

Prüfer: Prof. Dr. rer. nat. Kurt Rothermel
Betreuer: Dipl.-Inform. Reiner Siebert
Beginn am: 1. September 1996
Beendet am: 21. März 1997
CR-Nummern: H.4, I.6.5, J.1

Diplomarbeit Nr. 1460

**Realisierung von Diensten
zur Anpassung von Workflows
während der Laufzeit**

Ralf Schröder

Fakultät Informatik
Institut für Parallele und Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Inhaltsverzeichnis

Abbildungsverzeichnis	6
1 Motivation	9
2 Einführung	13
2.1 Workflow-Management	13
2.1.1 Grundbegriffe	14
2.1.2 Phasen des Workflow-Managements	15
2.1.3 Workflow-Beschreibung	17
2.2 Anpassungsfähige Workflows	20
2.2.1 Motivation und Begriffe	20
2.2.2 Anpassung und Anpassungsdienst	22
2.2.3 Anforderungen an das WFMS	22
2.3 Umfeld der Arbeit	24
2.4 Notation der strukturierten Modellierung	27
2.4.1 Einleitendes	27
2.4.2 Strukturierungselemente	27
2.4.3 Konsistenzbedingungen	31
3 Spezifikation	33
3.1 Anforderungen an den Anpassungsdienst	33
3.2 Spezifikation von Einzelanpassungen	34
Anpassungen am verhaltensbezogenen Aspekt	35
3.2.1 Entfernen einer Arbeitseinheit	35
3.2.2 Einfügen einer Arbeitseinheit	38
3.2.3 Änderung von Regeln in bedingten Verzweigungen	43

3.2.4	Explizite Pfadauswahl in bedingten Verzweigungen	44
3.2.5	Aktivität/Subworkflow scheitern lassen	44
3.2.6	Aktivität/Subworkflow wiederholen	45
3.2.7	Arbeitseinheit vorziehen	45
3.2.8	Arbeitseinheit auslassen	46
3.2.9	Zusatzaktivität starten	46
	Anpassungen am operationalen Aspekt	47
3.2.10	Änderung von Rollen	47
3.2.11	Explizite Rollenauflösung	47
3.2.12	Aktivität abgeben	48
3.2.13	Termine ändern	48
	Anpassung am informationsbezogenen Aspekt	49
3.2.14	Änderung von Workflow-relevanten Daten	49
3.3	Gruppierung von Einzelanpassungen	49
4	Entwurf	51
4.1	Notation zur Entwurfsdarstellung	51
4.2	Architektur	53
	4.2.1 Prozeßarchitektur	53
	4.2.2 Subsysteme des Anpassungsdienstes	54
4.3	Das BOW-CORBA-Gateway	56
	4.3.1 Verzeichnisdienste	56
	4.3.2 Zugriff auf die Workflow-Beschreibung	57
	4.3.3 Zugriff auf Zustandsinformationen und Workflow-relevante Daten	58
	4.3.4 Verwendete Datentypen	59
4.4	Die Workflow-Instanz-Repräsentation	60
	4.4.1 Repräsentation der PDL-Ebene	60
	4.4.2 Repräsentation von Laufzeitinformationen	63
	4.4.3 Repräsentation der strukturierten Ebene	64
	4.4.4 Integration der einzelnen Bereiche	66
4.5	Der Anpassungs-Server	67
	4.5.1 Die objektorientierte Schnittstelle	67

4.5.2	Anpassungsmethoden	69
4.5.3	Hilfsmethoden	78
5	Implementierung	81
5.1	Entwicklungsumgebung	81
5.2	Bemerkungen zur Implementierung	83
5.2.1	Das BOW-CORBA-Gateway in C++	83
5.2.2	Der Anpassungsdienst in Java	84
6	Zusammenfassung und Ausblick	89
	Abkürzungsverzeichnis	93
	Glossar	95
	Literaturverzeichnis	97

Abbildungsverzeichnis

2.1	Beziehungen zwischen den Grundbegriffen [WfMC96]	15
2.2	Phasen des Workflow-Managements [WfMC96]	16
2.3	Der Workflow-Anpassungsdienst im Umfeld	23
2.4	Anpassungsfähigkeit für WFMS [Siebert96a]	24
2.5	Graphische Notation einer Arbeitseinheit	28
2.6	Graphische Notation einer Sequenz	29
2.7	Graphische Notation eines Parallelkonstrukts	29
2.8	Graphische Notation einer bedingten Verzweigung	30
3.1	Zusammenarbeit und resultierende Anforderungen	33
3.2	Strikt lokale Sicht: Entfernen einer Arbeitseinheit	35
3.3	Entfernen aus einer Sequenz	36
3.4	Entfernen aus einer minimalen Sequenz	37
3.5	Entfernen aus einem Parallelkonstrukt	37
3.6	Entfernen aus einem minimalen Parallelkonstrukt	38
3.7	Sequentielles Einfügen	39
3.8	Sequentielles Einfügen am Anfang einer Sequenz	40
3.9	Sequentielles Einfügen innerhalb einer Sequenz	40
3.10	Paralleles Einfügen in ein bestehendes Parallelkonstrukt	41
3.11	Paralleles Einfügen	42
3.12	Alternatives Einfügen	42
3.13	Einfügen in eine bedingte Verzweigung	43
4.1	Darstellungselemente in Klassendiagrammen	52
4.2	Prozeßarchitektur des Anpassungsdienstes und seiner Umgebung .	53
4.3	Die Subsysteme des Anpassungsdienstes	55

4.4	Klassen zur Repräsentation von PDL-Knoten	61
4.5	Klasse zur Repräsentation von PDL-Kanten	62
4.6	Klassen zur Repräsentation des Templates	62
4.7	Klassen zur Repräsentation von Laufzeitinformationen	63
4.8	Klassen zur Repräsentation der strukturierten Ebene	64
4.9	Klassen zur Repräsentation der Strukturierungselemente	65
4.10	Gesamtzusammenhang der Klassen zur Workflow-Instanz- Repräsentation	66
4.11	Die objektorientierte Schnittstelle des Anpassungsdienstes	67

Kapitel 1

Motivation

Workflow-Management

Workflow-Management ist die Steuerung und Kontrolle des Ablaufes von Geschäftsprozessen mit Hilfe von Informationstechnologie, also ein wichtiger Zweig der Büroautomation.

Zur Verwirklichung und Durchführung von Workflow-Management werden Workflow-Management-Systeme (WFMS) eingesetzt. Ein WFMS verwaltet Informationen über Vergangenheit (Historie), Gegenwart (derzeitiger Zustand des Systems) und Zukunft (Modellwissen) von Geschäftsprozessen. Durch Auswertung dieser Informationen und entsprechende Schlußfolgerungen leistet das WFMS die Ablaufsteuerung der Geschäftsprozesse.

WFMS finden in Unternehmen zunehmend Anwendung. Dabei wird in erster Linie eine Qualitätssteigerung bezüglich der Abwicklung von Geschäftsprozessen erwartet – insbesondere verringerte Durchlaufzeiten, bessere Informationsverfügbarkeit und Innovation in der Prozeßabwicklung.

Eine wichtige Anforderung für den Einsatz von WFMS ist die Integrierbarkeit der vorhandenen Anwendungen und Datenbestände in das WFMS. Eine gute Integration trägt wiederum indirekt zu einer höheren Qualität der Prozeßabläufe bei, da sie eine gemeinsame Basis für das Zusammenspiel verschiedener Anwendungen und Datenbestände schafft [Jablonski95].

Die Geschichte des Workflow-Managements zeigt eine starke Weiterentwicklung der Flexibilität der Systeme [Siebert96a]:

Zuerst wurde im Banken- und Versicherungsbereich begonnen, Geschäftsprozesse durch Informationstechnologie zu steuern. Die in diesem Bereich vorliegenden, von vornherein feststehenden und bekannten Prozeßabläufe wurden direkt in den Applikationscode abgebildet.

Einen großen Fortschritt stellt die Entkopplung von Prozeßmodell und Applikationscode dar. So werden Prozeßabläufe in Workflow-Beschreibungen abgebildet, mit deren Hilfe das WFMS die Prozeßabwicklung steuern kann. Dies ermöglicht es, zusätzlich neue Workflow-Beschreibungen zu erstellen, bzw. bereits existierende entsprechend neuer Gegebenheiten abzuändern. Hierdurch gewinnt diese Entwicklungsstufe gegenüber der vorherigen enorm an Flexibilität, da die Einführung neuer Geschäftsprozesse und die Änderung bereits existierender nicht mehr Änderungen am Applikationscode nötig machen.

Eine Weiterentwicklung der WFMS, an der derzeit geforscht wird und die Gegenstand dieser Arbeit ist, ist der Schritt zur Anpassungsfähigkeit von Workflows zur Laufzeit. Workflows sollen während ihres Ablaufs noch modifizierbar sein, um auch dann für eventuell notwendig werdende Änderungen an der weiteren Prozeßabwicklung größtmögliche Flexibilität anzubieten.

Themenstellung

Heutige WFMS sind relativ starr und unflexibel und eignen sich daher nicht zur Ausführung unstrukturierter oder sich ändernder Prozesse. Für ein unternehmensweites Prozeßmanagement mit WFMS bilden jedoch gerade die Flexibilität und Anpassungsfähigkeit wesentliche Anforderungen.

Durch die Bereitstellung von Änderungsdiensten sollen definierte Schnittstellen angeboten werden, um Workflows auch während der Laufzeit ändern zu können. Somit können die Prozesse auch nach dem Start noch verfeinert oder an geänderte Rahmenbedingungen angepaßt werden.

Im Rahmen der vorliegenden Diplomarbeit ist eine Workflow-Komponente zu realisieren, mit der verschiedene Änderungsdienste nach außen bereitgestellt werden. Damit können aktive Workflows von Administratoren oder Bearbeitern über definierte Schnittstellen geändert werden.

Nach einer Anforderungsanalyse, bei der auf vorangegangene Arbeiten zurückgegriffen werden kann, sind die zulässigen Änderungen und deren Schnittstellen zu spezifizieren. Basierend auf einer gegebenen Workflow-Beschreibungssprache und einer vorhanden Workflow-Engine ist die Komponente zu entwerfen, zu implementieren und zu testen. Abschließend sind alle Ergebnisse ausführlich zu dokumentieren.

Überblick

Um einen Überblick über die vorliegende Arbeit zu geben, werden an dieser Stelle zu jedem Kapitel einige kurze Angaben zu ihrem Inhalt gemacht.

Kapitel 2 führt in die für das Thema der Diplomarbeit grundlegenden Bereiche Workflow-Management und Anpassungsfähigkeit ein, stellt das Umfeld dar, gibt Begriffsdefinitionen und erläutert die graphische Notation der strukturierten Modellierung, die in den nachfolgenden Kapiteln verwendet wird.

Die Spezifikation des Anpassungsdienstes ist in Kapitel 3 dargestellt. Dort sind die angebotenen Anpassungen, ihre Auswirkungen und Schnittstellenanforderungen angegeben.

Kapitel 4 beinhaltet die Darstellung des Entwurfs der Arbeit. Hierzu gehören Prozeßarchitektur und Subsystemstruktur sowie die Beschreibung der entworfenen Klassen und der Methoden zur Realisierung der Anpassungen.

Informationen zur Implementierung sind in Kapitel 5 gegeben. Es sind dort die Entwicklungsumgebung und Systemanforderungen, sowie bei der Implementierung aufgetretene Probleme und Lösungsideen beschrieben.

Auf Zusammenfassung und Ausblick in Kapitel 6 folgen ein Abkürzungsverzeichnis, ein Glossar mit den wichtigsten Begriffen der Arbeit und das Literaturverzeichnis.

Kapitel 2

Einführung

Auf die Motivation für Workflow-Management im vorigen Kapitel folgend soll in diesem in für das Thema der Arbeit grundlegende Bereiche eingeführt werden.

Der erste Abschnitt definiert Grundbegriffe des Workflow-Managements und vermittelt dem Leser wichtige Zusammenhänge wie eine Phaseneinteilung des Workflow-Managements und die Betrachtung verschiedener Aspekte der Workflow-Beschreibung.

Hierauf basierend wird im zweiten Abschnitt an das Thema Anpassungsfähigkeit herangeführt und eine Motivation für die Notwendigkeit anpassungsfähiger Workflows für ein flexibles Workflow-Management gegeben. Die Vorstellung eines geeigneten Gesamtrahmens für einen Dienst zur Realisierung von Anpassungen beschließt diesen Abschnitt.

Das Thema des dritten Abschnittes ist die Darstellung des Umfeldes der vorliegenden Diplomarbeit. Das Projekt, in dessen Rahmen die Arbeit sich einfügt, wird vorgestellt und es wird ein Überblick über die Entwicklungsumgebung zur Realisierung des Anpassungsdienstes gegeben.

Abschließend ist noch eine Einführung in die strukturierte Modellierung erforderlich, da dieses Konzept der Kontrollflußdarstellung in den Kapiteln 3 und 4 verwendet wird und auf die Realisierung des Anpassungsdienstes einen großen Einfluß hatte.

2.1 Workflow-Management

Workflow-Management-Systeme werden von Unternehmen eingesetzt, um den Ablauf ihrer Geschäftsprozesse mit Hilfe von Informationstechnologie zu steuern, zu kontrollieren und zum Zweck der Entscheidungsfindung überblicken zu können.

Insbesondere wird vom Workflow-Management erwartet, daß es die Qualität der

Abwicklung von Geschäftsprozessen verbessert. Nach [Jablonski95] kann man fünf allgemeingültige Faktoren der Qualitätssteigerung nennen:

Effektivität: Geschäftsprozesse werden durch Workflows repräsentiert, die an variierende Gegebenheiten angepaßt werden können.

Effizienz: Verringerte Durchlaufzeiten von Prozessen, beispielsweise durch mögliche Parallelisierung von Prozeßschritten und zeitgerechte Übermittlung von Information zwischen deren Produzenten und Konsumenten, tragen zur Effizienzsteigerung bei.

Informationstransparenz: Information ist unabhängig vom Ort ihrer Erfassung und Speicherung unternehmensweit verfügbar.

Konsistenz: Die Daten und ihre Verarbeitung werden durch das WFMS kontrolliert. Somit ist eine Konsistenzüberwachung möglich.

Innovation: Workflow-Management erfordert die Überarbeitung der für ein Unternehmen wesentlichen Prozesse.

2.1.1 Grundbegriffe

Abbildung 2.1 zeigt die im folgenden beschriebenen Grundbegriffe des Workflow-Managements im Überblick und ihre Beziehungen untereinander.

Ein **Geschäftsprozeß**, auch Vorgang oder Büroablauf genannt, setzt sich aus mehreren oft voneinander abhängigen Teilaufgaben zusammen, die entsprechend kombiniert eine Gesamtaufgabe ergeben. Beispiele für Geschäftsprozesse sind „Bestellungsauftrag bearbeiten“ oder „Urlaubsantrag stellen“.

Um einen Geschäftsprozeß durch ein WFMS steuern und verwalten zu können, wird er modelliert, d.h. in eine (semi-)formale Darstellung gebracht, die dem WFMS notwendige Informationen darüber vermittelt, wie der Prozeß ablaufen soll. Diese Darstellung nennt man **Workflow-Beschreibung** oder auch Prozeßbeschreibung. Abschnitt 2.1.3 stellt diesen zentralen Begriff des Workflow-Managements ausführlicher dar.

Eine Workflow-Beschreibung beinhaltet mehrere **Aktivitäten**, die Pendant der oben erwähnten Teilaufgaben des Geschäftsprozesses. Komplexere Teilaufgaben, die ihrerseits mehrerer Aktivitäten zu ihrer Abarbeitung bedürfen, können selbst als eigenständige Workflows beschrieben und als **Subworkflows** in der übergeordneten Workflow-Beschreibung referenziert werden. Ein und derselbe Subworkflow kann somit in mehreren Workflow-Beschreibungen verwendet werden.

Das WFMS erzeugt zur Kontrolle und Steuerung für jeden tatsächlich ablaufenden Geschäftsprozeß aus dessen Workflow-Beschreibung eine **Workflow-Instanz**, die alle zur Repräsentation des laufenden Geschäftsprozesses nötigen

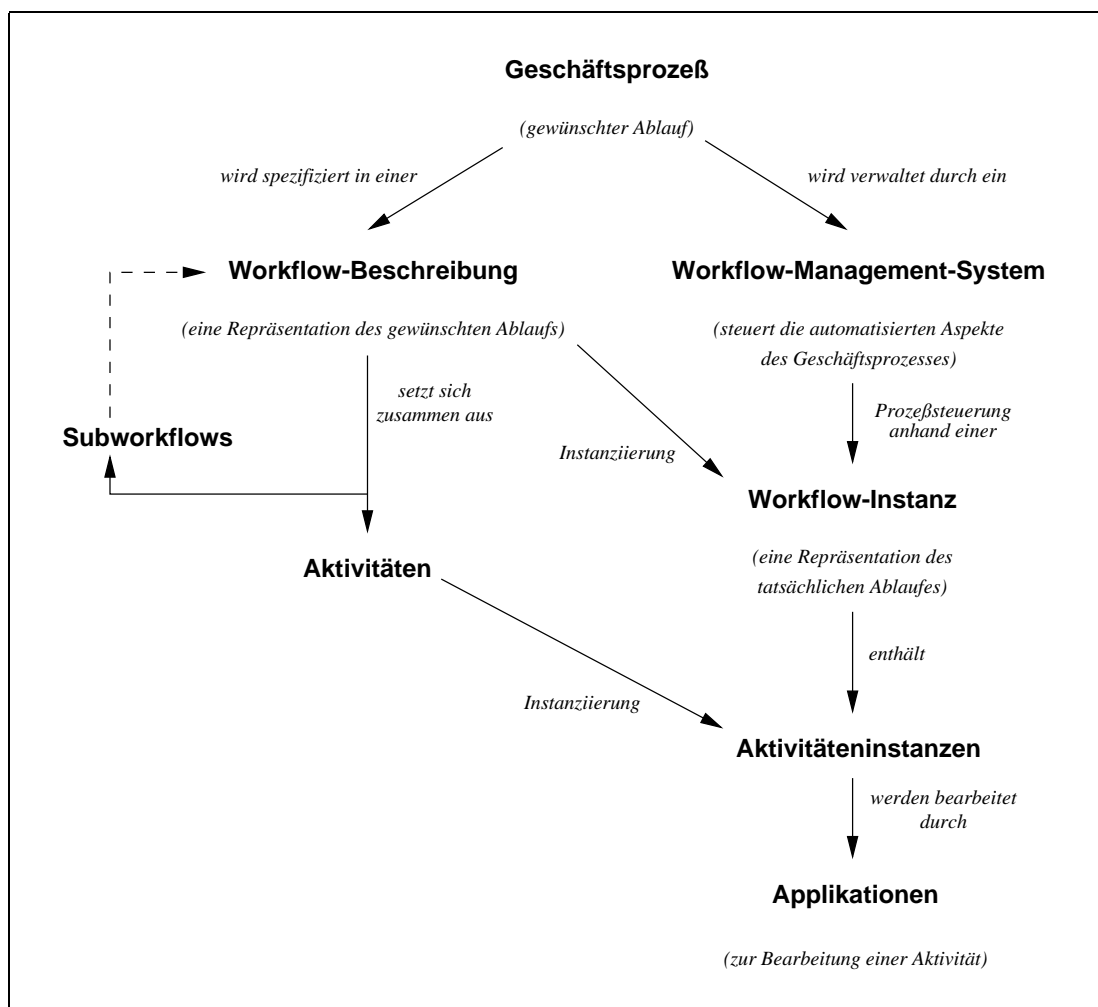


Abbildung 2.1: Beziehungen zwischen den Grundbegriffen [WfMC96]

Informationen enthält. Zu diesen Informationen gehört auch die Repräsentation der Aktivitäten des Prozesses, die in den **Aktivitäteninstanzen** gehalten wird.

Applikationen werden Aktivitäteninstanzen zugeordnet, um die Aktivität automatisch zu bearbeiten oder einen Bearbeiter dabei zu unterstützen.

2.1.2 Phasen des Workflow-Managements

Das Workflow-Management läßt sich in zwei Phasen unterteilen: Modellierungsphase und Laufzeitphase. Abbildung 2.2 stellt die Beziehungen zwischen den beiden Phasen und ihre Inhalte dar.

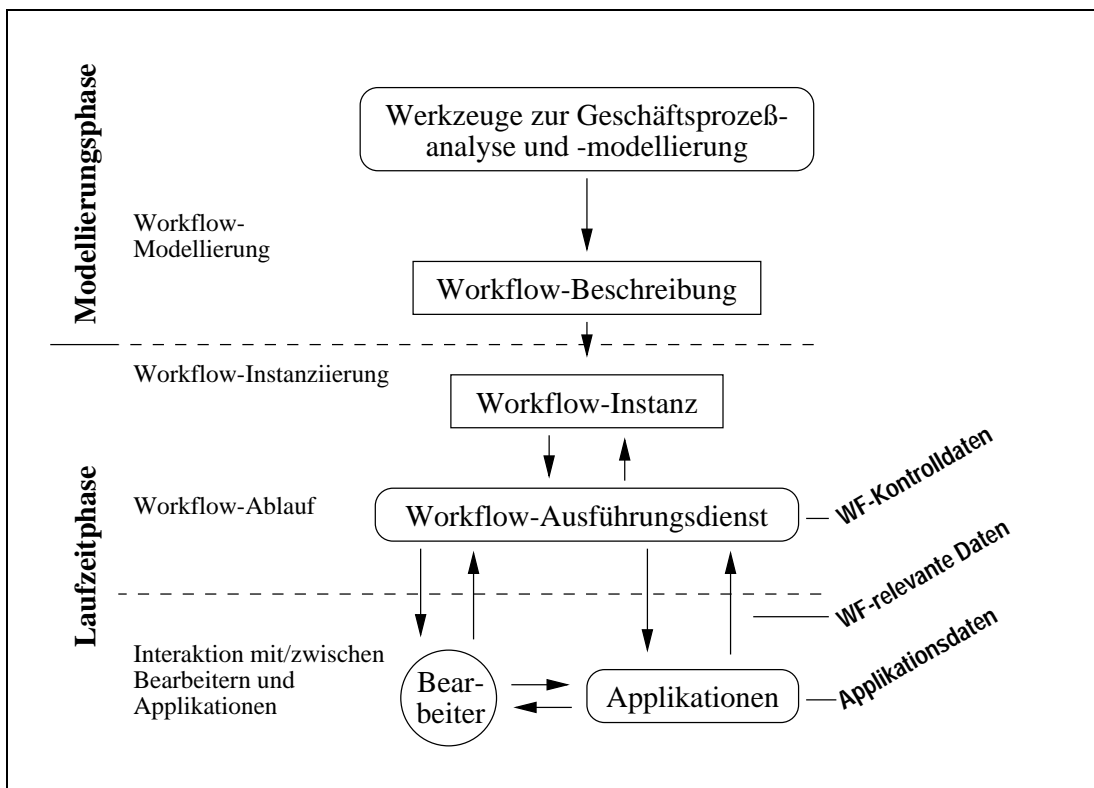


Abbildung 2.2: Phasen des Workflow-Managements [WfMC96]

Die Modellierungsphase

In der Modellierungsphase werden Geschäftsprozesse modelliert, d.h. es werden Spezifikationen derselben in Form von Workflow-Beschreibungen erstellt. Zur Unterstützung dieses Vorgangs stellt das WFMS Analyse- und Modellierwerkzeuge bereit.

Das Ergebnis der Modellierungsphase, die Workflow-Beschreibung, spezifiziert verschiedene Aspekte des Geschäftsprozesses und steht dem WFMS für die Ablaufsteuerung zur Auswertung und Information über diese Aspekte zur Verfügung. Somit ist die Workflow-Beschreibung das Bindeglied zwischen der Modellierungsphase und der Laufzeitphase.

Die Laufzeitphase

Der Inhalt der Laufzeitphase ist die eigentliche Vorgangsbearbeitung.

Zu diesem Zweck erzeugt das WFMS aus der Workflow-Beschreibung, die den gewünschten Prozeßtyp repräsentiert, zunächst eine Workflow-Instanz. Die Workflow-Instanz enthält die Workflow-Beschreibung sowie Workflow-bezogene Daten, die weiter unten beschrieben werden.

Das Verhältnis „Workflow-Beschreibung – Workflow-Instanz“ kann mit dem Verhältnis „Klasse – Objekt(instanz)“ aus der Objektorientierung verglichen werden. Die Workflow-Beschreibung spezifiziert die Eigenschaften aller Geschäftsprozesse eines bestimmten Typs bzw. einer bestimmten Klasse; die Workflow-Instanzen übernehmen diese Eigenschaften, halten aber noch weitere Informationen über den derzeitigen Zustand des dargestellten Vorgangs. In diesen Informationen unterscheiden sich die gleichzeitig existierenden Workflow-Instanzen ein und derselben Workflow-Beschreibung voneinander: sie repräsentieren verschiedene Vorgänge desselben Typs, beispielsweise „Bestellungsauftrag der Firma Meier bearbeiten“ und „Bestellungsauftrag der Firma Müller bearbeiten“.

Mit der **Workflow-Ausführung** (Synonym: Workflow-Ablauf) wird die Steuerung eines zu bearbeitenden Vorgangs durch das WFMS bezeichnet. Die WFMS-Komponente, die in erster Linie hierfür zuständig ist, heißt **Workflow-Ausführungsdienst** (Workflow-Enactment-Service). Sie verwaltet die Informationen der Workflow-Instanz und treibt den Ablauf entsprechend der Workflow-Beschreibung voran. Zur Bearbeitung der Aktivitäten des Workflows findet Interaktion zwischen dem Workflow-Ausführungsdienst auf der einen Seite und den Bearbeitern und Applikationen auf der anderen Seite statt. Genauer sind die Interaktionspartner des Workflow-Ausführungsdienstes im Fall vollständig automatisierter Aktivitäten nur Applikationen, im Fall halbautomatisierter (DV-unterstützter) Aktivitäten Bearbeiter und Applikationen, die ihrerseits interagieren, und im Fall manueller Aktivitäten nur die Bearbeiter.

Abbildung 2.2 bezeichnet am rechten Rand die unterschiedlichen Kategorien von Daten, die in der Vorgangsabwicklung eine Rolle spielen:

Applikationsdaten befinden sich in der Kontrollsphäre der jeweiligen Applikationen, sind dem WFMS nicht zugänglich und haben somit auch keinen Einfluß auf die Arbeit des Workflow-Ausführungsdienstes.

Zwischen Workflow-Ausführungsdienst und Applikationen ausgetauscht werden **Workflow-relevante Daten**, die von den Applikationen gelesen und/oder geschrieben werden und vom WFMS zur Weitergabe an andere Applikationen und zur Beeinflussung des Workflow-Ablaufs verwaltet werden.

In der Kontrollsphäre des Workflow-Ausführungsdienstes und den Applikationen nicht zugänglich sind die **Workflow-Kontrolldaten**, die zur Steuerung des Workflow-Ablaufs herangezogen werden. Unter die Workflow-Kontrolldaten fallen u.a. Zustandsinformationen über Workflow-Instanz und Aktivitäteninstanzen.

2.1.3 Workflow-Beschreibung

In [Jablonski95] wird die Spezifikation von Geschäftsprozessen von mehreren separaten Gesichtspunkten aus betrachtet und die Workflow-Beschreibung gemäß dieser Aspekte gegliedert. Ein Teil der Aspekte soll im folgenden herangezogen werden um darzustellen, welche Informationen eine Workflow-Beschreibung

enthält.

Funktionaler Aspekt

Der funktionale Aspekt spezifiziert, *was* ausgeführt werden soll. Er gibt an, aus welchen Subworkflows und Aktivitäten ein Workflow zusammengesetzt ist (siehe auch Abbildung 2.1 auf Seite 15).

Operationaler Aspekt

Die Aktivitäten eines Workflows stellen dessen elementare Verarbeitungseinheiten dar. Im operationalen Aspekt wird beschrieben *wie* diese elementaren Einheiten realisiert werden: Applikationen, die verwendet werden sollen, deren Ein- und Ausgabeparameter und eventuelle Einschränkungen.

Verhaltensbezogener Aspekt

Der verhaltensbezogene Aspekt beschreibt den Kontrollfluß zwischen den Subworkflows und Aktivitäten eines Workflows und somit *wann* Subworkflows bzw. Aktivitäten ausgeführt werden sollen.

Hierzu bedient man sich verschiedener Kontrollflußkonstrukte:

1. Grundlegende aus Programmiersprachen bereits bekannte präskriptive Konstrukte:
Sequenz, bedingte Verzweigung, unbedingte Verzweigung und Schleifen
2. Konstrukte zur kompakten Beschreibung komplexerer Kontrollflußbeziehungen. Diese sollten möglichst als Makros nach Bedarf aus vorgegebenen Basiskonstrukten und bereits definierten Makros konstruierbar sein.

Beispiele:

- Optionale Ausführung:
Aktivität bzw. Subworkflow ausführen oder überspringen.
- „m aus n“-Verzweigung:
Aus einer Menge von n Aktivitäten bzw. Subworkflows müssen genau m ausgeführt werden.
- Parallele Wiederholung:
Mehrere Instanzen einer Aktivität bzw. eines Subworkflows werden unabhängig voneinander (d.h. auch möglicherweise parallel) ausgeführt, bis eine Abbruchsbedingung eintritt.

- Reihe:
Eine Menge von Aktivitäten/Subworkflows werden in beliebiger Reihenfolge sequentiell ausgeführt.
3. Deskriptive Konstrukte zur Beschreibung komplexer Zusammenhänge ohne Formulierung von Bedingungen. Beispiele:
- Limitierung:
„A ist durch B limitiert“ bedeutet, daß A nicht mehr ausgeführt werden kann, sobald B seine Ausführung begonnen hat.
 - Verzögerung:
„A ist durch B verzögert“ bedeutet, daß A seine Ausführung erst starten kann, wenn B bearbeitet wurde oder nie ausgeführt werden wird.
 - Existenzabhängigkeit:
„A ist existentiell von B abhängig“ bedeutet, daß A nur ausgeführt werden kann, wenn B bereits bearbeitet wurde oder sicher in der Zukunft ausgeführt wird.

Informationsbezogener Aspekt

Der informationsbezogene Aspekt beschreibt Eigenschaften und Fluß Workflow-relevanter Daten.

Unter die beschriebenen Eigenschaften fallen unbedingt Datentypen. Dies können für das WFMS verarbeitbare Typen mit klarer Semantik sein, wie sie aus Programmiersprachen, Datenbanksystemen und Schnittstellenbeschreibungssprachen bekannt sind. Aber auch ein Typ für sogenannte nichttypisierte Daten¹, die vom WFMS nicht interpretiert werden sollen, sondern nur für den Datenfluß zwischen Aktivitäten/Subworkflows benötigt werden, sollte existieren. Darüberhinaus kann im Rahmen des informationsbezogenen Aspekts vom WFMS auch die Konstruktion neuer Datentypen ermöglicht werden.

Die Datenflußbeschreibung definiert, welche Daten zwischen Aktivitäten/Subworkflows fließen, und gibt somit Produzent und Konsument jedes Datums an. Auftretende Probleme beim Datenfluß sind Typ- und Interpretationsinkompatibilitäten. Deren Lösung durch Konvertierung wird ebenfalls in der Datenflußbeschreibung angegeben.

Organisatorischer Aspekt

Der organisatorische Aspekt beschreibt, *wer* zur Bearbeitung der Teilaufgaben eines Geschäftsprozesses zur Verfügung steht. Hier wird also die Aufbauorganisa-

¹In SQL beispielsweise steht hierzu der Typ Binary Large Object (BLOB) zur Verfügung.

tion definiert. Sie besteht aus der Unternehmensstruktur und der Unternehmenspopulation.

Die Unternehmensstruktur wird durch die Angabe existierender Rollen und der Beziehungen zwischen diesen Rollen beschrieben. Beispielsweise kann so eine streng hierarchische Organisation beschrieben werden.

Die Unternehmenspopulation ist sozusagen eine Aufzählung aller Mitarbeiter². Das Bindeglied zwischen Unternehmensstruktur und -population stellt die Angabe dar, welcher Mitarbeiter welche Rollen einnimmt.

Unter den organisatorischen Aspekt fällt auch die Beschreibung von Strategien zur Rollenauflösung. Wenn zur Bearbeitung einer Aktivität mehrere Repräsentanten der gewünschten Rolle zur Verfügung stehen, wird entsprechend einer solchen Strategie der tatsächliche Bearbeiter ermittelt.

2.2 Anpassungsfähige Workflows

2.2.1 Motivation und Begriffe

Im Kapitel 1 haben wir gesehen, daß bei der Weiterentwicklung der WFMS es ein wichtiges Ziel ist, deren Flexibilität zu vergrößern. Hierdurch werden auch die Anwendungsbereiche und Einsatzmöglichkeiten der Systeme ausgeweitet.

Flexibilität ist die Fähigkeit, für gegebene Anforderungen angemessene Lösungen anzubieten. Die Anforderung der letzten Entwicklungsstufe aus Kapitel 1, somit die Anforderung an diese Arbeit, ist, daß Workflows auch zur Laufzeit modifizierbar sein müssen. Anpassungsfähige Workflows erfüllen diese Forderung, die im folgenden begründet werden soll.

Bei der Betrachtung der Phasen des Workflow-Managements in Abschnitt 2.1.2 zeigt sich, daß die zur Workflow-Ausführung nötigen Informationen zu unterschiedlichen Zeitpunkten verfügbar sind und festgelegt werden können. Ein großer Teil dieser Informationen wird in der Modellierungsphase ermittelt und in der Workflow-Beschreibung niedergelegt. Bei der Instanziierung kommen weitere Informationen hinzu, um die Workflow-Instanz aufzubauen, und der Rest fällt zur Laufzeit an.

Ein zweiter Gesichtspunkt ist, *wie* die Informationen ermittelt werden können. Hier unterscheidet man die folgenden Fälle: 1. in der Workflow-Beschreibung festgelegt, 2. automatisch ermittelbar und 3. interaktiv zu ermitteln.

Die beiden Aspekte der Informationsfestlegung „wann“ und „wie“ bestimmen nach [Siebert96a] den Grad der Strukturiertheit eines Prozesses. Ein Prozeß ist

²Im Bereich Workflow-Management kann man unter „Mitarbeitern“ auch Applikationen zur vollautomatischen Bearbeitung von Aktivitäten verstehen.

strukturiert, wenn alle zur Ausführung notwendigen Informationen in der Modellierungsphase verfügbar sind oder zur Laufzeit noch fehlende Information wenigstens automatisch ermittelt werden kann. Je mehr Details eines Prozesses erst zur Laufzeit interaktiv, also manuell festgelegt werden können, desto unstrukturierter ist dieser Prozeß.

Nach dieser Klärung der Begriffe „strukturiert – unstrukturiert“ und den Überlegungen, wann und wie zur Workflow-Ausführung notwendige Information festgelegt werden kann, sollen mehrere Fälle beschrieben werden, die anpassungsfähige Workflows zur Realisierung erfordern.

Unstrukturierte Prozesse

Unstrukturierte Prozesse können nur mit Hilfe anpassungsfähiger Workflows in ein WFMS integriert werden. Die in der Modellierungsphase nicht festlegbaren Details werden vereinfacht modelliert, und dann zur Laufzeit an die Gegebenheiten angepaßt.

Verfeinerung der Modellierung zur Laufzeit

Eine vollständige Modellierung ist bei vielen Geschäftsprozessen theoretisch möglich, aber mit zuviel Aufwand verbunden. Ein ökonomischer Ansatz ist hier eine Grobmodellierung im sinnvollen Rahmen während der Modellierungsphase und zur Laufzeit die Verfeinerung der Workflow-Beschreibung entsprechend der Anforderungen des Prozeßverlaufs mit Hilfe von Anpassungen.³

Dieses Vorgehen kann maßgeblich zu einer Innovation der Prozeßabwicklung (siehe Abschnitt 2.1) beitragen, indem in mehreren Prozeßabläufen gefundene notwendige Verfeinerungen in die Workflow-Beschreibung eingearbeitet werden.

Unvorhersehbare Notwendigkeit zur Änderung

Auch in vermeintlich strukturierten Prozessen können zur Laufzeit Änderungen am Workflow-Ablauf notwendig werden. Ursachen hierfür können beispielsweise Gesetzesänderungen sein, oder das Fehlen von nötiger Information zur Weiterarbeit macht eine zusätzliche Aktivität „Rückfrage“ notwendig.

Ohne Anpassungsfähigkeit wird der Workflow im ungünstigsten Fall nicht mehr weiter abarbeitbar, d.h. der Qualitätsfaktor „Effektivität“ ist gefährdet. Im etwas günstigeren Fall können die notwendigen Veränderungen unabhängig vom Workflow-Ablauf bearbeitet werden. Dies vermindert allerdings den Qualitätsfaktor „Informationstransparenz“. Es findet ein Bruch

³Insbesondere hier sollten jedoch nicht alle Probleme durch Anpassungen gelöst werden. Die Bereitstellung möglichst flexibler Modellierungskonstrukte – beispielsweise deskriptiver Kontrollkonstrukte – reduziert ebenfalls den Modellierungsaufwand ohne die Übersichtlichkeit der Workflow-Beschreibung zu vermindern.

im Geschäftsprozeß statt: ein Teil wird innerhalb und der Rest außerhalb der Kontrollsphäre des WFMS bearbeitet. Die Anpassungsfähigkeit verhindert diesen Bruch, indem durchgeführte Anpassungen durch das WFMS kontrolliert und erfaßt werden und somit in der Historie des Workflows nachvollziehbar sind.

2.2.2 Anpassung und Anpassungsdienst

Unter Anpassung versteht man eine Änderung an einer Workflow-Instanz, d.h. ihrer Workflow-Beschreibung, ihren Workflow-Kontrolldaten oder ihren Workflow-relevanten Daten, die zur Laufzeit vorgenommen wird. Anpassung kann außer einer konkret durchgeführten auch eine ganze Klasse von Anpassungen bezeichnen, beispielsweise „Sequentielles Einfügen“.

Anpassungen werden auch Ad-hoc-Modifikationen genannt, was deutlich zum Ausdruck bringt, daß der Anlaß zur Durchführung einer Anpassung oft unvorhergesehen auftritt.

Zur Realisierung von Anpassungen wird das WFMS um einen Anpassungsdienst erweitert. Seinen Platz im System stellt Abbildung 2.3 dar (vgl. Abbildung 2.2 auf Seite 16).

2.2.3 Anforderungen an das WFMS

Die bei einer Anpassung durchzuführenden Änderungen sind nur möglich, wenn das WFMS lesenden und schreibenden Zugriff auf die Prozeßrepräsentation zur Laufzeit, d.h. auf die Workflow-Instanz, bereitstellt. Wie Abbildung 2.3 zeigt, betrifft dies die Workflow-Beschreibung der Instanz und die Instanz-bezogenen Daten (Workflow-relevante und Workflow-Kontrolldaten). Die Zugriffe müssen unter geeignetem Schutz erfolgen, ähnlich dem Transaktionsschutz von Datenbanksystemen. Gegenseitige Störungen der Zugriffe von Workflow-Ausführungsdienst und Workflow-Anpassungsdienst sind zu verhindern. Der Anpassungsdienst muß auch soweit ins System integriert sein, daß vorgenommene Anpassungen in der Historie des Workflows vermerkt werden.

Überdies sollte ein geeigneter Gesamtrahmen für eine einerseits sinnvoll kontrollierte und andererseits benutzerfreundliche Realisierung von Anpassungen gegeben sein. In [Siebert96a] wird hierzu eine konzeptionelle Architektur vorgestellt, die Abbildung 2.4 veranschaulicht.

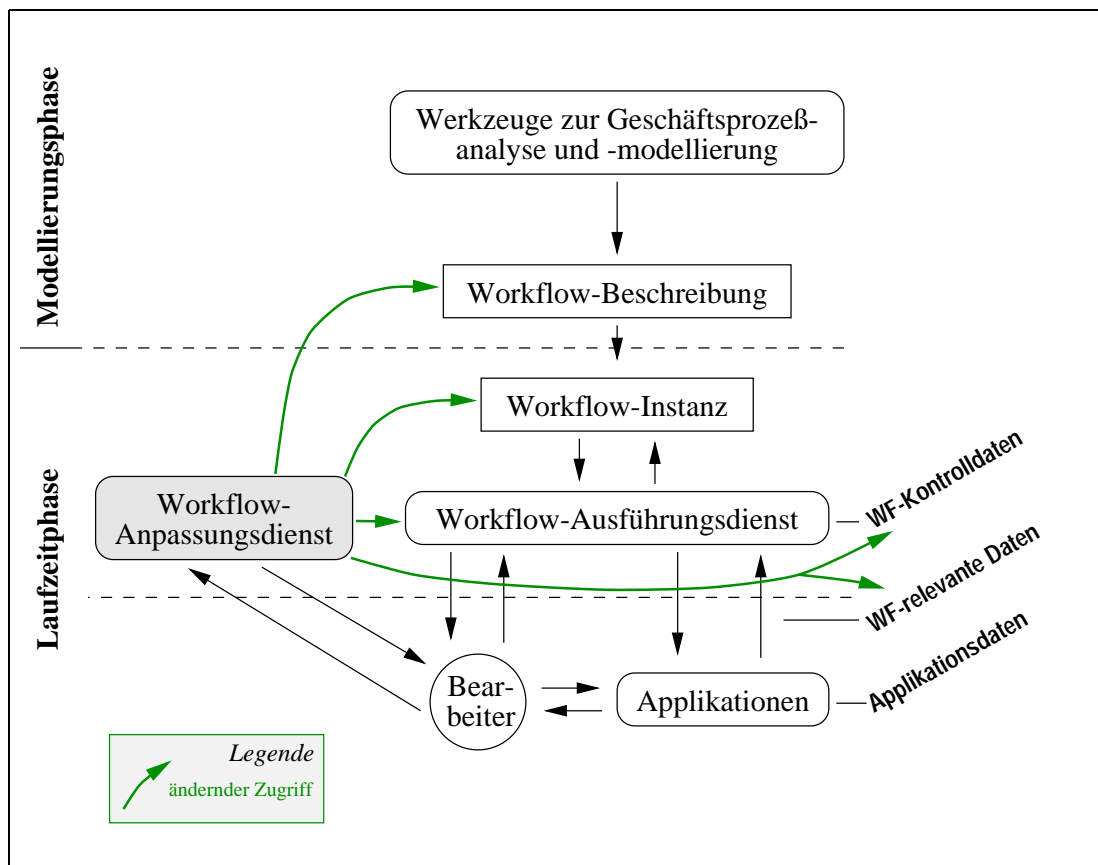


Abbildung 2.3: Der Workflow-Anpassungsdienst im Umfeld

Durch eine ausreichende Mächtigkeit der Workflow-Beschreibungssprache können viele Ausnahmefälle abgedeckt werden, ohne die Übersichtlichkeit der Workflow-Beschreibung zu beeinträchtigen oder den Modellierungsaufwand über Gebühr zu erhöhen. Hierdurch wird ein Teil der typischen Anpassungen zur Laufzeit nicht mehr anfallen.

Anpassungen müssen mehreren Kontrollen unterworfen sein:

Anpassungsrechte definieren wer welche Anpassungen unter welchen Umständen durchführen darf. Die Konsistenz, d.h. die Korrektheit bezüglich Workflow-Beschreibungsregeln, sowie die Integrität des Workflows, d.h. Bedingungen, die speziell für den betroffenen Workflow eingehalten werden müssen, dürfen durch die Anpassung nicht verletzt werden.

Schließlich muß mit Hilfe einer geeigneten Benutzerschnittstelle den Anwendern der Zugang zur Durchführung von Anpassungen in angemessener Weise ermöglicht werden. Hier sollte auf die unterschiedlichen Vorkenntnisse bezüglich Workflows eingegangen werden.

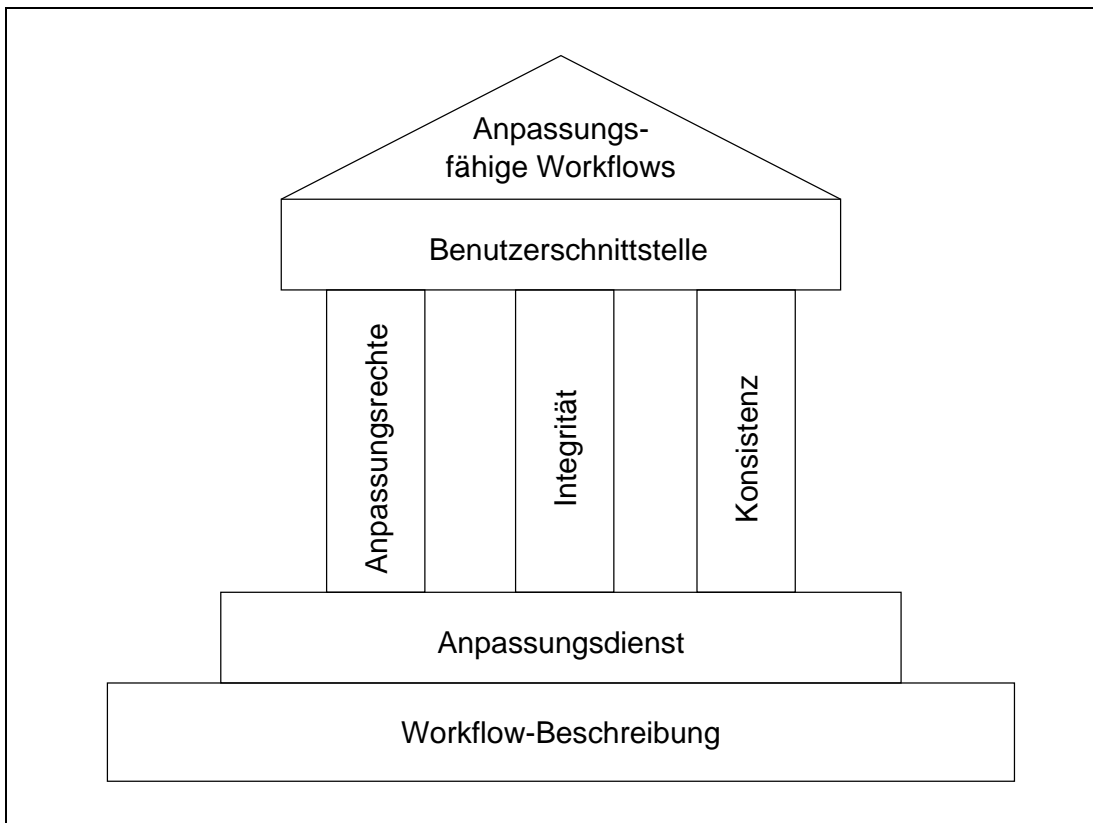


Abbildung 2.4: Anpassungsfähigkeit für WFMS [Siebert96a]

2.3 Umfeld der Arbeit

Diese Diplomarbeit entsteht im Rahmen von PoliFlow [Siebert96b], einem von vier vom Bundesministerium für Forschung und Technik geförderten Verbundprojekten der Förderinitiative POLIKOM. Diese Verbundprojekte erarbeiten grundlegende technische und wissenschaftliche Lösungen, um Kooperation und Koordination bei Vorgängen in der öffentlichen Verwaltung effizient unterstützen zu können. Der Hintergrund ist die Entscheidung zur Verteilung von Regierungsfunktionen auf Berlin und Bonn.

Im Projekt PoliFlow soll durch Telekooperation der Zwang zur räumlichen Konzentration abgebaut und die gemeinsame Bearbeitung verschiedener Aufgaben durch Menschen in räumlich verteilten Organisationen ermöglicht werden. Die Koordination der Teilaufgaben wird durch Workflow-Management-Systeme unterstützt.

Das PoliFlow-Konsortium setzt sich aus einem Hersteller (Hewlett-Packard GmbH), einem Anwender (Oberfinanzdirektion Berlin) und zwei Instituten der Universität Stuttgart (Institut für Arbeitswissenschaften und Technologiemanagement und Institut für Parallele und Verteilte Höchstleistungsrechner) zusammen.

men. Durch diese Kombination werden einerseits die Bedürfnisse des Anwenders von Beginn an berücksichtigt; andererseits wird durch die wissenschaftliche Begleitung sichergestellt, daß innovative Lösungsansätze erarbeitet und kontinuierlich in bestehende Systeme integriert werden.

Der Arbeitsbereich „Anpassungsfähige Workflow-Systeme“ am Institut für Parallele und Verteilte Höchstleistungsrechner stellt mit seiner im vorigen Abschnitt 2.2.3 dargestellten konzeptionellen Architektur den engeren Rahmen der vorliegenden Arbeit dar.

Die Studienarbeit „Modellierung von Rechten zur Anpassung von Workflows“ [Ott96] zeigt theoretische Grundlagen der Anpassungsrechte auf. Diese Arbeit diente der vorliegenden zur Orientierung bezüglich häufig auftretender Anpassungen (siehe Kapitel 3).

In einer laufenden Diplomarbeit wird derzeit an der „Realisierung eines Dienstes zur Kontrolle von Workflow-Anpassungen“ und somit an einer Workflow-Komponente zur Überprüfung von Anpassungsrechten gearbeitet.

Mit der Benutzerschnittstelle befaßte sich zeitgleich mit der vorliegenden Diplomarbeit die Diplomarbeit „Erstellung eines Workflow-Editors“ [Aldinger97]. Der Workflow-Editor bietet dem Anwender verschiedene angemessene Sichten auf die einzelnen Aspekte der Workflow-Beschreibung, um ihm einen intuitiven Zugang zur Durchführung von Anpassungen zu geben. Bei der Spezifikation der Anpassungen und der Schnittstellendefinition des Anpassungsdienstes wurden die Diplomarbeit „Erstellung eines Workflow-Editors“ und die vorliegende Arbeit aufeinander abgestimmt. Auch beim Entwurf der Klassen zur Repräsentation von Workflows waren eine enge Zusammenarbeit zur gegenseitigen Information über Anforderungen und gemeinsame Entscheidungen nötig, um die erfolgreiche Kooperation beider Komponenten zu gewährleisten.

Entwicklungsumgebung

Die Technologien, die im Projekt PoliFlow und somit auch in dieser Arbeit zur Realisierung eingesetzt werden, sind durch entsprechende Entscheidungen auf den verschiedenen Ebenen Verbundprojekt PoliFlow, Arbeitsgruppe am Institut für Parallele und Verteilte Höchstleistungsrechner bzw. Arbeitsbereich „Anpassungsfähige Workflow-Systeme“ bereits vorgegeben.

Zur Entwicklung des WFMS steht das noch in Entwicklung befindliche „Enterprise Process Management System“ (Codename „Montana“)⁴ [HP96a, HP96b] der Hewlett-Packard Corporation zur Verfügung. Das Herz dieses Systems stellt die Montana-Engine dar, die u.a. die Funktionalität zur Workflow-Beschreibung und Workflow-Ausführung über ein objektorientiertes API anbietet. Zur Anbindung diverser Infrastrukturkomponenten – beispielsweise auch des Anpassungsdien-

⁴Das Enterprise Process Management System „Montana“ ist urheberrechtlich geschützt. Das Urheberrecht hält die Hewlett Packard Corporation.

stes – existiert eine Entwicklungsumgebung für die Programmiersprache C++. Anfallende Kommunikation wird über einen CORBA-Transportdienst (Common Object Request Broker Architecture, [OMG95, YangDuddy95]) abgewickelt, wodurch die Arbeit in heterogenen Umgebungen möglich ist.

Montana stellt mit seiner Process Definition Language (PDL, [HP96a])⁵ eine Workflow-Beschreibungssprache bereit, die durch große Flexibilität und Vielfalt, aber auch durch eine Modellierungsweise auf einer Ebene ähnlich maschinen-naher Programmiersprachen gekennzeichnet ist. Die einzelnen Beschreibungsaspekte, insbesondere der verhaltensbezogene, werden in der Hauptsache mit Hilfe gerichteter Kanten, sehr mächtiger regelgesteuerter Knoten zur Verzweigung und Synchronisation oder Ereignisauslösung und -bearbeitung und Knoten für Aktivitäten bzw. Subworkflows spezifiziert. Mit den PDL-Sprachelementen sind selbst deskriptive Kontrollkonstrukte ausdrückbar, jedoch ist die ursprüngliche Absicht aus dem resultierenden PDL-Code nicht mehr ersichtlich. So ist es auch nicht vorgesehen, Workflow-Beschreibungen von Hand in PDL zu codieren, sondern zu diesem Zweck sollen Workflow-Editoren ins WFMS integriert werden, die dem Benutzer eine intuitive Repräsentation der Workflow-Beschreibung bieten und diese dann in PDL dem System mitteilen.

In der vorliegenden Arbeit wurde eine Kontrollfluß-Beschreibungsweise nach Art der strukturierten Modellierung gewählt. Hierdurch wird Übersichtlichkeit in Spezifikation, Entwurf und Realisierung erreicht, was auch für die in anderen Arbeiten hinzukommenden Konzepte der Konsistenz- und Integritätskontrollen von Vorteil sein dürfte. Die Notation der strukturierten Modellierung in dieser Arbeit wird im folgenden Abschnitt 2.4 vorgestellt.

Die PoliFlow-Arbeitsgruppe am Institut für Parallele und Verteilte Höchstleistungsrechner hat sich im Hinblick auf die heterogene Anwendungsumgebung dazu entschlossen, diverse Infrastrukturkomponenten in Java zu realisieren. Die Implementierung der vorliegenden Arbeit erfolgte dementsprechend bis auf eine C++-Komponente in Java.

Java [Flanagan96, CampioneWalrath96] von Sun Microsystems Inc. ist eine objektorientierte, plattformunabhängige Programmiersprache, die für den Anwendungsbereich heterogener verteilter Umgebungen konzipiert wurde. Die Kooperation mit in C++ implementierten Montana-Komponenten wird mittels der CORBA-Realisierungen für Java (OrbixWeb [IONA95]) und für C++ (HP ORB Plus [HPORB96]) ermöglicht.

⁵„PDL“ ist urheberrechtlich geschützt. Das Urheberrecht hält die Hewlett Packard Corporation.

2.4 Notation der strukturierten Modellierung

2.4.1 Einleitendes

Die strukturierte Modellierung⁶ überträgt die Art der Kontrollflußbeschreibung der strukturierten Programmierung in den Bereich der Workflow-Modellierung.

Die Grundidee ist die Zerlegung des Gesamtproblems in Teilprobleme, sowie die rekursive Anwendung der Zerlegung auf identifizierte Teilprobleme, bis man auf der Ebene der elementaren Schritte angelangt ist. Bei der Workflow-Modellierung entspricht das Gesamtproblem dem kompletten Workflow, die Teilprobleme verschiedenen Strukturierungselementen und die elementaren Schritte Basiselementen, die nicht weiter zerlegt werden können.

Diverse Vorteile, die für die strukturierte Programmierung sprechen [DudenInfo89], lassen sich auch auf die strukturierte Modellierung übertragen:

- Strukturierung fördert Übersichtlichkeit.
- Die Idee des Prozesses geht aus seiner Struktur hervor.
- Teillösungen können an anderer Stelle wiederverwendet werden.

2.4.2 Strukturierungselemente

Die strukturierte Modellierung bietet unterschiedliche Arten von Strukturierungselementen an, um Kontrollflußbeziehungen ausdrücken zu können. Für die prototypische Realisierung in dieser Arbeit wurden einige wichtige Elemente ausgewählt: die Arbeitseinheit als Basiselement und die zusammengesetzten Strukturierungselemente Sequenz, Parallelkonstrukt und bedingte Verzweigung. Schon in einer frühen Phase der Arbeit wurde entschieden, das Schleifenkonzept nicht zu betrachten.

Die Strukturierungselemente haben gewisse gemeinsame Eigenschaften:

- Jedes Strukturierungselement hat genau einen Eingang und einen Ausgang.
- Der vorigen statischen Eigenschaft entspricht die folgende dynamische: Wenn der Kontrollfluß ein Strukturierungselement **einmal** durch den Eingang betritt, dann verläßt er es auch genau **einmal** durch den Ausgang.

⁶Das Wort „strukturiert“ bei der strukturierten Modellierung wird unter einem anderen Aspekt verwendet als im Begriff „strukturierte Prozesse“! Der Unterschied geht aus dem Text hervor.

- Strukturierungselemente sind rekursiv aufgebaut: Die zusammengesetzten Strukturierungselemente enthalten wiederum Strukturierungselemente, deren Kontrollflußbeziehungen untereinander sie beschreiben. Das Ende dieser Rekursion stellen die Basiselemente der strukturierten Modellierung dar.

Die einzelnen Strukturierungselemente sollen im folgenden beschrieben und ihre für die vorliegende Arbeit erstellte graphische Notation vorgestellt werden.

Basiselement: Arbeitseinheit

Abbildung 2.5 stellt das Basiselement der strukturierten Modellierung, die Arbeitseinheit dar.

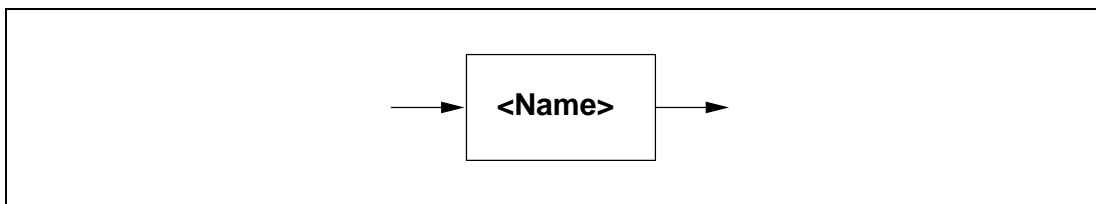


Abbildung 2.5: Graphische Notation einer Arbeitseinheit

In der Abbildung sind Ein- und Ausgang als Pfeile dargestellt. Die Pfeile illustrieren den Verlauf des Kontrollflusses zur Arbeitseinheit hin und nach deren Abarbeitung wieder von ihr weg zu einem nachfolgenden Element. Der Name der Arbeitseinheit wird innerhalb des Rechtecks angegeben.

Die Arbeitseinheit enthält einen Verweis auf eine Aktivität oder einen Subworkflow, der in der Notation nicht dargestellt zu werden braucht. Wenn der Kontrollfluß den Eingang der Arbeitseinheit erreicht, wird die Aktivität bzw. der Subworkflow durchgeführt und anschließend der Kontrollfluß über den Ausgang weitergeleitet. Die Arbeitseinheit entspricht somit der Anweisung (respektive dem Prozeduraufruf im Fall eines Subworkflows) in der strukturierten Programmierung.

Entsprechend der leeren Anweisung in der Programmierung – z.B. für den leeren ELSE-Zweig – sollte es auch eine leere Arbeitseinheit geben, um einen leeren Pfad in einer bedingten Verzweigung zu modellieren.

Sequenz

Das zusammengesetzte Strukturierungselement Sequenz ist eine Aneinanderreihung von Strukturierungselementen, die hintereinander ausgeführt werden sollen. Somit ist die Sequenz das Pendant des Anweisungsblocks aus der strukturierten Programmierung.

Abbildung 2.6 stellt die Sequenz dar.

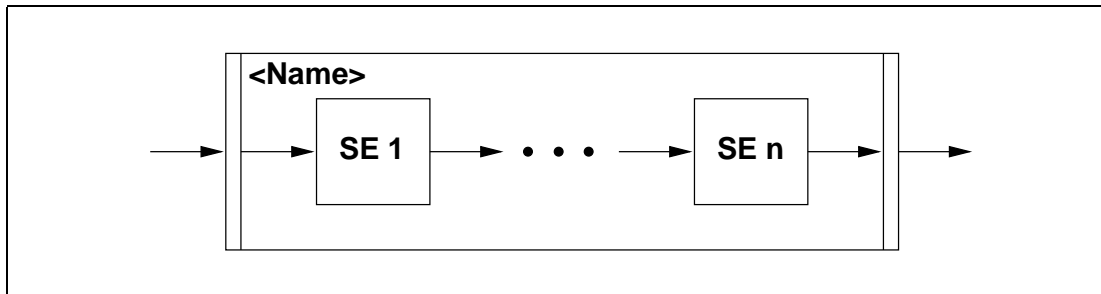


Abbildung 2.6: Graphische Notation einer Sequenz

Der Kontrollfluß der Sequenz verläuft folgendermaßen:

Wenn der Sequenzeingang erreicht ist, werden die in der Sequenz dargestellten Strukturierungselemente „SE 1“ bis „SE n“ in dieser Reihenfolge hintereinander ausgeführt. Bei Erreichen des Ausgangs von „SE n“ wird die Sequenz verlassen.

Parallelkonstrukt

Mit dem zusammengesetzten Strukturierungselement Parallelkonstrukt ist die Parallelausführung von mehreren Strukturierungselementen ausdrückbar. Abbildung 2.7 stellt das Parallelkonstrukt dar.

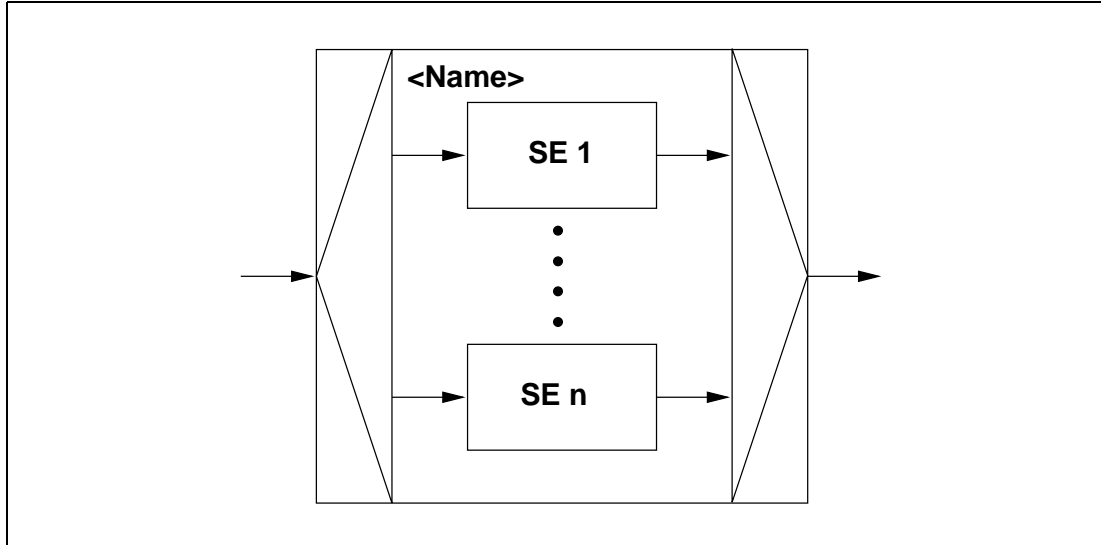


Abbildung 2.7: Graphische Notation eines Parallelkonstrukts

Der Kontrollfluß des Parallelkonstruktes:

Wenn der Eingang erreicht ist, wird der Kontrollfluß auf die enthaltenen Strukturierungselemente „SE 1“ bis „SE n“ aufgeteilt, wodurch diese nebenläufig ausgeführt werden. Wenn alle parallel auszuführenden Strukturierungselemente ab-

gearbeitet sind und somit den Kontrollfluß an ihren Ausgang weitergeleitet haben, kann das Parallelkonstrukt verlassen werden und seinerseits den Kontrollfluß – bildlich gesprochen – gebündelt zum Ausgang leiten.

Bedingte Verzweigung

Die bedingte Verzweigung hat sehr viel Ähnlichkeit mit dem gerade beschriebenen Parallelkonstrukt. Dies schlägt sich auch in der Darstellung nieder. Abbildung 2.8 zeigt die graphische Notation.

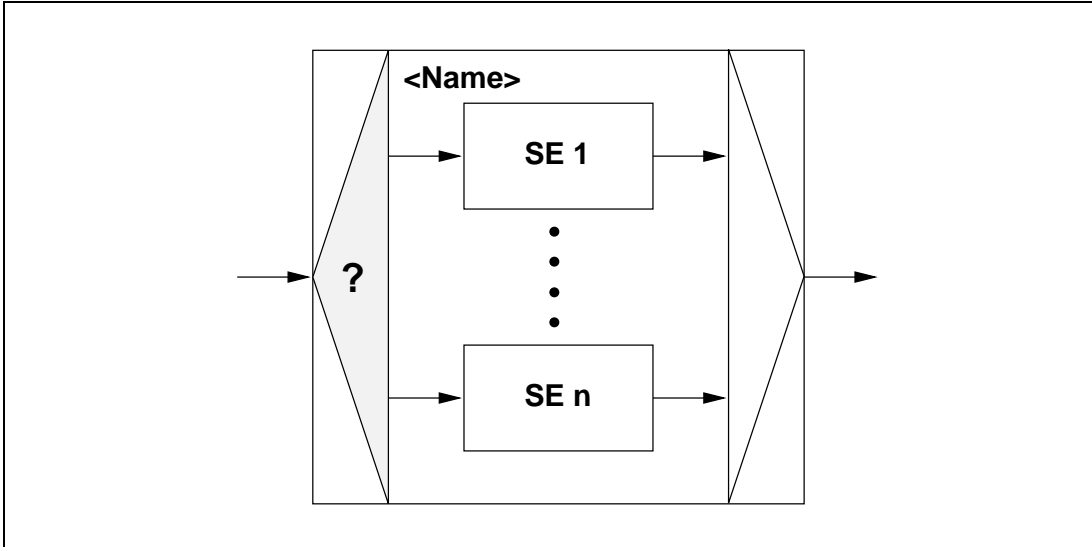


Abbildung 2.8: Graphische Notation einer bedingten Verzweigung

Wie aus der Abbildung sofort ersichtlich wird, handelt es sich nicht um eine feste „eins aus zwei“-Alternative, wie sie bei Programmiersprachen mit dem „IF-THEN-ELSE“-Konstrukt angeboten wird. Vielmehr ist die dargestellte bedingte Verzweigung eine „ m aus n “-Verzweigung. Das bedeutet, daß aus den n vorhandenen Pfaden der bedingten Verzweigung m Pfade⁷ zur Ausführung ausgewählt werden können. m kann auf 1 festgelegt werden, was aber bei der hier vorgestellten Notation offengelassen werden soll.

Der Kontrollfluß der bedingten Verzweigung verläuft folgendermaßen:

Ist der Eingang erreicht, so werden die durch ein Fragezeichen repräsentierten Verzweigungsregeln ausgewertet und entsprechend des Ergebnisses der Kontrollfluß weitergeleitet. Im Fall einer „1 aus n “-Alternative heißt das, daß eines der Strukturierungselemente „SE 1“ bis „SE n “ ausgeführt wird, im Fall „ m aus n “ werden mehrere parallel ausgeführt. Sind die durch die Verzweigungsbedingung ausgewählten Strukturierungselemente abgearbeitet, so wird die Verzweigung verlassen.

⁷ mit $0 \leq m \leq n$

2.4.3 Konsistenzbedingungen

Für die zusammengesetzten Strukturierungselemente Sequenz, Parallelkonstrukt und bedingte Verzweigung gelten einige Konsistenzbedingungen, die hier kurz aufgezählt werden sollen:

- *Sequenz*
 - Eine Sequenz muß mindestens zwei Sequenzelemente haben.
Begründung: Eine Sequenz mit nur einem Sequenzelement stellt keine Sequenz im eigentlichen Sinne mehr dar und kann durch ihr Element ersetzt werden, ohne die Ausführungseigenschaften des Workflows zu verändern.
 - Keine Sequenz kann Element einer Sequenz sein.
Begründung: Eine Sequenz *Seq1* innerhalb einer Sequenz *Seq2* kann unter Beibehaltung der Ausführungseigenschaften aufgelöst werden, indem die Elemente von *Seq1* in *Seq2* an der Position von *Seq1* eingereiht werden.
- *Parallelkonstrukt*
 - Ein Parallelkonstrukt muß mindestens zwei Parallelpfade haben.
 - Kein Parallelkonstrukt kann Pfad eines Parallelkonstruktes sein.
Begründung: Die Argumentation entspricht der bei der Sequenz.
- *Bedingte Verzweigung*
 - Eine bedingte Verzweigung muß mindestens zwei Verzweigungspfade haben.
Begründung: Falls der zweite Pfad leer sein soll (entspricht dem leeren ELSE-Zweig), muß er auch als solcher modelliert werden, d.h. mit einer leeren Arbeitseinheit belegt werden.

Kapitel 3

Spezifikation

In diesem Kapitel wird der Anpassungsdienst spezifiziert. Der Schwerpunkt liegt dabei auf der Darstellung der angebotenen Anpassungen, wobei die Auswirkungen jeder Anpassung und ihre Schnittstellenanforderungen aufgezeigt werden. Zuvor sollen im ersten Abschnitt (3.1) noch die Anforderungen an den Anpassungsdienst unter Betrachtung seiner Verbindungen nach außen im Rahmen der Zusammenarbeit mit anderen WFMS-Komponenten betrachtet werden.

3.1 Anforderungen an den Anpassungsdienst

Der Anpassungsdienst nimmt zur Erfüllung seiner Aufgaben eine Position zwischen der Benutzerschnittstelle einerseits und zentralen WFMS-Komponenten¹ andererseits ein (Abbildung 3.1):

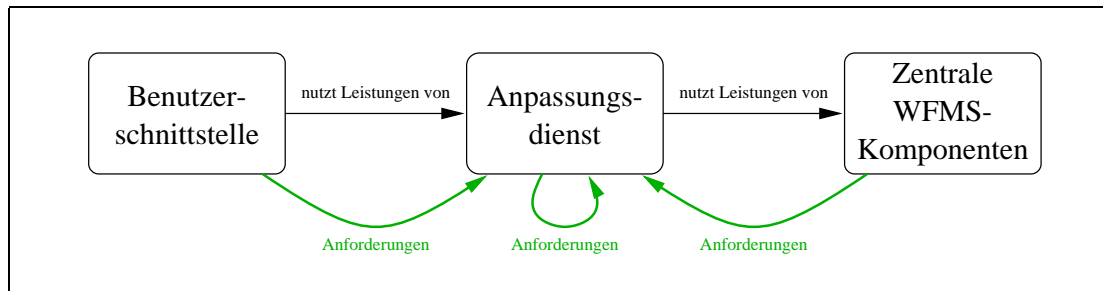


Abbildung 3.1: Zusammenarbeit und resultierende Anforderungen

Über die Benutzerschnittstelle bietet der Anpassungsdienst seine Leistungen dem Anwender an. Zentrale WFMS-Komponenten stellen ihrerseits Dienste zur Verfügung, die der Anpassungsdienst zur Realisierung seiner Leistungen benötigt,

¹Zu den „zentralen“ WFMS-Komponenten zählt zumindest der Workflow-Ausführungsdienst.

nämlich die Steuerung der Workflow-Ausführung während einer Anpassung sowie lesender und schreibender Zugriff auf Workflow-Instanzen und deren Daten.

Diese beiden Partner – Benutzerschnittstelle und zentrale WFMS-Komponenten – stellen Anforderungen an den Anpassungsdienst zur sinnvollen Zusammenarbeit, die gemeinsam mit den Anforderungen, die der Anpassungsdienst selbst stellt, im folgenden kurz dargelegt werden sollen. Ein Abwägen der jeweiligen Interessen ist zur Entscheidungsfindung nötig.

- **Von der Benutzerschnittstelle gestellte Anforderungen**

Aufgabe der Benutzerschnittstelle ist es, wie wir in den Abschnitten 2.2.3 und 2.3 gesehen haben, es dem Benutzer zu ermöglichen, daß er die von ihm gewünschten Anpassungen in einer intuitiven Weise dem System mitteilen kann. Die hieraus entspringende graphische Orientierung der Benutzerschnittstelle sollte sich in den Parametern zu den Anpassungsdiensten niederschlagen, so daß die Abbildung der vom Benutzer durchgeführten (graphischen) Aktionen auf die entsprechende Funktion des Anpassungsdienstes und die Versorgung desselben mit Parametern möglichst einfach wird.

Selbstverständlich ist die Grundlage, daß der Anpassungsdienst die vom Benutzer gewünschten und von der Benutzerschnittstelle angebotenen Anpassungen überhaupt anbietet.

- **Von den zentralen WFMS-Komponenten gestellte Anforderungen**

Die WFMS-Komponenten, die dem Anpassungsdienst Zugriff auf Workflow-Instanzen und ihre Daten bieten, präsentieren diese Informationen an ihrer Schnittstelle nach außen in einer von ihnen festgelegten Weise. Die Repräsentation der Workflows innerhalb des Anpassungsdienstes sollte hiermit eng gekoppelt sein, um wiederum den Abbildungsaufwand so gering wie möglich zu halten.

- **Interne Anforderungen**

Die von der Benutzerschnittstelle geforderte Parameterversorgung sowie die Anpassungsdienst-interne Repräsentation der Workflows müssen die Realisierung der spezifizierten Anpassungen ermöglichen und sollen den Realisierungsaufwand begrenzen.

3.2 Spezifikation von Einzelanpassungen

In [Ott96] wurden anhand eines umfangreichen Szenarios eines Geschäftsprozesses häufig auftretende Anpassungen ermittelt. Die Spezifikation der vorliegenden

Arbeit verwendet die Ergebnisse der dortigen Analyse und trifft eine Auswahl aus den gefundenen Anpassungen.

Die ausgewählten Anpassungen werden im Rahmen der in Abschnitt 2.4 eingeführten strukturierten Modellierung erklärt und auch in Abbildungen dargestellt. Zur Spezifikation einer Anpassung ist oft die Betrachtung eines gewissen Kontextes nötig, da die Auswirkungen entsprechend dieses Kontextes unterschiedlich sind. Der Kontext ist in der Darstellung der einzelnen Anpassungen so klein wie möglich und so groß wie nötig gewählt.

Zu jeder Anpassung wird eine grobe Schnittstellenspezifikation angegeben, d.h. die nötigen Parameter werden tabellarisch aufgeführt und kurz erläutert.

Die in diesem Abschnitt dargestellten Anpassungen werden Einzelanpassungen genannt. Sie sind kleinste Einheiten von sinnvollen Anpassungen, können aber – wie in Abschnitt 3.3 dargelegt – auch gruppiert werden: mehrere Einzelanpassungen hintereinander ausgeführt ergeben wieder eine sinnvolle Gesamtanpassung.

Die Darstellung der Einzelanpassungen erfolgt in drei Teilen: Der erste und größte Teil sind die Anpassungen, die den verhaltensbezogenen Aspekt der Workflow-Beschreibung betreffen (Unterabschnitte 3.2.1 bis 3.2.9). Darauf werden die Änderungen am operationalen Aspekt aufgeführt (Unterabschnitte 3.2.10 bis 3.2.13), und zuletzt folgt noch eine Anpassung, die den informationsbezogenen Aspekt betrifft (Unterabschnitt 3.2.14).

3.2.1 Entfernen einer Arbeitseinheit

Während der Workflow-Ausführung kann sich zeigen, daß eine in der Zukunft liegende Arbeitseinheit überflüssig wird. Durch das Entfernen dieser Arbeitseinheit kann der Workflow an die geänderten Gegebenheiten angepaßt werden.

Bei strikt lokaler Sicht bietet sich ein sehr einfaches Bild dieser Anpassung, wie es in Abbildung 3.2 dargestellt ist. Der Kontext ist hier außer Acht gelassen und durch Punkte repräsentiert. So ist nach dem Entfernen der Arbeitseinheit *AE* einfach der Kontext vor *AE* direkt mit dem Kontext nach *AE* verbunden.



Abbildung 3.2: Strikt lokale Sicht: Entfernen einer Arbeitseinheit

Ein Eingriff in den Kontrollfluß, wie er durch das Entfernen einer Arbeitseinheit geschieht, hat jedoch nachhaltigen Einfluß auf den Kontext, wenn dessen

Konsistenz erhalten bleiben soll. Aus diesem Grund werden im folgenden die verschiedenen möglichen Fälle dargestellt und erläutert.

Im allgemeinen gilt die folgende Schnittstellenspezifikation (das Entfernen aus einer bedingten Verzweigung benötigt eine eigene):

Entfernen einer Arbeitseinheit	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aeName	– Name der zu entfernenden Arbeitseinheit

Entfernen aus einer Sequenz

Abbildung 3.3 zeigt das Entfernen aus einer Sequenz im allgemeinen Fall.

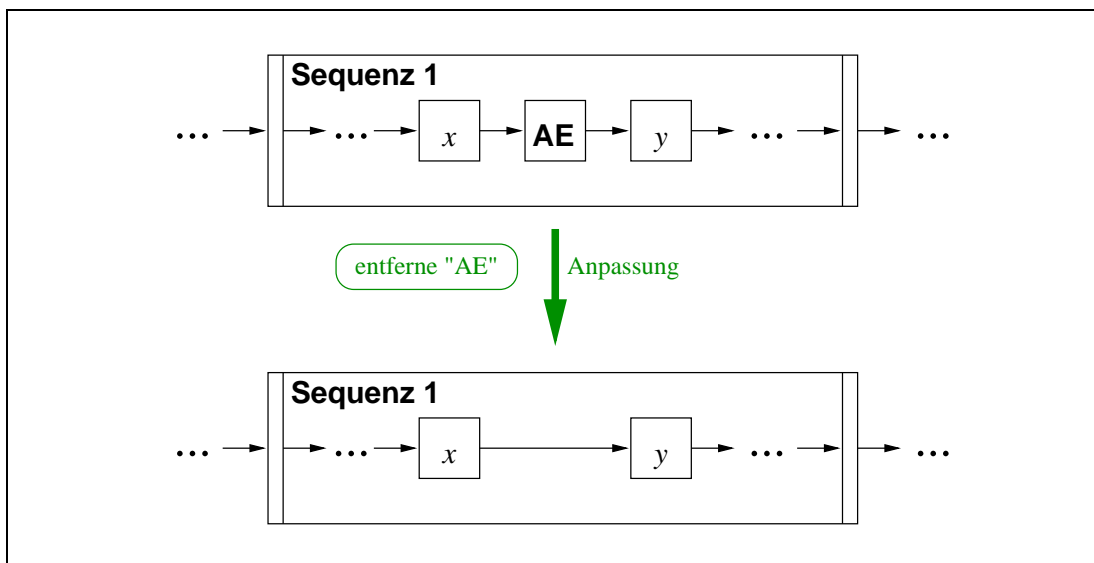


Abbildung 3.3: Entfernen aus einer Sequenz

Eine Konsistenzbedingung einer Sequenz ist, daß sie mindestens zwei Elemente enthält. So gibt es den Spezialfall des Entfernens aus einer minimalen Sequenz, einer Sequenz mit zwei Elementen (siehe Abbildung 3.4). In diesem Fall wird die Sequenz im Kontrollfluß durch das verbleibende Sequenzelement y ersetzt.

Entfernen aus einem Parallelkonstrukt

Beim Entfernen einer Arbeitseinheit aus einem Parallelkonstrukt treten zwei Fälle auf, die den soeben betrachteten beiden Fällen in der Sequenz entsprechen. Für

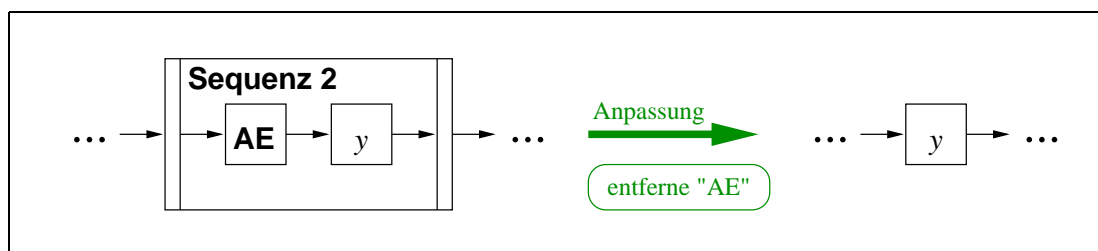


Abbildung 3.4: Entfernen aus einer minimalen Sequenz

das Parallelkonstrukt gilt dieselbe Konsistenzbedingung: es muß mindestens zwei Elemente enthalten, da ansonsten keine Parallelausführung mehr vorliegt.

Der reguläre Fall ist in Abbildung 3.5, der Sonderfall des minimalen Parallelkonstrukts in Abbildung 3.6 dargestellt. Man sieht in den Abbildungen, daß das Entfernen der Arbeitseinheit im Grunde ein Entfernen eines kompletten Parallelpfades ist, da ein leerer Parallelpfad überflüssig ist.

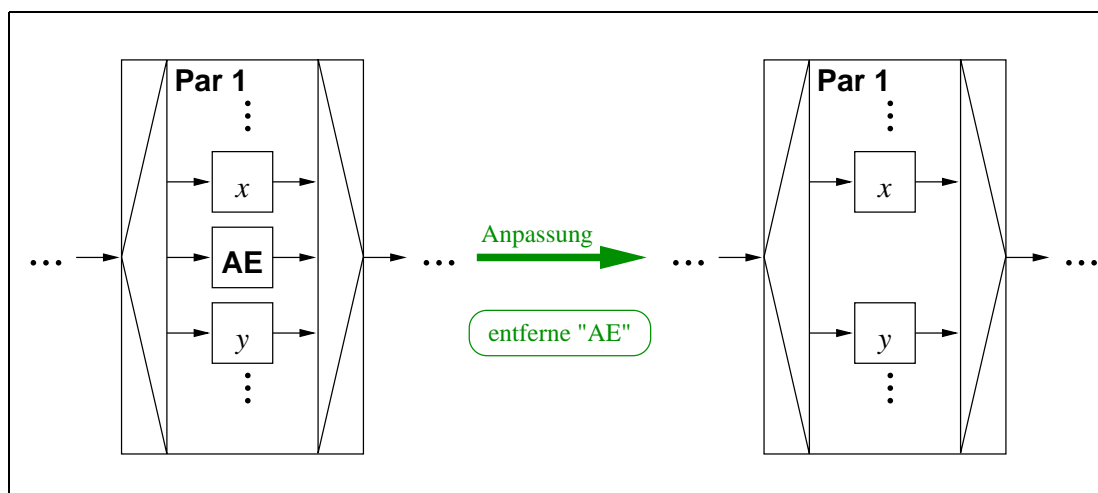


Abbildung 3.5: Entfernen aus einem Parallelkonstrukt

Entfernen aus einer bedingten Verzweigung

Das Entfernen einer Arbeitseinheit aus einer bedingten Verzweigung verhält sich genauso wie das aus einem Parallelkonstrukt mit dem regulären Fall und dem Sonderfall (Entfernen aus einer minimalen bedingten Verzweigung). Wegen der Strukturähnlichkeiten sei zur Veranschaulichung auf die dortigen Abbildungen verwiesen.

Hinzu kommt allerdings noch, daß eine Veränderung bezüglich der Zweige normalerweise auch das Modifizieren der Verzweigungsregeln notwendig macht. Somit kann für das Entfernen aus einer bedingten Verzweigung nicht dieselbe Schnitt-

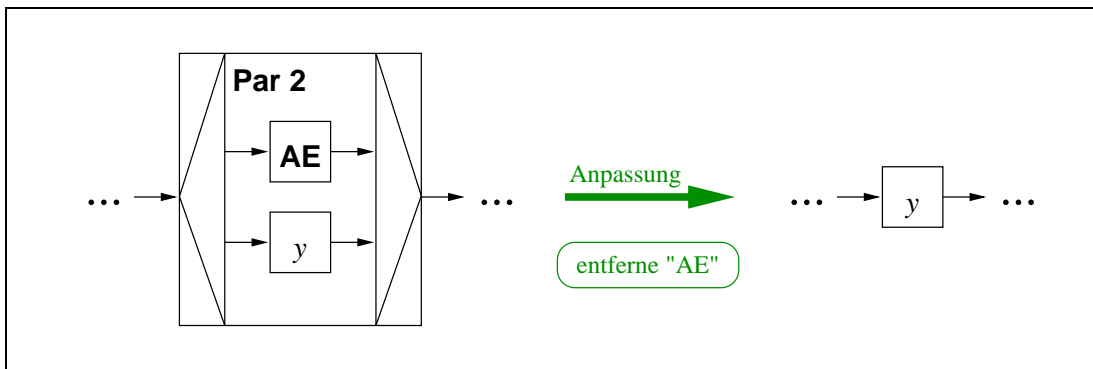


Abbildung 3.6: Entfernen aus einem minimalen Parallelkonstrukt

stelle verwendet werden wie für die anderen Fälle des Entfernens. Die Schnittstelle sieht hier so aus:

Entfernen einer Arbeitseinheit aus einer bedingten Verzweigung	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aeName	– Name der zu entfernenden Arbeitseinheit
regeln	– Modifizierte Verzweigungsregeln

Wenn der Sonderfall (Entfernen aus einer minimalen bedingten Verzweigung) eintritt, ist der Parameter **regeln** überflüssig und wird ignoriert.

3.2.2 Einfügen einer Arbeitseinheit

Häufig wird es bei der Abwicklung eines Geschäftsprozesses notwendig, eine zusätzliche Aktivität in den Vorgang aufzunehmen. Dies kann, sofern der Punkt in der Workflow-Beschreibung, an dem die zusätzliche Aktivität ausgeführt werden soll, noch in der Zukunft liegt, durch Einfügen einer Arbeitseinheit in die Workflow-Beschreibung getan werden.

Die Anpassung Einfügen bezieht sich immer auf ein im Workflow bereits existierendes Strukturierungselement und gibt die Kontrollflußbeziehung zwischen diesem und der einzufügenden Arbeitseinheit an. Die Schnittstellen sind je nach Kontrollflußbeziehung verschieden.

Sequentielles Einfügen

Beim sequentiellen Einfügen muß außer der einzufügenden Arbeitseinheit und dem Bezugspunkt noch spezifiziert werden, ob vor oder nach dem Bezugspunkt eingefügt werden soll. Es ergibt sich folgende Schnittstellenspezifikation.

Sequentielles Einfügen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
ae	– einzufügende Arbeitseinheit
bezugsSeName	– Name des Bezugs-Strukturierungselements
relation	– <i>vor</i> oder <i>nach</i> bezugsSeName einfügen

Die Berücksichtigung der strukturellen Konsistenzbedingungen von Sequenzen führt zur Unterscheidung von vier Fällen, die nachfolgend betrachtet werden. Die obige Schnittstellenspezifikation hat für alle vier Fälle Gültigkeit.

1. *Einfügen vor oder nach einem Strukturierungselement, das weder Element einer Sequenz, noch selbst Sequenz ist:*

In diesem Fall muß eine Sequenz erzeugt werden, die die beiden Strukturierungselemente, das Bezugsselement sowie die neu eingefügte Arbeitseinheit, als Sequenzelemente enthält. Abbildung 3.7 illustriert das Einfügen *vor* einem solchen Strukturierungselement.

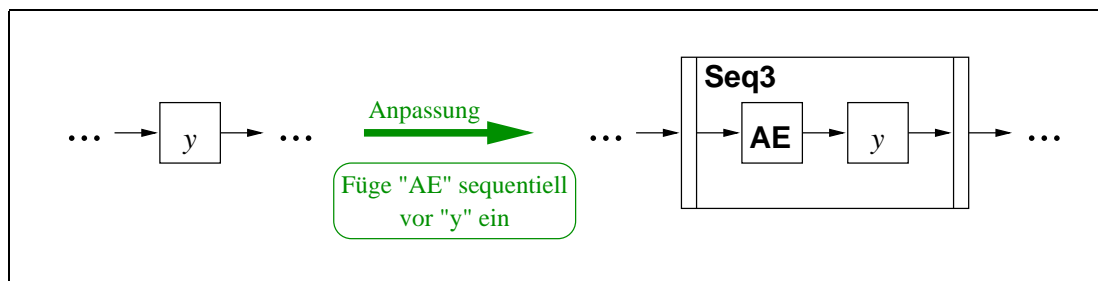


Abbildung 3.7: Sequentielles Einfügen

Die nachfolgenden drei Fälle zeichnen sich dadurch aus, daß eine Sequenz bereits existiert und nur noch in diese eingefügt werden muß.

2. *Einfügen vor dem ersten Element einer Sequenz bzw. vor einer Sequenz:*

Das Einfügen vor einer Sequenz darf nicht dazu führen, daß eine Sequenz entsteht, die die neu eingefügte Arbeitseinheit und die bereits vorher existierende Sequenz als Elemente enthält, da entsprechend der strukturellen Konsistenzbedingungen eine Sequenz nicht Element einer Sequenz sein darf. Daher wird dieser Fall gleich behandelt wie das Einfügen vor dem ersten Element der Sequenz, nämlich als Einfügen am Anfang einer Sequenz. Abbildung 3.8 stellt die Anpassung dar.

3. *Einfügen nach dem letzten Element einer Sequenz bzw. nach einer Sequenz:*

Dieser Fall entspricht – analog dem vorherigen – einem Einfügen am Ende einer Sequenz.

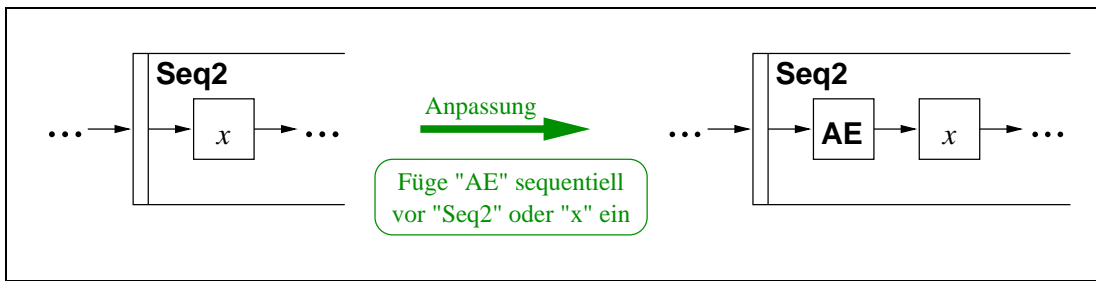


Abbildung 3.8: Sequentielles Einfügen am Anfang einer Sequenz

4. Einfügen innerhalb einer Sequenz:

Innerhalb einer Sequenz, also weder am Anfang (Fall 2) noch am Ende (Fall 3), gestaltet sich das Einfügen am einfachsten. Die einzufügende Arbeitseinheit wird an der gewünschten Stelle zwischen zwei Sequenzelemente „geschoben“, wie dies in Abbildung 3.9 dargestellt ist.

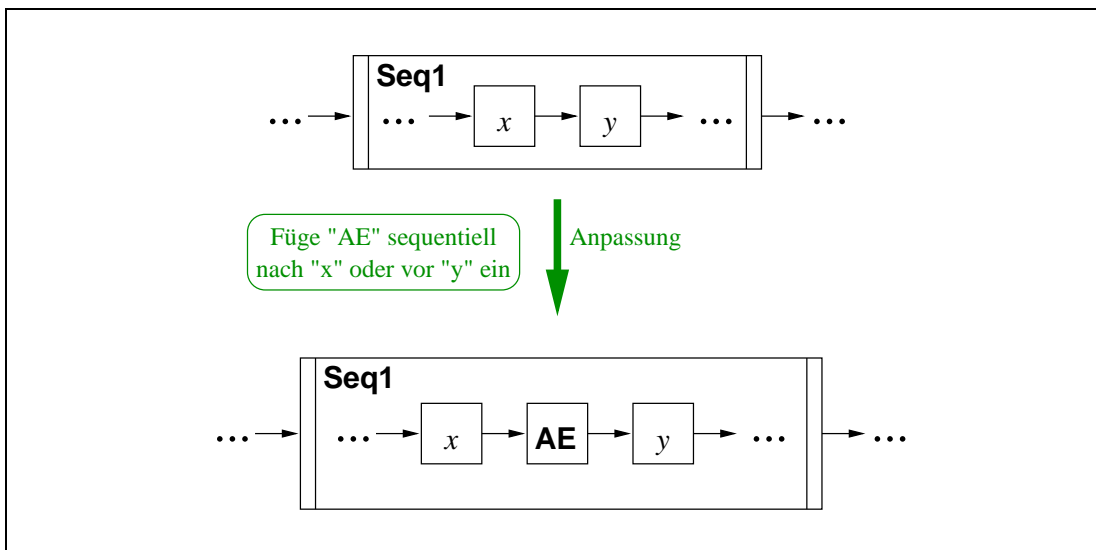


Abbildung 3.9: Sequentielles Einfügen innerhalb einer Sequenz

Paralleles Einfügen

Eine neue Arbeitseinheit soll in den Workflow aufgenommen werden. Sie soll parallel zu einem bereits existierenden Strukturierungselement ausgeführt werden.

Die Schnittstellenspezifikation lautet:

Paralleles Einfügen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
ae	– einzufügende Arbeitseinheit
bezugsSeName	– Name des Bezugs-Strukturierungselements

Die Anpassung gliedert sich in zwei Fälle: das Einfügen in ein schon bestehendes Parallelkonstrukt der Workflow-Beschreibung und das Einfügen, bei dem ein Parallelkonstrukt erst erzeugt werden muß. Die Fallunterscheidung geschieht durch Betrachten des Bezugs-Strukturierungselements:

1. *Einfügen parallel zu einem Parallelkonstrukt oder einem Element eines Parallelkonstruktes*

In diesem Fall muß nicht erst ein Parallelkonstrukt zur Parallelausführung erzeugt werden, sondern die einzufügende Arbeitseinheit kann als neues Element in das bestehende Parallelkonstrukt aufgenommen werden. Abbildung 3.10 stellt die Anpassung graphisch dar.

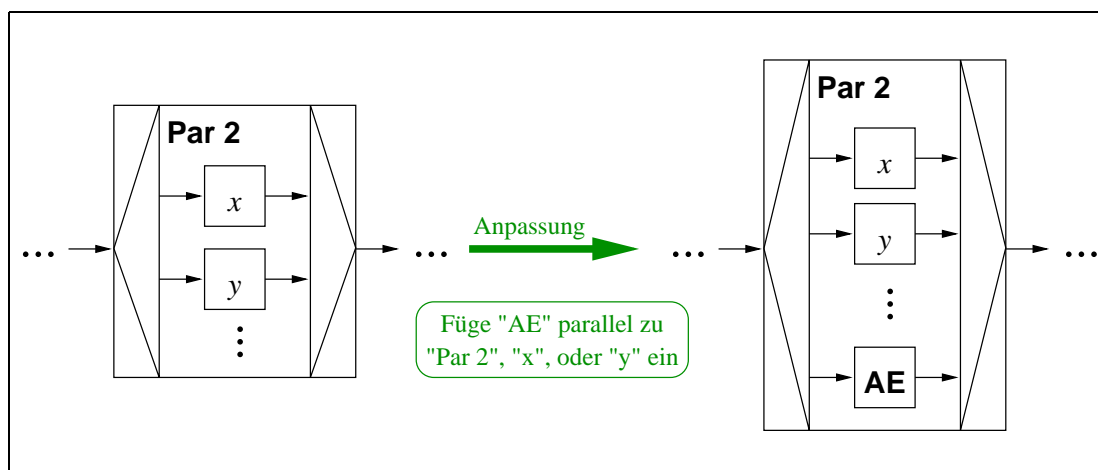


Abbildung 3.10: Paralleles Einfügen in ein bestehendes Parallelkonstrukt

2. *Übrige Fälle*

Soll das Einfügen parallel zu einem Strukturierungselement stattfinden, das weder selbst ein Parallelkonstrukt noch Element eines solchen ist, so muß zuerst ein Parallelkonstrukt erzeugt werden, in das dann eingefügt werden kann. Die Auswirkung der Anpassung für diesen Fall zeigt Abbildung 3.11.

Alternatives Einfügen

Eine neue Arbeitseinheit soll die Alternative zu einem bereits existierenden Strukturierungselement bilden. Hierzu wird das betroffene Strukturierungsele-

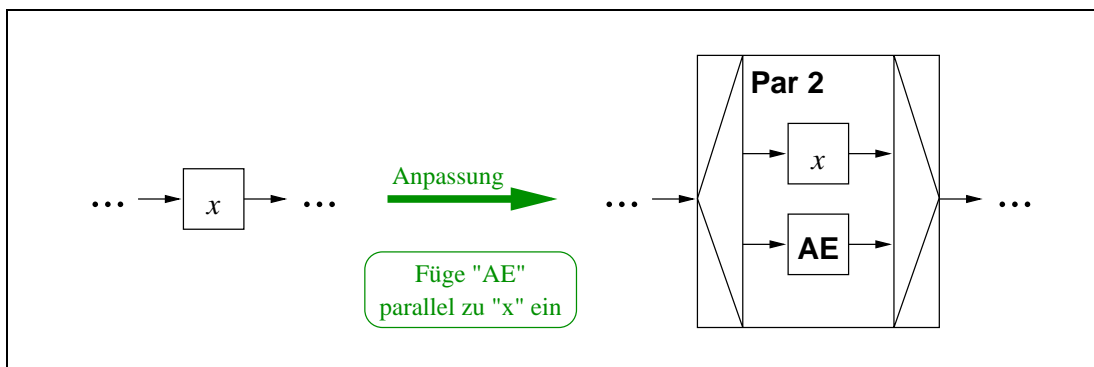


Abbildung 3.11: Paralleles Einfügen

ment durch eine neu erzeugte bedingte Verzweigung ersetzt. Die bedingte Verzweigung enthält das Strukturierungselement und die neue Arbeitseinheit als Elemente. Ihre Verzweigungsregeln müssen durch die Anpassung auch wie gewünscht gesetzt werden. Die Schnittstelle ist folgendermaßen spezifiziert:

Alternatives Einfügen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
ae	– einzufügende Arbeitseinheit
bezugsSeName	– Name des Bezugs-Strukturierungselements
regeln	– Verzweigungsregeln

Bedingte Verzweigungen können auch bedingte Verzweigungen als Elemente enthalten; daher ist hier keine aufwendige Fallunterscheidung wie beim parallelen Einfügen notwendig. Die Anpassung ist in Abbildung 3.12 graphisch dargestellt.

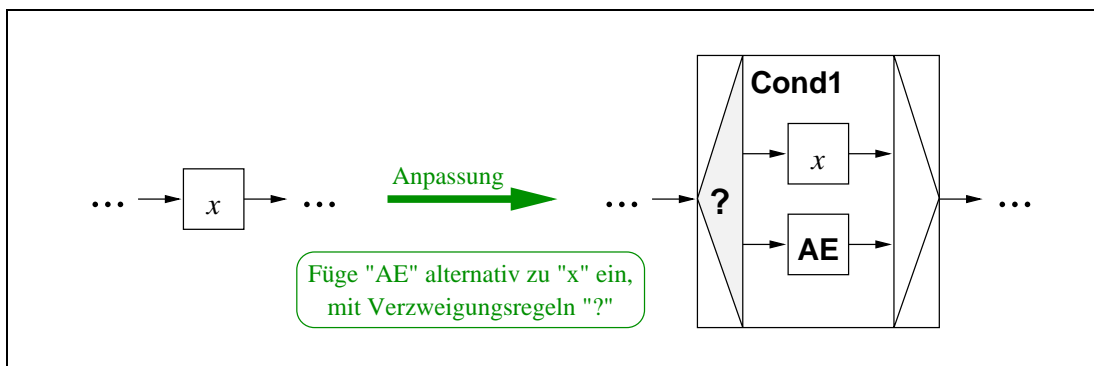


Abbildung 3.12: Alternatives Einfügen

Einfügen in eine bedingte Verzweigung

Mit Hilfe des alternativen Einfügens kann eine bestehende bedingte Verzweigung nicht um ein zusätzliches Element, einen weiteren Alternativpfad, erweitert werden. Zu diesem Zweck dient die Anpassung „Einfügen in eine bedingte Verzweigung“. Das Einfügen eines weiteren Alternativpfades erfordert normalerweise auch eine Änderung der Verzweigungsregeln. Somit ergibt sich die folgende Schnittstelle:

Einfügen in eine bedingte Verzweigung	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
ae	– einzufügende Arbeitseinheit
bedVerzwName	– Name der zu erweiternden bedingen Verzweigung
regeln	– Modifizierte Verzweigungsregeln

Die Anpassung ist in Abbildung 3.13 illustriert.

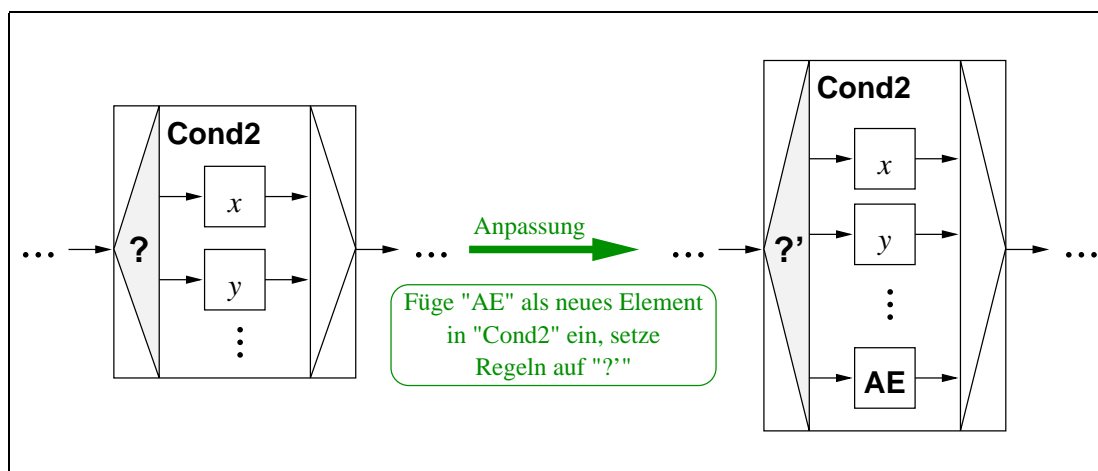


Abbildung 3.13: Einfügen in eine bedingte Verzweigung

3.2.3 Änderung von Regeln in bedingten Verzweigungen

Die Regeln in bedingten Verzweigungen steuern den Kontrollfluß innerhalb des Konstrukts durch Auswahl der zu durchlaufenden Pfade aus den Elementen der Verzweigung. Eine Modifikation dieser Regeln kann auch ohne sonstige Änderung der bedingten Verzweigung erforderlich werden. Die Anpassung hat die folgende Schnittstellenspezifikation:

Regeln in bedingten Verzweigungen ändern	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
bedVerzwName	– Name der betroffenen bedingten Verzweigung
regeln	– Modifizierte Verzweigungsregeln

3.2.4 Explizite Pfadauswahl in bedingten Verzweigungen

Diese Anpassung ähnelt der im vorigen Abschnitt beschriebenen „Änderung von Regeln in bedingten Verzweigungen“. Sie bietet dem Anwender jedoch eine einfachere Schnittstelle. Er soll sich nicht mit der Formulierung von Regeln befassen müssen, wenn sein Ziel ist, einen oder mehrere Pfade einer bedingten Verzweigung explizit zur Ausführung auszuwählen.

Wenn der Workflow-Ausführungsdienst eine bedingte Verzweigung bearbeitet, bei der ein Anwender zuvor eine explizite Pfadauswahl getroffen hat, so „überstimmt“ diese Pfadauswahl die ansonsten stattfindende Auswahl durch Auswertung der Verzweigungsregeln. D.h. es werden alle ausgewählten Pfade und nur diese abgearbeitet. Die Schnittstellenspezifikation dieser Anpassungsklasse lautet:

Explizite Pfadauswahl	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
bedVerzwName	– Name der betroffenen bedingten Verzweigung
pfadListe	– ausgewählte Pfade, repräsentiert durch die Namen der Strukturierungselemente

3.2.5 Aktivität/Subworkflow scheitern lassen

Bei der Bearbeitung einer Aktivität bzw. eines Subworkflows wird möglicherweise festgestellt, daß sie nicht erfolgreich abgeschlossen werden kann und eine weitere Bearbeitung sinnlos ist. Die Aktivität bzw. der Subworkflow muß beendet und dem Workflow-Ausführungsdienst das Scheitern mitgeteilt werden.

Da der Parameter `instanzBez` eine Aktivitäten- bzw. Subworkflow*instanz* bezeichnet, ist sichergestellt, daß nur das Scheiternlassen von gerade in Bearbeitung befindlichen Aktivitäten/Subworkflows möglich ist. Die Schnittstelle wird folgendermaßen spezifiziert:

Aktivität/Subworkflow scheitern lassen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktSubwf	– Aktivität oder Subworkflow?
instanzBez	– Bezeichnung der Aktivitäten- bzw. Subworkflowinstanz, die scheitern gelassen werden soll

3.2.6 Aktivität/Subworkflow wiederholen

Eine gerade in Bearbeitung befindliche Aktivität (bzw. Subworkflow, so auch im folgenden) würde ohne Neuanfang scheitern, kann aber durch Wiederholen zum erfolgreichen Abschluß geführt werden. In diesem Fall wird die Aktivität nicht mit der im vorigen Abschnitt beschriebenen Anpassung scheitern gelassen, sondern mit Hilfe der Wiederholung wird die Aktivität noch einmal von neuem begonnen. Der Aktivität zur Verfügung gestellte Workflow-relevante Daten werden wieder mit den Werten belegt, die sie vor dem ersten Versuch hatten.

Die Schnittstellenspezifikation dieser Anpassung sieht so aus:

Aktivität/Subworkflow wiederholen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktSubwf	– Aktivität oder Subworkflow?
instanzBez	– Bezeichnung der zu wiederholenden Aktivitäten- bzw. Subworkflowinstanz

3.2.7 Arbeitseinheit vorziehen

Während der Abwicklung eines Geschäftsprozesses kann es z.B. in Leerlaufzeiten sinnvoll sein, eine erst später eingeplante Arbeitseinheit vorzeitig zu bearbeiten, wenn alle zu deren Bearbeitung nötigen Informationen bereits vorliegen.

Die Anpassung „Arbeitseinheit vorziehen“ startet die Aktivität bzw. den Subworkflow der betreffenden Arbeitseinheit nebenläufig zur regulären Workflow-Ausführung und vermerkt deren vorzeitige Bearbeitung. Wenn die vorzeitige Bearbeitung abgeschlossen ist, wird auch dies vermerkt. Trifft der Workflow-Ausführungsdienst bei Fortführung seiner regulären Workflow-Abarbeitung auf eine Arbeitseinheit, die sich in vorzeitiger Bearbeitung befindet, so wird die reguläre Workflow-Ausführung verzögert, bis die vorzeitige Bearbeitung beendet ist, und dann nach der vorgezogenen Arbeitseinheit fortgesetzt. Trifft der Workflow-Ausführungsdienst auf eine Arbeitseinheit, deren vorgezogene Bearbeitung be-

reits beendet ist, so kann er die reguläre Workflow-Abarbeitung sofort nach der vorgezogenen Arbeitseinheit fortsetzen.

Die Schnittstelle für diese Anpassung ist wie folgt spezifiziert:

Arbeitseinheit vorziehen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aeName	– Name der vorzuziehenden Arbeitseinheit

3.2.8 Arbeitseinheit auslassen

Diese Anpassung ähnelt der Anpassung „Arbeitseinheit entfernen“. Die Aktivität bzw. der Subworkflow der Arbeitseinheit wird für die Bearbeitung des Geschäftsprozesses überflüssig. Jedoch soll die Arbeitseinheit weiterhin in der Beschreibung der Workflow-Instanz verbleiben, nur als „auszulassen“ gekennzeichnet werden und zum Zeitpunkt ihrer Ausführung auch tatsächlich ausgelassen werden. Die Historie verzeichnet dann, daß die Arbeitseinheit ausgelassen wurde.

Im Fall einer Erweiterung des Workflow-Beschreibungs-Konzeptes um Schleifen wird der Unterschied zu „Arbeitseinheit entfernen“ noch größer: die Arbeitseinheit wird dann nur beim direkt auf die Anpassung folgenden Schleifendurchlauf ausgelassen und bei weiteren Durchläufen wieder ausgeführt.

Die Schnittstellenspezifikation ist wie folgt:

Arbeitseinheit auslassen	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aeName	– Name der auszulassenden Arbeitseinheit

3.2.9 Zusatzaktivität starten

Beim Abarbeiten einer Arbeitseinheit kann der Bearbeiter feststellen, daß eine zusätzliche Aktivität notwendig wird. Er startet diese als Zusatzaktivität, die parallel zum momentanen Workflowablauf bearbeitet wird. Beispiele für Zusatzaktivitäten sind nach [Ott96]: Rücksprache, Mitzeichnung, Information Dritter und Einholen von Informationen.

Die Zusatzaktivität kann unterschiedlich mit dem übrigen Workflowablauf synchronisiert werden:

1. Das Ende der Zusatzaktivität wird mit dem Abschluß der Arbeitseinheit, von der aus die Zusatzaktivität gestartet wurde, synchronisiert.
2. Das Ende der Zusatzaktivität wird mit dem Ende einer anderen, später abzuarbeitenden Arbeitseinheit (oder Strukturierungselement) synchronisiert.
3. Keine explizite Synchronisation wird spezifiziert. (In diesem Fall sollte jedoch sinnvollerweise implizit mit dem Ende des Workflows synchronisiert werden.)

Die Schnittstellenspezifikation für diese Anpassung:

Zusatzaktivität starten	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktName	– Name der zusätzlich zu startenden Aktivität
syncArt	– Synchronisationsart (siehe obige Aufzählung)
syncSeName	– bei Synchronisationsart 2: Name des Strukturierungselements, mit dessen Ende die Zusatzaktivität synchronisiert werden soll

3.2.10 Änderung von Rollen

In der Beschreibung einer Aktivität ist deren Akteur als Rolle angegeben. Diese Rolle wird zu Beginn der Aktivitätsbearbeitung aufgelöst, indem ein Repräsentant der Rolle als Bearbeiter der Aktivität gewählt wird. Wenn im Verlauf der Prozeßabwicklung klar wird, daß eine Aktivität im weiteren Verlauf der Workflow-Instanz von einem Repräsentanten einer anderen Rolle als der bisher angegebenen bearbeitet werden soll, so muß die Rolle geändert werden können.

Die Schnittstellenspezifikation für die Rollenänderung ist:

Rollenänderung	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktName	– Name der anzupassenden Aktivität
rolle	– einzutragende Rolle

3.2.11 Explizite Rollenauflösung

Wenn beispielsweise ein Vorgesetzter eine Aktivität einem von ihm gewünschten Mitarbeiter zur Bearbeitung zuweisen möchte, kann dies mit Hilfe der expliziten

Rollenauflösung geschehen. Ähnlich wie bei der expliziten Pfadauswahl wird hier die automatische Rollenauflösung durch die gewünschte Zuweisung „überstimmt“. Diese Anpassung muß somit *vor* dem Start der Aktivität durchgeführt werden.

Die Schnittstelle ist folgendermaßen spezifiziert:

Explizite Rollenauflösung	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktName	– Name der anzupassenden Aktivität
aktor	– gewünschter Aktor für die Aktivität

3.2.12 Aktivität abgeben

Während der Bearbeitung einer Aktivität – die Rollenauflösung hat also bereits stattgefunden – kann der Fall auftreten, daß die Aktivität von ihrem derzeitigen Bearbeiter nicht zuende bearbeitet werden kann und an einen anderen Bearbeiter abgegeben werden muß. Beispielsweise kann dies bei Überlastung durch den Bearbeiter manuell oder bei Krankheit des Bearbeiters durch eine Vertreterregelung automatisch erfolgen. In der Historie sollten beide (und evtl. noch folgende) Bearbeiter vermerkt werden.

Die Schnittstellenspezifikation dieser Anpassung:

Aktivität abgeben	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktInstBez	– Bezeichnung der Aktivitäteninstanz, die abgegeben werden soll
aktor	– Bearbeiter, an den die Aktivität abgegeben werden soll

3.2.13 Termine ändern

Für die Bearbeitung eines Workflows existiert ein Zeitmodell, das u.a. Termine umfaßt, an denen eine Aktivität oder ein Subworkflow abgearbeitet sein sollte.² Termine können absolut angegeben werden, aber auch relativ zur Workflow- oder Aktivitätsinstanziierung (relative Termine entsprechen Fristen). Die Änderung

²Auf die Behandlung von Terminüberschreitungen durch den Workflow-Ausführungsdienst soll hier nicht weiter eingegangen werden.

von Terminen kann nötig werden, wenn beispielsweise eine Prozeßabwicklung beschleunigt werden muß, oder aber bestimmte anormale Verzögerungen im Verlauf eines Workflows toleriert werden sollen.

Es ergibt sich die folgende grobe Schnittstellenspezifikation, die entsprechend dem im WFMS verwendeten Terminmodell verfeinert werden muß:

Termin ändern	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
aktInstBez	– Bezeichnung der anzupassenden Aktivitäteninstanz
termin	– neuer Termin

3.2.14 Änderung von Workflow-relevanten Daten

Workflow-relevante Daten können vom Workflow-Ausführungsdienst aber auch von Arbeitseinheiten, bzw. deren Aktivität oder Subworkflow, verändert und auch ausgewertet werden. Über diese Daten kommunizieren die beschriebenen Gruppen auch miteinander. In bestimmten Situationen kann es notwendig werden, von außen durch eine Änderung in dieses geschlossene System einzugreifen.

Die Schnittstelle zur Anpassung von Workflow-relevanten Daten:

Änderung von Workflow-relevanten Daten	
<i>Parameter:</i>	
wfInstanzBez	– Bezeichnung der anzupassenden WF-Instanz
daten	– neue Datenbelegung

3.3 Gruppierung von Einzelanpassungen

In einigen Fällen sollen mehrere Anpassungen gemeinsam durchgeführt werden, d.h. jede einzelne soll nur zur Wirkung gelangen, wenn auch die andern garantiert ausgeführt werden. Hierzu ist es sinnvoll, Einzelanpassungen gruppieren zu können. Dies geschieht in Anlehnung an Datenbanktransaktionen, indem der Benutzer dem Anpassungsdienst mitteilt, daß eine Anpassungsgruppe beginnt (BEGIN), daraufhin die Einzelanpassungen spezifiziert und die Gruppe durch eine Bekräftigung zur Durchführung freigibt (COMMIT) oder für ungültig erklärt (ABORT).

Kapitel 4

Entwurf

Dieses Kapitel stellt den Entwurf des Anpassungsdienstes dar.

Einführend gibt Abschnitt 4.1 eine kurze Erläuterung der Darstellungselemente, die in den Klassendiagrammen verwendet werden.

Im zweiten Abschnitt wird die Architektur des Anpassungsdienstes dargestellt, indem zuerst die Aufteilung in zwei Prozesse und dann die Subsystemstruktur des Hauptprozesses motiviert und beschrieben werden.

Eine kurze Darstellung der vom Anpassungsdienst verwendeten WFMS-Funktionen ist Inhalt des dritten Abschnitts. Diese Funktionen werden dem Hauptprozeß, dem eigentlichen Anpassungsdienst, durch den anderen Prozeß, ein Gateway, vermittelt.

Die Abschnitte 4.4 und 4.5 beschreiben die Entwurfseinzelheiten des Hauptprozesses. Im ersten von beiden werden die zur Repräsentation einer Workflow-Instanz entworfenen Klassen dargestellt und erläutert. Inhalt des zweiten von beiden ist die Schnittstelle des Anpassungsdienstes. Dort werden der Entwurf der einzelnen Methoden zur Realisierung von Anpassungen vorgestellt und die Methodenschnittstellen angegeben.

4.1 Notation zur Entwurfsdarstellung

Mit Blick auf Java als Implementierungssprache für den Anpassungsdienst werden in der Darstellung des Entwurfs Methodenschnittstellen in Form von Java-Methodendefinitionen angegeben. Die Parameter dieser Methoden werden auf die Definitionen folgend in den aus Abschnitt 3.2 bekannten Tabellen erläutert.

Zur Veranschaulichung von Entwurfszusammenhängen wurde in dieser Arbeit auf die Notation von Grady Boochs „Objektorientierte Analyse und Design“ [Booch95] zurückgegriffen. Die Darstellungselemente in Klassendiagrammen sollen an dieser Stelle anhand von Abbildung 4.1 kurz erläutert werden.

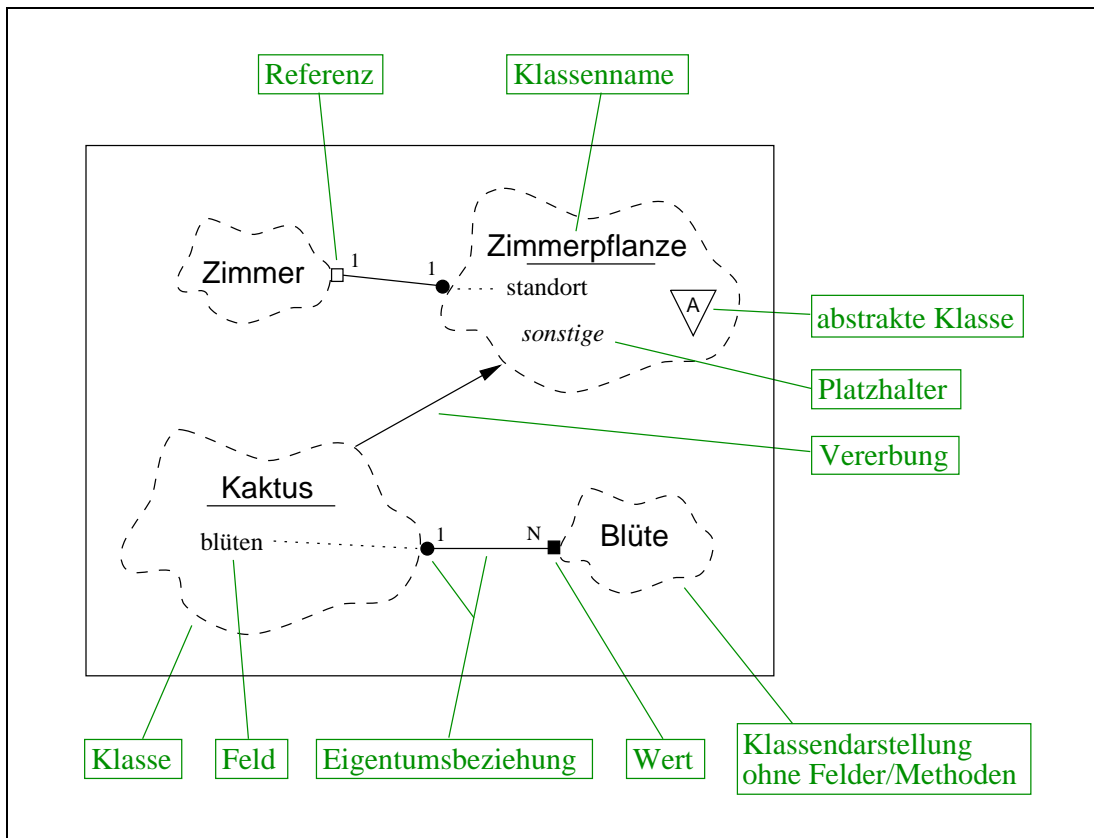


Abbildung 4.1: Darstellungselemente in Klassendiagrammen

Das abgebildete Klassendiagramm stellt beispielhaft vier Klassen (in gestrichelten „Wolken“) und ihre Zusammenhänge dar. Die beiden Klassen *Zimmerpflanze* und *Kaktus* sind etwas ausführlicher, die beiden Klassen *Zimmer* und *Blüte* ohne Nennung von Feldern und Methoden dargestellt. Der kursiv gedruckte Platzhalter *sonstige* soll andeuten, daß *Zimmerpflanze* außer ihrem explizit angegebenen Feld *standort* auch noch andere Felder oder Methoden enthält, die in diesem Diagramm nur durch den Platzhalter erwähnt aber nicht genau dargestellt werden sollen. *Zimmerpflanze* ist eine abstrakte Klasse, was durch das A im Dreieck signalisiert wird.

Eigentumsbeziehungen zwischen Klassen werden üblicherweise durch Felder in einer der durch die Beziehung verbundenen Klassen repräsentiert. Die besitzende Klasse, die durch einen gefüllten Kreis an der Stelle gekennzeichnet ist, wo die Beziehungslinie auf die Klassenwolke trifft, enthält dieses Feld. Die zu einer Beziehung gehörigen Felder – hier *Zimmerpflanze.standort* und *Kaktus.blüten* sind mit gepunkteten Linien mit der entsprechenden Eigentumsbeziehung verbunden. Ob eine Eigentumsbeziehung durch Referenz oder Wert in der Klasse repräsentiert wird läßt sich dem leeren respektive gefüllten Quadrat am anderen Ende der Beziehungslinie entnehmen. Die Beziehung zwischen *Kaktus* und *Blüte* ist eine „1:N“-Beziehung, d.h. ein *Kaktus*-Objekt besitzt mehrere *Blüte*-Objekte. Diese

sind als Werte (gefülltes Quadrat) im Feld `blüten` abgelegt. Ein Zimmerpflanze-Objekt besitzt genau eine Referenz (leeres Quadrat) auf ein Zimmer-Objekt in seinem `standort`-Feld.

Vererbungsbeziehungen werden mit Hilfe von Pfeilen ausgedrückt: `Kaktus` ist eine Subklasse von `Zimmerpflanze`.

4.2 Architektur

4.2.1 Prozeßarchitektur

Im Abschnitt 3.1 wurde die Position des Anpassungsdienstes zwischen der Benutzerschnittstelle und den zentralen WFMS-Komponenten dargestellt: Die Benutzerschnittstelle spielt gegenüber dem Anpassungsdienst die Rolle des Dienstanwenders, die zentralen WFMS-Komponenten die der Dienstbringer. Aus dieser Konstellation ergibt sich eine Prozeßarchitektur, die in Abbildung 4.2 dargestellt ist und im folgenden weiter erläutert werden soll.

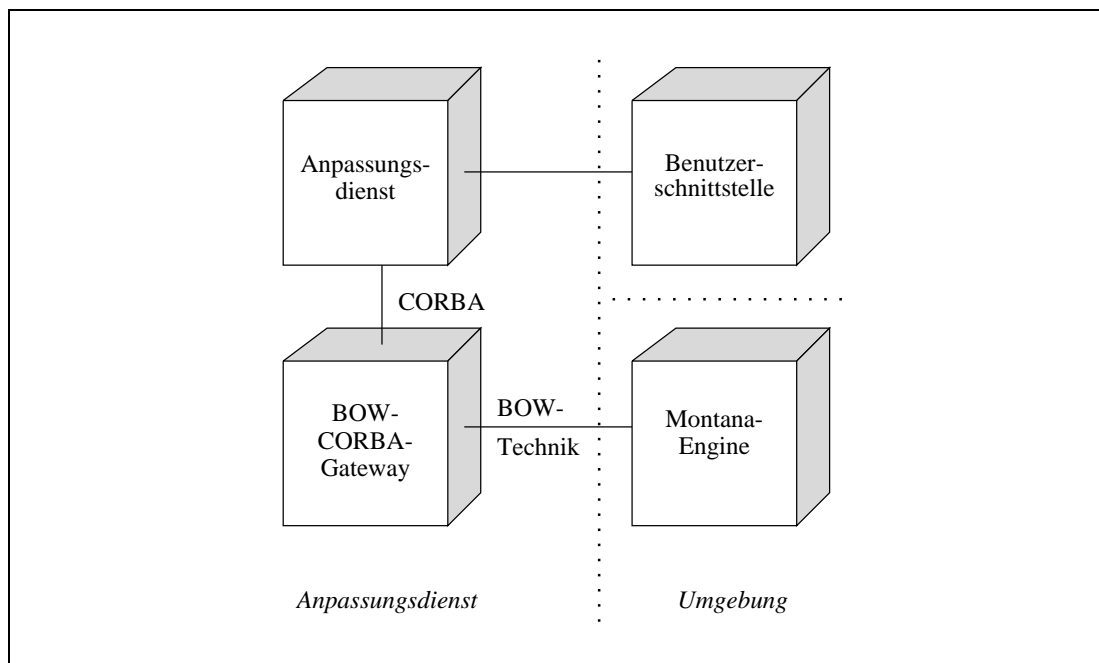


Abbildung 4.2: Prozeßarchitektur des Anpassungsdienstes und seiner Umgebung

In der Entwicklungsumgebung der vorliegenden Diplomarbeit (siehe Abschnitt 2.3) wird die Funktionalität der zentralen WFMS-Dienste – Workflow-Ausführung und Zugriffe auf Workflow-Instanz-Beschreibung und -Daten – von der Montana-Engine zur Verfügung gestellt. Dies geschieht mit Hilfe von Montana's Business-Object-Wrapper-Bibliothek (BOW), die die Anbindung von In-

Infrastrukturkomponenten an die Montana-Engine und die dabei anfallende Kommunikation sehr vereinfacht. Die BOW-Klasse sowie die gesamte Entwicklungsumgebung von Montana liegen in C++ vor. Der Anpassungsdienst, der in Java realisiert wird, kann somit nicht direkt an die Montana-Engine angebunden werden.

Zur Vermittlung der für den Anpassungsdienst relevanten Dienste der Montana-Engine wird in der vorliegenden Arbeit ein BOW-Modul in C++ implementiert, das **BOW-CORBA-Gateway**. Es bindet sich mit Hilfe der BOW-Technik als Infrastrukturkomponente an die Montana-Engine an und stellt die benötigte Untermenge der Montana-API-Funktionalität in Form von CORBA-Methoden zur Verfügung, wie auch aus der Abbildung zu entnehmen ist.

Der **Anpassungsdienst** kann nun mit Hilfe der Methoden des BOW-CORBA-Gateways mit der Montana-Engine kommunizieren. Aus diesen beiden Prozessen – dem Anpassungsdienst in Java und dem BOW-CORBA-Gateway in C++ – besteht die Realisierung der vorliegenden Arbeit. Die Partnerprozesse aus der Umgebung sind, wie in der Abbildung dargestellt, die Montana-Engine und die Benutzerschnittstelle des Anpassungsdienstes (beispielsweise der Workflow-Editor).

Die Zusammenarbeit zwischen dem Anpassungsdienst und der Benutzerschnittstelle entspricht dem Model-View-Controller-Paradigma¹:

Der Anpassungsdienst der vorliegenden Arbeit stellt das *Modell* der Workflow-Instanz sowie diverse Methoden für Anpassungen zur Verfügung. Die Benutzerschnittstelle umfaßt verschiedene *Sichten* auf dieses Modell und die *Steuerung* in der Form, daß der Anwender mit Hilfe der Benutzerschnittstelle die gewünschten Anpassungen spezifiziert und geeignete Sichten des Modells auswählt.

4.2.2 Subsysteme des Anpassungsdienstes

Der Anpassungsdienst-Prozeß wird in diesem Abschnitt zur Darstellung der funktionalen Struktur in vier Subsysteme gegliedert. Abbildung 4.3 zeigt die Subsysteme und ihre Abhängigkeiten. Die einzelnen Subsysteme werden in den folgenden Unterabschnitten beschrieben.

Montana-API-Zugang

In diesem Subsystem wird die Funktionalität zum Zugang zur Montana-Engine bereitgestellt. Die Basis des Subsystems stellen daher die Klassen der Java-Repräsentation des BOW-CORBA-Gateways dar. Diese Klassen sind durch die Übersetzung der IDL-Schnittstellen-Definition des BOW-CORBA-Gateways entstanden. Auf dieser Basis aufbauend werden weitere vereinfachte Zugangsmetho-

¹Eine Erläuterung des Model-View-Controller-Paradigmas ist in fast allen Werken zu Smalltalk zu finden. Siehe auch [KrasnerPope88].

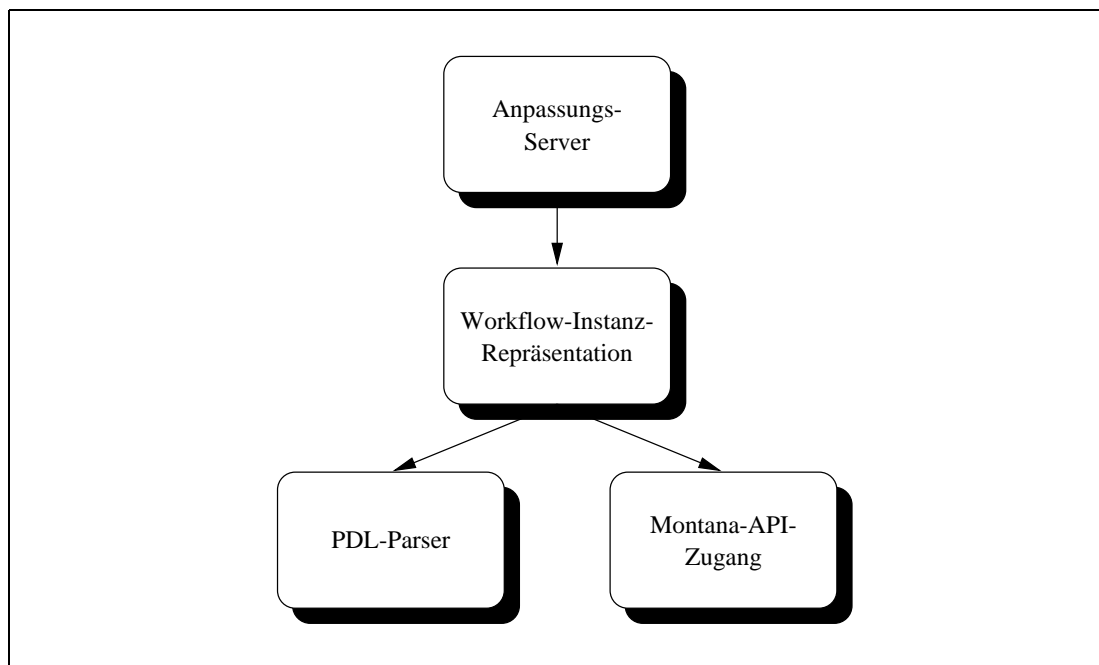


Abbildung 4.3: Die Subsysteme des Anpassungsdienstes

den hinzugefügt, die schon die nötige Konvertierung zwischen den Strukturen des BOW-CORBA-Gateways und denen des Workflow-Instanz-Objektes leisten.

PDL-Parser

Auf Anfragen bezüglich der Beschreibung einer Workflow-Instanz liefert Montana diese in PDL, seiner Workflow-Beschreibungssprache, zurück. Um die gewünschten Informationen aus PDL-Code zu gewinnen, muß dieser geparkt werden. Die Funktionalität hierzu steckt im Subsystem PDL-Parser. Dieses Subsystem wird mit Hilfe einer PDL-Grammatik-Spezifikation und einem Parsergenerator für Java implementiert (näheres in Kapitel 5).

Workflow-Instanz-Repräsentation

Eine ganze Gruppe von Klassen dient der Repräsentation von Workflow-Instanzen, dem Gegenstand von Anpassungen. Die zum Aufbau nötigen Informationen werden mit Hilfe der Funktionalität der Subsysteme Montana-API-Zugang und PDL-Parser ermittelt (siehe Abhängigkeitspfeile in Abbildung 4.3).

Das Subsystem Workflow-Instanz-Repräsentation stellt die verschiedenen Komponenten einer Workflow-Instanz dar: Workflow-Beschreibung, Zustand der Instanz als Kombination der Zustände aller Instanzteile, sowie die Workflow-relevanten Daten. Die Repräsentation der Workflow-Beschreibung ist in zwei Ebe-

nen aufgebaut:

Die Objekte der *PDL-Ebene* korrelieren direkt mit dem beschreibenden PDL-Code. Aus diesen Objekten kann nach einer Anpassung wieder PDL-Code generiert werden, um die Anpassung wirksam zu machen. Die Objekte der *strukturierten Ebene* werden aus der PDL-Ebene abgeleitet und repräsentieren die Struktur der strukturierten Modellierung.

Anpassungs-Server

Das Subsystem Anpassungs-Server realisiert die angebotenen Anpassungen mit Hilfe der Workflow-Instanz-Repräsentation und stellt diese an der in Abschnitt 4.5 beschriebenen Schnittstelle zur Verfügung.

4.3 Das BOW-CORBA-Gateway

Das BOW-CORBA-Gateway stellt die vom Anpassungsdienst benötigte Montana-API-Funktionen in Form von CORBA-Methoden bereit und ermöglicht dem in Java implementierten Anpassungsdienst auf diese Weise den Zugang zur Montana-Engine.

Im folgenden sollen die aus Entwurfssicht bereitzustellenden Montana-Dienste aufgezählt und kurz unter Angabe ihrer Argumente und Resultate erläutert werden. [HP96a] dokumentiert die Dienste.

4.3.1 Verzeichnisdienste

GroupDefnQuery: Welche Workflow-Gruppen gibt es?

Argumente: groupName, groupVersion

Resultate: returnStatus, groupDefnQueryReplyInfo

Die Argumente schränken den Suchraum ein. Die Resultate informieren über Erfolg oder Mißerfolg der Operation (returnStatus) und liefern das Anfrageergebnis (groupDefnQueryReplyInfo).

ProcDefnQuery: Welche Workflow-Beschreibungen gibt es?

Argumente: groupName, groupVersion, procName, procVersion

Resultate: returnStatus, procDefnQueryReplyInfo

Argumente/Resultate siehe GroupDefnQuery.

ActivityDefnQuery: Welche Aktivitätenbeschreibungen gibt es?

Argumente: activityName, activityVersion

Resultate: returnStatus, activityDefnQueryReplyInfo

Argumente/Resultate siehe GroupDefnQuery.

ProcInstQuery: Welche Workflow-Instanzen existieren momentan?

Argumente: groupName, groupVersion, procName, procVersion, activity-
Name, activityVersion

Resultate: returnStatus, procInstQueryReplyInfo

Argumente/Resultate siehe GroupDefnQuery.

4.3.2 Zugriff auf die Workflow-Beschreibung

GroupDefnGet: Lesender Zugriff auf eine Gruppenbeschreibung

Argumente: groupName, groupVersion

Resultate: returnStatus, groupPdlSrcCode

Die Argumente benennen die gewünschte Gruppenbeschreibung. Die Resultate informieren über Erfolg bzw. Mißerfolg der Operation und liefern die gewünschte Beschreibung in PDL zurück.

ProcDefnGet: Lesender Zugriff auf eine Workflow-Beschreibung

Argumente: groupName, groupVersion, procName, procVersion

Resultate: returnStatus, procPdlSrcCode

Argumente/Resultate siehe GroupDefnGet.

ActivityDefnGet: Lesender Zugriff auf eine Aktivitätenbeschreibung

Argumente: activityName, activityVersion

Resultate: returnStatus, activityPdlSrcCode

Argumente/Resultate siehe GroupDefnGet.

PdlAdd: Hinzufügen neuer Workflow-Beschreibungsteile

Argument: pdlString

Resultate: returnStatus, pdlcErrMsg

Das Argument `pdlString` enthält einen Workflow-Beschreibungsteil. Was beschrieben wird (z.B. Aktivität oder Workflow), ist in `pdlString` spezifiziert und muß daher nicht in Form gesonderter Argumente angegeben werden. Die Resultate informieren über Erfolg bzw. Mißerfolg der Operation (`returnStatus`) und liefern bei Mißerfolg Fehlermeldungen (`pdlcErrMsg`).

4.3.3 Zugriff auf Zustandsinformationen und Workflow-relevante Daten

ProcInstStatusGet: Lesen von Statusinformation und Workflow-relevanten Daten einer Workflow-Instanz

Argument: `procInstID`

Resultate: `returnStatus`, `workNodesInfo`, `ruleNodesInfo`, `activityInfo`, `procStartTime`, `procEndTime`, `procDeadline`, `latestCasePacketContents`

Das Argument `procInstID` identifiziert die Workflow-Instanz. Die Resultate informieren über Erfolg bzw. Mißerfolg der Operation (`returnStatus`) und liefern Statusinformationen der verschiedenen Knoten und Aktivitäten der Workflow-Instanz sowie Start-/Endzeitpunkt, Abschlußtermin und die aktuelle Belegung der Workflow-relevanten Daten.

ProcInstStatusSet: Schreiben von Workflow-relevanten Daten

Argumente: `procInstID`, `typeNameValueTripleList`

Resultat: `returnStatus`

Es werden Workflow-relevante Daten der durch `procInstID` identifizierten Workflow-Instanz gemäß den Angaben über Typ, Name und Wert in `typeNameValueTripleList` gesetzt.

ProcInstGetState: Lesen des Ausführungsstatus einer Workflow-Instanz

Argument: `procInstID`

Resultate: `returnStatus`, `procInstState`

Der Ausführungsstatus der durch `procInstID` identifizierten Workflow-Instanz wird in `procInstState` zurückgeliefert.

ProcInstChangeState: Ändern des Ausführungsstatus einer Workflow-Instanz

Argumente: `procInstID`, `procInstState`

Resultat: `returnStatus`

Der Ausführungsstatus der durch `procInstID` identifizierten Workflow-Instanz wird auf `procInstState` gesetzt.

NodeInstGetState: Lesen des Ausführungsstatus einer Knoteninstanz

Argumente: `procInstID`, `nodeInstID`, `nodeType`

Resultate: `returnStatus`, `nodeInstState`

Der Ausführungsstatus der durch `nodeInstID` und `nodeType` identifizierten Knoteninstanz innerhalb der mit `procInstID` bezeichneten Workflow-Instanz wird in `nodeInstState` zurückgeliefert.

NodeInstChangeState: Ändern des Ausführungsstatus einer Knoteninstanz

Argumente: `procInstID`, `nodeInstID`, `nodeType`, `nodeInstState`

Resultat: `returnStatus`

Der Ausführungsstatus der durch `nodeInstID` und `nodeType` identifizierten Knoteninstanz innerhalb der mit `procInstID` bezeichneten Workflow-Instanz wird auf `nodeInstState` gesetzt.

NodeInstRestart: Wiederholte Ausführung einer Knoteninstanz

Argumente: `procInstID`, `nodeInstID`, `nodeType`

Resultat: `returnStatus`

Die durch `nodeInstID` und `nodeType` identifizierte Knoteninstanz innerhalb der mit `procInstID` bezeichneten Workflow-Instanz wird von der Montana-Engine wiederholt ausgeführt.

4.3.4 Verwendete Datentypen

Montana stellt in seiner Entwicklungsumgebung eigene C++-Klassen für die intern verwendeten Datentypen (beispielsweise Workflow-relevanter Daten) bereit. Die meisten dieser Datentypen werden im Anpassungsdienst als Strings repräsentiert, da sie in dieser Form von der Benutzerschnittstelle am besten zu verarbeiten sind. Einige müssen in die Darstellung der Java-Klassen des Subsystems `Workflow-Instanz-Repräsentation` überführt werden. Ein Beispiel hierfür ist die Java-Klasse `PDLNameValueList` (siehe Abschnitt 4.4.2) zur Darstellung der Workflow-relevanten Daten einer Workflow-Instanz.

Die Umwandlung von Strings in diverse Montana-Typen (z.B. `Version` oder `ObjectID`) und zurück wird innerhalb des BOW-CORBA-Gateway geleistet, so daß die entsprechenden Argumente und Resultate an der Schnittstelle als Strings definiert sind.

4.4 Die Workflow-Instanz-Repräsentation

In diesem Abschnitt wird der Entwurf eines Teils der wichtigsten Klassenhierarchien und Objektbeziehungen des Anpassungsdienstes dargestellt. Es handelt sich um die Klassen des Subsystems Workflow-Instanz-Repräsentation.

Die Workflow-Instanz-Repräsentation erfordert geeignete Klassen zur Objektdarstellung der einzelnen Bestandteile einer Workflow-Instanz. Die Darstellung der in PDL vorliegenden Workflow-Beschreibung erfolgt in Klassen, die sich an der von PDL vorgegebenen Struktur orientieren. Die Klassen zur Repräsentation verschiedener Laufzeitinformationen wie Workflow-relevanter Daten und Zustandsinformationen sind in ähnlicher Weise ihrer Struktur nach eng an die Gegebenheiten von Montana angelehnt. Die Objektrepräsentation der strukturierten Ebene der Kontrollflußbeschreibung orientiert sich an der in Abschnitt 2.4 eingeführten strukturierten Modellierung, deren Elemente durch verschiedene Klassen repräsentiert werden.

4.4.1 Repräsentation der PDL-Ebene

Aufgabe der hier beschriebenen Klassen ist es, die verschiedenen in der PDL-Workflow-Beschreibung definierten Objekte zu repräsentieren. Um diese Aufgabe zu erfüllen, stellen die entworfenen Klassen im Grunde ein direktes Abbild der in PDL gegebenen Struktur dar. Die Felder der einzelnen Klassen repräsentieren zumeist ein Attribut des dargestellten PDL-Objekts. Oft ist dies in Form von Strings möglich. In einigen wenigen Fällen sind komplexere Klassen (z.B. `PDLNameValueList`) zur Repräsentation eines PDL-Attributs nötig.

Im folgenden werden die Klassenhierarchien und -beziehungen beschrieben und in mehreren Abbildungen illustriert. Die Klassen umfassen nicht die ganze Mächtigkeit von PDL sondern nur die in der vorliegenden Arbeit verwendete Untermenge.

Knoten

Der Kontrollfluß wird in PDL durch Knoten dargestellt, die durch Kanten miteinander verbunden sind. Für die vorliegende Diplomarbeit wurden aus Montana's Knotenarten drei ausgewählt: `WorkNodes`, `RouteNodes` und `HyperNodes`.

`WorkNodes` sind Knoten, die die Ausführung einer Aktivität oder eines Subworkflows im Kontrollfluß vermerken.

`RouteNodes` dienen der Verzweigung oder Synchronisation. Mit Hilfe von Regeln (`RouteRules`) in den `RouteNodes` wird deren Verhalten in Abhängigkeit von Workflow-relevanten und Workflow-Kontrolldaten spezifiziert.

`HyperNodes` spielen eine besondere Rolle: Sie dienen nicht der Kontrollflußdar-

stellung, sondern können beispielsweise von Modellierwerkzeugen verwendet werden, um zusätzliche, vom Workflow-Ausführungsdienst nicht berücksichtigte Information in der Workflow-Beschreibung unterzubringen. In der vorliegenden Arbeit werden die Strukturmerkmale eines Workflows in HyperNodes abgelegt, da Montana selbst sich nicht der strukturierten Modellierung bedient. Ein HyperNode macht somit Aussagen über den Typ eines Strukturierungselementes und die zu seiner Realisierung benutzten Knoten und Kanten, sowie deren Funktion.

Abbildung 4.4 stellt die Klassen zur Repräsentation von Knoten, sowie die Hierarchie- und Enthaltenseinsbeziehungen zwischen den Klassen dar.

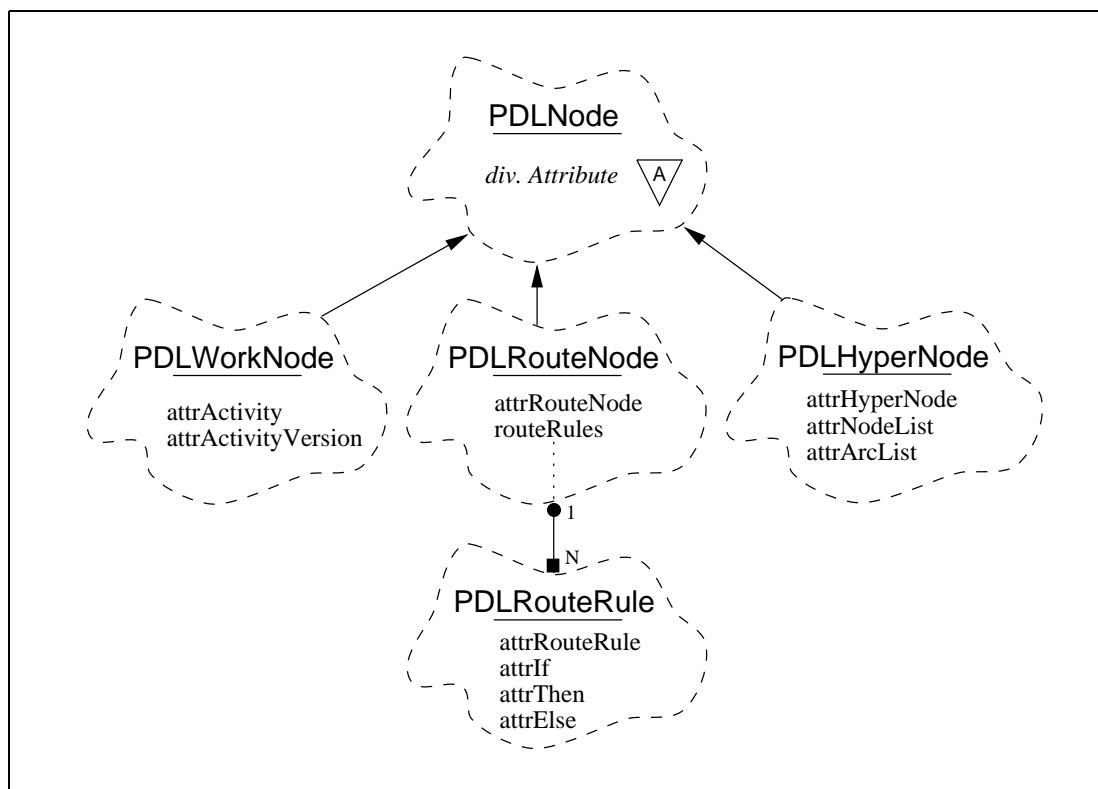


Abbildung 4.4: Klassen zur Repräsentation von PDL-Knoten

Die abstrakte Klasse `PDLNode` faßt die allen Knotenarten gemeinsamen Eigenschaften zusammen und vererbt diese an die Subklassen `PDLWorkNode`, `PDLRouteNode` und `PDLHyperNode`. Die Klasse `PDLRouteRule` dient der Darstellung von `RouteRules`, von denen mehrere in einem `PDLRouteNode` zu dessen Steuerung enthalten sein können.

Kanten

Die notwendigen Verbindungen, die den Kontrollfluß von Knoten zu Knoten weiterleiten, werden in PDL mittels gerichteter Kanten dargestellt. Zur Repräsentation

tion dieser Kanten dient die in Abbildung 4.5 gezeigte Klasse `PDLArc`.

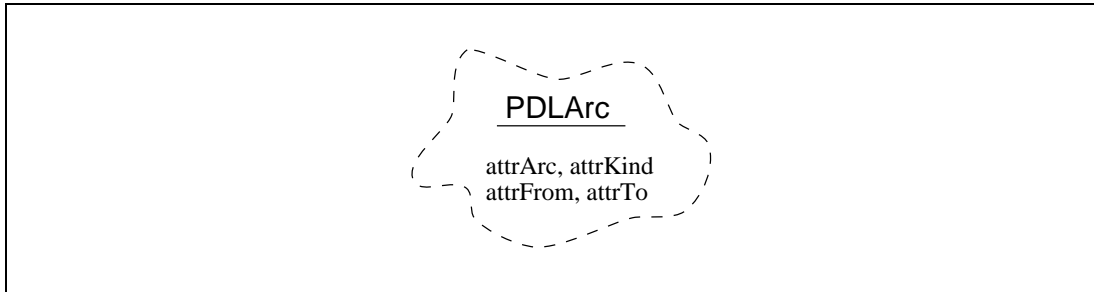


Abbildung 4.5: Klasse zur Repräsentation von PDL-Kanten

Die Felder `attrFrom` und `attrTo` enthalten die Namen des Knotens, von dem die Kante ausgeht, und des Zielknotens der Kante. `attrKind` dient der Unterscheidung zwischen Startkanten des Workflows, deren `attrFrom`-Feld unbesetzt ist, und normalen weiterleitenden Kanten. In `attrArc` ist der Name der Kante abgelegt.

Das Template

Zur Beschreibung der Variablen, die zur Aufnahme der Workflow-relevanten Daten eines Workflows verwendet werden sollen, dient in PDL das Template. Templates werden in der vorliegenden Arbeit durch Objekte der Klasse `PDLTemplate` repräsentiert. Abbildung 4.6 stellt diese und zu ihrem Aufbau notwendige weitere Klassen dar.

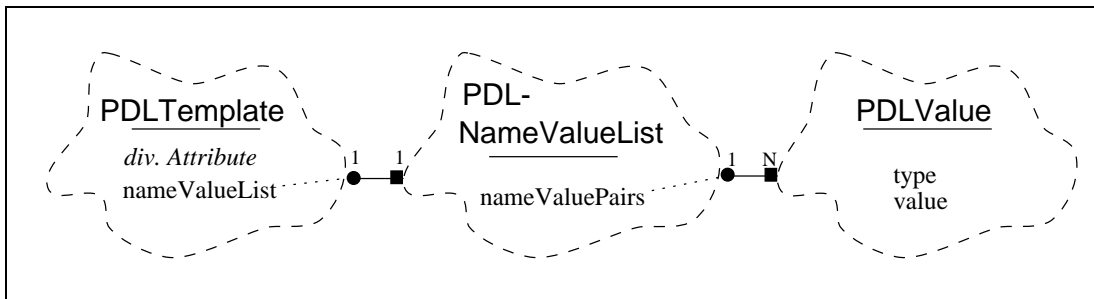


Abbildung 4.6: Klassen zur Repräsentation des Templates

Das Feld der Klasse `PDLTemplate`, das hier genauer betrachtet werden soll, ist `nameValueList`. Es beschreibt in Form einer Liste von Name-Wert-Paaren die Variablen für die Workflow-relevanten Daten, d.h. deren jeweiligen Namen und Initialisierungswert. `nameValueList` enthält ein Objekt der Klasse `PDLNameValueList`, die der Darstellung solcher Listen von Name-Wert-Paaren dient. `PDLNameValueList` hält die Liste in der Hashtable `nameValuePairs`, die ein Auffinden eines Wertes anhand des Namens ermöglicht. Jeder einzelne Wert wird durch eine Instanz der Klasse `PDLValue` repräsentiert. Diese Klasse hält die Information

über den Typ des Wertes sowie den Wert selbst. Die möglichen Typen sollen hier nicht beschrieben werden, sondern können [HP96a] sowie der Implementierung von PDLValue entnommen werden.

4.4.2 Repräsentation von Laufzeitinformationen

Die gesamte ermittelbare Laufzeitinformation einer Workflow-Instanz kann mit der Montana-API-Funktion ProcInstStatusGet abgefragt werden. Zur Repräsentation dieser Daten wurden die in Abbildung 4.7 dargestellten Klassen entworfen.

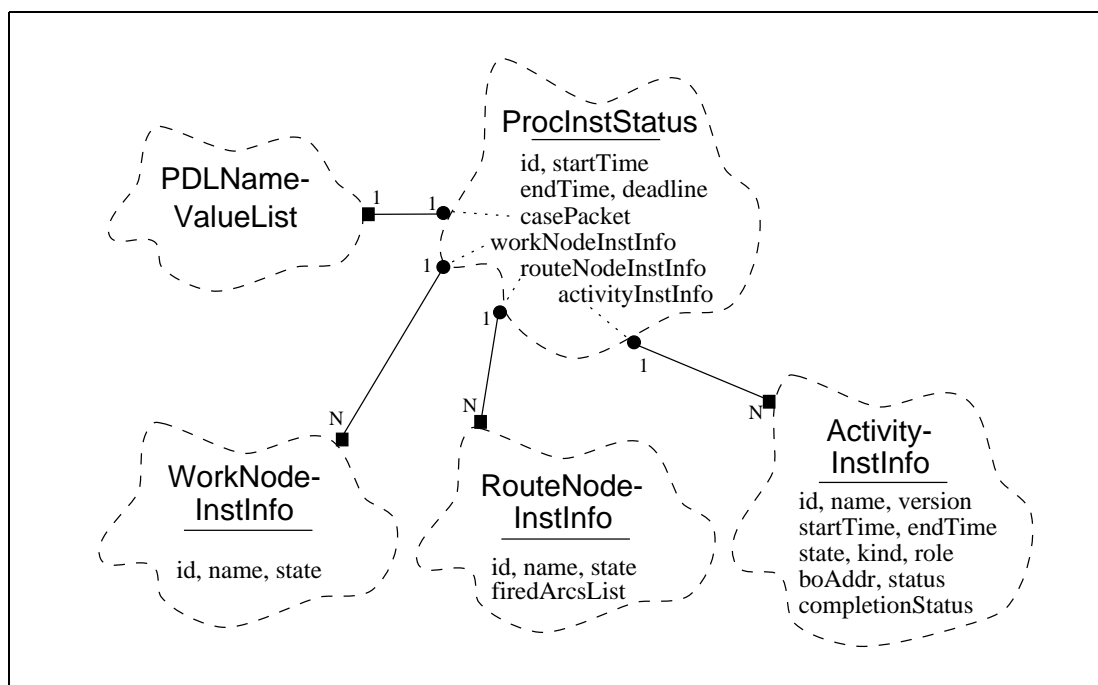


Abbildung 4.7: Klassen zur Repräsentation von Laufzeitinformationen

Die Gesamtheit der Laufzeitinformationen einer Workflow-Instanz wird demnach in einem Objekt der Klasse ProclnstStatus dargestellt. Dieses enthält die Instanz-Bezeichnung `id`, den Startzeitpunkt der Instanz `startTime` sowie den Zeitpunkt des Bearbeitungsabschlusses `endTime`, wenn die Instanz schon beendet wurde, und den vorgegebenen spätesten Beendigungstermin `deadline`. Das Feld `casePacket` enthält in einem Objekt der Klasse PDLNameValueList die Belegung der Workflow-relevanten Daten. Die Felder `workNodeInstInfo`, `routeNodeInstInfo` und `activityInstInfo` repräsentieren die Laufzeitinformationen der zum Workflow gehörigen Knoten- und Aktivitäteninstanzen. Hierbei enthält jeweils ein Objekt der Klassen WorkNodeInstInfo, RouteNodeInstInfo bzw. ActivityInstInfo die Daten einer WorkNode-, RouteNode- bzw. Aktivitäteninstanz. Die Felder dieser drei Klassen sollen hier nicht weiter erläutert werden.

4.4.3 Repräsentation der strukturierten Ebene

Zur Kontrollflußbeschreibung bedient sich die vorliegende Diplomarbeit der strukturierten Modellierung. Die strukturierte Darstellungsebene liegt über der PDL-Ebene; der Kontrollfluß einer Workflow-Instanz wird dem Benutzer des Anpassungsdienstes also in seiner strukturierten Modellierung präsentiert, und auf die Elemente derselben beziehen sich auch die Parameter der verschiedenen kontrollflußbezogenen Anpassungsmethoden.

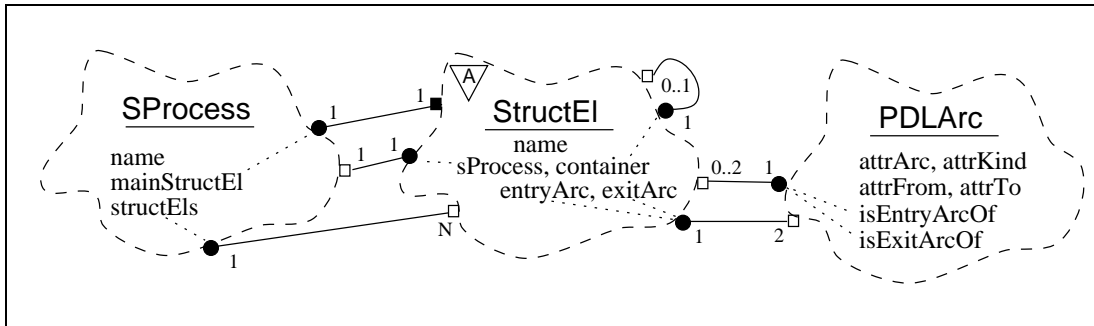


Abbildung 4.8: Klassen zur Repräsentation der strukturierten Ebene

Abbildung 4.8 zeigt die Klasse **StructEl**, die die allen Strukturierungselementen gemeinsamen Eigenschaften auf sich vereinigt. **StructEl** ist eine abstrakte Klasse; sie dient nur dazu, die gemeinsamen Eigenschaften an ihre Subklassen zu vererben. Die Eigenschaft, daß ein Strukturierungselement nur einen Eingang und einen Ausgang hat, ist durch die Felder **entryArc** und **exitArc** abgebildet, die auf die Ein- und Ausgangskante verweisen.

Ein- und Ausgangskanten werden durch PDL-Kanten repräsentiert. Die Klasse **PDLArc** enthält die in Abbildung 4.5 noch nicht dargestellten Felder **isEntryArcOf** und **isExitArcOf** als Rückverweise auf das Strukturierungselement, deren Ein- bzw. Ausgangskante die entsprechende PDL-Kante ist.

Das **container**-Feld eines **StructEl**-Objekts kann auf ein übergeordnetes Strukturierungselement verweisen. Die Elemente von Sequenzen, Parallelkonstrukten und bedingten Verzweigungen verweisen mit diesem Feld auf das jeweilige umgebende Konstrukt.

Ein Objekt der Klasse **SProcess** umfaßt sämtliche Informationen zu einer Workflow-Instanz. Das Feld **structEls** enthält eine Hashtable, über die alle zum Workflow gehörigen **StructEl**-Objekte abhängig von ihrem Namen gefunden werden können. In **mainStructEl** ist das „oberste“ Strukturierungselement abgelegt, also das Strukturierungselement, das rekursiv alle anderen umfaßt. Das **sProcess**-Feld eines **StructEl**-Objekts *SE* verweist auf das **SProcess**-Objekt, in dem *SE* enthalten ist.

Abbildung 4.9 zeigt die von **StructEl** abgeleiteten Klassen zur Darstellung verschiedener Arten von Strukturierungselementen.

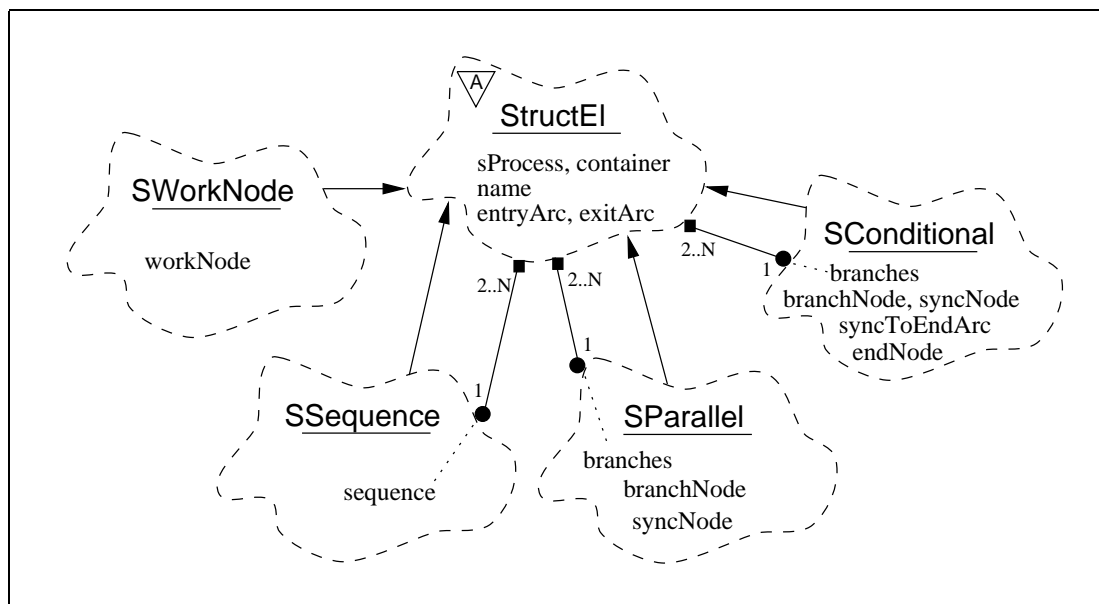


Abbildung 4.9: Klassen zur Repräsentation der Strukturierungselemente

Im folgenden werden die **StructEl**-Subklassen einzeln beschrieben:

Die Klasse **SWorkNode** stellt die Arbeitseinheit dar. Sie enthält in ihrem Feld **workNode** einen Verweis auf ein **PDLWorkNode**-Objekt, durch das das Verhalten der Arbeitseinheit beschrieben wird.

Sequenzen werden durch Objekte der Klasse **SSequence** repräsentiert. Deren Feld **sequence** enthält – geordnet entsprechend der gewünschten Ausführungsreihenfolge – die Elemente der Sequenz, von denen es mindestens zwei geben muß („2..N“ in der Abbildung). Diese Elemente sind wiederum **StructEl**-Objekte.

Die Klasse **SParallel** dient der Darstellung von Parallelkonstrukten. In ihrem Feld **branches** sind die parallel auszuführenden Strukturierungselemente enthalten. Zur Verzweigung vor und Synchronisation nach der Parallelausführung werden die beiden **PDLRouteNode**-Objekte **branchNode** und **syncNode** verwendet.

Schließlich repräsentieren Objekte der Klasse **SConditional** Konstrukte zur bedingten Verzweigung. Die zur Auswahl bereitstehenden Pfade sind im Feld **branches** abgelegt. Die Verzweigung zu den Pfaden und die Synchronisation nach deren Ausführung werden durch die **PDLRouteNode**-Objekte **branchNode** und **syncNode** realisiert. Das Verzweigungs- und Synchronisationsverhalten einer bedingten Verzweigung soll in dieser Arbeit durch den modellierenden oder anpassenden Benutzer äußerst frei spezifiziert werden können, indem ihm volle Freiheit über die Formulierung der **RouteRules** in **branchNode** und **syncNode** gewährt wird. Um aber die Eigenschaft „Strukturierungselement **einmal** durch den Eingang betreten \Rightarrow genau **einmal** durch den Ausgang verlassen“ weiter zu gewährleisten, wird der Kontrollfluß innerhalb einer bedingten Verzweigung über die **PDL**-Kante

`syncToEndArc` an einen abschließenden `PDLRouteNode` `endNode` weitergeleitet, der die Eigenschaft durch seine Regeln sichert, indem er nur die erste Aktivierung von `syncToEndArc` an die Ausgangskante der bedingten Verzweigung weiterleitet und evtl. auftretende weitere Aktivierungen „verschluckt“. Der Inhalt der Regeln im `endNode` kann vom Benutzer nicht beeinflusst werden, sondern er wird entsprechend der Aufgabe dieses Knotens festgelegt.

4.4.4 Integration der einzelnen Bereiche

Die in den vorigen Unterabschnitten beschriebenen Teilbereiche der Workflow-Instanz-Repräsentation werden so zusammengefaßt, daß eine Workflow-Instanz durch ein Objekt und dessen Bestandteile dargestellt wird. Zu diesem Zweck dienen die Klasse `PDLProcess` und deren Subklasse `SProcess`. Die Integration durch diese beiden Klassen wird in Abbildung 4.10 illustriert.

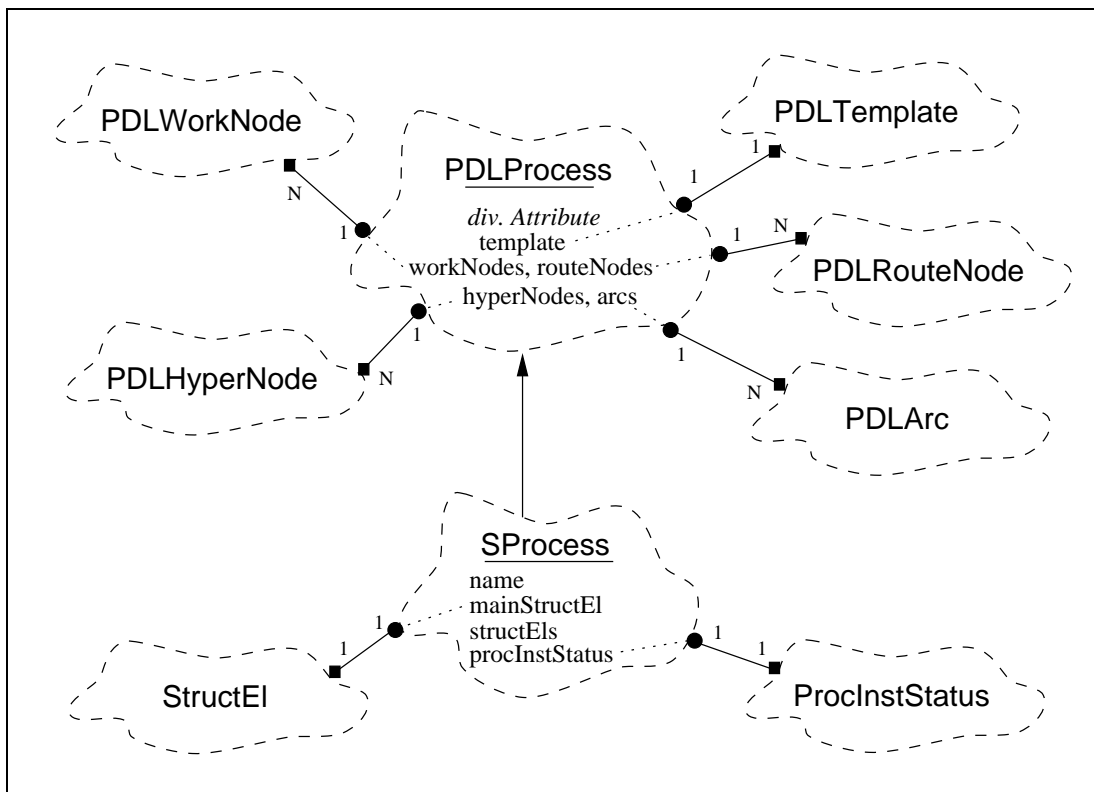


Abbildung 4.10: Gesamtzusammenhang der Klassen zur Workflow-Instanz-Repräsentation

Die Objekte, die die PDL-Ebene der Workflow-Instanz beschreiben, werden in den Feldern der Klasse `PDLProcess` abgelegt. Es sind dies die Objekte zur Repräsentation der verschiedenen Knoten und Kanten sowie das Template der Workflow-Beschreibung. Die Klasse `SProcess` erbt diese Felder und die zugehörigen Me-

thoden und erweitert ihre Superklasse um die Informationen der strukturierten Ebene sowie die Laufzeitinformationen der Workflow-Instanz.

4.5 Der Anpassungs-Server

Der Anpassungs-Server ist das Subsystem, das durch den Zugriff auf die Objekte der im vorigen Abschnitt 4.4 erläuterten Workflow-Instanz-Repräsentation die vom Anpassungsdienst angebotenen Anpassungen realisiert und die Schnittstelle des Anpassungsdienstes definiert, an der die Anpassungen dem Benutzer bereitgestellt werden.

4.5.1 Die objektorientierte Schnittstelle

Ein Programm, das den Anpassungsdienst nutzt, erzeugt zu diesem Zweck ein „Anpassungsobjekt“, ein Objekt der Klasse **Modification**. Abbildung 4.11 stellt diese Klasse dar.

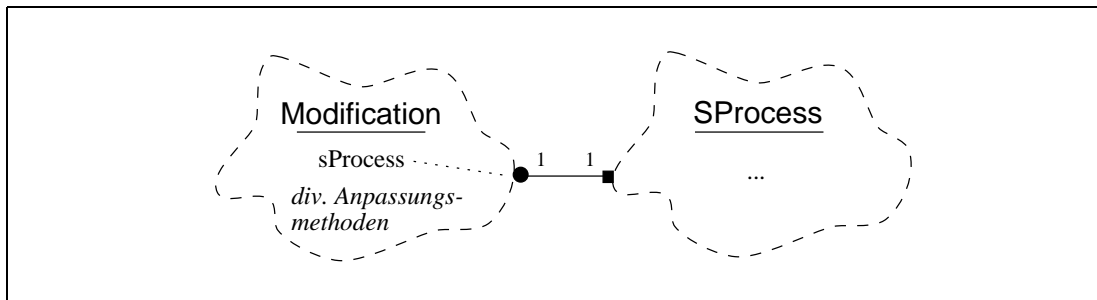


Abbildung 4.11: Die objektorientierte Schnittstelle des Anpassungsdienstes

Ein Objekt der Klasse **Modification** enthält in seinem Feld **sProcess** die Repräsentation der Workflow-Instanz, an der der Benutzer Anpassungen durchführen will. Durch Aufruf der diversen Anpassungsmethoden und Versorgung derselben mit Parametern werden Anpassungen spezifiziert und ihre Durchführung veranlaßt. Auf die Workflow-Instanz-Repräsentation in **sProcess** kann nur mit Hilfe der Methoden des **Modification**-Objekts verändernd zugegriffen werden, wodurch die Kontrolle der Veränderungen gänzlich beim Anpassungsdienst bleibt. Zum lesen Zugriff werden daher generell nur *Kopien* der Objektstruktur unter **sProcess** zur Verfügung gestellt, niemals Verweise auf die Originale.

Die durch eine Anpassung stattfindenden Veränderungen werden zuerst an der Objektdarstellung der Workflow-Instanz innerhalb des **Modification**-Objekts vollzogen und dann auf die wirkliche Workflow-Instanz innerhalb der Montana-Engine übertragen.

Bereitstellung des Anpassungsdienstes

Im Rahmen verteilter Umgebungen und Interprozeßkommunikation gibt es für die vorliegende Arbeit generell zwei Möglichkeiten für die Bereitstellung des Anpassungsdienstes seinen Dienstnutzern gegenüber, die im folgenden kurz diskutiert werden sollen.

1. Verwendung eines lokalen Anpassungsobjektes

Anpassungsdienstnutzer wie beispielsweise der Workflow-Editor als Benutzerschnittstelle zum Anpassungsdienst legen ein Anpassungsobjekt in einem ihrer Felder ab und können dann direkt dessen Methoden aufrufen. Anpassungsdienst und Dienstnutzer laufen so im selben Java-Prozeß ab.

2. Verwendung eines entfernten Anpassungsobjektes

Programme, die den Anpassungsdienst nutzen, erzeugen ein entferntes Anpassungsobjekt und arbeiten mit diesem zusammen. Mögliche Technologien zur Realisierung der Zusammenarbeit sind:

- Java Remote Method Invocation (RMI)

In der nächsten Version (Release 1.1) von Sun's Java Development Kit wird das Package `java.rmi` die Bereitstellung und Verwendung entfernter Objekte ermöglichen [Sun96RMI]. Diese Technologie wird bei der Programmierung mit Java sehr einfach zu nutzen sein, dient jedoch nur der Kommunikation zwischen Java-Objekten und umfaßt nicht die Anbindung von Objekten, die in anderen Programmiersprachen realisiert werden.

- CORBA

Die vollständig heterogene und Programmiersprachen-unabhängige Lösung ist eine Bereitstellung des Anpassungsdienstes als CORBA-Objekt.

Die in dieser Arbeit verwendete CORBA-Implementierung OrbixWeb [IONA95] ermöglicht in der Version 1.0 noch nicht die Realisierung von CORBA-Objektservern in Java. In Version 2.0 wird dies jedoch möglich sein.

Da die Implementierung der vorliegende Diplomarbeit prototypischen Charakter hat, wird auf die Bereitstellung des Anpassungsdienstes in Form eines entfernten Anpassungsobjektes verzichtet und anstatt dessen die lokale Variante gewählt. Insbesondere mit der Java-RMI-Technologie ist jedoch eine spätere Erweiterung der Implementierung leicht möglich.

Erzeugung eines Anpassungsobjektes

Bei der Erzeugung eines Anpassungsobjektes wird bereits angegeben, an welcher Workflow-Instanz Anpassungen vorgenommen werden sollen. Hieraus ergibt sich folgender Konstruktor für die Klasse `Modification`:

```
public Modification(String groupName, String groupVersion,
                   String procName, String procVersion,
                   String procInstId)
```

Modification-Konstruktor	
<i>Parameter:</i>	
<code>groupName</code>	– Name der Gruppe der Workflow-Beschreibung
<code>groupVersion</code>	– Version — " —
<code>procName</code>	– Name der Workflow-Beschreibung
<code>procVersion</code>	– Version — " —
<code>procInstId</code>	– Bezeichner der anzupassenden Workflow-Instanz

Die Parameter `groupName`, `groupVersion`, `procName` und `procVersion` sind zusätzlich zu `procInstId` erforderlich, damit die Workflow-Beschreibung mittels `ProcDefnGet` von der Montana-Engine ausgelesen werden kann.

Die Methoden eines so konstruierten `Modification`-Objektes realisieren die verschiedenen Arten von Anpassungen (Abschnitt 4.5.2) und stellen zusätzlich noch diverse Hilfsdienste (Abschnitt 4.5.3) z.B. zum lesenden Zugriff auf Beschreibung und Laufzeitinformationen der Workflow-Instanz im Anpassungsobjekt zur Verfügung.

4.5.2 Anpassungsmethoden

In diesem Unterabschnitt werden die entworfenen Schnittstellen der Anpassungsmethoden angegeben. Die Anpassungsmethoden realisieren die in Kapitel 3 beschriebenen Anpassungen und halten sich eng an deren Spezifikation.

Aus der Beschreibung der bedingten Verzweigung in Abschnitt 4.4.3 wird jedoch schon ein Unterschied zur Spezifikation deutlich: Wenn dort von Verzweigungsregeln die Rede war, so muß hier immer die Kombination von Verzweigungs- und Synchronisationsregeln betrachtet werden. Sonstige kleine Veränderungen gehen aus der Beschreibung der jeweiligen Anpassungsmethode hervor.

Anpassungsmethoden sind Methoden der Klasse `Modification`. Da schon bei der Erzeugung eines `Modification`-Objekts festgelegt wird, welche Workflow-Instanz

RemoveBranchFromCondition	
<i>Parameter:</i>	
branchName	– Name des Strukturierungselementes, das den zu entfernenden Pfad repräsentiert
branchRouteRules	– modifizierte Verzweigungsregeln (Vektor von PDLRouteRule-Objekten)
syncRouteRules	– modifizierte Synchronisationsregeln (Vektor von PDLRouteRule-Objekten)

Sequentielles Einfügen einer Arbeitseinheit

In der Spezifikation wurden für das sequentielle Einfügen die Angaben zweier Parameter verlangt, bezüglich welchem Strukturierungselement und ob *vor* oder *nach* diesem eingefügt werden soll. Diese Ortsangabe des Einfügens läßt sich auch durch einen Parameter ausdrücken, indem man nämlich eine bestehende Kante (von den Ein-/Ausgangskanten der Strukturierungselemente) angibt. Diese wird dann ‘aufgetrennt’ und in die ‘Lücke’ wird das neue Element eingefügt.

Die Schnittstelle der entsprechenden Anpassungsmethode sieht folgendermaßen aus:

```
public Return InsertInSequence(String arcName,
                               PDLWorkNode workNode,
                               String newEntryArcName)
```

InsertInSequence	
<i>Parameter:</i>	
arcName	– Name der Kante, „in“ die eingefügt werden soll
workNode	– PDL-WorkNode der einzufügenden Arbeitseinheit
newEntryArcName	– Name der neu zu erzeugenden Eingangskante der durch die Anpassung eingefügten Arbeitseinheit

Der Name der eingefügten neuen Arbeitseinheit wird durch die Methode vom Namen des PDL-WorkNode `workNode` abgeleitet.

Paralleles Einfügen einer Arbeitseinheit

Die im folgenden beschriebene Methode realisiert das parallele Einfügen:

```
public Return InsertInParallel(String refSEName,
```

```
PDLWorkNode workNode,
String sParallelName)
```

InsertInParallel	
<i>Parameter:</i>	
refSEName	– Name des Strukturierungselementes, zu dem parallel eingefügt werden soll
workNode	– PDL-WorkNode der einzufügenden Arbeitseinheit
sParallelName	– Name des neu entstehenden Parallelkonstrukts

Wenn **refSEName** einen Parallelzweig oder eine Parallelstruktur bezeichnet, so wird ein neuer Zweig in die bereits bestehende Parallelstruktur eingefügt. In diesem Fall hat der Parameter **sParallelName** keine Bedeutung. Ansonsten wird eine neue Parallelstruktur mit dem Namen **sParallelName** erzeugt, die dann das mit **refSEName** bezeichnete Strukturierungselement und die neue Arbeitseinheit als Pfade enthält.

Alternatives Einfügen einer Arbeitseinheit

Die Schnittstelle der Methode zum alternativen Einfügen:

```
public Return InsertAlternatively(String refSEName,
                                PDLWorkNode workNode,
                                String sConditionName,
                                Vector branchRouteRules,
                                Vector syncRouteRules)
```

InsertAlternatively	
<i>Parameter:</i>	
refSEName	– Name des Strukturierungselementes, zu dem alternativ eingefügt werden soll
workNode	– PDL-WorkNode der einzufügenden Arbeitseinheit
sConditionName	– Name der neu entstehenden bedingten Verzweigung
branchRouteRules	– Verzweigungsregeln der neu entstehenden bedingten Verzweigung
syncRouteRules	– Synchronisationsregeln der neu entstehenden bedingten Verzweigung

Eine Sonderbehandlung abhängig vom Kontext, in den eingefügt werden soll, gibt es beim alternativen im Gegensatz zum parallelen Einfügen nicht. Die Schachtelung von Alternativen kann durchaus erwünscht sein.

Einfügen in eine bedingte Verzweigung

Das Einfügen eines neuen Pfades in eine bestehende bedingte Verzweigung wird mit dem oben beschriebenen alternativen Einfügen nicht ermöglicht. Hierzu dient die folgende Anpassungsmethode:

```
public Return InsertConditionBranch(String refSConditionName,
                                   PDLWorkNode workNode,
                                   Vector branchRouteRules,
                                   Vector syncRouteRules)
```

InsertConditionBranch	
<i>Parameter:</i>	
<code>refSConditionName</code>	– Name der bedingten Verzweigung, in die eingefügt werden soll
<code>workNode</code>	– PDL-WorkNode der einzufügenden Arbeitseinheit
<code>branchRouteRules</code>	– modifizierte Verzweigungsregeln
<code>syncRouteRules</code>	– modifizierte Synchronisationsregeln

Regeln einer bedingten Verzweigung ändern

Die folgende Methode ermöglicht das Ändern der Regeln von bedingten Verzweigungen:

```
public Return SetConditionRules(String refSConditionName,
                                Vector branchRouteRules,
                                Vector syncRouteRules)
```

SetConditionRules	
<i>Parameter:</i>	
<code>refSConditionName</code>	– Name der bedingten Verzweigung, deren Regeln geändert werden sollen
<code>branchRouteRules</code>	– modifizierte Verzweigungsregeln
<code>syncRouteRules</code>	– modifizierte Synchronisationsregeln

Explizite Pfadauswahl in bedingten Verzweigungen

Die Schnittstelle zur expliziten Pfadauswahl enthält anstatt der Regeln in der vorigen Methode eine Namensliste der ausgewählten Pfade:

```
public Return ChooseBranches(String refSConditionName,
                             Vector branchNames)
```

ChooseBranches	
<i>Parameter:</i>	
refSConditionName	– Name der bedingten Verzweigung, für die eine explizite Pfadauswahl getroffen werden soll
branchNames	– ausgewählte Pfade, repräsentiert durch Namensliste der Strukturierungselemente (Vektor von String-Objekten)

Aktivität/Subworkflow scheitern lassen

Subworkflows werden in Montana als Aktivitäten beschrieben, die nicht eine Applikation außerhalb der Kontrollsphäre des Workflow-Ausführungsdienstes sondern eine Instanz eines Workflows durch den Ausführungsdienst abarbeiten lassen. Somit bezieht sich diese Anpassung immer auf Aktivitäteninstanzen und verlangt folgende Schnittstelle:

```
public Return EndFailedActivity(String actInstId)
```

EndFailedActivity	
<i>Parameter:</i>	
actInstId	– Bezeichner der Aktivitäteninstanz, die gescheitert beendet werden soll

Aktivität/Subworkflow wiederholen

Bezüglich Subworkflows gilt hier dasselbe wie für die zuvor beschriebene Anpassungsmethode `EndFailedActivity`. Zum Wiederholen von Aktivitäteninstanzen dient folgende Methode:

```
public Return RepeatActivity(String actInstId)
```

RepeatActivity	
<i>Parameter:</i>	
actInstId	– Bezeichner der Aktivitäteninstanz, die wiederholt werden soll

Arbeitseinheit vorziehen

Die folgende Anpassungsmethode veranlaßt die vorgezogene Ausführung einer Arbeitseinheit:

```
public Return PreStartSWorkNode(String sWorkNodeName)
```

PreStartSWorkNode
<i>Parameter:</i> sWorkNodeName – Name der Arbeitseinheit, die vorgezogen werden soll

Arbeitseinheit auslassen

Durch Aufruf der Methode SkipSWorkNode kann eine Arbeitseinheit ausgelassen werden:

```
public Return SkipSWorkNode(String sWorkNodeName)
```

SkipSWorkNode
<i>Parameter:</i> sWorkNodeName – Name der Arbeitseinheit, die ausgelassen werden soll

Zusatzaktivität starten

Zum Starten einer Zusatzaktivität dient die hier dargestellte Methode. Zur Synchronisation mit dem übrigen Workflowablauf sei auf die Spezifikation in Abschnitt 3.2.9 verwiesen.

```
public Return StartExtraActivity(String actName,
                                String actVersion,
                                int syncMode,
                                String syncSEName)
```

StartExtraActivity	
<i>Parameter:</i>	
<code>actName</code>	– Name der Aktivität, die zusätzlich gestartet werden soll
<code>actVersion</code>	– Version — ” —
<code>syncMode</code>	– Synchronisationsart gemäß Spezifikation
<code>syncSeName</code>	– Name eines Strukturierungselementes, mit deren Ende das Ende der Zusatzaktivität synchronisiert werden soll. Dieser Parameter hat nur bei der entsprechenden Synchronisationsart Bedeutung.

Rollenänderung

Die folgende Methode ermöglicht die Änderung der Rolle einer Aktivität:

```
public Return ChangeRole(String actName, String actVersion,
                        String role)
```

ChangeRole	
<i>Parameter:</i>	
<code>actName</code>	– Name der betroffenen Aktivität
<code>actVersion</code>	– Version — ” —
<code>role</code>	– einzutragende Rolle

Explizite Rollenauflösung

Die explizite Zuweisung einer Aktivität an einen bestimmten Bearbeiter geschieht durch die folgende Methode:

```
public Return ChooseActor(String actName, String actVersion,
                        String actor)
```

ChooseActor	
<i>Parameter:</i>	
<code>actName</code>	– Name der betroffenen Aktivität
<code>actVersion</code>	– Version — ” —
<code>actor</code>	– gewünschter Aktor für die Aktivität

Die Methode bewirkt, daß alle Instanzen der bezeichneten Aktivität, die innerhalb der von der Anpassung betroffenen Workflow-Instanz gestartet werden, ohne Berücksichtigung ihrer Rollenangabe dem gewünschten Aktor zugewiesen werden.

Aktivität abgeben

Die Zuweisung einer bereits existierenden Aktivitäteninstanz an einen anderen Bearbeiter als den durch die Rollenauflösung ermittelten geschieht durch die folgende Methode:

```
public Return DelegateActivity(String actInstId,
                               String newActor)
```

DelegateActivity	
<i>Parameter:</i>	
<code>actInstId</code>	– Bezeichner der Aktivitäteninstanz, die abgegeben werden soll
<code>newActor</code>	– Akteur, an den die Aktivität abgegeben werden soll

Termine ändern

Die folgende Methode dient der Änderung des Deadline-Termins einer Aktivitäteninstanz:

```
public Return SetDeadline(String actInstId,
                          String deadline)
```

SetDeadline	
<i>Parameter:</i>	
<code>actInstId</code>	– Bezeichnung der Aktivitäteninstanz, deren Deadline verändert werden soll
<code>deadline</code>	– neue Deadline für die Aktivitäteninstanz

Änderung von Workflow-relevanten Daten

Um Workflow-relevante Daten einer Workflow-Instanz zu ändern, wird dem Anpassungsdienst die Neubelegung der betroffenen Daten übergeben. Dies geschieht mit Hilfe der Datenstruktur, die auch zur Repräsentation der Workflow-relevanten Daten einer Workflow-Instanz benutzt wird: eines `PDLNameValueList`-Objektes. Die Schnittstelle sieht somit so aus:

```
public Return UpdateCaseData(PDLNameValueList updateData)
```

UpdateCaseData
<i>Parameter:</i> updateData – modifizierte Daten

Gruppierung von Einzelanpassungen

Zur Gruppierung von einzelnen Anpassungen entsprechend der Spezifikation in Abschnitt 3.3 dienen die folgenden Methoden:

```
public Return CommitModification()
```

```
public Return AbortModification()
```

Die Operation BEGIN fehlt, da sie implizit mit der zuerst aufgerufenen Anpassungsmethode ausgelöst wird. Daraus folgt, daß der Abschluß auch nur *einer* Anpassung eines COMMIT oder ABORT bedarf. D.h. die Wirkung von Anpassungen, sei es eine oder seien es mehrere gruppiert, wird durch die Methode `CommitModification` auf die Workflow-Instanz innerhalb der Montana-Engine übertragen, bzw. durch die Methode `AbortModification` unwirksam gemacht.

4.5.3 Hilfsmethoden

Außer den oben beschriebenen Methoden, die die verschiedenen Anpassungen realisieren, müssen noch einige Hilfsmethoden bereitgestellt werden, mit deren Hilfe übrige im Zusammenhang mit den Anpassungen notwendige Aufgaben erledigt werden können.

Workflow-Instanz-Repräsentation auslesen

Zum Auslesen der in einem Anpassungsobjekt enthaltenen Workflow-Instanz-Repräsentation stellt die Klasse `Modification` die folgende Methode:

```
public SProcess sProcessClone()
```

sProcessClone
<i>Parameter:</i> – (<i>keine</i>)
<i>Rückgabewert</i> – Kopie der Workflow-Instanz-Repräsentation im Anpassungsobjekt

Verzeichnisdienste

Die bisher nicht beschriebene Klasse `Query` stellt mit Hilfe der Methoden des BOW-CORBA-Gateways einige Verzeichnisdienste bereit:

- *Gruppen-Verzeichnis ermitteln*

```
public NameVerStatList groupList()
```

groupList	
<i>Parameter:</i>	– (keine)
<i>Rückgabewert</i>	– Listenstruktur, der Name und Version aller existierenden Gruppen in der Datenbasis der Montana-Engine entnommen werden können

- *Workflow-Beschreibungs-Verzeichnis ermitteln*

```
public NameVerStatList processList(String groupName,
                                   String groupVersion)
```

processList	
<i>Parameter:</i>	
<code>groupName</code>	– Name der Gruppe, dessen Workflow-Beschreibungs-Verzeichnis ermittelt werden soll
<code>groupVersion</code>	– Version — ” —
<i>Rückgabewert</i>	– Listenstruktur, der Name und Version aller existierenden Workflow-Beschreibungen der spezifizierten Gruppe entnommen werden können

- *Existierende Workflow-Instanzen ermitteln*

```
public InstList procInstList()
```

procInstList	
<i>Parameter:</i>	– (keine)
<i>Rückgabewert</i>	– Listenstruktur, der zu jeder mit Name und Version spezifizierten Workflow-Beschreibung die momentan existierenden Workflow-Instanzen entnommen werden können

- *Aktivitätenbeschreibungs-Verzeichnis ermitteln*

```
public NameVerStatList activityList()
```

activityList	
<i>Parameter:</i>	– (keine)
<i>Rückgabewert</i>	– Listenstruktur, der Name und Version aller existierenden Aktivitätenbeschreibungen in der Datenbasis der Montana-Engine entnommen werden können

- *Aktivitätenbeschreibung auslesen*

```
public PDLActivity activityDefnGet(String actName,
                                   String actVersion)
```

activityDefnGet	
<i>Parameter:</i>	
actName	– Name der auszulesenden Aktivitätenbeschreibung
actVersion	– Version — ” —
<i>Rückgabewert</i>	– Objekt, das die gesuchte Aktivitätenbeschreibung repräsentiert

Kapitel 5

Implementierung

In diesem Kapitel wird die Implementierung des Anpassungsdienstes dargestellt. Abschnitt 5.1 beschreibt die Entwicklungsumgebung sowie deren Systemanforderungen und die des Anpassungsdienstes. Im zweiten Abschnitt werden einige Aussagen zum Verlauf der Implementierung und aufgetretenen Problemen gemacht. Dort sind auch Ideen für weiterführende Arbeiten zu finden.

5.1 Entwicklungsumgebung

Zur Implementierung der vorliegenden Arbeit wurden die beiden Programmiersprachen C++ [Stroustrup92] für das BOW-CORBA-Gateway und Java [CampionWalrath96, Flanagan96] für den Anpassungsdienst eingesetzt. Die Zusammenarbeit und Kommunikation dieser beiden Komponenten wird mittels CORBA [OMG95, YangDuddy95] realisiert.

Nachfolgend sollen die Systemanforderungen der Komponenten der Diplomarbeit und ihrer Entwicklungsumgebung kurz dargestellt werden.

Das Workflow-Management-System

Die Grundlage für das verwendete WFMS stellt HP's Enterprise Process Management System Montana in der Version „X.01.06.00 Developers Pre-Release“ dar. Dieses System ist im Projekt PoliFlow am Institut für Parallele und Verteilte Höchstleistungsrechner auf einer HP-9000-Workstation der 700-er Reihe installiert und läuft dort unter dem Betriebssystem HP-UX 10.20. Die Workstation hat die Adresse `allo.informatik.uni-stuttgart.de`.

Das BOW-CORBA-Gateway

Das BOW-CORBA-Gateway muß auf derselben Maschine wie Montana laufen, in der vorliegenden Arbeit also auf `allo`.

Zur Übersetzung des BOW-CORBA-Gateways ist der C++-Compiler „HP C++“ in der Version A.10.09 verwendet worden. Die Verwendung der BOW-Bibliothek macht es erforderlich, daß die Klassen- und Template-Bibliothek „STL<Toolkit>“ von ObjectSpace in der Version 1.2.2 installiert ist.

Als CORBA-Plattform unter HP-UX wurde HP's ORB Plus Version 2.01 [HPORB96] verwendet.

Der Anpassungsdienst

Da die Implementierung des Anpassungsdienstes in Java erfolgt, ist dieser Teil der Arbeit auf verschiedenen Plattformen ausführbar, insofern die entsprechende Plattform Java-Code ausführen kann. Die in dieser Arbeit eingesetzte Java-Umgebung ist Sun's Java Developers Kit (JDK) in der Version 1.02 [Sun96JDK].

Das Anpassungsdienst-Subsystem PDL-Parser wurde mit Hilfe einer Grammatikspezifikation entwickelt, aus der von Sun's Parsergenerator „Java Compiler Compiler“ in der Beta-Version 0.6.-8 [Sun97] Java-Code generiert wird.

Die Abbildung von IDL nach Java und somit die Anbindung an die CORBA-Kommunikation wurde mit Hilfe von IONA's OrbixWeb [IONA95] realisiert.

Folgende Einschränkungen für die Lokation der Anpassungsdienst-Klassen sowie für die den Anpassungsdienst ausführende Maschine sind zu beachten:

- Im Fall der Verwendung eines lokalen Anpassungsobjektes in einem Java-Applet müssen die Klassen-Dateien des Anpassungsdienstes auf dem Herkunftsrechner des Applets abgelegt sein, um keine Applet-Security-Verletzung zu verursachen.
- Das BOW-CORBA-Gateway schreibt seine CORBA-Objektreferenz in eine Datei. Diese Datei muß vom Anpassungsdienst lesbar sein. D.h. das Verzeichnis, in dem die Datei abgelegt ist, muß im Verzeichnisbaum der den Anpassungsdienst ausführenden Maschine enthalten sein, sei es direkt oder per Network Filesystem über das Netzwerk eingebunden.

5.2 Bemerkungen zur Implementierung

5.2.1 Das BOW-CORBA-Gateway in C++

Grundlage und Orientierungshilfe für die Implementierung des BOW-CORBA-Gateways waren [HP96b] und insbesondere die Code-Beispiele, die Montana beigefügt sind. Viel Code konnte direkt aus dem Beispiel `cli.cc` übernommen werden, einem Kommandozeileninterpreter, der die Benutzung mehrerer Montana-API-Funktionen von der Shell aus ermöglicht.

Die CORBA-Schnittstelle zum BOW-CORBA-Gateway ist in der IDL-Datei `montanaAPI.idl` spezifiziert. Dort sind die Datentypen zur Repräsentation von Argumenten und Ergebnissen der CORBA-Methoden sowie diese selbst als Methodendefinitionen angegeben. Zur Veranschaulichung diene folgender Auszug aus `montanaAPI.idl`, der die Methode `PdlAdd` und den in ihr benutzten Datentyp `ReturnStatusEnum` beschreibt:

```

module API {

    enum ReturnStatusEnum {
        FAILURE,
        SUCCESS
    };

    // ... Auslassung

    interface MontanaAPI {

        void PdlAdd(    in string pdlSrcCode,
                       out ReturnStatusEnum returnStatus,
                       out string pdlcErrMsg);

        // ... Auslassung

    }; // interface MontanaAPI

}; // module API

```

Die so definierten Methoden werden in der Klasse `montanaAPI_impl` implementiert. Der Prozeß `montanaAPI_server` stellt ein `montanaAPI_impl`-Objekt bereit und schreibt dessen CORBA-Objektreferenz in eine Datei, die es anderen Prozessen ermöglicht, ein Referenzobjekt zu erzeugen und über dieses die Methoden des `montanaAPI_impl`-Objektes aufzurufen.

5.2.2 Der Anpassungsdienst in Java

Die grundlegende Arbeit zur Realisierung des Anpassungsdienstes stellte die Entwicklung einer geeigneten Objektrepräsentation von Workflow-Instanzen dar. Die Repräsentation der vorliegenden Arbeit ist in Abschnitt 4.4 ausführlich beschrieben. Die Entwurfsdarstellung wurde direkt in Java-Klassen abgebildet.

Der Aufbau der Objektrepräsentation mit Hilfe dieser Klassen wird durch das Subsystem PDL-Parser durchgeführt. Hierzu ist eine PDL-Grammatik erstellt worden, die den Umfang an PDL umfaßt, der in der vorliegenden Arbeit Verwendung findet. Dies ist bereits ein Großteil des Gesamtumfangs von PDL. Der syntaktische Teil der PDL-Grammatik wurde mit semantischen Aktionen angereichert, die aus den in PDL vorliegenden Workflow-Instanz-Informationen ein SProcess-Objekt aufbauen.

Kommunikation mit der Montana-Engine findet im Verlauf einer Anpassung zu zwei Zeitpunkten statt:

Vor der Anpassung erfolgt ein lesender Zugriff auf die anzupassende Workflow-Instanz, um die Objektrepräsentation dieser Workflow-Instanz im Anpassungsdienst aufzubauen. *Nach* der Anpassung an dieser Objektrepräsentation müssen die Änderungen auf die Workflow-Instanz innerhalb der Montana-Engine übertragen werden, d.h. es erfolgt dann ein schreibender Zugriff auf die anzupassende Workflow-Instanz.

Im Verlauf der Arbeit stellte sich heraus, daß im Gegensatz zum ersten, dem lesenden Zugriff mit dem zweiten, dem schreibenden Zugriff auf Workflow-Instanzen der Montana-Engine erhebliche Schwierigkeiten verbunden sind. Diese Schwierigkeiten, ihre Auswirkungen und das Vorgehen in der vorliegenden Arbeit sind – gegliedert nach unterschiedlichen betroffenen Teilen einer Workflow-Instanz – in den folgenden drei Unterabschnitten dargelegt.

Änderung an Kontrollflußbeschreibung einer Workflow-Instanz

Montana bietet in seinem derzeitigen Umfang keine API-Funktionen an, mit deren Hilfe Änderungen an der Kontrollflußbeschreibung einer Workflow-Instanz durchgeführt werden können. Somit ist es nicht möglich, die Änderungen, die an der Anpassungsdienst-internen Objektrepräsentation der Workflow-Instanz durchgeführt wurden, auf die Workflow-Instanz innerhalb der Montana-Engine zu übertragen und so für den weiteren Ablauf der Workflow-Instanz zur Wirkung zu bringen.

Ein konzeptionell verfolgter Ausweg war, aus der veränderten Objektrepräsentation im Anpassungsdienst eine neue Version der Workflow-Beschreibung zu erzeugen, diese dann zu instanzieren und auf den Stand der von der Anpassung betroffenen Workflow-Instanz zu bringen, d.h. die Abarbeitung an der richtigen

Stelle wiederaufzunehmen. Die alte Instanz könnte dann verworfen werden und es könnte mit der neuen weitergearbeitet werden.

Diese Vorgehensweise hat sich jedoch als enorm aufwendig und kaum überblickbar gezeigt, was allein die unter Kontrolle der Montana-Engine befindlichen Teile der Workflow-Instanz betrifft. Daß diese Lösung nicht realisierbar ist, zeigt sich allerdings erst im Blick auf die schon zum Zeitpunkt der Anpassung gerade in Bearbeitung befindlichen Aktivitäteninstanzen. Diese befinden sich nicht mehr in der Kontrollsphäre der Montana-Engine sondern werden von den sie realisierenden Applikationen kontrolliert. Die Wiedereingliederung dieser zum Zeitpunkt der Anpassung bereits aktiven Aktivitäteninstanzen in die neu erzeugte angepaßte Workflow-Instanz ist mit den Mitteln, die Montana anbietet, nicht möglich.

Aufgrund dieser Situation wurde die Realisierung der Anpassungen am Kontrollfluß in dieser Arbeit auf die Modifikationen an der Anpassungsdienst-internen Objektrepräsentation beschränkt. Die realisierten Anpassungsmethoden lassen sich so beispielsweise zur schrittweisen strukturierten Modellierung von Workflows einsetzen.

Betroffen von dieser Einschränkung sind die Anpassungen, die sich auf die Kontrollflußbeschreibung auswirken (siehe Abschnitt 4.5.2):

Entfernen eines Strukturierungselementes, Einfügen einer Arbeitseinheit, Regeln einer bedingten Verzweigung ändern und explizite Pfadauswahl in bedingten Verzweigungen.

Änderung an Aktivitäteninstanzen

Auch zum schreibenden Zugriff auf Aktivitäteninstanzen werden von Montana in der vorliegenden Version noch keine API-Funktionen zur Verfügung gestellt. Hier gibt es entgegen dem oben beschriebenen Fall jedoch einen realisierbaren Ausweg, der auf der Art und Weise beruht, wie Montana gestartete Aktivitäten Applikationen zuordnet. Der folgende Einschub beschreibt das Aktivitäten-Enactment von Montana:

Montana's Aktivitäten-Enactment [HP96a, HP96b]¹

Die Beschreibung eines PDL-WorkNodes enthält die Bezeichnung der in diesem WorkNode auszuführenden Aktivität in Form von Name und Version der Aktivitätenbeschreibung. Die Aktivitätenbeschreibung spezifiziert u.a. die Rolle, die ein Bearbeiter der Aktivität haben

¹In diesem Einschub werden absichtlich einige für eine bloße Beschreibung überflüssige Bezeichnungen aus der Montana-Literatur und dem verfügbaren Quellcode (insbesondere `montana/include/export/msgexport.hh`) genannt, um einen Einstieg in weiterführende Arbeiten zu erleichtern. Trotzdem ist die Beschreibung um einige Details gekürzt, da sonst der Rahmen gesprengt würde.

muß, welche Daten aus dem Case Packet (den Workflow-relevanten Daten der Workflow-Instanz) der Aktivität zum Lesen, sowie welche zum Schreiben bereitgestellt werden müssen.

Der erste Schritt des Aktivitäten-Enactments ist die Rollenauflösung: Die Montana-Engine schickt die Rolleninformation in einer Nachricht namens `ActivityAssign` an einen Ressourcen-Manager. Ein Ressourcen-Manager ist ein Prozeß, der die angegebene Rolle auflöst, in dem er abhängig von der Rolle ein geeignetes Business Object – d.h. einen anderen Prozeß – auswählt, dem die Bearbeitung der Aktivität überlassen werden kann. Ein Business Object ist entweder selbst die Applikation zur Bearbeitung der Aktivität oder eine Hülle um eine solche Applikation bzw. mehrere zu koordinierende Applikationen. Die Adresse des geeigneten Business Objects wird in einer Antwortnachricht namens `ActivityAssignReply` an die Montana-Engine zurückgesandt.

Nach erfolgter Rollenauflösung durch einen Ressourcen-Manager schickt die Montana-Engine dem ausgewählten Business Object eine `ActivityHandle`-Nachricht, in der ihm die zum Lesen bzw. Schreiben bereitstehenden Workflow-relevanten Daten zur Verfügung gestellt werden. Nach der Bearbeitung der Aktivität durch das Business Object sendet dieses eine `ActivityHandleReply`-Nachricht an die Montana-Engine zurück, die u.a. die geschriebenen Workflow-relevanten Daten enthält. In dieser Nachricht kann auch mit Hilfe einer Statusangabe ein Abgeben (Status `NOT_PERFORMED`) oder Scheitern der Aktivität (Status `FAILURE`) angezeigt werden.

Aus dieser Darstellung des Aktivitäten-Enactments ist bereits ersichtlich, daß es zwei Stellen gibt, bei denen ein Eingriff in den Ablauf von Aktivitäteninstanzen möglich ist: Ressourcen-Manager und Business Objects. In der PoliFlow-Gruppe am Institut für Parallele und Verteilte Höchstleistungsrechner wird die Funktionalität des Ressourcen-Managers durch einen Worklisthandler realisiert, der Arbeitslisten mit zu bearbeitenden Aktivitäten verwaltet, aus denen sich die Mitarbeiter die Aktivitäten auswählen können, die sie bearbeiten wollen. Der Worklisthandler wird derzeit im Rahmen einer Diplomarbeit überarbeitet und reimplementiert.

Anpassungen, die eine Änderung an Aktivitäteninstanzen vornehmen und sich in Kooperation mit Worklisthandler und den einzelnen Business Objects realisieren lassen, sollen im folgenden kurz unter Darlegung der Realisierungsideen genannt werden:

- *Rollenänderung*

Der Worklisthandler merkt sich gewünschte Rollenänderung und führt diese für die entsprechenden Aktivitäten vor der Rollenauflösung durch.

- *Explizite Rollenauflösung*

Der Worklisthandler merkt sich gewünschte explizite Rollenauflösung und berücksichtigt diese für die entsprechenden Aktivitäten.

- *Aktivität abgeben*

Das Business Object, das die Aktivität kontrolliert, wird benachrichtigt und sendet eine ActivityHandleReply-Nachricht mit der Rolle des gewünschten anderen Bearbeiter² und dem Status `NOT_PERFORMED` an die Montana-Engine, die dann die Aktivität mit Hilfe des Ressourcen-Managers der neuen Rolle zuordnet.

- *Aktivität/Subworkflow scheitern lassen*

Das Business Object, das die Aktivität kontrolliert, wird benachrichtigt, bricht die Bearbeitung ab und sendet eine ActivityHandleReply-Nachricht mit Status `FAILURE` an die Montana-Engine zurück.

- *Aktivität/Subworkflow wiederholen*

Die Realisierung entspricht der des Abgebens mit der Besonderheit, daß an den momentanen Bearbeiter 'abgegeben' wird. Er bekommt also die Aktivität noch einmal zugewiesen.

Die Terminänderung als Änderung an Aktivitäteninstanzen läßt sich jedoch nicht auf diese Weise realisieren, da die Information über Aktivitätentermine nur intern von der Montana-Engine verwaltet wird. Hierfür ist eine Erweiterung der Montana-API-Funktionen um den schreibenden Zugriff auf Aktivitätentermine erforderlich.

Änderung an Workflow-relevanten Daten

Die Realisierung der Anpassungsmethode `UpdateCaseData` ist mit Hilfe der Montana-API-Funktion `ProcInstStatusSet` (siehe Abschnitt 4.3.3) möglich.

Aus dem Datenbankbereich ist das Phänomen der verlorengegangenen Änderung („Lost Update“) bei konkurrierendem Zugriff auf dieselben Daten bekannt geworden. Um das Verlorengehen von Änderungen zu verhindern, muß ein exklusiver Zugriff auf die betroffenen Daten gewährleistet sein. Dies wird durch Sperren ermöglicht.

Montana bietet momentan noch keinen Sperrmechanismus für Workflow-relevante Daten an. Evtl. ist es möglich, den Workflow-Ablauf anzuhalten³, so daß wenig-

²Für jeden Bearbeiter sollte zu diesem Zweck eine Rolle existieren, die nur nach diesem Bearbeiter aufgelöst werden kann.

³Hierzu könnte `ProcInstChangeState` mit dem Zustand `Suspended-Active` verwendet werden, dies ist aber nicht dokumentiert.

stens durch den Workflow-Ausführungsdienst keine zwischenzeitliche Änderung stattfindet. Allerdings können andere WFMS-Komponenten, die ähnlich wie der Anpassungsdienst mittels ProcInstStatusSet auf die Workflow-relevanten Daten zugreifen, immer noch ein Verlorengehen von Änderungen verursachen.

Kapitel 6

Zusammenfassung und Ausblick

Zusammenfassung

Workflow-Management-Systeme werden zunehmend von Unternehmen eingesetzt, um die Qualität der Geschäftsprozeßabwicklung zu steigern. Die Anwendung von Workflow-Management bringt Kontrolle und Überblick über die laufenden Vorgänge des Unternehmens durch bessere Informationsverfügbarkeit, verringerte Durchlaufzeiten und Innovation in der Prozeßabwicklung.

Ein wichtiges Ziel bei der Weiterentwicklung der WFMS ist die Vergrößerung ihrer Flexibilität, also ihrer Fähigkeit, sich verändernden Gegebenheiten anzupassen. Hierbei hat es sich gezeigt, daß Workflows auch zur Laufzeit noch modifizierbar sein sollten, da bei der Bearbeitung Ereignisse auftreten können, die für die weitere Abwicklung eine Abweichung vom in der Modellierung definierten Ablauf erforderlich machen. Das Ziel dieser Arbeit ist es, einen Anpassungsdienst zu realisieren, d.h. eine WFMS-Komponente, mit deren Hilfe Anwender Anpassungen an Workflows zur Laufzeit durchführen können.

Für den Modellierungsaspekt der Kontrollflußbeschreibung wurde die strukturierte Modellierung gewählt, da sie eine übersichtliche Kontrollfluß-Modellierung ermöglicht und die Übersichtlichkeit in Entwurf und Implementierung des Anpassungsdienstes erhöht. Dieser Vorteil wird auch für weiterführende Arbeiten zum Thema „Kontrolle der strukturellen Integrität und Konsistenz von Workflows“ von Nutzen sein.

Auf einer vorangegangenen Arbeit basierend, in der eine Analyse häufig auftretender Anpassungen durchgeführt wurde, wurden ausgewählte Anpassungen spezifiziert, die der Anpassungsdienst bereitstellen soll. Dabei wurde dessen Rolle in der Zusammenarbeit mit dem WFMS und dem in einer parallel laufenden Diplomarbeit erstellten Workflow-Editor – der Benutzerschnittstelle zum Anpassungsdienst – berücksichtigt. Die hierbei auftretenden Anforderungen spielten

auch beim Entwurf eine große Rolle. Die Workflow-Instanz-Repräsentation wurde in zwei Ebenen unterteilt, von denen die untere sich eng an die Gegebenheiten des WFMS anlehnt und die obere die übersichtliche strukturierte Modellierung darstellt.

Die in der Arbeit verwendete Entwicklungsumgebung hat als Grundlage des verwendeten WFMS das Enterprise Process Management System „Montana“¹ von Hewlett Packard, das seine API-Funktionen für C++ bereitstellt. Die Nutzung dieser Funktionen durch den in Java implementierten Anpassungsdienst machte einen Gatewayprozeß in C++ erforderlich, der die benötigte Montana-Funktionen dem eigentlichen Anpassungsdienst als CORBA-Methoden zur Verfügung stellt.

Im Verlauf der Arbeit an der Realisierung der Anpassungen zeigte sich ein Problem der derzeitigen Version von Montana: verändernde Zugriffe auf Workflow- und Aktivitäteninstanzen sind nur teilweise – für Workflow-relevante Daten beispielsweise – möglich. Insbesondere schreibende Zugriffe auf die Kontrollflußbeschreibung einer Workflow-Instanz sind zwar in Planung, werden derzeit aber noch nicht angeboten. Die Realisierung der hiervon betroffenen Anpassungsmethoden wurde insofern eingeschränkt, daß die durch die Methoden verursachten Veränderungen an der Workflow-Instanz-Repräsentation im Anpassungsdienst nicht auf die Workflow-Instanz innerhalb der Montana-Engine übertragen werden können.

Ausblick

Die prototypische Implementierung der vorliegenden Arbeit zeigt, daß die zur weiteren Flexibilisierung des Workflow-Managements nötige Anpassungsfähigkeit realisierbar ist. Die Forschung zum Thema Anpassungsfähigkeit und die offensichtliche Notwendigkeit anpassungsfähiger Workflows für flexibles, effektives Workflow-Management werden die Entwicklung der WFMS zur Integration eines Anpassungsdienstes vorantreiben.

Im Verlauf der Arbeit wurden bei Überlegungen und in Diskussionen weitere Arbeiten identifiziert, die im Zusammenhang mit der vorliegenden stehen bzw. diese weiterführen:

- *Aufnahme weiterer Kontrollflußkonstrukte*

Die in dieser Arbeit benutzte strukturierte Modellierung kann sehr gut um weitere Konstrukte erweitert werden. Dringend notwendig sind Schleifen und wünschenswert wären deskriptive Konstrukte.

¹Das Enterprise Process Management System „Montana“ ist urheberrechtlich geschützt. Das Urheberrecht hält die Hewlett Packard Corporation.

- *Modellierung von Anpassungen*

Anpassungen können ähnlich wie Workflows modelliert werden. Hiermit wird Flexibilität für die Anpassungen selbst erreicht: Das Spektrum angebotener Anpassungen kann so erweitert werden und komplexe Anpassungen können sich der Steuerung durch das WFMS bedienen, wenn ihr Ablauf als spezieller „Workflow“ modelliert wurde.

Abkürzungsverzeichnis

API	Application Programmer's Interface
BOW	Business Object Wrapper
CORBA	Common Object Request Broker Architecture
DV	Datenverarbeitung
HP	Hewlett Packard
IDL	Interface Definition Language
PDL	Process Definition Language
WF	Workflow
WFMS	Workflow-Management-System

Glossar

Aktivität: Aktivitäten erfüllen Teilaufgaben eines Workflows, die aus Sicht der Workflow-Modellierung nicht mehr weiter unterteilt werden sollen. Sie repräsentieren Bearbeitungsschritte eines Geschäftsprozesses.

Aktivitäteninstanz: Dieselbe Aktivität kann in verschiedenen Workflow-Instanzen – und dort ggfs. auch mehrfach – auftreten. Sie wird jeweils durch eine Aktivitäteninstanz repräsentiert.

Anpassung: Änderung an einer Workflow-Instanz, die zu deren Laufzeit vorgenommen wird. Siehe Abschnitt 2.2.2 (Seite 22).

Anpassungsdienst: Komponente im WFMS, das die Durchführung von Anpassungen ermöglicht. Siehe Abschnitt 2.2.2 (Seite 22).

Applikation: Unter Applikationen werden im Bereich Workflow-Management Anwendungen verstanden, die zur Durchführung einer Aktivität verwendet werden.

Geschäftsprozeß: Geschäftsprozesse sind Arbeitsabläufe, die zum Erreichen der verschiedenen Arbeitsziele eines Unternehmens nötig sind. Der Begriff Geschäftsprozeß existiert unabhängig vom Workflow-Management.

Prozeß: Synonym für *Geschäftsprozeß*

Strukturierte Modellierung: Workflow-Modellierung mit Kontrollflußkonstrukten, die an die strukturierte Programmierung angelehnt sind. Siehe Abschnitt 2.4 (Seite 27).

Subworkflow: Workflow, der zur Beschreibung einer Teilaufgabe in einem anderen Workflow – seinem Superworkflow – verwendet wird.

Vorgang: Synonym für *Geschäftsprozeß*

Workflow-Ablauf: Synonym für *Workflow-Ausführung*

Workflow-Ausführung: Verwaltung der Informationen der Workflow-Instanz und Vorantreiben des Ablaufes entsprechend der Workflow-Beschreibung. Wird vom Workflow-Ausführungsdienst geleistet.

Workflow-Ausführungsdienst: WFMS-Komponenten, die die *Workflow-Ausführung* leistet.

Workflow-Beschreibung: Die Workflow-Beschreibung ist das Resultat der Modellierung eines Geschäftsprozesses. Sie spezifiziert verschiedene Aspekte des Workflows. Siehe Abschnitt 2.1.3 (Seite 17).

Workflow-Daten: Daten, die im Workflow-Management eine Rolle spielen:

- Workflow-Kontroll-Daten
- Workflow-relevante Daten
- Applikations-Daten

Siehe Abschnitt 2.1.2 (Seite 15), Abbildung 2.2 (Seite 16) und insbesondere die Ausführungen auf Seite 17.

Workflow-Enactment-Service: *Workflow-Ausführungsdienst*

Workflow-Instanz: Jede Workflow-Instanz repräsentiert einen derzeit unter Kontrolle des WFMS ablaufenden Geschäftsprozeß. Eine Workflow-Instanz umfaßt somit die Workflow-Beschreibung des Vorgangs sowie die während der Ausführung hinzugekommenen Informationen.

Workflow-Management: Die Steuerung und Kontrolle des Ablaufes von Geschäftsprozessen mit Hilfe von Informationstechnologie. (DV-gesteuerte Vorgangsabwicklung)

Workflow-Management-System: Ein Workflow-Management-System wird zur Verwirklichung und Durchführung des *Workflow-Management* eingesetzt.

Literaturverzeichnis

[Aldinger97]

Jörg Aldinger: *Erstellung eines Workflow-Editors*,
Diplomarbeit 1456, IPVR, Universität Stuttgart, März 1997.

[Booch95]

Grady Booch: *Objektorientierte Analyse und Design*,
Addison-Wesley, Bonn, 1995.

[CampioneWalrath96]

Mary Campione, Kathy Walrath: *The Java Tutorial*,
Online Version vom 6. Juli 1996.

<http://java.sun.com/books/series/tutorial/index.html>

The Java Tutorial: Object-Oriented Programming for the Internet,
Addison-Wesley Publishing Company, 1996.

[DudenInfo89]

Hermann Engesser (Hrsg.), Volker Claus (Bearb.): *Duden Informatik*,
Dudenverlag Mannheim/Wien/Zürich, 1989.

[Flanagan96]

David Flanagan: *Java in a Nutshell*,
O'Reilly & Associates, Sebastopol, 1996.

[HPORB96]

Hewlett Packard Corporation: *HP ORB Plus Version 2.01*,
Online Help, Palo Alto, 2. Ausgabe Mai 1996.

[HP96a]

Hewlett-Packard Corporation: *Enterprise Process Management – Project
Codename ‘Montana’, External Specification v1.7*,
Mai 1996.

[HP96b]

Hewlett-Packard Corporation: *Enterprise Process Management – Project
Codename ‘Montana’, Developers Guide & Technical Reference v1.2*,
Mai 1996.

[IONA95]

IONA Technologies Ltd.: *OrbizWeb Programming Guide*,
Online Help, Version 1.0, Dublin, 1995.

OrbizWeb Programming Guide (Beta Release),
Online Help, Version 2.0 Beta, Dublin, 1995.

[Jablonski95]

Stefan Jablonski: *Workflow-Management-Systeme: Modellierung und Architektur*,
International Thomson Publishing, Bonn, 1. Auflage 1995.

[KrasnerPope88]

G.E. Krasner, S.T. Pope: *A Cookbook for Using the Model-View-Controller User Interface Paradigma in Smalltalk-80*, Journal of Object-Oriented Programming, Aug./Sept. 1988.

[OMG95]

OMG: *The Common Object Request Broker: Architecture and Specification (Revision 2.0)*,
Framingham MA, Juli 1995.

[Ott96]

Anita Ott: *Modellierung von Rechten zur Anpassung von Workflows*,
Studienarbeit 1569, IPVR, Universität Stuttgart, November 1996.

[Siebert95]

Reiner Siebert: *Anpassungsfähigkeit in Workflow-Systemen – Modellierung und Beschreibung anpassungsfähiger Workflows*,
Interner Bericht IPVR, Universität Stuttgart, Juni 1995.

[Siebert96a]

Reiner Siebert: *Adaptive Workflow for the German Public Administration*,
Proceedings of the First International Conference on Practical Aspects of Knowledge Management (PAKM), Workshop on Adaptive Workflow, Basel 1996.

[Siebert96b]

Reiner Siebert: *Informationen zum Verbundprojekt PoliFlow – Arbeitsbereich Anpassungsfähige Workflow-Systeme*,
IPVR, Universität Stuttgart, April 1996.

[Stroustrup92]

Bjarne Stroustrup: *Die C++-Programmiersprache*,
2. Auflage, Addison Wesley, Bonn, 1992.

[Sun96JDK]

Sun Microsystems Inc.: *The Java Developers Kit – Version 1.0.2*,
Online-Dokumentation, 1996.
(<http://www.javasoft.com:80/products/jdk/1.0.2/>)

[Sun96RMI]

Sun Microsystems Inc.: *Java Remote Method Invocation – Specification*,
Prebeta Draft Revision 1.1, November 1996. (<http://java.sun.com/>)

[Sun97]

Sun Microsystems Inc.: *Java Compiler Compiler (JavaCC) – The Java
Parser Generator*,
<http://www.suntest.com/JavaCC/>, Februar 1997.

[WfMC94]

Workflow Management Coalition: *The Workflow Reference Model*,
Document Number TC00-1003, Issue 1.1, Brüssel, November 1994.
(<http://www.aiai.ed.ac.uk:80/WfMC>)

[WfMC96]

Workflow Management Coalition: *Terminology & Glossary*,
Document Number WFMC-TC-1011, Issue 2.0, Brüssel, Juni 1996.
(<http://www.aiai.ed.ac.uk:80/WfMC>)

[YangDuddy95]

Zhonghua Yang, Keith Duddy: *CORBA: A Platform for Distributed Ob-
ject Computing*,
University of Queensland, Australia, 1995.

Ich versichere, daß ich diese Arbeit selbständig verfaßt
und nur die angegebenen Hilfsmittel verwendet habe.

Stuttgart, den 11. März 1997