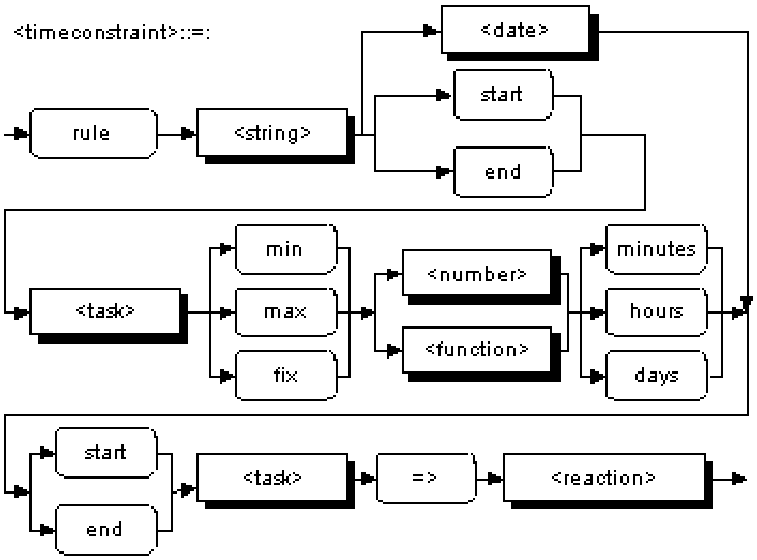

Abbildung 7.18: Syntax der externen Zeitregel

Auf Seite 17 wird im Beispiel 2.4 bestimmt, daß die Bestätigung KB für den Kunden spätestens 24 Stunden nach der Flugbuchung B bearbeit sein muß. Gelingt das nicht, wird von Fall zu Fall entschieden, wie ein Kunde verständigt werden soll. Die alternative Entscheidung `r1_proc` wird vom Designer entworfen und an die Regel gebunden.

Durch eine externe Regel wird dieser Sachverhalt so formuliert:

```
RULE r1
  END FB MAX 24 HOURS END KB
=>
  USER r1_proc
```

Sie ist folgend zu lesen: „Nach dem Ende von FB dürfen maximal 24 Stunden vergehen, bis das Ende von KB eintritt. Ist das nicht der Fall, muß der Benutzer entscheiden, ob abgebrochen, weiter fortgefahren oder die Prozedur `r1_proc` ausgeführt wird.“ □

Weiters ist die Konjunktion und Disjunktion von solchen Regeln erlaubt, um komplexe zeitliche Bedingungen formulieren zu können.

Beispiel 7.10 *Spezifikation einer komplexen externe Zeitregel*

Die folgende Prozeßspezifikation ist eine Erweiterung des Lebensversicherungsantrag-Prozesses von Seite 86. Zusätzlich zu den dort festgelegten Abläufen des Prozesses muß man noch die folgenden zeitlichen Einschränkungen in die Abarbeitung mit einbeziehen:

1. Zwischen der Gesundheitsprüfung des Normalantrags (Aktivität C) und der Polizzierung (F) muß mindestens ein Zeitraum von 7 Tagen vergehen, um die Ergebnisse eventueller Arztanfragen oder Anamneseanforderungen abzuwarten.
2. Wenn ein Aktionsantrag (A) aufgenommen wird, dürfen bis zur Polizzierung nur 5 Tage vergehen.

Gelingt die Einhaltung der Vorgaben nicht, muß im ersten Punkt von Fall zu Fall das weitere Vorgehen bestimmt werden. Im zweiten Fall wird der Prozeßverantwortliche benachrichtigt. Die Prozessbeschreibung wird jetzt um diese Regeln erweitert.

```
PROCESS lv (antrag INOUT LVantragT)
  DESCR 'Einfacher Lebensversicherungsantrag';
  SUBJ IS antrag.subj;
  ...
  (RULE Delay4Answer
    END C MIN 7 DAYS START F => USER DelayOccured(antrag))
  AND
  (RULE SpeedUp
    START A MAX 5 DAYS END F => WARNING);
  ...
BEGIN
  ...
END;
```

□

Kapitel 8

Implementierung

Die Umsetzung der in den vorigen Kapiteln beschriebenen Konzepte in den Prototypen Panta Rhei erfolgte durch die in die relationale Datenbank ORACLE integrierte Datenbankprogrammiersprache `PL/SQL`. Neben allen Steuerungselementen imperativer Programmiersprachen bietet diese Sprache einen einfachen Zugriff auf die Datenbank durch die Einbettung von SQL-Konstrukten an.

Ein Workflow-Managementsystem muß die Funktionalität für die *build-time* und *run-time*-Komponenten zur Verfügung stellen. Die Aufgabe des entwickelten Programms besteht darin, nach der Übersetzung der WDL-Prozeßbeschreibung in die interne Darstellung von Panta Rhei aus diesen generierten Daten eine ePERT-Netzstruktur mit allen notwendigen Zeitinformationen aufzubauen. Diese Prozedur kann man als erweiterte Funktionalität des Compilers betrachten und deshalb dem *build-time*-Bereich zuordnen.

Die durch den WDL-Compiler [Tha96] erzeugten Informationen werden in ein Metaschema, das als relationale Datenbank in ORACLE 7.0 verwirklicht ist, abgelegt. Darauf greift das Programm `net.genNet(<process>)` (siehe Anhang B.1) zu. Der Parameter dieser Prozedur, die nach dem Kompilieren der WDL-Programme aufgerufen wird, ist der Name des zu bearbeitenden Prozesses. Die Haupttätigkeiten sind:

1. Erzeugen der Netzstruktur durch die Prozedur `genStruct`. Die Menge aller Kontrollflüsse wird kontinuierlich abgearbeitet und nach dem Einfügen in das Netz wird der bearbeitete Task aus dieser Menge entfernt.

2. Berechnen der Werte E durch rekursive Traversierung des Netzes. Die Funktion `calcEVal` wird mit der aktuellen Prozeßbezeichnung und dem Anfangszustand aufgerufen. Der Rückgabewert dieser Funktion ist der letzte Zustand des ePERT-Netzes.

3. Berechnen der Werte L mittels der Prozedur `calcLVal`.

4. Durch `tc` (*time check*) wird überprüft, ob inkompatible externe Zeitrestriktionen entworfen wurden. Fehlermeldungen werden in der Datenbank abgelegt.

Die Netzstruktur wird vorwiegend aus der Analyse des Kontrollflusses erzeugt. Bei der Implementierung wurde vorerst noch kein Wert auf Optimierung gelegt, da das Hauptaugenmerk auf die Demonstration der Umformung gerichtet wurde.

8.1 Strukturaufbau

Die Prozedur `genStruct` liest sukzessive alle Kontrollflüsse der WDL ein und erzeugt in jedem Schritt eine entsprechende Netzstruktur. Die Cursor-Struktur `taskFlows`, über den die Hauptschleife iteriert, liest alle Kontrollflüsse aus der Datenbanktabelle `wf_flow`. Diese Kontrollflüsse sind einfache Zuordnungen der Form „Vorgänger, Nachfolger, Kontrollflußtyp". Diese Flüsse werden so abgearbeitet, daß der Vorgänger bereits im Netz sein muß, bevor ein Nachfolger eingefügt werden kann.

Im ersten Schritt der Prozedur werden bereits vorhandene alte Netzstrukturen des aktuell bearbeiteten Prozesses gelöscht, danach wird der erste Task des Workflows gesucht. Aus diesen Informationen werden der Anfangs- und Endzustand der ersten Netzaktivität erzeugt und der Task durch die Funktion `newNetEvent(<aktueller Prozess>, <Workflow-Task>, <Startzustand>, <Endzustand>` dazwischen gehängt.

Danach wird der Cursor `taskFlows` in einer Schleife abgearbeitet, wobei `v_wfFlow` das aktuelle Tupel des Cursors ist. Ein wichtiger Schritt ist die Bestimmung des Endzustands des Vorgängers des aktuell bearbeiteten Workflow-Tasks`v_wfFlow.succ`. Dieser ist nämlich der Anfangszustand des aktuellen Tasks. Je nach der Anzahl und Typ der vom Vorgängerzustand ausgehenden Verbindungen wird entschieden, ob es sich um eine Sequenz (bisherige Anzahl der ausgehenden Verbindungen ist Null), oder um eine Verzweigung handelt. Diese Information wird im jeweilige Netzzustand gespeichert.

Nachdem festgestellt wurde, welche Netzkontrollstruktur mit dem Vorgängerzustand des aktuellen Tasks beginnt, muß nun bestimmt werden, welches Kontrollstruktur-Ende dieser Zustand markiert. Zu diesem Zweck wird die Zahl der WDL-Kontrollflüsse, deren Endpunkt der aktuelle Task ist, abgezählt. Falls der aktuelle Task noch nie bearbeitet und ins Netz eingefügt wurde, muß eine Scheinaktivität erzeugt werden, die zum Anfangszustand des aktuellen Task führt, um den expliziten Endzustand (um so einen handelt es sich im vorliegenden Fall) zu markieren. Erst dann wird diese WDL-Tätigkeit in das Netz eingebunden. Außerdem wird noch der Typ der Verzweigung protokolliert, um die Berechnungen korrekt durchführen zu können.

Wenn aber der Tasks bereits als Netzaktivität vorhanden ist, dann wird nur eine Scheinaktivität zwischen dem Vorgängerendzustand und dem Startzustand der Netzaktivität eingefügt. Handelt es sich beim aktuellen `v_wfFlow` nicht um eine Verzweigung, dann wird nur die neue Aktivität mit einem Endzustand dazugehängt.

Das Ergebnis dieser Prozedur ist ein ePERT-Netzplan, der die Struktur der Kontrollflüsse der WDL-Prozeßbeschreibung korrekt widerspiegelt. Dieser Netzplan ist in den Datenbankrelationen `net_state` und `net_event` abgelegt (siehe Anhang A.3).

Im nächsten Schritt werden die Werte für den Erwartungswert und die Varianz der frühesten Beginnzeitpunkte der Netzaktivitäten errechnet.

8.2 Berechnung der frühesten Startwerte

Die Funktion `calcEVal` berechnet das früheste Eintreten eines Zustands einer Netzaktivität. Ausgehend vom Startzustand eines ePERT-Netzplans werden entsprechend der Berechnungsvorschriften (Kapitel 5.2) die Werte aktualisiert. Das generelle Vorgehen ist eine Variante des *iterativ deepening*, d. h. ausgehend von einem Startzustand wird nach der Aktualisierung solange über den nächstgrößeren Nachfolgeknoten in die Tiefe (nach rechts) des Netzes gegangen, bis das Ende einer Verzweigung erreicht wurde. Erst wenn alle an einem Verzweigungsende ankommenden Wege bearbeitet wurden, wird weiter in die Tiefe gerechnet. Durch Backtracking können auf diese Weise alle Wege, die durch das Netz führen, mit den korrekten Werten aktualisiert werden.

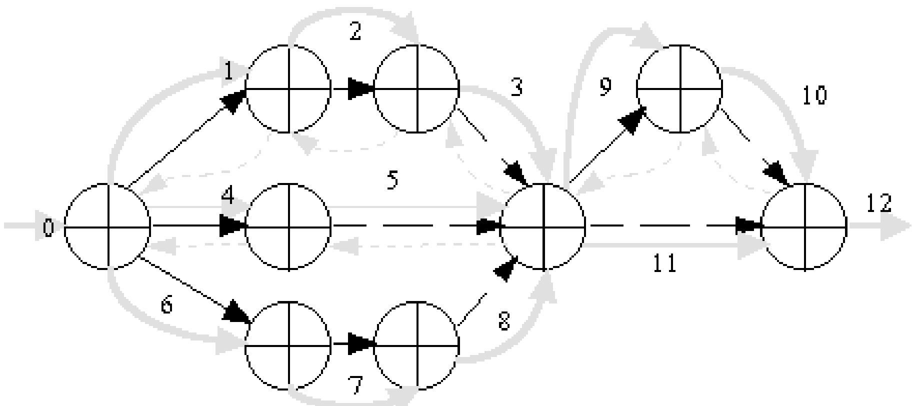

Abbildung 8.1: Abarbeitungsfolge bei Berechnung der frühesten Werte

In der Abbildung 8.1 wird diese Vorgehensweise dargestellt, jeder numerierte Pfeil stellt einen rekursiven Aufruf von `calcEVal` dar, der den betreffenden Zustand des Netzes aktualisiert. Nachdem das Ende einer Verzweigung erkannt wurde, wird der nächste Zweig versucht, bis alle bearbeitet wurden.

Der Rückgabewert dieser Funktion ist die Nummer des letzten Zustands des Netzes.

8.3 Berechnung der spätesten Startwerte

Die Prozedur `calcLVal` zur Errechnung der spätesten Startwerte einer Aktivität wird mit der Nummer des letzten Zustands eines ePERT-Netzes als Parameter aufgerufen. Vom Aufbau her ist sie das Spiegelbild der Funktion `calcEVal`. Auch hier wird mit einer Variante des *iterativ deepening* das Netz entsprechend der Berechnungsvorschriften aktualisiert.

Anzumerken ist nur, daß beide Unterprogramme eine Hilfsstruktur `nodeSet` benötigen, die durch den Aufruf der Prozedur `accumNodes` gebildet wird. Diese Knotenmenge enthält alle Zustände des aktuellen ePERT-Netzes. Bei der Hinrechnung werden für jeden Zustand die bereits bearbeiteten Verzweigungsenden protokolliert, um erkennen zu können, ob die gegenwärtige Rekursion die Berechnung bereits komplettierte. Dieses Prinzip wird auch bei der Rückrechnung angewandt. Wenn erkannt wurde, daß weiter in die Tiefe (nun nach links) des Netzes gearbeitet werden darf, wird der entsprechende Zustand aus dieser temporären Struktur gelöscht.

8.4 Überprüfung auf unverträgliche Zeitvorgaben

Der abschließende Schritt, der noch zur Kompilierzeit durchgeführt wird, ist die Überprüfung auf Kompatibilität zwischen inhärenten und externen Zeitvorschriften. Da jetzt die Werte des ePERT-Netzplans vorliegen, kann die Menge der externen Regeln gegen die Zeitgrenzen der Netzaktiväten geprüft werden.

Die Regelmenge, die sich auf den gegenwärtigen Prozess bezieht, wird durch Einlesen der Abhängigkeiten aus der Datenbankrelation `wf_time_dep` in den Cursor `orthDep` gebildet. Dieser Cursor wird Regel für Regel in einer Schleife abgearbeitet.

Im ersten Schritt werden entsprechend des Abhängigkeitstypen (EA, EE, AE oder AA) die korrespondierenden Zustände der Aktivitäten selektiert.

Die nächste Aktion ist die Bestimmung des Beziehungstypen `min`, `max` oder `equal`. Danach erfolgt die Überprüfung des korrekten Zeitintervalls, siehe dazu auch die Ausführungen und die Tabelle 7.2 auf der Seite 91. Diese Berechnungsvorschriften werden nun angewendet, um die prinzipielle Möglichkeit einer externen Zeitvorschrift zu erkennen. Falls eine Inkonsistenz existiert, werden die Prozessbezeichnung, die Regelnummer, die Bezeichnung des abhängigen und der abhängigkeitserzeugenden Tasks in einer Datenbankrelation abgelegt und deren Inhalte nach dem Programmablauf ausgegeben.

Kapitel 9

Resümee

9.1 Zusammenfassung

In dieser Arbeit wurde untersucht, wie Zeitmodellierung in ein Workflow-Managementsystem eingebracht und angewendet werden kann. Teile der dabei gewonnen Ergebnisse sind auf der Basis des prototypischen Workflow-Managementsystems Panta Rhei realisiert worden. Dabei wurden zwei Klassen von Zeitabhängigkeiten identifiziert, die in diesem Anwendungsgebiet von Bedeutung sind. Sie wurden als inhärente und externe Zeitrestriktionen bezeichnet.

Nach der Prüfung der Eignung von Konzepten der Transaktionsmodellierung und der temporalen Logik zur Modellierung und Umsetzung dieser Einschränkungen, wurde mit der Netzplantechnik eine Methode des Projektmanagements gewählt. Netzplantechniken bieten viele Vorteile, die ihren Einsatz zur Darstellung inhärenter Zeitrestriktionen rechtfertigen. Die hier gewählte stochastische PERT-Methode bringt wegen der Verwendung normalverteilter Ausführungszeiten für Basisaktivitäten gute Resultate bei der Schätzung der Dauer und bei der Bewertung der Ergebnisse .

Da aber konventionelle Netzpläne nicht genügend Funktionalität zur verlustfreien Abbildung des Verhaltens von Workflow-Prozessen bieten, wurden weitere Elemente zur Netzplantechnik hinzugefügt. Ein solches Element ist die Alternative, die die nun als ePERT-Netzplantechnik bezeichnete Methode um die Darstellung unterschiedlicher Exekutionspfade erweitert. Aufgrund der veränderten Semantik wurden auch die Berechnungsvorschriften für die Zeitschranken der Aktivitätenzustände adaptiert.

Weiters wurden auch Konzepte vorgestellt, mit denen Zeitgrenzen der Kontrollstrukturen Schleife und Auswahl in einem ePERT-Netz dargestellt und berechnet werden können.

Durch die Vorberechnung struktureller Zeitgrenzen kann der Zustand eines Workflow-Prozesses jederzeit bewertet und auf sich abzeichnende Zeitüberschreitungen sofort reagiert werden. Dem Prozeßverantwortlichen wird dadurch ein Werkzeug in die Hand gegeben, mit dem er **aktiv** Abweichungen entgegensteuert.

Wichtig ist auch die Definition externer Zeitrestriktionen durch ein Konstrukt, das in die Workflow-Beschreibungssprache WDL der Modellierungskomponente von Panta Rhei aufgenommen wurde. Damit können zeitbezogene Geschäftsregeln ohne Zusatzaufwand festgelegt werden. Das WfMS überprüft die prinzipielle Machbarkeit dieser Spezifikationen und verwaltet die Überprüfung zur Laufzeit.

Das System erkennt nicht nur aufgetretene Fehler, es bietet auch einen Mechanismus zur Reaktion auf entdeckte Ausnahmesituationen. Durch die Festlegung voreingestellter Reaktionstypen für die Verletzung von Zeitregeln wird ein Ausnahmefall auf eine Standardsituation zurückgeführt. Damit entfällt eine manuelle Interaktion des Prozeßverantwortlichen zur Behebung der Anomalie.

Das vorrangige Ziel ist es jedoch, Fehlersituationen überhaupt nicht auftreten zu lassen. Auch hier kann der vorgestellte Mechanismus gute Dienste leisten. Aufgrund der stochastischen Natur der Zeitberechnung sind Engpässe früh erkennbar und der Workflow-Scheduler hat die Möglichkeit, durch Manipulation der Prioritäten einer Aktivität, Verspätungen zu verhindern.

In diesem Zusammenhang sei hier noch zu erwähnen, daß die in dieser Arbeit gewonnen Erkenntnisse sich speziell auf Panta Rhei beziehen. Doch die Konzepte zur Behandlung der externen und inhärenten Zeitrestriktionen sind auch für andere Architekturen gültig. Zwar können sich die verschiedenen Workflow-Beschreibungssprachen erheblich unterscheiden (Petri-Netze, Zustandsübergangsdiagramme, Scriptsprachen,...), ihr Aufgabengebiet ist aber immer die Prozeßbeschreibungen. Deshalb ist die interne Verarbeitung zeitbezogender Konzepte durch ePERT-Netzpläne in den meisten Fällen denkbar. Wie groß der Aufwand zur Einbringung von Regelmengen zur Beschreibung externer Beziehungen ist, kann sich von Fall zu Fall erheblich unterscheiden.

9.2 Ausblick

Die in dieser Arbeit vorgestellte Integration von Zeit in ein Workflow-System ist ein weiterer Schritt zur automatischen Unterstützung administrativer Prozesse. Um dieses Ziel zu erreichen, müssen aber noch einige offene Fragen untersucht werden.

Ein wichtiger Punkt in diesem Zusammenhang ist die Zusammenführung von Konzepten der Zeitmodellierung mit jenen der Transaktionsspezifikation. Ziel dieses Vorgehens ist eine umfassende und konsistente Modellierung allgemeiner Abhängigkeiten, um die Modellierungskomponente mächtiger und das Workflow-Managementsystem sicherer zu machen.

Ein weiterer wichtiger Aspekt ist auch die Problematik der Risikobewertung von Prozeßexekutionen. Dazu muß der Ansatz, der bereits in der Zeitbewertung erfolgreich angewendet wird, erweitert werden. Das System bewertet dann nicht nur die erfolgreichen Exekutionspfade eines Prozeßablaufs, sondern es stellt auch die möglichen Abbrüche und Wiederausführungen von Exekutionspfaden dar und validiert diese. Interessante Fragen in diesem Zusammenhang sind dann die Bewältigung der kombinatorischen Explosion der Wege oder die sinnvolle Interpretation der Bewertungen.

Die Vision, die durch die Verwirklichung dieser technischen Herausforderungen näher kommt, ist die Entwicklung eines „Workflow-Leitstandes". Ein solcher Leitstand soll die Grundlagen schaffen, die administrativen betrieblichen Vorgänge zu steuern und Schwachstellen zu bekämpfen. Der Mitarbeiter wird dadurch von lästiger Routinearbeit befreit und kann sich höherqualifizierteren Herausforderungen stellen.

Doch der durchschlagende Erfolg von Workflow-Systemen läßt sich nicht nur auf die Automatisierung von Bürotätigkeiten zurückführen. Die Vorteile eines Systems, daß langweilige administrative Tätigkeiten abnimmt und dadurch mehr Spielraum für wichtige und wertschöpfende Arbeiten läßt, macht es auch für andere Gebiete interessant. Als ein Beispiel sei hier [SV96] angeführt, wo versucht wird, ein WfMS zur Unterstützung projektähnlicher Forschungsarbeiten einzusetzen, um wertvolle Kapazitäten freizuhalten.

Workflow-Management ist ein intuitives und mächtiges Konzept, das revolutionäre Auswirkungen auf viele administrative Tätigkeiten haben wird. Die Palette aller Einsatzmöglichkeiten wird auch in Zukunft immer reicher, und die Liste der technischen und sozialen Herausforderungen, die es noch zu bewältigen gilt, ist lange.

Anhang A

Panta Rhei

A.1 Sprachbeschreibung WDL

Quelle [Tha96]

```
<WDL>         ::=  { <ProcDef>            // Prozesz definieren
                   | <TaskDef>            //Tasks
                   | <RoleDef>            // Rollen
                   | <OrgDef>             // Aufbauorganisation
                   | <RecDef> }           // Kommunikation
<ProcDef>     ::=  <ProcHead>             // Prozeszkopf
                   <ProcDecl>             // Deklarationsteil
                   <ProcBody> ;           // Prozeszkoerper
<ProcHead>    ::=  process <Name>         // Prozeszname+Argumente
                   ( [ <Name> [in | out | inout ] <Name> {,<Name>
                   [in | out | inout ] <Name>} ] )
<ProcDecl>    ::=  [processgroup <Name>;]  // Prozeszgruppe
                   [owner <Name>;]        // Eigentuemer
                   [descr <String>;]      // Beschreibung
                   [<MaxTime>]            // max. Ausfuehrungszeit
                   {<Name> <Name>; }      // Variable Formulartyp
                   [subject is
                   <Formfield>;]
<MaxTime>     ::=  maxtime                // max. Ausfuehrungszeit
                   (<Number> (minutes |   // Nummer oder Feld
                   hours | days) | <Formfield>);
<Formfield>   ::=  <Name>.<Name>          // Formular- + Feldname
<ProcBody>    ::=  begin                  // Prozeszkoerper
                   <Statseq>              // Sequenz von Statements
                   end
<Statseq>     ::=  {<Statement> ;}+       // Statements mit ; abgeschlossen

<Statement>   ::=  [nonvital]
                   (<Alternatives>        // Wenn-Dann-Sonst-Konstrukt
                   | <Loops>              // Schleifen
                   | <Par_Choice>         // Parallelitaet
                   | <Call>               // Aufrufe
                   | <Assignment> )       // Zuweisungen
<Alternative> ::=  if <Expr> then         // if
                   <Statseq>
                   {elsif <Expr> then     // elsif (optional)
                   Statseq}
                   [else <Statseq>]       // else (optional)
                   end
```

```
<Loops>       ::=  <WhileLoop>             // Abweisende Schleife
                   | <UntilLoop>           // Nicht-Abweisende Schleife
                   | <ForeachLoop>         // For-Schleife ueber Formfeld
<WhileLoop>   ::=  while <Expr> do
                   <Statseq>
                   end
<UntilLoop>   ::=  repeat
                   <Statseq>
                   until <Expr>
<ForeachLoop> ::=  foreach <Name> in
                   <Formfield> do
                   <Statseq>
                   end
<Par_Choice>  ::=  ( andpar | orpar |      // und/oder-Parallelitaet
                   ranked choice | cost
                   choice index is         // Statements mit | getrennt
                   <Number>) <Statseq> {| <Statseq>}
                   end
<Call>        ::=  <StartTask> | <StartProcess> // Task oder Prozesz starten
<Assignment>  ::=  ( <Formfield>
                   | <Name> in <Formfield> [where <Expr>] )
                   :=
                   ( Value | Aggfunction ( <Name> )
                     in <Formfield> [where <Expr>] )
<Aggfunction> ::=  avg                     // Durchschnitt
                   | max                   // Maximum
                   | min                   // Minimum
                   | sum                   // Summe
<Expr>        ::=  <Value> (== | # | < | <= | > | >=) <Value>
<Value>       ::=  <Formfield>             // ein Feld in einem Formular
                   | <Number>              // eine Nummer
                   | <String>              // oder ein String
                   | null                  // NULL-Wert
                   | <Name>:user           // User eines Tasks
<StartTask>   ::=  <Userdesc>  <Name>
                   ([<Name>{,<Name>}])
<Userdesc>    ::=  <Formfield>             // ein Feld mit Benutzer oder Rolle
                   | <Name>[:user]         // Benutzer, Rolle oder Task:user
                   | <Formfield>           // Formfield mit Benutzer oder Rollen
<StartProcess>::=  [fork] <Url> <Name>
                  (<Name>{ ,<Name>})
<TaskDef>     ::=  <TaskHead> <TaskDecl> ; // Definition eines Tasks
<TaskHead>    ::=  task <Name> (         // Kopf des Tasks
                   <Arglist> )
<TaskDecl>    ::=  [descr <String>]        // Deklarationsteil des Tasks
                   [postcondition <String>]
                   [procedure <String>]
                   [stornotype (none
                        | compensateable <Name>
                        | undoable <Name>
                        | critical ) ]
                   [force]
                   [costfunction (<Number> | <String>)]
                   [formprops {<Formfield> <Properties> ; }+ end]
                   [<MaxTime>]
<Properties>  ::=  write | read | unvisible// Properties fuer Formfelder
<RoleDef>     ::=  role <Name>             // Rollendefinition
                   contains <Name>
                   {,<Name>};
```

```
<OrgDef>      ::=  department <Name>       // Aufbauorganisation
                   head <Name>
                   contains <Name> {, <Name>}  ;
<RecDef>      ::=  on receive (<Name>    // bei Erhalt von Formularen Prozesz starten
                   <Name> { , <Name>  <Name>})
                   start <Name>(<Name>{,Name>}) ;
<Arglist>     ::=  [ Name Name {, Name Name } ]
<String>      ::=   <beliebige Zeichen>
<Name>        ::=  <Zeichenkette>
<Url>         ::=  <gueltige URL>
```

A.2 Metaschemata in OMT-Notation

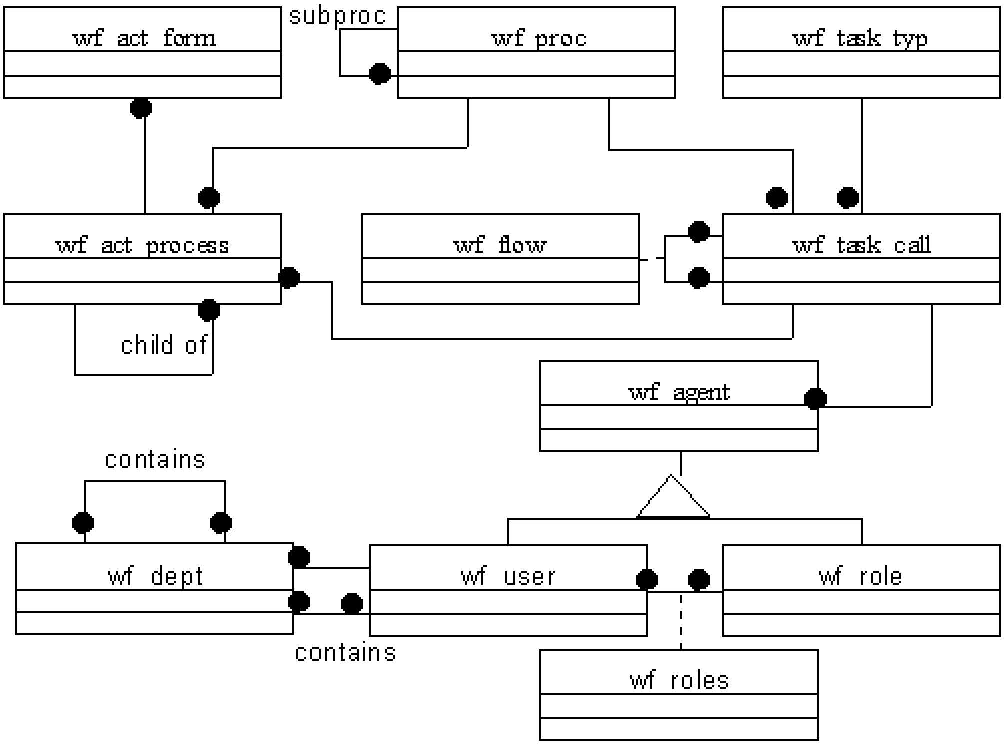

Abbildung A.1: Ausschnitt aus dem Metaschema Panta Rhei

A.3 Wichtige Datendefinitionen

```
create table wf_time_dep( -- workflow time dependency
  source_proc   varchar2(80) not null,-- links to proc of task which creates dep
  source_task   number,       -- refers to dep creating task_call
  dep_proc      varchar2(80), -- link to proc of dependent task_call
  dep_task      number,       -- refers to dependent task_call
  dep_type      number,       -- 0 EA, 1 EE, 2 AE, 3 AA
  dep_rel       number,       -- 0 equal, 1 min, 2 max, 3 date
  intervall     number,       -- duration between events
  source_date   date,         -- used only for fixed date dependency
  reaction      number,       -- references the reaction
  primary key   (source_task,dep_proc, dep_task, dep_type, dep_rel),
```

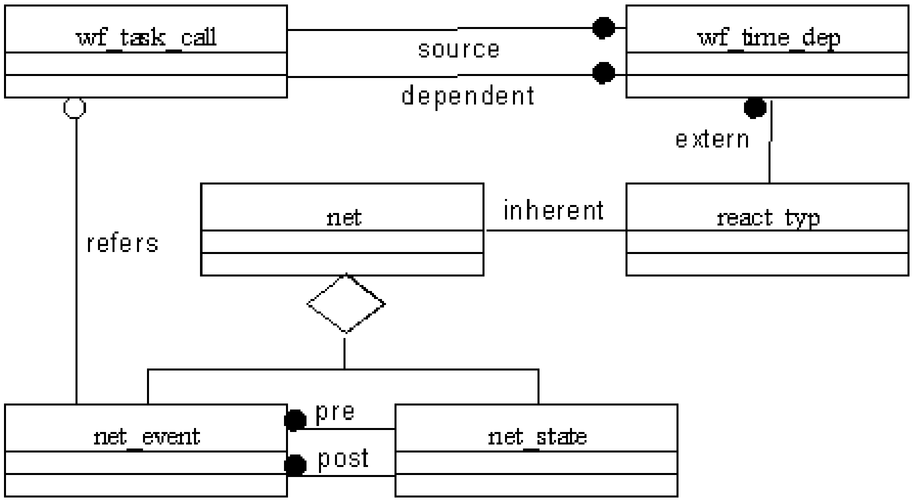

Abbildung A.2: Metaschema für Zeitabhängigkeit

```
  foreign key   (source_task) REFERENCES wf_task_calls(tid));

--net_state: explicit start or endstate of an event
create table net_state(
  sid     number  not null,
  pid     varchar2(80), -- refers to wf_proc
  e_bc    number, --expected value of the earliest best case
  e_bcVar number, --variance of the earliest best case
  l_bc    number, --expected value of the latest best case
  l_bcVar number, --variance of the latest best case
  e_wc    number, --expected value of the earliest worst case
  e_wcVar number, --variance of the earliest worst case
  l_wc    number, --expected value of the latest worst case
  l_wcVar number, --variance of the latest worst case
  isEndOf char(3),-- the type of the begin of the currenct structure
  isStartOf char(3), -- start of control structure, allowed values: INI SEQ PAR ALT END
  input number,   -- number of incoming tasks
  output number,  -- number of outgoing tasks
  foreign key(pid) REFERENCES wf_proc(pid),
  primary key(sid,pid));

-- net_event: represents activity of wf_task as a netstructure
create table net_event(
  eid       number not null, -- event id
  tid       number,       -- task id - refers to wf_task_call
  mu      number,       -- expectet value of duration time
  var     number,       -- variance of duration time
  pre     number,       -- refers to pre-state of the event
  post    number,       -- refers to post-state of the event
  proc    varchar2(80), -- refers to the act wf_process
  foreign key(pre, proc) REFERENCES net_state(sid,pid),
  foreign key(post, proc) REFERENCES net_state(sid,pid),
  primary key(eid,proc));
```

Anhang B

Quellcode

B.1 Erzeugen der Netzstruktur

```
/*
  file: compnet.sql
  date: 21. 6. 1996
  author: Heinz Pozewaunig
  function: generating an e-pert from a wdl metaschema
*/
-- forward declarations
CREATE OR REPLACE PACKAGE net IS -- visible interface objects
  procedure genNet(v_proc IN varchar2);
END;
/

CREATE OR REPLACE PACKAGE BODY net IS
---------------------------newNetState--------------------------------
-- purpose:  generating an initialized Node of epert-state
FUNCTION newNetState(proc IN varchar2)
  RETURN number AS
  num number;
  MAXNUM CONSTANT number := 0;
BEGIN
  SELECT state_id_seq.nextval INTO num FROM dual; -- generatin unique num
  -- first state
  INSERT INTO net_state (sid, pid, e_bc, e_bcVar,
    l_bc, l_bcVar, e_wc, e_wcVar,
    l_wc, l_wcVar, input, output)
    VALUES (num, proc, 0, 0, MAXNUM, 0, 0, 0, MAXNUM, 0, 0, 0);
  RETURN num;
END; --newNetState

---------------------------newNetEvent--------------------------------
-- purpose:  generating an activity with current values
-- and connecting it with its predecessor state
-- param: v_proc..current process; v_tid..task id;
-- v_predState..predecessor state of tid
-- v_endState..successor state of tid
FUNCTION newNetEvent(v_proc IN varchar2, v_tid IN number, --current task id
  v_predState IN number, --net pre state
  v_endState IN number)  --net post state

  RETURN number AS
```

```
  v_num number; --new event id
  v_mu number:=0; --expected value
  v_var number:=0; --variance of duration of the event
  v_a number; --optimitical value
  v_b number; --pessimistical value
  v_m number; -- mean
BEGIN

  -- Calculating the mean and the variance duration of the current task
  IF v_tid = 0 THEN --dummy task
    v_a:=0; v_b:=0; v_m :=0;
  ELSE
    SELECT a, b, m INTO v_a, v_b, v_m FROM aux_task_typ
      WHERE ttypid IN (
    SELECT ttypid FROM wf_task_calls
      WHERE tid=v_tid);
  END IF;

  -- under assumption of beta-distribution the mean value and variance are calculated as follows
  v_mu := (v_a+4*v_m+v_b)/6;
  v_var := ((v_b-v_a)/6)**2;

  SELECT event_id_seq.nextval INTO v_num FROM dual; -- generating unique num

  -- INSERT the correct values INTO net_event
  INSERT INTO net_event(eid, tid, mu, var,pre, post, proc)
    values(v_num, v_tid, v_mu, v_var,v_predState, v_endState, v_proc);

  -- actualizing the number of inputs and outputs of the states
  UPDATE net_state SET output=output+1
    WHERE pid=v_proc AND sid=v_predState;

  UPDATE net_state SET input=input+1
    WHERE pid=v_proc AND sid=v_endState;

  RETURN v_num;
END; -----newNetEvent--------------

---------------------------erasOldStruct--------------------------------
-- purpose: Erasing old net-structures of v_proc
PROCEDURE erasOldStruct(v_proc IN varchar2) AS
  v_count number; -- counter of rows
  v_min number; -- dummy id

BEGIN
  SELECT count(*) INTO v_count FROM net_event
    WHERE proc = v_proc;
  IF v_count > 0 THEN
    DELETE FROM net_event
      WHERE proc = v_proc;
  END IF;
  SELECT count(*) INTO v_count FROM net_state
    WHERE pid=v_proc;
  IF v_count > 0 THEN
    DELETE FROM net_state
      WHERE pid=v_proc;
  END IF;
  -- generating a dummy activity for modeling explicit state transitions
  DELETE FROM wf_task_calls WHERE pid=v_proc and ttypid='DUMMY';
```

```
  SELECT MIN(tid) INTO v_min FROM wf_task_calls; -- generating new dummy-id
  INSERT INTO wf_task_calls (pid,tid,ttypid)
    VALUES(v_proc,v_min-1,'DUMMY');
  COMMIT;
END;  -- erasOldStruct

---------------------------accumNodes--------------------------------
-- purpose: generating a set of all states of the ePERT-net
PROCEDURE accumNodes (v_proc IN varchar2) AS
  CURSOR states IS
    SELECT sid, input FROM net_state
      WHERE pid=v_proc;
BEGIN -- erasing old structures
  DELETE FROM nodeSet WHERE pid=v_proc;
  FOR curState IN states LOOP
    INSERT INTO nodeSet (sid, pid, yetWorkedInput,yetWorkedOutput)
      VALUES (curState.sid, v_proc, 0, 0);
  END LOOP;
END; -- accumNodes

---------------------------genStruct--------------------------------
-- purpose: transforming wdl-metaschema to epert-structure
PROCEDURE genStruct(v_proc IN varchar2) AS

  v_pred number; -- predecessor task
  v_branch number; -- branch type of the current task

  n_preS number;   -- id of new net start state
  n_auxPreS number;   -- helps to switch to a new prestate, if a dummy activity must be inserted
  n_postS number;     -- id of new net END state
  n_sOutput number; -- number of output flows of a state
  n_sInput number; -- counter for incoming flows
  v_isEndOf varchar2(80); -- determines the type of the control structure
  n_isEndOf char(3);
  v_sid number;   -- stackvariable for shifting control structures

  n_eId number;     -- id of new net event
  n_isStartOf net_state.isStartOf%TYPE; --marks the type of control structure
  v_wfFlow wf_flow%ROWTYPE; --record of one flow
  numElab  number; -- counter for yet alaborated tasks

  CURSOR taskFlows IS --all flows of the workflow process
    SELECT * FROM wf_flow WHERE pid=v_proc ORDER BY pred;

  illegal EXCEPTION;  -- general exception for handling errors

BEGIN
  --erasing old net-structure of current process, if exists
  erasOldStruct(v_proc);
  -- generating a new initialised state
  -- searching for the initial task
  SELECT DISTINCT pred, branch INTO v_pred, v_branch FROM wf_start_task
    WHERE pid=v_proc;

  n_preS := newNetState(v_proc); --start state
  n_postS := newNetState(v_proc); --END state of activity

  -- completing to s<->E<->s.
  n_eId :=  newNetEvent(v_proc, v_pred, n_preS, n_postS);
```

```
--------*******MAIN LOOP ********--------------------------------------------------
FOR v_wfFlow IN taskFlows LOOP  --main cursor loop over all wf_flows

  n_isStartOf:=null; -- initalicing after every loop

  --searching for the post-state of pre-task, which is the pre-state of current task
  -- this data is very often used in this loop
  SELECT post INTO n_preS FROM net_event
    WHERE  proc=v_proc AND tid = v_wfFlow.pred;

  -- we have to to check if pre-task is the beginning of a ramification,
  -- because the state connectors input and output have been updated.
  SELECT output INTO n_sOutput FROM net_state --number or output flows
    WHERE pid=v_proc AND sid = n_preS; -- of the start state of the current task

  IF n_sOutput>0 THEN -- beginning of a branch
    IF (v_wfFlow.branch = 1) OR (v_wfFlow.branch=2) THEN
      n_isStartOf := 'ALT'; -- previous task is start of an
      -- alternative structure
    ELSE
      n_isStartOf := 'PAR';
    END IF;
  ELSE
    n_isStartOf := 'SEQ';
  END IF;

  UPDATE net_state SET isStartOf=n_isStartOf
    WHERE sid=n_preS;

  -- now we have to determine, if task is END of a control structure
  --detecting the current branch-type
  SELECT count(*) INTO n_sInput FROM wf_flow
    WHERE pid=v_proc AND succ=v_wfFlow.succ; -- it is end of a branching

  IF n_sInput > 1 THEN --current task is end of a ramification
    -- and not yet elaborated
    SELECT count(tid) INTO numElab FROM net_event
      WHERE proc=v_proc AND tid=v_wfFlow.succ;

    IF numElab=0 THEN  -- task not yet elaborated

      -- generate new postState for the dummy activity
      n_postS:= newNetState(v_proc);

      -- inserting dummy activities for synchronizing all branches
      n_eId :=  newNetEvent(v_proc, 0, n_preS, n_postS);

      --now we must insert the current task into the net
      n_auxPreS:=n_postS;
      n_postS := newNetState(v_proc); --end state of activity

      --generating Event and connecting it with it's predecessor state n_preS
      n_eId :=  newNetEvent(v_proc, v_wfFlow.succ, n_auxPreS, n_postS);

      -- determing type of the current end of ramification
      SELECT ttypid INTO v_isEndOf FROM wf_task_calls
        WHERE tid=v_wfFlow.succ; -- determines endtype
```

```
        IF v_isEndOf='[nop]' THEN -- itf's an endif
          n_isEndOf := 'ALT';
        ELSIF v_isEndOf='[andjoin]' THEN
          n_isEndOf := 'PAR';
        ELSIF v_isEndOf='[orjoin]' THEN
          n_isEndOf := 'RCH';  -- ranked choice
        ELSE
          RAISE illegal;
        END IF;

        UPDATE net_state SET isEndOf = n_isEndOf
          WHERE sid=n_auxPreS; -- where is the start of this end state

      ELSE -- task already inserted into the net
        IF v_wfFlow.pred>v_wfFlow.succ THEN -- it is a LOOP
          dbms_output.put('ATTENTION: ILLEGAL LOOP CONSTRUCT');
          RAISE illegal;
        ELSE
          --what is the post state of the current flow
          SELECT pre INTO n_postS FROM net_event
            WHERE proc=v_proc AND tid = v_wfFlow.succ;

          -- inserting dummy activities for synchronizing all branches
          n_eId :=  newNetEvent(v_proc, 0, n_preS, n_postS);
        END IF; --LOOP
      END IF;

    ELSE  -- currently read control structure not end of a branch
      -- or this fact has not yet been detected
      n_postS := newNetState(v_proc); --end state of activity

      -- generating Event and connecting it with it's predecessor state n_preS
      n_eId :=  newNetEvent(v_proc, v_wfFlow.succ, n_preS, n_postS);
      END IF;
    COMMIT;
  END LOOP;

EXCEPTION
  WHEN OTHERS THEN
    IF taskFlows%isopen THEN
      close taskFlows;
    END IF;
    ROLLBACK;
END; -- genStruct

---------------------------calcEVal--------------------------------
-- purpose: calculating the values of the earliest occurence of an
-- net state for the best and worst case, returning the last worked state

FUNCTION calcEVal (v_proc IN varchar2, curState IN number) RETURN number AS
  v_mu number; --mean value of earlist best case
  v_var number; --variance of the earliest best case
  cur_e_bc number; -- current early value in the best case
  cur_e_bcVar number;
  cur_e_wc number;
  cur_e_wcVar number;

  -- successor states of the already updated state
  CURSOR postStates (p_id IN varchar2, st_id IN number) IS
```

```
    SELECT sid, e_bc, e_bcVar, e_wc, e_wcVar, isEndOf, input FROM net_state
      WHERE  pid = p_id AND sid IN (
        SELECT post FROM net_event
          WHERE pre=st_id);

  currWorkedInput number; -- already worked number of inputs for the poststate
  curOutput number; -- number of output flows of the current state
  res   number; -- the result of the function (the last worked state)
BEGIN
  -- the current values
  SELECT e_bc, e_bcVar, e_wc, e_wcVar, output
    INTO cur_e_bc, cur_e_bcVar, cur_e_wc, cur_e_wcVar, curOutput
    FROM net_state WHERE pid = v_proc AND sid = curState;

  IF curOutput=0 THEN -- the last state of the net
    UPDATE net_state SET -- the earliest results of the last state are the goals of the workflow
      l_bc= e_bc, l_bcVar=0, --! Variance set to zero !
      l_wc= e_wc, l_wcVar=0
      WHERE pid=v_proc AND sid=curState;

    res := curState;
  ELSE

    --calculating values of all successor states
    FOR curPost IN postStates (v_proc,curState) LOOP

      SELECT mu, var INTO v_mu, v_var FROM net_event
        WHERE proc=v_proc AND pre = curState AND post = curPost.sid;

      -- handling the worst case values
      IF cur_e_wc + v_mu > curPost.e_wc THEN -- new maximum worst case
        UPDATE net_state SET
          e_wc= cur_e_wc+v_mu,
          e_wcVar=cur_e_wcVar+v_var
          WHERE pid=v_proc AND sid=curPost.sid;
      END IF;

      -- handling the best case values
      IF curPost.isEndOf = 'PAR'  OR curPost.isEndOf = 'RCH' THEN -- end of parallel structure

        IF cur_e_bc + v_mu > curPost.e_bc THEN -- new maximum in the best case
          UPDATE net_state SET
            e_bc= cur_e_bc+v_mu,
            e_bcVar=cur_e_bcVar+v_var
            WHERE pid=v_proc AND sid=curPost.sid;
        END IF;
      ELSIF curPost.isEndOf ='ALT' THEN
        IF (curPost.e_bc=0) OR (cur_e_bc + v_mu < curPost.e_bc) THEN
          -- new minimum found
          UPDATE net_state SET
          e_bc= cur_e_bc+v_mu,
          e_bcVar=cur_e_bcVar+v_var
          WHERE pid=v_proc AND sid=curPost.sid;
        END IF;
      ELSE -- a sequence
        UPDATE net_state SET
          e_bc= cur_e_bc+v_mu,
          e_bcVar=cur_e_bcVar+v_var
          WHERE pid=v_proc AND sid=curPost.sid;
```

```
      END IF; --determining endtype of the structure

      --indicating, that the current state was yet worked out
      UPDATE nodeSet SET
        yetWorkedInput = yetWorkedInput + 1
        WHERE pid=v_proc AND sid=curPost.sid;

      SELECT yetWorkedInput INTO currWorkedInput FROM nodeSet
        WHERE pid=v_proc AND sid=curPost.sid;

      IF currWorkedInput = curPost.input THEN -- are all inputs worked out
        -- now we can step deeper into the net
        res := calcEVal(v_proc, curPost.sid);
      END IF;

    END LOOP;
  END IF; --if it's the last state of the net
  RETURN res; -- return worked state
END; --calcEVal

---------------------------calcLVal--------------------------------
-- purpose: calculating the values of the latest occurence of an
-- net state for the best and worst case
PROCEDURE calcLVal (v_proc IN varchar2, curState IN number) AS
  v_mu number; --mean value of earlist best case
  v_var number; --variance of the earliest best case
  cur_l_bc number; -- current early value in the best case
  cur_l_bcVar number;
  cur_l_wc number;
  cur_l_wcVar number;

  -- successor states of the already updated state
  CURSOR preStates (p_id IN varchar2, st_id IN number) IS
    SELECT sid, e_bc,l_bc, l_bcVar, l_wc, l_wcVar, isStartOf, output FROM net_state
      WHERE  pid = p_id AND sid IN (
        SELECT pre FROM net_event
          WHERE post=st_id);

  currWorkedOutput number; -- already worked number of inputs for the poststate
BEGIN
  -- the current values
  SELECT l_bc, l_bcVar, l_wc, l_wcVar
    INTO cur_l_bc, cur_l_bcVar, cur_l_wc, cur_l_wcVar FROM net_state
    WHERE pid = v_proc AND sid = curState;

  --calculating values of all successor states
  FOR curPre IN preStates (v_proc,curState) LOOP

    SELECT mu, var INTO v_mu, v_var FROM net_event
      WHERE proc=v_proc AND pre = curPre.sid AND post = curState;

    -- determining the start type of the structure
    IF curPre.isStartOf = 'PAR' THEN -- start of parallel structure

      IF curPre.l_bc <= 0 OR cur_l_bc - v_mu < curPre.l_bc THEN
        -- new minimum in the best case
        UPDATE net_state SET
          l_bc= cur_l_bc-v_mu,
          l_bcVar=cur_l_bcVar+v_var
```

```
        WHERE pid=v_proc AND sid=curPre.sid;
    END IF;
    IF curPre.l_wc = 0 OR cur_l_wc - v_mu < curPre.l_wc THEN
      -- new minimum in the worst case
      UPDATE net_state SET
        l_wc= cur_l_wc-v_mu,
        l_wcVar=cur_l_wcVar+v_var
        WHERE pid=v_proc AND sid=curPre.sid;
    END IF;
  ELSIF curPre.isStartOf ='ALT' THEN

    IF (curPre.l_bc <= 0) OR (cur_l_bc - v_mu < curPre.l_bc) THEN
      -- new minimum in the best case
      UPDATE net_state SET
        l_bc= cur_l_bc-v_mu,
        l_bcVar=cur_l_bcVar+v_var
        WHERE pid=v_proc AND sid=curPre.sid;
    END IF;

    IF cur_l_wc - v_mu > curPre.l_wc THEN
      -- new maximum in the worst case
      UPDATE net_state SET
        l_wc= cur_l_wc-v_mu,
        l_wcVar=cur_l_wcVar+v_var
        WHERE pid=v_proc AND sid=curPre.sid;
    END IF;

  ELSE -- a sequence
    IF cur_l_bc - v_mu < curPre.e_bc THEN -- coordination required
      UPDATE net_state SET
        l_bc = curPre.e_bc, -- set to the early value !
        l_bcVar = cur_l_bcVar+v_var -- variance treated as usual !
        WHERE pid=v_proc AND sid=curPre.sid;
    ELSE
      UPDATE net_state SET
        l_bc= cur_l_bc - v_mu,
        l_bcVar=cur_l_bcVar+v_var
        WHERE pid=v_proc AND sid=curPre.sid;

    END IF;
    -- no distinction necessary to manipulate the worst case
    UPDATE net_state SET
      l_wc= cur_l_wc-v_mu,
      l_wcVar=cur_l_wcVar+v_var
      WHERE pid=v_proc AND sid=curPre.sid;

  END IF; --determining start type of the structure

  --indicating, that the current state was yet worked out
  UPDATE nodeSet SET
    yetWorkedOutput = yetWorkedOutput + 1
    WHERE pid=v_proc AND sid=curPre.sid;

  SELECT yetWorkedOutput INTO currWorkedOutput FROM nodeSet
    WHERE pid=v_proc AND sid=curPre.sid;

  IF currWorkedOutput = curPre.output THEN -- are all outputs worked
    DELETE FROM nodeSet
      WHERE pid=v_proc AND sid=curPre.sid;
```

```
      -- now we can take one step deeper into the net
      calcLVal(v_proc, curPre.sid);
    END IF;

  END LOOP;
END; --calcLVal

------------------genNet------------------------
-- start point of construction an epert from wdl
PROCEDURE genNet (v_proc IN varchar2) AS
  state number; -- the start or end  state of the net
  nodeSet varchar2(80); -- the set of all nodes

BEGIN
  -- generating the static epert-net structure from the workflow description
  genStruct(v_proc);

  -- searching for the initial state under the restriction that a net has only one
  -- initial state
  SELECT sid INTO state FROM net_state
    WHERE pid=v_proc AND input=0;

  -- generate a set of all nodes of the net, the set ist stored in the table 'nodeSet'
  accumNodes(v_proc);

  -- calculating all earliest latest values of the epert net
  calcLVal(v_proc, calcEVal(v_proc, state));

  -- cheching time integrity constraints of the epert, eventually errors are reported to the table 'mess'
  tc(v_proc); -- Time Check of the current net-structure of the procedure v_proc
  COMMIT;
EXCEPTION
  WHEN others THEN
    ROLLBACK;
END; -- genNet
END; -- of package body
/
```

B.2 Unverträglichkeit zwischen inhärenten und externen Regeln

```
-- check Time constraint integrity
-- purpose: checks if a time restriction is generally possible
CREATE OR REPLACE PROCEDURE tc(v_proc IN varchar2) AS
  s_state   number;     -- the source of dependency
  s_bc  number;     -- the earliest best case start
  s_wc  number;     -- the latest worst case start
  d_state   number;     -- the aim of dependency
  d_bc  number;     -- the earliest best case start
  d_wc  number;     -- the latest worst case start
  illegal EXCEPTION;
  error BOOLEAN;        -- indicates an injured integrity constraint
  CURSOR orthDep IS -- orthogonal time dependencies
    SELECT source_task, dep_proc, dep_task, dep_type, dep_rel, intervall
      FROM wf_time_dep
      WHERE source_proc=v_proc
```

```
    ORDER BY source_task;

BEGIN
  DELETE FROM mess
    WHERE proc=v_proc; -- erasing old messages
  FOR curDep IN orthDep LOOP
    error := FALSE;
    -- we have to determine the type relation and therefor the
    -- correct states for the source and dependent state
    IF curDep.dep_type = 0 THEN -- EA
      SELECT post INTO s_state FROM net_event
        WHERE tid=curDep.source_task AND proc=v_proc;
      SELECT pre INTO d_state FROM net_event
        WHERE tid=curDep.dep_task AND   proc=v_proc;
    ELSIF curDep.dep_type = 1 THEN --EE
      SELECT post INTO s_state FROM net_event
        WHERE tid=curDep.source_task AND    proc=v_proc;
      SELECT post INTO d_state FROM net_event
        WHERE tid=curDep.dep_task AND   proc=v_proc;
    ELSIF curDep.dep_type = 2 THEN --AE
      SELECT pre INTO s_state FROM net_event
        WHERE tid=curDep.source_task AND proc=v_proc;
      SELECT post INTO d_state FROM net_event
        WHERE tid=curDep.dep_task AND   proc=v_proc;
    ELSIF curDep.dep_type = 3 THEN --AA
      SELECT pre INTO s_state FROM net_event
        WHERE tid=curDep.source_task AND    proc=v_proc;
      SELECT pre INTO d_state FROM net_event
        WHERE tid=curDep.dep_task AND   proc=v_proc;
    ELSE -- not defined dependency type
      RAISE illegal;
    END IF;
    -- getting the fitting time data
    SELECT e_bc, l_wc INTO s_bc, s_wc FROM net_state
      WHERE pid= v_proc AND sid = s_state;
    SELECT e_bc, l_wc INTO d_bc, d_wc FROM net_state
      WHERE pid= v_proc AND sid = d_state;
    -- handling the diverse relation types
    IF curDep.dep_rel = 0 THEN -- equality
      IF s_wc + curDep.intervall < d_bc OR
        s_bc + curDep.intervall > d_wc THEN
        error := TRUE;
      END IF;
    ELSIF curDep.dep_rel = 1 THEN -- minimum distance between states
      IF s_bc + curDep.intervall > d_wc THEN
      error := TRUE;
      END IF;
    ELSIF curDep.dep_rel = 2 THEN -- maximum distance between states
      IF s_wc + curDep.intervall < d_bc THEN
        error := TRUE;
      END IF;
    ELSE -- not defined relation type
      RAISE illegal;
    END IF;
    IF error THEN
      INSERT into mess (proc , mess) values(v_proc, ' DEPENDENCY ' ||
        curDep.source_task || ' TO ' || curDep.dep_task ||
      ' ILLEGAL DUE TO INCONSISTENT TIME CONSTRAINT');
    END IF;
```

```
  END LOOP;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    IF orthDep%isopen THEN
        CLOSE orthDep;
    END IF;
  ROLLBACK;
COMMIT;
END; -- tc time check
/
```

Literaturverzeichnis

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced Transaction Models in Workflow Contexts. In 12^{th} *International Conference on Data Engineering*, New Orleans, Lousiana, Feb 1996.

[AL+94] Martin Ader, Gang Lu, et al. Woorks, an Object Oriented Workflow System for Offices. Technical report, Bull S. A., Paris; Technics en Automatitzacio d'Officines S. A. Barcelona; Universita' di Milano; Communication and Management Systems Unit, 1994.

[ASSR93] Paul C. Attie, Munindar P. Singh, Amit Sheth, and Marek Rusinkiewicz. Specifying and Enforcing Intertask Dependencies. In Agrawal, Baker, and Bell, editors, *Proceedings of 19th International Converence on Very Large Databes*, 1993.

[ASW96] M. Amberg, R. Striemer, and M. Weske. Modellierung von Workflows: Ein Rahmenmodell. Beitrag zum GI Arbeitskreis Workflow, März 1996.

[BB94] A. Bestavros and S. Braoudakis. Timeliness via Speculation for Real-Time Databases. In *Real-Time Systems Symposium* [IEE94], pages 36–45.

[BLC95] T. Berners-Lee and D. Connolly. Hypertext markup language 2.0. URL http://www.w3.org/hyptertext/WWW/MarkUp/html-spec/html-spec_toc.html, June 1995. Internet Draft.

[BQRT95] Franz Burger, Gerald Quirchmayr, Siegfried Reich, and A Min Tjoa. Using HyTime for Modeling Publishing Workflows. *ACM SIGOIS Bulletin, ACM Press, New York NY*, 16(1):39–45, August 1995.

[BS95a] Markus Böhm and Wolfgang Schulze. Grundlagen von Workflow-Managementsystemen. *Wissenschaftliche Beiträge zur Informatik, Technische Universität Dresden*, 8(2):50–65, 1995.

[BS95b] Leonard Bolc and Andrzej Szalas, editors. *Time & Logic - a computational approach.* UCL Press, London, 1995.

[CC88] J. Cartier and P. Chretienne. Timed Petri net schedules. In Rozenberg [Roz88], pages 62–84.

[Cha92] Sharma Chakravarthy. Data Engineering Bulletin, Special Issue on Active Databases, December 1992.

[Cor94] J. C. Corbett. Modeling and Analysis of Real-Time Ada Tasking Programms. In *Real-Time Systems Symposium* [IEE94], pages 132–141.

[CR92] P. K. Chrysanthis and K. Ramamritham. Acta: The Saga Continues. In Elmagarmid [Elm92], chapter 10, pages 350–397.

[CR94] P. K. Chrysanthis and K. Ramamritham. Synthesis of Extended Transaction Models using Acta. In *ACM Transactions on Database Systems*, volume 19 of *3*, SEP 1994.

[Dat90] C. J. Date. *An Introduction to Database Systems*, volume 1. Addison-Wesley Publishing Company, Inc., Massachusetts, California, New York et. al., 5 edition, 1990.

[DD90] W. Domschke and A. Drexl. *Einführung in Operations Research.* Springer Verlag, Berlin–Heidelberg, 1990.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-Running Activities. In *Proceedings of the 16th International Conference on Very Large Data Bases*, August 1991.

[Dit93] Klaus R. Dittrich. Time Issues in Active Database Systems. In *International Workshop on an Infrastructure for Temporal Databases, Arlington Texas.* Institut für Informatik, Universität Zürich, JUN 1993.

[DJOR90] Christian Dorninger, Otto Janschek, Erlefried Olearczick, and Hans Röhrenbacher. *PPS Produktionsplanung und -steuerung - Konzepte, Methoden und Kritik.* Ueberreuter, Wien, 1990.

[DK88] Dietmar Dorninger and Günther Karigl. *Mathematik für Wirtschaftsinformatiker.* Springer, Wien, New York, 1988.

[DP88] Willibald Dörfler and Werner Peschek. *Einführung in die Mathematik für Informatiker.* Hanser, München, Wien, 1988.

[Dud88] Duden. *Informatik – Ein Sachlexikon für Studium und Praxis.* Mannheim, Wien, Zürich, 1 edition, 1988.

[EG96] Johann Eder and Herbert Groiss. Ein Workflow-Managementsystem auf der Basis aktiver Datenbanken. In Jörg Becker and Gottfried Vossen, editors, *Geschäftsprozeßmodellierung und Workflow-Management: Modelle, Methoden, Werkzeuge*, chapter 23. International Thomson Publ., Bonn, et. al., 1996.

[EGL96] Johann Eder, Herbert Groiss, and Walter Liebhart. Workflow-Systeme im WWW. In *ADV-Kongreß*, Wien, 1996.

[EL95a] Johann Eder and Walter Liebhart. A Transaction-Oriented Workflow Activity Model. Technical report, Institut für Informatik, Universität Klagenfurt, 1995.

[EL95b] Johann Eder and Walter Liebhart. The Workflow Activity Model WAMO. In *Proceedings of the 3^{rd} International Conference on Cooperative Information Systems*, 1995.

[EL96] Johann Eder and Walter Liebhart. Workflow recovery. In *First IFCIS International Conference on Cooperative Information Systems (CoopIS 96)*, Brussels, Belgium, Jun 1996. IEEE Computer Society Press.

[Elm92] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1992.

[EMSS93] E. Allen Emerson, A. Mok, A. Prasad Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. *Real Time Systesm Journal*, 2, Jan 1993.

[EV95] Jürgen Ebert and Gottfried Vossen. I-Serializability: Generalized Correctness for Transaction-Based Environments. Technical report, University of Koblenz, Germany; University of Münster, Germany, June 1995.

[EvF77] Horst A. Eiselt and Helmut von Frajer. *Operations Research Handbook - Standard Algorithms and Methods with Examples.* Walter de Gruyter, Berlin, 1977.

[Fü96] Hannes Fürpaß. Schnittstelle zwischen Bonapart 2.0 und Workflow 4.0. Master's thesis, Universität Klagenfurt, 1996.

[Gal93] R. Les Galloway. *Principles of operatons management.* Routledge, London, New York, 1993.

[Gal95] J. Galler. Metamodelle des Workflow-Managements. In Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Universität des Saarlandes, D-66123 Saarbrücken, Heft 21, Dec 1995.

[GE95] Herbert Groiss and Johann Eder. Interoperability with World Wide Workflows. In 1^{st} *World Conference on Integrated Design & Process Technology*, Austin TX, Dec 1995.

[GH94] Dimitrios Georgakopoulos and Mark F. Hornick. A framework for enforceable specification of extended transaction models and transactional workflows. *International Journal of Intelligent and Cooperative Information Systems*, Sep 1994.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.

[GKP92] Fred Glover, Darwin Klingman, and Nancy V. Phillips. *Network Models in Optimization and Their Applications in Practice.* John Wiley & Sons, Inc, New York, Chister, et. al., 1992.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing - Concepts and Techniques.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[GRL94] D. Georgakopoulos, M. Rusinkiewicz, and W. Litwin. Chronological Scheduling of Transactions with Temporal Dependencies. *The VLDB Journal*, 3(1), 1994.

[Han93] H. M. Hanisch. Analysis of Place/Transition with Timed Arcs and its Application to Batch Process Control. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets - 14^{th} International Conference*, pages 282–299, Berlin, Heidelberg, et. al., 1993. Springer.

[HR88] R. R. Howell and L. E. Rosier. On questions of fairness and temporal logic for conflict-free Petri-nets. In Rozenberg [Roz88], pages 200–226.

[HS95] M. Hammer and A. S. Stanton. *The Reengineering Revolution - A Handbook.* Harper Business, 1995.

[IEE93] IEEE. *Proceedings of the Real-Time Systems Symposium, Raleigh-Durham, North Carolina, Dec. 1–3*, Los Alamos CA, Washington, 1993. IEEE Computer Society Press.

[IEE94] IEEE. *Proceedings of the Real-Time Systems Symposium, San Juan, Puerto Rico, Dec. 7 – 9*, Los Alamos CA, Washington, 1994. IEEE Computer Society Press.

[Jab95] Stefan Jablonski. Workflow-management-systeme: Motivation, modellierung, architektur. *Informatik Spektrum, Springer-Verlag*, 18:13–24, 1995.

[JB86] Richard Johnson and Gouri Bhattacharyya. *Statistics – Principles and Methods.* John Wiley & Sons, New York, Chister, et. al., 1986.

[JZ96] Heinrich Jasper and Olaf Zukunft. Zeitaspekte bei der Modellierung und Ausführung von Workflows. Erweiterte Kurzfassung, FB Informatik, Abt. Informationssysteme, Universität Oldenburg, Escherweg 2, 26121 Oldenburg, 1996.

[KAGM96] M. Kamath, G. Alonso, R. Günthör, and C. Mohan. Providing High Availability in Very Large Workflow Management systems. In *The Fifth International Conference on Extending Database Technology (EDBT 96)*, Avignon, France, Mar 1996.

[Kle91] Johannes Klein. Advanced Rule Driven Transaction Management. In *Compcon 91 - Thirty-Sixth IEEE Computer Society International Conference*, Mar 1991.

[KR96] Mohan Kamath and Krithi Ramamritham. Bridging the gap between Transaction Management and Workflow Management. In *NSF Workshop on Workflow and Process Automation in Information Systems*. Department of Computer Science, University of Massachusetts, Amherst MA 01003, May 1996. URL http://wwww-ccs.cs.umass.edu/~kamath/nsf-wf.html.

[Lie95] Walter Liebhart. The Workflow Activity Description Language WADL. Technical report, Institut für Informatik, Universität Klagenfurt, 1995. Technischer Report für CoopIS95.

[Mar94] Ronni T. Marshak. Perspectives on workflow. In *New Tools for New Times - The Workflow Paradigm - The Impact of Information Technology on Business Process Reengineering*. Future Strategies Inc., Almeda, California, 1^{st} edition, 1994.

[Moh95] C. Mohan. Tutorial: Advanced Transaction Models - Survey and Critique. Presented at ACM SIGMOD International Conference on Management of Data, Minneapolis, May 1994, Jun 1995.

[MP91] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer Verlag, New York, 1991.

[NDS96] Anne H. H. Ngu, Toncan Duong, and Uma Srinivasan. Modeling Workflow using Tasks and Transactions. In Amith Sheth, editor, *National Science Foundation Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, Athens, Georgia, 1996.

[Nek95] Harald Nekvasil. Realisierung eines Workflow-Managementsystems auf der Basis aktiver Datenbanken. Master's thesis, Universität Klagenfurt, 1995.

[Phi86] Susy Philipose. *Operations Research - A Practical Approach*. Tata McGraw-Hill, New Delhi, New York, 1986.

[PHLR93] P.C.Clements, C. L. Heitmeyer, B. G. Labaw, and A. T. Rose. Mt: A Toolset for Specifying and Analyzing Real-Time Systems. In *Real-Time Systems Symposium* [IEE93], pages 12–22.

[RMSM+93] Y. S. Ramakrishna, P. M. Melliar-Smith, L. E. Moser, L. K. Dillon, and G. Kutty. Really Visual Temporal Reasoning. In *Real-Time Systems Symposium* [IEE93], pages 262–273.

[Roz88] Grzegorz Rozenberg, editor. *Lecture Notes in Computer Science - Advances in Petri Nets*, Berlin, Heidelberg, et. al., 1988. Springer Verlag.

[RS93] Marek Rusinkiewicz and Amit Sheth. Specification and Execution of Transactional Workflows. In *Proceedings of the FODO Conference June 1993*, 1993.

[RU71] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic*. Springer, Wien, New York, 1971.

[RzM96] Michael Rosemann and Michael zur Mühlen. Der Lösungsbeitrag von Metadatenmodellen beim Vergleich von Workflowmanagementsystemen. Arbeitsberichte des Instituts für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster, June 1996. Arbeitsbericht Nr. 48.

[Sch91] Ludger Schäfers. *Temporal Modeling in Relational Database Systems*. Verlag Dr. Kovač, Hamburg, 1991.

[Sin96] Munindar P. Singh. Formal Semantics for Workflow Computations. Technical report, Department of Computer Science, North Carolina State University, North Carolina, Raleigh, USA, Jan 1996. A previous version of this paper appeared in the Proceedings of the 5th International Workshop on Database Programming Languages (DBPL), held in Gubbio, Italy, in September 1995.

[SR69] S. I. Suchowizki and I. A. Radtschik. *Mathematische Methoden der Netzplantechnik.* B. G. Teubner, Leipzig, 2 edition, 1969.

[SST+96] Rajendran M. Sivasankaran, John A. Stankovic, Don Towsley, Bhaskar Purimetla, and Krithi Ramamritham. Priority assignment in real-time active databases. *The VLDB Journal*, 5:19–34, 1996.

[SV96] Munindar P. Singh and Mladen A. Vouk. Scientific Workflows: Scientific Computing Meets Transactional Workflows. URL http://lsdis.cs.uga.edu/activities/NSF-workflow/singh/, 1996.

[SW95] A. Prasad Sistla and Ouri Wolfson. Temporal Conditions and Integrity Constraints in Active Database Systems. In *SIGMOD '95, San Jose, CA, USA*, 1995.

[Sza95] Andrzej Szalas. Temporal Logic of programs: standard approach. In Bolc and Szalas [BS95b], chapter 1, pages 1–50.

[TCG+93] Abdullah Uz Tansel, James Clifford, Shashi Gadia, Sushil Jajodia, Arie Segev, and Richard Snodgrass. *Temporal Databases - Theory, Design and Implementation.* Benjamin/Cummings Publishing Company, Inc, Redwood City, Califonien et. al., 1993.

[Tha96] Horst Thaller. WDL - Eine Workflow-Beschreibungssprache. Technical report, Institut für Informatik, Universität Klagenfurt, May 1996. Draft zur Entwicklung der WDL, Teil der Diplomarbeit.

[VGH93] Gottfried Vossen and Margret Groß-Hardt. *Grundlagen der Transaktionsverarbeitung.* Addison-Wesley, Bonn, Paris, 1993.

[Vos90] Gottfried Vossen. *Transaktionsverarbeitung in Datenbanksystemen unter Ausnutzung semantischer Information.* Hüthig Buch Verlag, Heidelberg, 1990.

[WFM95] The Workflow Reference Model. The Workflow Management Coalition, July 1995. Author David Hollingsworth, Document Number TC00-1003 Draft 1.1.

[Wid92] Jennifer Widom. Active database rule systems. In *3rd International Conference on Extending Database Technology*, Vienna, 1992.

[Zim92] Werner Zimmermann. *Operations Research - Quantitative Methoden zur Entscheidungsvorbereitung.* R. Oldenbourg Verlag, München, Wien, 6. edition, 1992.