Entwicklung eines Workflow Systems auf Basis eines aktiven objektorientierten Datenbanksystems

DIPLOMARBEIT
zur Erlangung des akademischen Grades
eines Diplom-Ingenieurs der Informatik

Eingereicht an der Johannes Kepler Universität Linz
Institut für Informatik
Abteilung für Informationssysteme
o.Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel

Mitbetreuende Assistenten:
Dipl.-Ing. Stefan Rausch-Schott
Mag. Werner Retschitzegger

Verfaßt von: **Peter Lang**

Linz, im Februar 1995

Eidesstattliche Erklärung

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

Linz, am 28. Februar 1995

Kurzfassung

Workflow Systeme sind vernetzte Computersysteme, die Mitglieder einer Organisation in der Beschreibung, in der Analyse und in der Ausführung von Arbeitsprozessen unterstützen sollen.

Ein Workflow System besteht gewöhnlich aus zwei Teilen: einer Modellierungskomponente und einer Ablaufkomponente. Die Modellierung beschäftigt sich mit der Definition von Arbeitsabläufen, deren Analyse und Simulation und der Zuordnung von einzelnen Arbeitsschritten zu Mitgliedern der Organisation. Die Ablaufkomponente beinhaltet die Unterstützung der Endbenutzer und die Steuerung der Abarbeitung von Arbeitsschritten.

Der in dieser Diplomarbeit entwickelte Workflow System Prototyp baut auf einer aktiven Erweiterung eines kommerziellen objektorientierten Datenbanksystems auf. Ziel der Diplomarbeit war es, eine Erhöhung der Flexibilität sowohl während der Modellierung als auch während der Ausführung von Arbeitsprozessen durch den Einsatz von aktiven und objektorientierten Konzepten zu erreichen. Zwei Beispiele seien herausgegriffen: die Ablauffolge von Arbeitsschritten wird mit Event-Condition-Action-Regeln spezifiziert, und die Zuordnung von Arbeitsschritten zu Organisationseinheiten wird durch ein objektorientiertes Rollenkonzept ermöglicht. Der implementierte Prototyp zielt (noch) nicht auf eine komfortable Benutzerschnittstelle ab. Vielmehr stand die Entwicklung von Grundkonzepten im Bereich des Einsatzes von Regeln und im Bereich der objektorientierten Modellierung von Organisationen, Aktivitäten und Daten in einem Workflow System im Vordergrund.

Abstract

Workflow systems are networked computer applications, which support members of an organization in defining, analysing, and enacting of work processes.

A workflow system usually consists of two components: a workflow modeling component and a workflow enactment component. The modeling component covers definition, analysis and simulation of workflows and the assignment of work steps to organizational units. The enactment component supports the scheduling, execution, and monitoring of work steps.

The workflow system prototype developed in this diploma thesis is based on an active extension of a commercial object-oriented database system. The main objective of the thesis is to increase flexibility both during modeling and enactment by applying active and object-oriented concepts. For example, event-condition-action rules are used for specifying the activity ordering of workflows, and the assignment of working steps to organizational units is due to an object-oriented role concept. The prototype mainly pays attention to the development of basic concepts for using rules and for object-oriented modeling of organisations, data and activities within a workflow system.

Danksagung

An dieser Stelle bedanke ich mich bei all jenen, die durch ihre fachliche, finanzielle, direkte oder indirekte Unterstützung zum Gelingen dieser Arbeit beigetragen haben.

Besonderer Dank gebührt meinen Eltern, die mich nicht nur finanziell sondern auch stets mit viel Verständnis unterstützt haben.

Frau Professor Dr. Gerti Kappel, Dipl.-Ing. Stefan Rausch-Schott und Mag. Werner Retschitzegger möchte ich für ihre zahlreichen wissenschaftlichen Ratschläge vielen Dank aussprechen. Ob in stundenlangen Besprechungen oder nur zwischen Tür und Angel gegeben, sie haben stets zur Verbesserung der Arbeit beigetragen. Für die ausgiebige Korrektur möchte ich ihnen ebenfalls danken.

Weiters bedanke ich mich bei Claudia für ihre Korrekturarbeit und ihr Verständnis für meine oft nächtelange Arbeit, und bei Dipl.-Ing. Werner Kurschl für seine Unterstützung bei der Arbeit mit LaTeX.

Peter Lang

GemStone ist ein eingetragenes Warenzeichen von Servio Corporation.

OPAL ist ein Warenzeichen von Servio Corporation.

VisualWorks/Smalltalk ist ein Warenzeichen von ParcPlace Systems.

Smalltalk-80 ist ein Warenzeichen von Xerox Corporation.

WorkFlow ist ein eingetragenes Warenzeichen von C.S.E. Systems.

ORACLE ist ein eingetragenes Warenzeichen von Oracle Systems.

AIX ist ein eingetragenes Warenzeichen von IBM.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Ziel der Arbeit

Der Begriff *Workflow* wird als neue Bezeichnung für einen schon bekannten Problemkreis verwendet. Es geht dabei um die Beschreibung und Unterstützung von unternehmensweiten Arbeitsabläufen. Unternehmensweit beinhaltet die Unterstützung sowohl von Büro- und Verwaltungstätigkeiten als auch von Produktionsprozessen. Früher waren derartige Systeme unter den Begriffen Kanzlei- und Büroinformationssysteme (*KIS*, *BIS*) zusammengefaßt. Diese unterstützten jedoch lediglich den Büro- und Verwaltungsbereich mit der Automatisierung der verschiedensten Büroabläufe. Bereits in den 70er Jahren wurden solche Systeme entwickelt und verwendet [Zism77]. Zisman entwickelte im Rahmen seiner Dissertation ein System, das über einen Mailserver die verschiedenen Anwendungen in einem Büro steuert, und legte damit den Grundstein für weitere Arbeiten im Bereich der Darstellung, Spezifikation und Automatisierung von Büroabläufen. Chroust definierte den Begriff „Workflow Management" in [Chro94] kürzlich als

> *„the computer assisted management of business process through the execution of software whose order of execution is controlled by a computerized representation of the business process"*

„Workflow Management" soll sich aber nicht, wie in der vorhergehenden Definition, nur auf Büroabläufe beschränken, sondern, wie bereits erwähnt, auch den Produktionsbereich einschließen. Ein Produktionsprozeß besteht ja wiederum aus vielen einzelnen Aktivitäten, die teilweise parallel oder seriell stattfinden und somit auch untereinander koordiniert werden müssen. Gerade in der Koordination dieser Prozesse liegt sehr großes Rationalisierungspotential.

Bei der maschinellen Produktion konnte in den letzten Jahren durch die technische Automatisierung die Produktivität erheblich gesteigert werden. Ähnliche Produktivitätssteigerungen konnten im Büro- und Verwaltungsbereich nicht beobachtet werden. Dies liegt u.a. darin begründet, daß im Büro- und Verwaltungsbereich wesentlich flexiblere Systeme notwendig sind, wodurch sich eine Automatisierung erheblich schwieriger gestaltet. Unter Flexibilität versteht man hier die Behandlung von Ausnahmen und die ständige Anpassung des Systems an neue Anforderungen, wie geänderte Abläufe und Organisationsstrukturen. Die Produktivität im Büro- und Verwaltungsbereich hat durch die Computerunterstützung bisher nur in geringem Maße zugenommen. Umsomehr gilt es jetzt, die Schwachpunkte in der Abwicklung der Büroabläufe zu erkennen und gleichzeitig ganzheitliche Lösungen für Produktion und Verwaltung zu finden.

In Workflow Systemen werden viele Daten produziert und verwendet. Bei diesen Daten handelt es sich sowohl um Prozeßdaten zur Steuerung des Systems, als auch um herkömmliche Benutzerdaten wie Texte, Grafiken, Tabellen, Kalkulationsdaten und ganze Datenbanken. Die Aufgabe von Workflow Systemen ist es, diese Daten vielen Benutzern (oft) gleichzeitig zur Verfügung zu stellen. Daher liegt es nahe, alle diese Daten in Datenbanken zu sammeln. Die meisten der bereits verfügbaren Workflow Systeme bauen aber nicht auf den Konzepten und Möglichkeiten von Datenbanksystemen auf und sind deshalb in dieser Hinsicht nur eingeschränkt verwendbar. Andere Workflow Systeme lassen nur das Design von Abläufen zu und unterstützen nicht deren Ausführung. Erst durch die Verwendung von Datenbankeigenschaften, wie Mehrbenutzerfähigkeit und Wiederanlauf, unterliegt man nicht mehr den Einschränkungen von Dateisystemen und kann zumindest teilweise die Flexibilität und Sicherheit von Workflow Systemen gewährleisten.

Ziel der Diplomarbeit ist es, anhand der Entwicklung eines Workflow System Prototyps, basierend auf einer aktiven Erweiterung des objektorientierten Datenbanksystems GemStone — aktive Komponente TriGS (Triggersystem für GemStone [Kapp94a]) — und der begleitenden Analyse bestehender Ansätze ([Zism77, Rein93, Jabl94a] und andere), die Grundanforderungen an ein solches System herauszuarbeiten und dabei die Stärken und Schwächen der verwendeten Systeme bzw. Konzepte aufzuzeigen. Mögliche Alternativen, Verbesserungen und Ergänzungen sollen ebenso ihren Platz finden. Der Anwendungsbereich wird in dieser Arbeit auf Geschäftsprozesse im Büro- und Verwaltungsbereich eingeschränkt, um zusätzliche Probleme aus der Produktionssteuerung wie Performanz, Echtzeitverarbeitung usw. vorerst nicht berücksichtigen zu müssen. Der Schwerpunkt der Arbeit liegt im Einsatz von ECA-Regeln (Event-Condition-Action-Regeln) des Triggersystems von GemStone, das die Implementierung erleichtern und dadurch flexiblere Lösungen ermöglichen soll.

1.2 Begriffsklärung

ECA-Regeln bestehen aus einem Ereignis, einer Bedingung und einer Aktion. Tritt ein Ereignis ein, so wird die Bedingung der Regel evaluiert und die Aktion ausgeführt, wenn die Bedingung erfüllt ist. Ein aktives Datenbanksystem unterstützt die Definition und Verwendung von ECA-Regeln.

Bevor näher auf die Anforderungen eines Workflow Systems eingegangen wird, sind noch einige Begriffe zu definieren, um Mißverständnisse — wie sie immer wieder durch unterschiedliche Verwendung der Termini entstehen — zu vermeiden.

Ein Workflow System kann in Analogie zu einem Datenbanksystem beschrieben werden. Während in einer Datenbank Daten gesammelt werden, werden in einer „Workflow Bank“ alle ablaufrelevanten Daten abgelegt. Die Definition, Kontrolle und Ausführung der gespeicherten Abläufe wird von einem Programmsystem, dem sogenannten *Workflow Management System* übernommen. Beides zusammen, Workflow Bank und Workflow Management System, ergibt ein *Workflow System*.

Der Einsatz eines Workflow Systems wird in zwei Teile geteilt, *Modellierung* und *Ablauf*. Die Modellierungskomponente unterstützt die Definition von Organisationsstrukturen, Arbeitsabläufen und Daten. Die Ablaufkomponente steuert Arbeitsabläufe und ermöglicht die Bearbeitung von Arbeitsschritten.

Ein *Workflow Typ* umfaßt die Beschreibung von Arbeitsabläufen mit einem konkreten Ziel. Ein *Workflow* repräsentiert eine konkrete Instanz eines Workflow Typs mit konkreten Daten. Die konkreten Daten eines Workflows werden zu einem *Akt* zusammengefaßt.

Der Begriff *Prozeß* wird im allgemeinen als Synonym für Workflow oder Workflow Typ gebraucht. Im engeren Sinn wird darunter auch manchmal ein einzelner Arbeitsschritt (Aktivität) verstanden. Die richtige Bedeutung ist dem jeweiligen Kontext zu entnehmen.

Alle weiteren Begriffe werden bei erstmaliger Verwendung in den einzelnen Kapiteln erklärt. Im Glossar (siehe Anhang E) sind noch einmal alle wichtigen Begriffe zusammengefaßt und kurz erläutert.

1.3 Anforderungen an ein Workflow System

Im August 1993 haben sich zahlreiche Anwender, Händler und Hersteller aus Wissenschaft und Praxis zu einer internationalen Organisation zusammengeschlossen. Die derzeit aus etwa 90 Mitgliedern bestehende *Workflow Management Coalition* hat in einem Referenzmodell die grundlegenden Anforderungen an Workflow Systeme definiert [WFMC94]. Dieses Modell versucht, die Hauptkomponenten und die wichtigsten Schnittstellen zu bestimmen.

Prozeßdefinitionswerkzeuge Zur Analyse, Modellierung und Spezifikation von Geschäftsprozessen gibt es eine Menge von Werkzeugen. Eine Vielzahl davon ist aber nur für ein bestimmtes Workflow System ausgelegt und/oder kann nur Teilaufgaben lösen. Es soll eine gemeinsame Schnittstelle geschaffen werden, damit mehrere Werkzeuge in einem System verwendet werden können.

Workflow Ablaufsteuerung Die Ablaufsteuerung übernimmt die Kontrolle und Ausführung der Prozesse. Diese Aufgabe kann von mehreren Systemen gelöst werden, die untereinander kommunizieren können, wofür wieder einheitliche Schnittstellen notwendig sind. Die Ablaufsteuerung wird in der Literatur oft als „Workflow Engine“ bezeichnet.

Anwendungen für Workflow Teilnehmer (Anwender) Diese Applikationen zeigen dem Endbenutzer die Arbeitsliste an und ermöglichen ihm, die Arbeitsschritte auszuführen oder neue Prozesse zu starten, soweit er dazu berechtigt ist. Da diese Werkzeuge Teil des Management Systems, aber auch Produkte von Fremdherstellern sein können, muß auch hier die Kommunikation mit den Steuereinheiten genau definiert werden. Bereits verwendete Anwendungsprogramme sollen, wenn möglich, weiterhin benützt werden können.

Einbindung vorhandener Applikationen Es gibt viele Anwendungsprogramme, die in ein Workflow System eingebunden werden oder damit kooperieren müssen, wie etwa Texteditoren, Kalkulationsprogramme und Grafikwerkzeuge. Um diese Anwendungen einbinden zu können, müssen sie offene Schnittstellen aufweisen, um dieses überhaupt zu ermöglichen.

Interoperabilität verschiedener Workflow Systeme Um Workflow Systeme verschiedener Hersteller miteinander verwenden zu können, muß deren Kommunikation genau definiert werden. Je umfangreicher die Kommunikation ist, desto mehr muß sie aufeinander abgestimmt sein. Eine detaillierte Angabe der Schnittstellen ist dabei unerläßlich.

Administrations- und Kontrollwerkzeuge Neben den Programmen zur Ausführung von Arbeitsschritten muß auch Software zur Überwachung zur Verfügung gestellt werden.

Viele der hier angeführten Anforderungen zielen bereits auf die Verwendung mehrerer Systeme ab. In dieser Arbeit jedoch gelten alle Anforderungen nur für den Einsatz eines Systems. Die Grundanforderungen bleiben aber dennoch bestehen. Das heißt, es fällt „lediglich“ die Verwendung einer einheitlichen Schnittstelle zwischen den Softwareprodukten weg. Außerdem wird als Anwendungsbereich das Büro angenommen, da, wie bereits erwähnt, bei der flexiblen Fertigung zusätzliche Anforderungen, wie Echtzeitverarbeitung, hinzukommen.

1.4 Einbettung in das Forschungsumfeld

Im Rahmen dieser Diplomarbeit können nicht alle forschungsrelevanten Problemstellungen im Bereich Workflow Management aufgegriffen werden. Eine nicht vollzählige Darstellung der Problemfelder soll dies illustrieren.

Die Entwicklung von Workflow Systemen verlangt stets nach deren Erweiterung. So ist es möglich, das System in einem globalen Netz zu installieren, und Gruppen von Benutzern auf verschiedenen Knoten zu unterstützen. Damit wird der Verteilungsaspekt eingebracht, der im Bereich von verteilten Datenbanksystemen viel Stoff für wissenschaftliche Arbeit liefert [Öszu91].

Ein weiterer Punkt ist der Umgang mit Ausnahmesituationen in einem Ablauf. Die Ausnahmebehandlung („Exception Handling") stellt ein sehr aktuelles Forschungsgebiet dar [Good75, Karb90, Saas92].

Die Fehlerbehandlung führt auch in das Gebiet der Graphentheorie. Arbeitsabläufe müssen bestimmten Richtlinien folgen, um das gewünschte Ziel zu erreichen. Als Beispiel läßt sich die Prüfung auf Deadlock-Freiheit eines Ablaufs anführen. Die Bedeutung der Graphentheorie wird durch zahlreiche Publikationen belegt [Dörf77]. Unter dem Einfluß der Graphentheorie und der Theorie der endlichen Automaten [Hopc79] entstanden Petri-Netze zur Beschreibung von Arbeitsabläufen [Pete81]. Viele, zur Zeit eingesetzte Modellierungsmethoden zur Beschreibung von Abläufen greifen auf Petri-Netze zurück, weshalb die Aktualität dieses Konzepts noch immer gegeben ist.

1.5 Beispiel einer Problemstellung

Es gibt genügend Anwendungsbeispiele für den Einsatz von Workflow Systemen. Und doch ist es schwierig, ein praxisnahes Beispiel zu finden, das eine möglichst kurze Beschreibung aller wesentlichen Teile des Systems zuläßt.

Oft wird als Beispiel das Anwendungsgebiet Bankwesen herangezogen. Kreditanträge scheinen ein gutes Beispiel für die Darstellung von Abläufen zu sein. Eigentlich eignen sich alle Vorgänge, bei denen Anträge gestellt werden, sehr gut zur Beschreibung der Funktionalität von Workflow Systemen. Ein externes Ereignis — ein Kunde stellt einen Antrag — hat das Anlegen eines Aktes zur Folge. Daraufhin durchläuft der Akt verschiedene Sachbearbeiterstellen. Bei jeder Stelle gelangt der Akt in den Posteingang, wird bearbeitet und wird schließlich wieder in den Postausgang gelegt. Am Ende wird die Entscheidung gefällt, ob der Antrag angenommen oder abgelehnt wird. Eine Vielzahl solcher Vorgänge findet sich in der öffentlichen Verwaltung.

Aber nicht nur die öffentliche Verwaltung bietet derartige Vorgänge. Scheer stellt in [Sche94] viele industrielle Geschäftsprozesse vor, die alle im Hinblick auf die Vorgangsbearbeitung modelliert und strukturiert sind. Der Begriff „Workflow Management" wird nur am Rande erwähnt, obwohl die Grundlagen dafür — nämlich Organisations-, Aktivitäten- und Datenmodellierung — sehr ausführlich behandelt werden. Als Beispiel wird aus dieser Menge von Geschäftsprozessen wieder eine Antragstellung ausgewählt. Wie bei einem Kreditantrag ist das Eintreffen und Antragstellen eines Kunden das auslösende Ereignis. Der Kunde fordert ein Angebot des Betriebes über ein bestimmtes Produkt. Es handelt sich also um die „Angebotsbearbeitung".

Nachdem die Kundenanfrage eingetroffen ist, und die Daten übernommen worden sind, müssen die Kalkulationsdaten ermittelt werden. Einflußfaktoren sind dabei die Menge und Kosten der verbrauchten Teile, die Herstellungsdauer und die Kosten der involvierten Betriebsmittel. Anschließend wird die Kalkulation durchgeführt. Vom Kalkulationsergebnis hängt es ab, ob (1) ein Angebot erstellt wird, (2) die Erstellung eines Angebots abgelehnt wird, weil eine Produktion bzw. Lieferung entweder nicht durchführbar oder nicht rentabel ist, oder ob (3) die Anfrage modifiziert und neuerlich kalkuliert wird.

Dieses kurze Beispiel zeigt bereits zwei wesentliche Teile eines Vorgangs: die durchzuführenden Funktionen und die dabei verwendeten und produzierten Daten. Der dritte Aspekt wird noch außer Acht gelassen: die Ausführenden und die Art der Ausführung. Die Ermittlung der Kalkulationsdaten erfolgt durch einen Mitarbeiter der Konstruktionsabteilung, also interaktiv. Die Kalkulation selbst und die etwaige Erstellung eines Angebotes wird automatisch in der Vertriebsabteilung vorgenommen. Damit ist auch der dritte wichtige Faktor in der Vorgangsbearbeitung, die Organisation, mit einbezogen.

Das gesamte Beispiel wird später in Kapitel 5 genau modelliert und in eine für den Workflow System Prototyp verständliche Form gebracht, um die Funktionsweise des Systems darzustellen. Anhand des Beispiels sollen dann auch die Unzulänglichkeiten des Prototyps aufgezeigt werden.

1.6 Aufbau der Arbeit

Kapitel 2, „Einsatz von Workflow Systemen", beinhaltet die theoretischen Grundlagen für die Entwicklung des Prototyps. Dabei wird der Einsatz in Modellierung und Ablauf getrennt. Viele der in der Arbeit verwendeten Begriffe werden in diesem Abschnitt definiert.

Kapitel 3, „Basiskonzepte und Realisierungsgrundlagen", beschreibt die verwendeten Konzepte von Objektorientierung, über (aktive) Datenbanksysteme und einem Rollenkonzept bis zur Integration von Systemen.

Kapitel 4, „TriGS$_{flow}$ — Prototyp eines Workflow Systems", stellt neben der Architektur die gesamte Funktionalität des entwickelten Prototyps vor. Die wesentlichen Implementierungsaspekte, wie der Einsatz von ECA-Regeln und der Einsatz des Rollenkonzepts, werden ebenfalls angeführt.

Kapitel 5, „Modellierung und Ablauf in TriGS$_{flow}$", behandelt das in Abschnitt 1.5 beschriebene Beispiel einer Problemstellung.

Kapitel 6, „Analyse eines kommerziellen Workflow Systems", bringt eine detaillierte Analyse eines kommerziell erhältlichen Workflow Systems und vergleicht dessen Funktionalität mit der des entwickelten Prototyps.

Kapitel 7, „Zusammenfassung und Ausblick", faßt das Basiswissen über den Prototyp und den darin verwendeten und entwickelten Konzepten zusammen und gibt einen Ausblick auf noch nicht gelöste Probleme und notwendige Ergänzungen des Prototyps.

Im Anhang finden sich die Dokumentation der Objektklassen des Prototyps, Anmerkungen zur aktiven Datenbanksystemkomponente und zum Rollenkonzept und ein Auszug aus der Regelbasis des Prototyps. Der Anhang schließt mit einem Glossar der wichtigsten Begriffe, die in dieser Arbeit eingeführt werden.

Kapitel 2

Einsatz von Workflow Systemen

Bevor auf die Einsatzgebiete von Workflow Systemen eingegangen wird, werden die in der Folge verwendeten Termini kurz definiert, um Mißverständnisse, die sich aus ähnlichen Begriffen in der Literatur ergeben, zu vermeiden. Alle Begriffe werden später ausführlich definiert.

Agenten können Arbeitsschritte ausführen (Aktivitäten bearbeiten). Sie werden in menschliche Agenten und Software Agenten eingeteilt. Der Unterschied besteht in der interaktiven Bearbeitung der Arbeitsschritte durch menschliche Agenten und in der automatischen Bearbeitung durch Software Agenten. Das Kompetenzprofil eines Agenten wird durch die Gesamtheit der Rollen beschrieben, die er innerhalb der Organisation annehmen kann. An eine Rolle sind Funktionen gebunden, die ein Agent ausführen kann, wenn er die Rolle annimmt. Eine Rolle kann einer bestimmten Gruppe (Abteilung) zugeordnet werden. Auch ein Software Agent kann im weiteren Sinne eine Rolle annehmen und damit auch einer Gruppe zugeordnet werden.

Rollen sind konkrete Ausprägungen eines Rollentyps. Eine Rolle entsteht, wenn ein Agent „in eine Rolle schlüpft", d.h. das Verhalten eines Rollentyps annimmt. Ein Agent kann jederzeit neue Rollen annehmen und bekannte Rollen aufgeben.

Rollentypen definieren das Verhalten, das Agenten annehmen können (Verhalten im Sinne von möglicher Aufgabenausführung). Wenn sie in einer Hierarchie angeordnet werden, kann das Verhalten eines übergeordneten Rollentyps dem untergeordneten vererbt werden.

Gruppen dienen der Zusammenfassung von Agenten in Abteilungen. Einer Abteilung in einem Betrieb entspricht eine Gruppe im Workflow System.

Aktivitäten beschreiben einen Arbeitsschritt, den ein Agent ausführen kann oder soll. Bearbeitet wird ein Akt.

Aktivitätennetze beschreiben Arbeitsabläufe, also die Reihenfolge, in welcher die Aktivitäten ausgeführt werden können.

Akte sind eine Art elektronische Umlaufmappe, deren Struktur durch den Akttyp festgelegt ist.

Akttypen beschreiben den Aufbau von Akten. Innerhalb eines Arbeitsablaufs kann nur ein Akttyp verwendet werden. Er beschreibt alle Daten, die im Zuge eines Aktenlaufs („Workflow") verwendet werden.

Organisationspolitiken verbinden Aktivitäten mit Agenten. Sie beschreiben, welche Agenten eine Aktivität auszuführen haben.

Workflow Typen fassen obige Begriffe zusammen und beschreiben somit die Organisationsstruktur (Agenten, Rollen, Gruppen) des Betriebes, das Aktivitätennetz und den Aufbau der Akte, die bearbeitet werden können.

Workflows sind die Instanzen eines Workflow Typs und entsprechen damit einem konkreten Aktenlauf. Sie halten die aktuelle Situation eines Ablaufs fest.

In dieser Arbeit wird zwar grundsätzlich zwischen einem Rollentyp und einer Ausprägung davon, einer Rolle bzw. eines Rollenobjekts, unterschieden. Der Begriff „Rolle" wird aber allgemein auch als Synonym für Rollentyp oder Rollenobjekt verwendet. In diesem Fall ist die richtige Bedeutung dem Kontext zu entnehmen.

Workflow Systeme werden zur Unterstützung von Geschäftsprozessen verwendet. Bevor ein solcher Ablauf jedoch unterstützt werden kann, muß dieser definiert werden. Das erste Verwendungs- bzw. Einsatzgebiet von Workflow Systemen ist daher die Modellierung aller Gegebenheiten, die zur Unterstützung eines Prozesses notwendig sind, und die Modellierung des Prozesses selbst. Die *Modellierung* („Modeling", Abschnitt 2.1) umfaßt die Definition der Organisationsstruktur des Betriebes, der Agenten, der Arbeitsabläufe und auch die Festlegung der Datenstruktur der verwendeten Akte. Erst wenn die Modellierung beendet ist, kann das System die Prozesse ausführen und kontrollieren. Der zweite Einsatzbereich ist der *Ablauf* („Enactment", Abschnitt 2.2) der Prozesse, kontrolliert durch das Workflow System. Anhand der Definitionen der Modellierung werden die Arbeitsschritte verteilt, den Agenten deren Ausführung ermöglicht und das System durch den Workflow System Administrator kontrolliert.

2.1 Modellierung

Grundsätzlich sollen die Abbildung der Organisation eines Betriebes und die Darstellung der Arbeitsabläufe getrennt voneinander vorgenommen werden können. Erst wenn alle Organisationseinheiten und Arbeitsschritte definiert

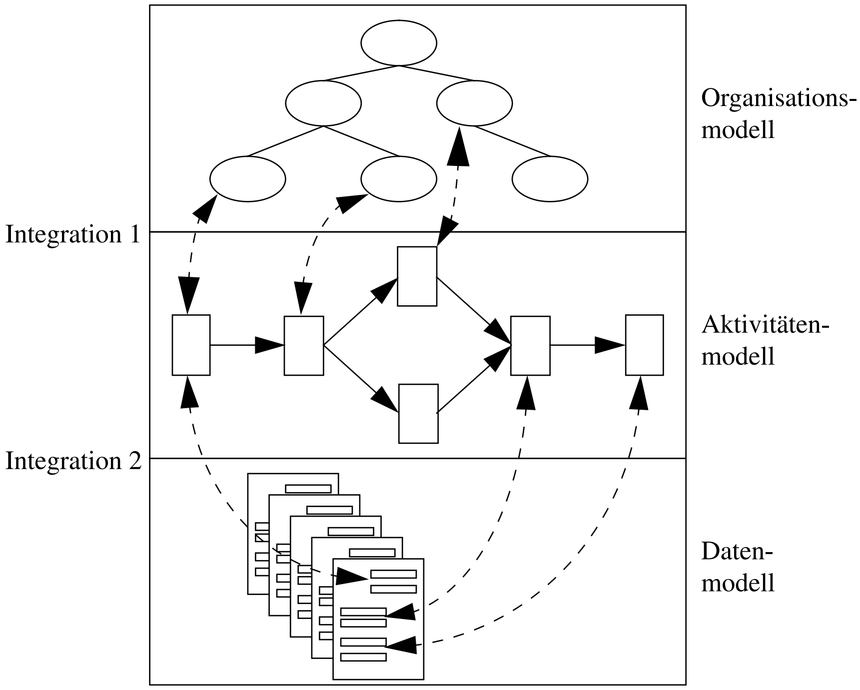


Abbildung 2.1: Modellierungsschichten und deren Integration [Gruh94]

sind, sollen diese mit Hilfe von Organisationspolitiken verbunden werden. Dies hat den Vorteil, daß Organisation und Ablauf von verschiedenen Personen zu verschiedenen Zeitpunkten modelliert werden können, und die Organisation unter Beibehaltung der Arbeitsabläufe ausgetauscht werden kann. Nachdem Arbeitsabläufe modelliert worden sind und im Workflow System eingesetzt werden, müssen sie jederzeit wieder geändert werden können, weshalb sich die Modellierungsphase und die Ablaufphase meist überschneiden. Die Zeitspanne der Modellierung wird auch als „Build Time“ bezeichnet.

Die Abbildung 2.1 aus [Gruh94] zeigt die drei Schichten Organisation, Aktivitäten und Datenstrukturen in der Modellierung. Die Integration der Schichten erfolgt durch Zuweisung der Aktivitäten zu den Agenten (Integration 1 — „Organisationspolitiken“, siehe 2.1.4.1) und durch die Beschreibung der in einer Aktivität verwendeten Daten (Integration 2 — „Datenfluß“, siehe 2.1.4.2). Obwohl die Integration der Schichten erst durchgeführt werden kann, wenn die Schichten selbst modelliert sind, ist die zeitliche Abfolge der Modellierung der einzelnen Schichten und deren Integration auch nicht streng sequentiell, sondern weist wieder Überlappungen auf. Wie im Software Engineering müssen auch Geschäftsprozesse immer wieder einem Redesign unterworfen werden. Da viele Methoden des Software Engineering, wie zum Beispiel der Top-Down-Entwurf und der Bottom-Up-Entwurf, in der Modellierung von Geschäftsprozessen Anwendung finden, könnte man diese Phase gewissermaßen auch als „Workflow Engineering“ bezeichnen. Die Modellierung von Geschäftsprozessen kann somit durch eine Art *Prototyping* erfolgen, unter der Voraussetzung, daß das Workflow System Simulation und Test von Geschäftsprozessen unterstützt.

Scheer teilt in [Sche94] die Modellierung der Vorgänge (= Prozesse im allgemeinen) auch in Daten, Funktionen und Organisation. In der vorliegenden Arbeit werden sie als Schichten bezeichnet. Scheer verwendet dafür den Ausdruck „Sicht". Eine weitere Sicht, die Steuerungssicht, beschreibt die Zusammenhänge zwischen den drei Sichten und entspricht deshalb dem Integrationsaspekt (Organisationspolitik und Datenfluß) dieser Arbeit. Der augenblickliche Zustand eines Vorgangs wird durch die Daten beschrieben. Vorgänge werden immer durch Ereignisse ausgelöst, wenn sie auch meist implizit durch das Ende der vorhergehenden Funktion vorliegen. Das Konzept der vorliegenden Arbeit unterscheidet sich von dem Scheers dadurch, daß dieser noch eine zusätzliche, von den anderen Sichten abgekoppelte Ebene einführt, die Ressourcensicht. Diese enthält alle informationstechnisch relevanten Komponenten wie z.B. Personal Computer, Anschluß an Datenbanken usw., die für die Ausführung von Prozessen von Bedeutung sind. Dargestellt werden die Arbeitsabläufe bei Scheer mittels Vorgangskettendiagrammen, die den Workflow Typen in der vorliegenden Arbeit entsprechen. Sie enthalten die Beschreibung der Organisationseinheiten, Ereignisse, Funktionen, Daten und schließlich auch die Art der Bearbeitung der Funktionen (interaktiv/automatisch).

In den meisten Betrieben liegt eine Organisationsstruktur vor, die im großen und ganzen in das Workflow System übernommen werden kann. Wenn sie noch nie abgebildet wurde, so wird durch die Einführung eines Workflow Systems auch eine Organisationsstruktur bereitgestellt, die beliebig erweitert und so auch von anderen Anwendungen genutzt werden kann. Bei der Definition der Abläufe muß man sich zunächst nicht darum kümmern, wer eine bestimmte Aktivität auszuführen hat, sondern kann Abläufe in einzelne Schritte zerlegen und die richtige Reihenfolge der Arbeitsschritte bestimmen.

In den meisten Fällen werden sich immer wieder — insbesondere nach den Testphasen von Abläufen — Änderungswünsche ergeben. Diese sollen ohne aufwendige Eingriffe vom Workflow System Administrator erfüllt werden können. Das Workflow System muß also auch während der Betriebszeit Änderungen an den Definitionen der Workflow Typen zulassen und auf Wunsch sofort beim nächsten gestarteten Aktenlauf verwenden. In der Praxis ist hier eine Möglichkeit zur Simulation von Abläufen zu schaffen oder wenigstens eine Art Versionierung von Prozessen einzuführen, sodaß ein Prozeß ausreichend getestet werden kann, bevor er von den Anwendern benutzt wird.

2.1.1 Organisationsmodellierung

Die Organisation eines Betriebes sollte möglichst realitätsgetreu modelliert werden können, damit die Vorteile einer gut funktionierenden Organisation auch im Workflow System ausgenützt werden können. Einerseits muß die statische Organisationsstruktur genau abgebildet, andererseits müssen auch die Verbindungen zu den einzelnen Aufgaben definiert werden. Diese Verbindungen werden aber erst später bei der Integration durch Angabe von Organisationspolitiken definiert.

Die *Organisationsstruktur* eines Betriebes setzt sich aus mehreren Organisationseinheiten zusammen, von denen die wichtigsten folgende sind: Personen, Programme, Rollen (Kompetenzen) und Abteilungen. Personen und Programme — im Folgenden *Agenten* genannt — führen Arbeitsschritte aus. Im Workflow System muß genau definiert werden, welche Funktionen ein Agent ausführen kann. Dieses Verhalten der Agenten wird durch ihre *Rollen* beschrieben. *Rollen* sind meist bestimmten *Gruppen* (Instanzen, Abteilungen) eines Betriebes zugeteilt, d.h. jene zugeteilten Rollen existieren nur innerhalb dieser Gruppen. In [Kies92] wird der Begriff der *Stellenbeschreibung* eingeführt:

> *„Stellenbeschreibungen oder Arbeitsplatzbeschreibungen legen den Umfang der Weisungs- und Entscheidungsbefugnisse fest, nennen die über- und untergeordneten Stellen, spezifizieren einzelne Rechte und Pflichten und enthalten oft auch Hinweise auf die anzuwendenden Programme und die maßgeblichen Pläne.“*

Damit kann ein Rollentyp mit seinen Funktionen und der Einbettung innerhalb einer Rollentyphierarchie als Stellenbeschreibung angesehen werden. Die Gesamtheit aller Rollen und der zugeteilten Gruppen eines Agenten, dessen Aufgaben genau definiert sind, wird in der Organisationslehre als Kompetenzprofil bezeichnet.

Wichtig bei der Abbildung der Organisationsstruktur ist, daß sowohl die im Betrieb beteiligten Personen, die Rollen als auch die Gruppen getrennt voneinander im System beschrieben werden können.

2.1.1.1 Agenten

Aus [Bußl93] wurde diese Bezeichnung für die Teilnehmer eines Workflow Systems übernommen, wo auch besonderer Wert auf die Einbindung der Organisation gelegt wird. Jeder Agent hat seine eigene Arbeitsliste, die alle noch auszuführenden Arbeitsschritte enthält. Menschliche Agenten (z.B. Sachbearbeiter) haben bei der Wahl der Arbeitsschritte einen Freiraum, indem sie die Reihenfolge der ihnen aufgetragenen Arbeitsschritte selbst bestimmen können. Software Agenten (z.B. Kalkulationsprogramm, Robotersteuerung) hingegen müssen immer einen Arbeitsschritt nach dem anderen aus ihrer Arbeitsliste ausführen, und zwar in der Reihenfolge, wie sie in die Arbeitsliste gestellt worden sind (siehe auch Abschnitt 2.2).

2.1.1.2 Rollen und Gruppen

Jeder Agent spielt innerhalb eines Betriebes eine oder mehrere *Rollen*. Ein Angestellter kann etwa sowohl Abteilungsleiter als auch Bilanzbuchhalter sein. Ein anderer ist „nur“ Feinmechaniker. Wesentlich dabei ist, daß solche Rollen einer

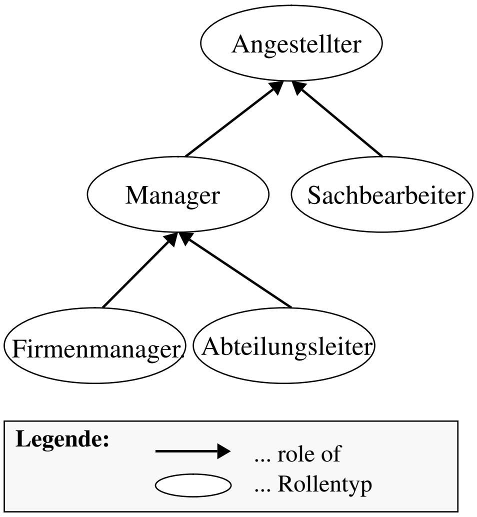


Abbildung 2.2: Rollentyphierarchie

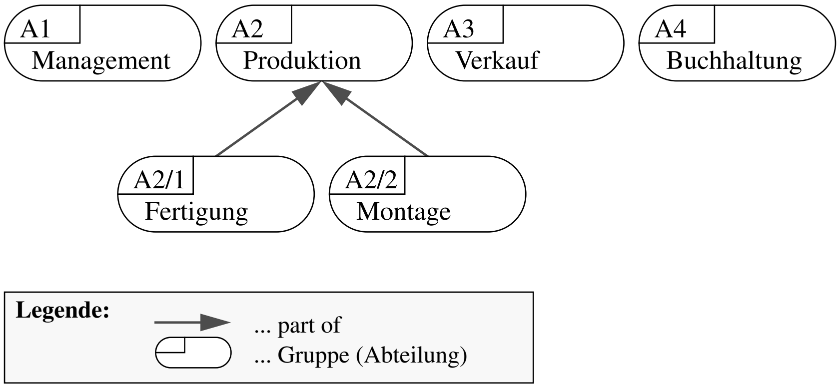


Abbildung 2.3: Abteilungsstruktur

bestimmten *Gruppe* zugeordnet sind. Im ersten Fall ist der Angestellte Abteilungsleiter der Abteilung Buchhaltung. Somit ist dessen Rolle „Abteilungsleiter“ der Gruppe „Buchhaltung“ zugeordnet. Die Organisationsstruktur eines Betriebes läßt sich also immer in Rollen und Gruppen unterteilen. Wenn sich Gruppen aus anderen zusammensetzen, so bilden sie eine „part of“-Hierarchie und beschreiben die Abteilungsstruktur (siehe Abbildung 2.3). Rollentypen bilden eine „subclass of“-Hierarchie, wenn das Verhalten spezialisiert wird (siehe Abbildung 2.2). Außerdem ist die Rollentyphierarchie zweigeteilt, in Rollentypen für menschliche Agenten und solche für Software Agenten. Um eine solche Organisation adäquat in einem Computersystem abbilden zu können, muß ein fundiertes Rollenkonzept zugrundeliegen. Ein Rollenkonzept wird später im Abschnitt 3.3 eingeführt und unter Punkt 4.1.2 näher beschrieben.

2.1.2 Aktivitätenmodellierung

Die *Aktivitätenmodellierung* (auch „Prozeßmodellierung“ genannt, siehe Abbildung 2.1) befaßt sich mit der Definition der Arbeitsabläufe („Was soll wann getan werden?“). Hauptaufgabe ist die Beschreibung der Arbeitsschritte und deren zeitliche Beziehungen durch Kontrollstrukturen. Wie im Software Engineering ist auch hier eine Top-Down-Vorgehensweise bei der Strukturierung der Prozesse sinnvoll. Da insbesondere bei der Modellierung in der untersten Ebene auch detaillierte Angaben über die zu bearbeitenden Datenstrukturen nötig sind, ist eine Kopplung mit einer Bottom-Up-Strategie und der Datenmodellierung unvermeidbar. Auch hier zeichnet sich wieder eine Verschiebung und Vermischung der Modellierungsschichten untereinander und auch deren Integration ab, die in der Entwicklung von Software durchaus üblich, ja sogar notwendig ist.

2.1.2.1 Aktivität

Eine *Aktivität* definiert einen Arbeitsschritt. Sie muß innerhalb eines Workflow Systems genau identifiziert werden können und ist deshalb einem Aktivitätennetz zugeordnet. Das wichtigste Element einer Aktivität ist die Aktion. Eine *Aktion* benennt eine Funktion, die bei einem Rollentyp definiert ist und bei der Bearbeitung der Aktivität ausgeführt wird.

Beispiel 2.1 *Der Rollentyp „Sachbearbeiter“ besitzt eine Funktion namens „Kreditantrag beschließen“, wobei dem Benutzer die zur Entscheidung über die Gewährung eines Kredites maßgeblichen Daten präsentiert werden. Bei der Bearbeitung der Aktivität „Entscheidung über Kreditgewährung“ durch einen Sachbearbeiter wird diese Aktion ausgeführt.*

Mit einer *Ausführungsbedingung* kann angegeben werden, unter welchen Umständen eine Aktivität ausgeführt werden soll. Sie bezieht sich meist auf die Daten des aktuellen Aktes (z.B. Kreditantrag: genaueste Haftungsüberprüfung nur bei sehr großen Krediten).

Aufgrund der Beschreibung der Reihenfolge der Aktivitäten im Aktivitätennetz gibt es drei Typen: *Startaktivität*, *Endeaktivität* und *Innere Aktivität*, die entweder Nachfolger, Vorgänger oder beides haben. Zur Ausnahmebehandlung wird noch eine *Ausnahmeaktivität* eingeführt.

Bei der Integration der Organisationsstruktur wird für jede Aktivität eine Organisationspolitik angegeben (siehe 2.1.4.1). Die Integration mit den Datenstrukturen (Datenflußbeschreibung) definiert jene Attribute des Akttyps, die gelesen (*input*) und geschrieben (*output*) werden (siehe 2.1.4.2).

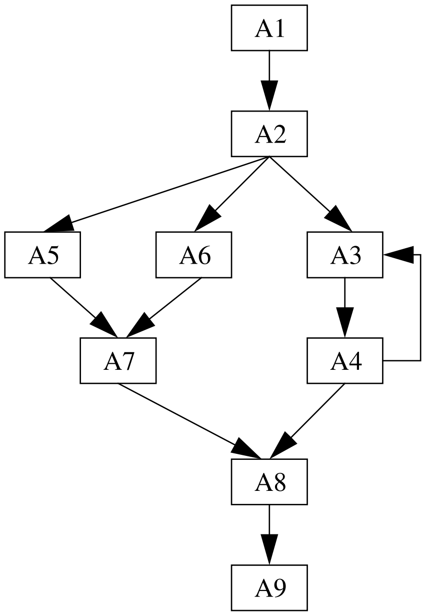


Abbildung 2.4: Aktivitätennetz

2.1.2.2 Aktivitätennetz

Ein *Aktivitätennetz* (siehe Abbildung 2.4) legt die Ausführungsreihenfolge der Aktivitäten fest und beschreibt somit den Kontrollfluß mit Hilfe dreier Kontrollstrukturen: Sequenz, Verzweigung und Zusammenführung. In [Rein93] wird ein Aktivitätennetz als bipartiter, gerichteter Graph mit einem Anfangs- und einem Endknoten, dessen Knoten die Aktivitäten und dessen Kanten die Ausführungsreihenfolge darstellen, bezeichnet. Die Kanten sind zudem noch gewichtet.

Es ist zwar notwendig, nur einen Startpunkt zu haben, es kann aber sehr wohl sinnvoll sein, mehrere Endpunkte zuzulassen, an denen die Ausführung endet. In einem Aktivitätennetz existiert höchstens eine Endeaktivität, die keine Beziehung zu den anderen Aktivitäten aufweist. Sie wird dann ausgeführt, wenn ein Aktenlauf einen Endknoten des Netzes erreicht. Die Endeaktivität stellt somit einen „impliziten Nachfolger“ aller Endknoten dar. Damit ist auch die Forderung von einem Anfangs- und einem Endknoten wieder erfüllt. Näheres dazu wird unter Punkt 2.2.2 beschrieben.

Der Graph muß nicht unbedingt bipartit sein, d.h. es gäbe nur Verzweigungen in zwei Pfade und Zusammenfassungen von zwei Pfaden. Warum soll eine Verzweigung nicht mehr als zwei Pfade aufweisen dürfen? Die Gewichtung der Kanten erfolgt nicht durch Zahlenangaben, sondern sie entsprechen den Ausführungsbedingungen der Aktivitäten.

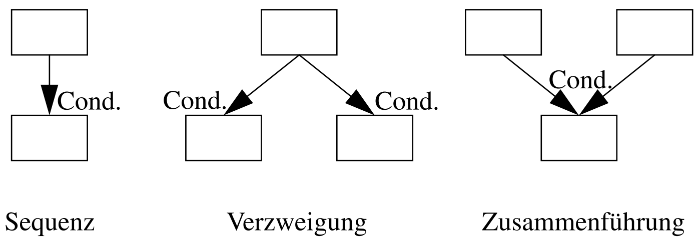


Abbildung 2.5: Die Kontrollstukturen: Sequenz, Verzweigung und Zusammenführung

2.1.2.3 Workflow Typ

Der *Workflow Typ* faßt alle drei Modellierungsebenen (Organisation, Aktivitäten und Datenstruktur) zusammen. Er besteht somit aus der konkreten Organisationsstruktur (Agenten, Rollen, Gruppen), einem Aktivitätennetz (Aktivitäten und deren Reihenfolge) und der behandelten Datenstruktur (Akttyp). Für jede zu unterstützende Art eines Geschäftsfalls ist ein Workflow Typ anzulegen. Organisation und Datenstruktur können natürlich mehrmals verwendet werden.

Ein *Workflow* ist ein konkreter Durchlauf eines Aktes durch das Aktivitätennetz des Workflow Typs. Er zeichnet also die Realität auf. Durch den Workflow Typ wird angegeben, wer welche Aktivität ausführen kann, deren Reihenfolge, und welche Daten dazu verwendet werden.

Nicht jeder Agent soll dazu berechtigt sein, einen Workflow zu starten. Aus diesem Grund wird auch beim Workflow Typ eine Organisationspolitik angegeben, die nur bestimmten Agenten die Initiierung erlaubt.

2.1.2.4 Kontrollfluß

Die Ausführungsreihenfolge von Aktivitäten ist durch deren Beziehung innerhalb des Aktivitätennetzes gegeben. Zur Beschreibung des Kontrollflusses gibt es, wie bereits erwähnt, folgende Basismechanismen (siehe Abbildung 2.5): Sequenz, Verzweigung und Zusammenführung.

Eine *Sequenz* bedeutet die Hintereinanderausführung von Aktivitäten. Wenn die eine fertig, d.h. abgearbeitet, ist, dann wird die Nachfolgende ausgeführt.

Gibt es mehrere Nachfolger einer Aktivität, so handelt es sich um eine *Verzweigung*. Von den angegebenen Verzweigungen werden nur jene Nachfolgeaktivitäten ausgeführt, deren Ausführungsbedingung wahr ist. Jede Ausführung der Aktivität ist also von dieser Bedingung abhängig. In der Literatur werden Verzweigungen immer als „and“-, „or“ und „xor“-Pfade angegeben. Logischen Operatoren stellen jedoch nur eine Belegung der Ausführungsbedingungen dar. Der Ansatz mit Ausführungsbedingungen ist wohl etwas praxisorientierter. Welche Belegung der Bedingungen der Nachfolgeaktivitäten welcher „logischen Ver-

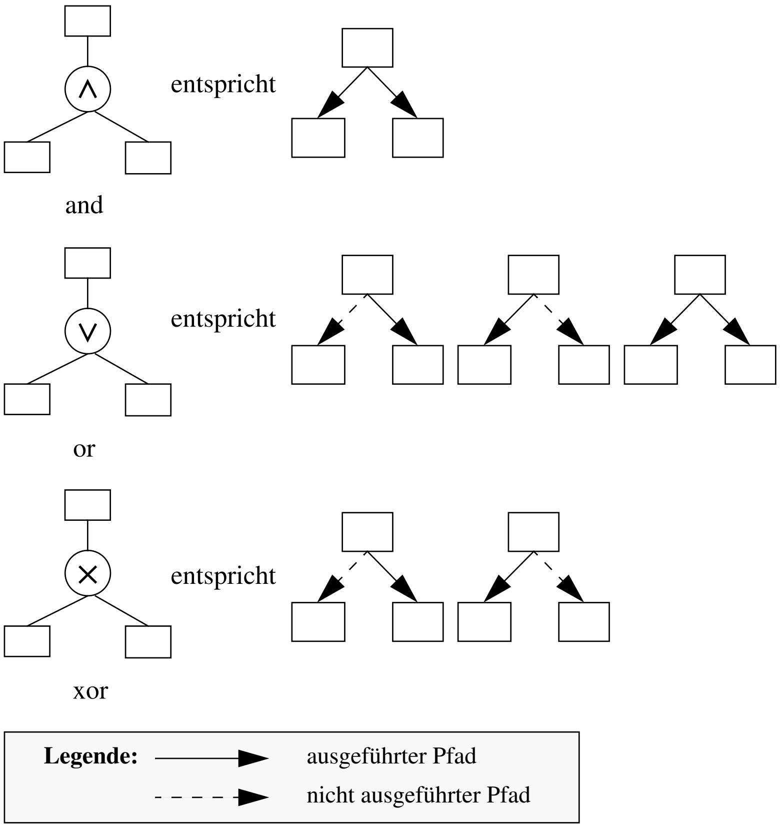


Abbildung 2.6: Vergleich von Verzweigung (mit Ausführungsbedingungen) und logischen Pfaden

zweigung“ entspricht, ist in Abbildung 2.6 dargestellt.

Das Pendant zur Verzweigung ist die *Zusammenführung*. Wie bei der Verzweigung werden auch hier in der Literatur die Pfade nur mit logischen Operatoren zusammengefaßt. In der Praxis wird es meist der Fall sein, daß alle Pfade, die bei der Verzweigung ausgeführt worden sind, hier zusammengefaßt und dann nach dem Beenden des letzten Pfades wieder fortgesetzt werden. Das bedeutet, daß alle Ausführungspfade zusammenwarten und dann erst fortgesetzt werden. In manchen Fällen kann es aber auch sinnvoll sein, verschiedene Zweige zu starten, den als ersten beendeten nach der Zusammenführung sofort fortsetzen zu lassen und alle anderen zu verwerfen.

Die logischen Zusammenführungen richten sich dabei nach der vorhergehenden Verzweigung. Dabei kann sehr einfach ein Deadlock erzeugt werden, wie das Beispiel 2.2 zeigt. Die Probleme liegen da aber an der nicht semantischen Definition der logischen Operatoren. In der Realität darf so eine Unstimmigkeit nicht passieren, deshalb werden die konkreten Belegungen der Pfade bei der Zusammenführung berücksichtigt und nur jene mit einbezogen, die auch tatsächlich ausgeführt wurden.

Beispiel 2.2 *Wenn eine Verzweigung als „xor“ definiert wird und unmittelbar* *nachher wieder mit „and“ zusammengeführt wird, so entsteht immer ein Deadlock. Die Zusammenführung wartet auf beide Pfade, bei der Verzweigung wird aber immer nur ein Pfad ausgeführt.*

Will man aber tatsächlich auch eine „or“-Zusammenführung darstellen, so genügt es nicht die Bedingung bei der Aktivität zu definieren, sondern jede Kante des Aktivitätennetzes muß damit gewichtet werden. Die Ausführungsbedingung wird dann zu einer Vorgänger-Nachfolger-Bedingung zweier Aktivitäten. Es gibt einen Grund dafür, warum nicht gleich die Bedingungen an die Kanten, sondern an die Aktivitäten gebunden wurden: Wenn bei einer solchen Zusammenführung ein Zweig übernommen und die Arbeit des anderen Zweiges verworfen wird, so müssen die durch den verworfenen Zweig geänderten Daten wieder in den Zustand vor seiner Ausführung zurückgesetzt werden. Im Rahmen der Entwicklung des in Kapitel 4 beschriebenen Prototyps eines Workflow Systems war es nicht möglich, eine Graphanalyse durchzuführen, um die Teilgraphen herauszusuchen, aufgrund deren Ausführung bestimmte Daten geändert wurden, die jetzt wieder zurückgesetzt werden müssen. Eine vollständige Graphanalyse stellt ein sehr umfangreiches Unterfangen dar, das einerseites den Rahmen dieser Arbeit sprengen, andererseits zur Zeit fehlende Vorraussetzungen in bezug auf das Transaktionskonzept des darunterliegenden Datenbanksystems verlangen würde.

Eine durch die Zusammenführung „xor“ herbeigeführte Verwerfung eines bereits begonnenen Pfades ist nicht möglich. Alle einmal begonnenen Pfade müssen auch zu Ende geführt werden.

Da ohne dieses Transaktionskonzept eine andere Realisierung als die „and“-Zusammenführung nicht möglich ist, wird im Prototyp nur diese behandelt.

In der Praxis kommt es häufig vor, daß eine oder mehrere Aktivitäten wiederholt auszuführen sind, solange eine bestimmte Bedingung nicht erfüllt ist.

Beispiel 2.3 *Review-Prozeß der Artikelannahme bei Fachzeitschriften: Solange die Reviewer der Zeitschrift der Veröffentlichung nicht zustimmen, muß der Autor seinen Artikel verbessern oder zurückziehen.*

Dieser Arbeitsablauf kann nur durch Einbringen von *Schleifen* in das Aktivitätennetz gelöst werden (siehe Abbildung 2.7). Eine Schleife ist eine besondere Art einer Verzweigung, bei der ein Folgepfad zu einer direkten oder indirekten Vorgängeraktivitität führt. Man erzeugt Schleifen, indem als Nachfolgeaktivität auch jene Aktivität angegeben wird, bei der im Falle einer falschen Bedingung wieder angefangen werden soll. Dies kann sich natürlich über mehrere Aktivitäten erstrecken. Dabei sind zwei Dinge zu beachten:

- Die Ausführungsbedingungen der Nachfolgeaktivitäten müssen komplementär sein, um ein Ende der Schleife (semantisch) zu garantieren.

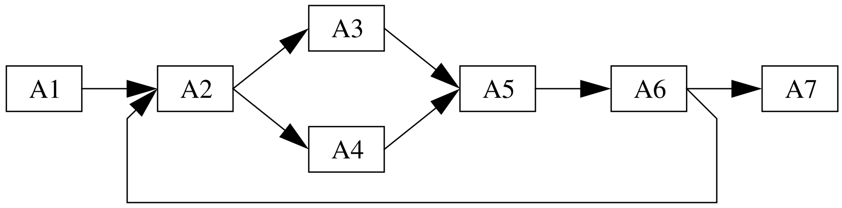


Abbildung 2.7: Schleifen im Aktivitätennetz

- Große Schleifen (über viele Aktivitäten und Pfade hinweg) erschweren die Lesbarkeit und vermindern die Übersichtlichkeit eines Aktivitätennetzes und sollen deshalb nur gut überlegt eingeführt werden.

2.1.3 Datenmodellierung

Die Datenobjekte, die von den Agenten bearbeitet werden, sind Akte. Ein *Akt* entspricht einer elektronischen Umlaufmappe, die durch ein Aktivitätennetz wandert. In einem Workflow Typ kann nur ein *Akttyp* verwendet werden, es können aber sehr wohl mehrere Workflow Typen denselben Akttyp benutzen. Jeder Akttyp soll neben den spezifischen zu bearbeitenden Daten folgende Standardinformationen enthalten:

- Ausführungsstatus
- Agent, der den Workflow gestartet hat
- Verarbeitungshistorie
- Kommentare der Agenten

Da über diese Informationen jeder Akt verfügen muß, entsteht wieder eine Hierarchie, an dessen Spitze der Standardakt steht (siehe Abbildung 2.8). Jeder in einem Workflow verwendete Akttyp ist eine direkte oder indirekte Spezialisierung des Standardaktes.

In der Verarbeitungshistorie werden Daten wie Verteilungszeitpunkt, Verarbeitungsbeginn und -ende für jede durchlaufene Aktivität gesammelt. Jedem Agenten sollte auch ermöglicht werden, Kommentare zur Bearbeitung anzugeben, die für nachfolgende Anwender oder für den ganzen Ablauf von Nutzen sein können. Für die Bearbeitung kann es auch nützlich sein zu wissen, wer einen Akt angelegt und somit den Workflow gestartet hat. Dieser möchte dann am Ende der Bearbeitung eventuell davon informiert werden.

2.1.4 Integration von Organisation, Aktivitäten und Daten

Nach der Modellierung der einzelnen Schichten Organisation, Aktivitäten und Daten muß bestimmt werden, wer welche Aktivität ausführen soll, und wel-

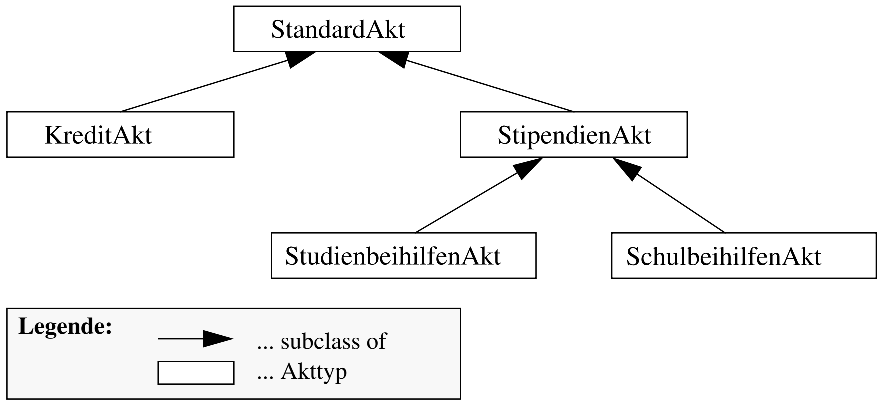


Abbildung 2.8: Hierarchie von Akttypen

che Daten des Akttyps in einer Aktivität gelesen oder geschrieben werden. Die Verbindung von Aktivitäten und Organisation wird unter dem Begriff *Organisationspolitik* zusammengefaßt, die Verbindung von Aktiviäten und Daten wird als *Datenflußbeschreibung* bezeichnet (siehe Abbildung 2.1).

2.1.4.1 Organisationspolitiken

Jablonski und Bußler beanstanden in [Bußl94], daß die Zuordnung von Aktivitäten und Agenten in den meisten Systemen zu statisch ist, wenn zu jeder Aktivität der ausführende Agent gespeichert wird. Sie stellen eine wesentlich flexiblere Art der Verbindung Aktivität-Agent vor, und zwar mit Hilfe von Organisationspolitiken. Eine *Organisationspolitik* besteht aus einer Menge von Bedingungen, die ein Agent erfüllen muß, um eine Aktivität ausführen zu können. Eine Organisationspolitik wird immer zur Laufzeit ausgewertet, und jene Agenten, die diese Politik erfüllen, werden als mögliche Ausführende identifiziert. Für jede Aktivität muß eine solche Politik angegeben werden.

Abfragekategorien:

Grob eingeteilt, gibt es drei verschiedene Kategorien von Abfragen in einer Politik, die wahlweise — wie ein boolscher Ausdruck — miteinander verknüpft werden können:

nummernbezogen: Ein Agent wird durch seine innerhalb der gesamten Organisation eindeutige Nummer identifiziert und zur Ausführung ausgewählt.

rollen- und/oder gruppenbezogen: Jene Agenten werden ausgewählt, die eine bestimmte Rolle in der angegebenen Gruppe spielen oder auch nur eine Rolle spielen (egal in welcher Gruppe) oder nur einer bestimmten Gruppe angehören (egal in welcher Rolle).

workflow- und aktivitätenbezogen: Hier sollen Agenten bestimmt werden können, die eine bestimmte Aktivität schon einmal ausgeführt haben oder eventuell auch einen bestimmten Workflow gestartet haben.

Zusätzlich muß es noch Hilfsfunktionen geben, die Werte für obige Abfragen liefern, damit möglichst alle Abfragemöglichkeiten abgedeckt sind.

Komplexe Politik:

Viele Politiken setzen sich aus mehreren Bedingungen zusammen und müssen deshalb geeignet verknüpft werden können. Die oben angeführten Abfragefunktionen können mit logischen Operatoren — wie etwa *and* und *or* — unter den allgemein gültigen Verknüpfungs- und Vorranggesetzen zu *komplexen Organisationspolitiken* zusammengesetzt werden.

Beispiel 2.4 *Der Manager der Produktionsabteilung oder der Manager der Verkaufsabteilung sollen einen bestimmten Arbeitsschritt ausführen.*

Kontextabhängige Politik:

Will man etwa die aktuellen Daten des Aktes auch in die Auswahl der Agenten mit einbeziehen, dann bedient man sich dem Konstrukt der *kontextabhängigen Organisationspolitik*. Mit Hilfe von Funktionen kann man den Bezug zu anderen Akten oder zum aktuellen Akt herstellen und darauf wieder Bedingungen aufsetzen.

Beispiel 2.5 *Wenn der Wert des aktuellen Kreditantrages ÖS 500.000,- übersteigt, dann soll der Leiter der Bank über die Genehmigung entscheiden, ansonsten der involvierte Sachbearbeiter. Der Kreditantrag ist hier das Datenobjekt, dessen Attribute für die Aufgabenverteilung herangezogen werden.*

2.1.4.2 Datenflußbeschreibung

Innerhalb eines Aktivitätennetzes kann nur eine Art von Akt (*Akttyp*) verwendet werden. Deshalb müssen alle im Laufe der Bearbeitung notwendigen Attribute eines Aktes zuvor im Akttyp definiert werden.

Der *Datenfluß* beschreibt nun, bei welcher Aktivität welche Attribute eines Aktes gelesen und geschrieben werden. Die involvierten Attribute werden in lesende und schreibende Attribute getrennt und bei jeder Aktivität angegeben. Beim Design muß darauf geachtet werden, daß bei parallel ausgeführten Aktivitäten die Daten nicht gegenseitig überschrieben werden. Um dabei alle möglichen Fehlerquellen auszuschalten, muß die Menge der zu lesenden Attribute (I ...input) und die Menge der zu schreibenden Attribute (O ...output) zweier paralleler Aktivitäten (i, j) folgende Bedingungen erfüllen:

$$I_i \cap O_j = \emptyset \text{ und } O_i \cap I_j = \emptyset \text{ und } O_i \cap O_j = \emptyset$$

Die gelesenen Attribute einer Aktivität und die geschriebenen Attribute der dazu parallelen Aktivität müssen einander wechselseitig ausschließen, und die geschriebenen Attribute beider Aktivitäten dürfen sich auch nicht überschneiden. Diese Konsistenzüberprüfung soll bereits bei der Definition der Aktivitäten durchgeführt werden.

2.1.5 Wiederverwendung von Workflow Komponenten

Bei jeder Modellierung eines Systems versucht man, dieses in zusammengehörende Teile zu zerlegen, die dann wiederverwendet werden können. Der Begriff *Wiederverwendung* ist ein wesentlicher Bestandteil des Software Engineering [McGr92]. In verschieden Stufen wurde über Prozeduren, Module, Datenkapseln bis hin zur Objektorientierung die neuerliche Verwendung bereits entwickelter Teile ermöglicht.

In Workflow Systemen können ebenfalls verschiedene Teile wiederverwendet werden. Agenten, Rollen, Gruppen, Aktivitäten und Akttypen sind Workflow Komponenten, die auf ihre Art und Weise wiederverwendet werden können.

2.1.5.1 Wiederverwendung der Organisation

Jeder Agent wird einem Betrieb zugeordnet. Da ein und derselbe Betrieb im Regelfall für mehrere Workflow Typen Geltung hat, kann man von einer Wiederverwendung von Agenten sprechen. Agenten selbst können aber zugleich nicht verschiedenen Betrieben angehören. Die Rollentyphierarchie kann in der Regel für mehrere Betriebe verwendet werden. Die Abteilungsstruktur gilt in der Regel aber nur für einen bestimmten Betrieb.

2.1.5.2 Wiederverwendung der Aktivitäten

Bei der Wiederverwendung von Aktivitäten gibt es Probleme. Denn jede Aktivität ist genau einem Aktivitätennetz bzw. einem Workflow Typ zugeordnet und kann somit nicht von mehreren Aktivitätennetzen gleichzeitig benutzt werden. Außerdem werden bei jeder Aktivität die Vorgängeraktivitäten gespeichert. Ein Sprung mitten in ein anderes Aktivitätennetz ist deshalb nicht möglich. Es können aber sehr wohl mehrere Workflow Typen miteinander gekoppelt werden. In einer Aktivität eines Workflow Typs wird dann ein anderer Workflow gestartet. Will man Teile eines Workflows in einem anderen verwenden, so muß ersterer in kleinere Einheiten zerlegt werden. Dann können Teile des ersten Workflows in den zweiten eingebunden werden.

Um auch eine Mehrfachverwendung einzelner Aktivitäten zu gestatten, müßte das ganze Konzept einem Redesign unterzogen werden und dabei geklärt werden, ob es wirklich notwendig ist, eine Aktivität genau einem Aktivitätennetz zuzuordnen.

2.1.5.3 Wiederverwendung der Akttypen und der Akte

Ein Akttyp kann von mehreren Workflow Typen verwendet werden. Durch die Spezialisierung kann auch die Struktur der Datenobjekte jederzeit weiterverwendet werden. Dazu muß der neue Akttyp als Subtyp eines bereits vorhandenen Akttyps definiert werden. Ähnlich dazu wird jeder Akttyp vom Standardakttyp abgeleitet. Die zusätzlichen Attribute werden dabei im Subtyp angegeben.

Auf die Daten der Akte anderer Workflows kann in den Aktivitäten auch zugegriffen werden. Dafür müssen lediglich geeignete Operationen auf den entsprechenden Objekten zur Verfügung gestellt werden.

2.2 Ablauf

Wenn Organisation, Aktivitäten und Daten modelliert und in Wortkflow Typen zusammengefaßt sind, dann können sie mit Hilfe der Ablaufsteuerung ausgeführt werden. Die Ausführung wird allgemein auch als „Enactment“, und die Zeit als „Run Time“ bezeichnet.

Zum Ablauf des Workflow Systems gehören alle Vorgänge, die zur Laufzeit stattfinden. Der „gewöhnliche Ablauf“ (ohne Sonderfälle) geht folgendermaßen vor sich: Jeder menschliche Agent benutzt ein Programm, das ihm seine auszuführenden Arbeitsschritte anzeigt. Hier liegt der wesentliche Unterschied zwischen den zwei Agententypen menschlicher Agent und Software Agent. Menschen können die Arbeitsschritte in beliebiger Reihenfolge auswählen, sofern nicht Einschränkungen wie Zeitbeschränkungen oder Prioritäten angegeben werden. Software Agenten hingegen entnehmen die Aufgaben in der Reihenfolge, in der sie in die Arbeitsliste gestellt werden, d.h. z.B. nach der FIFO-Methode („First In First Out“) und auch nur dann, wenn keine Einschränkungen vorliegen. Menschen arbeiten interaktiv mit einem Programm, das ihnen die Auswahl von Arbeitsschritten und das Starten eines Ablaufs erlaubt. Jeder Software Agent wird durch einen übergeordneten Prozeß gesteuert, der vom Workflow System Administrator überwacht wird. Software Agenten sind „Software-Objekte“ ohne Pendant in der Realität, wie z.B. ein `Calculator` und `PrinterSpooler`, oder „Software-Objekte“ mit Pendant in der Realität, wie z.B. eine `DrillingMachine` zur softwaremäßigen Steuerung einer realen Bohrmaschine.

2.2.1 Beschreibung eines Durchlaufs im Aktivitätensystem

Jede Aktion in einem Workflow System beginnt damit, daß ein menschlicher Agent interaktiv bzw. ein Software Agent automatisch eine Aktivität aus seiner Arbeitsliste auswählt. Das System führt hierauf die folgenden Schritte durch, sofern keine Fehler auftreten. Die Fehlerbehandlung wird später erläutert.

2.2.1.1 Ausführung (Execution)

Zuerst werden jene Daten, die laut Datenflußbeschreibung gelesen werden, der Datenbank entnommen (*Datenbereitstellung*). Die bereitgestellten Daten werden dem auszuführenden Programm übermittelt, das dann gestartet wird (*Aktionsausführung*). Nach erfolgreichem Beenden der Anwendung (z.B. Texteditor) werden die in der Datenflußbeschreibung als Ausgabeattribute deklarierten Daten in den Akt übernommen und in die Datenbank abgelegt (*Datenübernahme*).

2.2.1.2 Arbeitsverteilung (Work Distribution)

Für jede Aktivität, die der gerade ausgeführten nachfolgt und deren Ausführungsbedingung erfüllt ist, werden die nachfolgenden Schritte ausgeführt. Wann die Arbeitsverteilung stattfindet, bestimmt die Art der Kontrollstruktur, die der gerade ausgeführten Aktivität vorausgeht. Auf die Kontrollstrukturen und die damit verbundenen Probleme wurde bereits unter Punkt 2.1.2.4 eingegangen. Hier sollen aber noch einmal die für die Arbeitsverteilung relevanten Aspekte angeführt werden.

Bei Sequenz und Verzweigung wird die Arbeitsverteilung sofort nach Ende der Ausführung vorgenommen. Handelt es sich aber um eine Zusammenführung, muß deren Art beachtet werden. In den meisten Fällen wird die Arbeitsverteilung erst dann erfolgen, wenn alle Aktivitäten der zusammengeführten Pfade ausgeführt sind. Es wird also zusammengewartet, und erst beim Beenden der letzten Vorgängeraktivität wird weiter verteilt. Es kann aber auch nützlich sein, die Bearbeitung fortzuführen, sobald ein Pfad beendet ist, die Ergebnisse der anderen Pfade zu verwerfen und die noch offenen Arbeitsschritte aus den Arbeitslisten der Agenten herauszunehmen. Dieser Fall tritt nur selten ein und ist wegen der dynamischen Änderung der Arbeitslisten aufwendiger. Um feststellen zu können, ob alle Pfade einer Zusammenführung bereits fertig ausgeführt sind, ist eine umfangreiche Graphanalyse notwendig. Dabei wird das Aktivitätennetz in jene Teilbäume aufgespalten, die bis zur Zusammenführung laufen, und dann die Fertigstellung der Aufgaben überprüft. Über die Analyse von Graphen und allgemein zur Graphentheorie gibt es einschlägige Literatur [Dörf77], sie wird hier nicht weiter behandelt.

Nachdem die Nachfolgeraktivität(en) festgestellt ist (sind), werden alle zur Ausführung in Frage kommenden Agenten durch Auswerten der Organisationspolitik bestimmt (*Agent Selection*). Erfüllt kein Agent die Politik, so wird eine Ausnahmebehandlung gestartet und der Workflow muß gestoppt werden. Dafür gibt es zwei Gründe: entweder steht tatsächlich zur Zeit kein geeigneter Agent, etwa wegen Urlaub oder Krankheit, zur Verfügung oder es wurden bei der Modellierung Fehler gemacht (Definition von Organisationspolitiken, die nie erfüllt werden können). In einer derartigen Ausnahmesituation wird in der Regel der Workflow System Administrator darüber informiert. Er kann dann Gegenmaßnamen treffen, was später unter den Sonder- und Ausnahmefällen noch genauer erläutert wird.

Sind die Agenten erst einmal ausgewählt, so muß jener Agent bestimmt werden, in dessen Arbeitsliste die Aktivität gestellt werden soll (*Agent Binding*). Auch hier gibt es wieder mehrere Möglichkeiten: Eine Möglichkeit ist, daß derjenige Agent die Aktivität ausführen soll, dessen Arbeitsliste die wenigsten Arbeitsschritte enthält. Eine Alternative ist, daß man mehreren Agenten die Aufgabe überträgt und sobald der erste diese Aktivität zur Ausführung auswählt, diese von den Arbeitslisten der anderen informierten Agenten entfernt wird. Eine weitere Möglichkeit besteht darin, wieder mehrere Agenten zu informieren, und alle ausführen zu lassen, aber nur die Ergebnisse jenes Agenten zu übernehmen, der als erster die Aufgabe beendet.

Wenn die Agenten zugeordnet sind, müssen sie benachrichtigt werden (*Agent Notification*). Benachrichtigen heißt, die Aktivität zur Arbeitsliste der zugeordneten Agenten hinzufügen.

2.2.2 Sonderfälle im Ablauf

Die zuvor beschriebene Ausführung einer Aktivität und die anschließende Verteilung der Nachfolgeaktivitäten trifft lediglich für die gewöhnliche Exekution von Arbeitsschritten (ohne Auftreten von Ausnahmesituationen) zu. Bei der vollständigen Abarbeitung eines Vorgangs gibt es mehrere Sonderfälle.

1. Bevor die erste Aktivität (Startaktivität) des Workflows ausgeführt werden kann, muß diese durch eine entsprechende Funktion des Workflow Typs gestartet werden. Die Software für den Agenten listet dazu alle dem System bekannten Workflow Typen auf. Der Agent kann aber nur bestimmte davon starten. Die Berechtigung dazu wird, wie in der Modellierung bereits besprochen, durch eine Organisationspolitik ausgedrückt. Wenn der Agent einen Workflow starten will, wird die Organisationspolitik ausgewertet. Befindet er sich unter den ausgewählten Agenten, so wird der Workflow gestartet und damit die Startaktivität verteilt, denn der Durchlauf beginnt mit der Verteilung der Startaktivität und nicht mit der Exekution.

2. Das Ende eines Workflows ist erreicht, wenn eine Aktivität keine Nachfolgeaktivität mehr besitzt. Hierauf wird die Endeaktivität verteilt, mit deren Hilfe meist ein Agent vom Ende des Workflows benachrichtigt wird. Informiert kann etwa jener Agent werden, der den Workflow gestartet hat.

3. Ein weiterer wichtiger Sonderfall entsteht, wenn die Ausführungsbedingungen aller Nachfolgeaktivitäten nicht erfüllt sind. Es gibt zwar Nachfolger, aber keiner kann verteilt werden. In diesem Fall liegt entweder ein Modellierungsfehler oder ein durchaus gewünschter Effekt vor. Es wird hierauf nämlich die Ausnahmeaktivität ausgeführt, wodurch z.B. der Workflow System Administrator davon informiert wird.

   Eine zweite Ausnahme ist das Stoppen eines Workflows, wenn kein Agent die Organisationspolitik erfüllt. Dieses Ereignis wird später noch im Zusammenhang mit den Einsatzmöglichkeiten eines Triggersystems in Kapitel 4 behandelt. Wenn neue Agenten angelegt oder bereits vorhandene wieder aktiviert werden, dann werden alle zur Zeit gestoppten Workflows gesucht, die Organisationspolitiken der Aktivitäten, an denen gestoppt wurde, neu ausgewertet, und wenn möglich die Arbeitsverteilung fortgesetzt.

   In einem solch komplexen System wie einem Workflow System kann natürlich eine Vielzahl von Fehlern auftreten, die alle zu einer Ausnahmesituation führen. Einerseits gibt es leider immer wieder Systemfehler durch defekte Hard- oder Software, andererseits treten auch in der Modellierung von Organisation und Abläufen Fehler auf, die später zu einer Ausnahme führen. In [Good75], [Karb90] und [Saas92] wird ausführlicher über die Arten von Ausnahmen und deren Behandlung berichtet. In dieser Arbeit wird unter Punkt 3.2.3 noch einmal auf die Ausnahmebehandlung im Zusammenhang mit dem Transaktionsmanagement von Datenbanksystemen eingegangen.

Bevor der Aufbau des entwickelten Workflow System Prototyps näher beschrieben wird, werden im nächsten Kapitel die Realisierungsgrundlagen und die dabei verwendeten Konzepte kurz erläutert.

Kapitel 3

Basiskonzepte und Realisierungsgrundlagen

Bei der Entwicklung des Workflow System Prototyps wurde besonderer Wert darauf gelegt, vorhandene Konzepte und Technologien zu übernehmen und wenn nötig zu adaptieren. Der Prototyp wurde nicht objektorientiert aufgebaut, um dem „Trend“ zu entsprechen, sondern es wurden ganz bewußt die Vorteile einer objektorientierten Entwicklung genutzt. Einerseits, um den Entwurf zu vereinfachen und die Flexibilität zu erhöhen, und andererseits, um bereits bestehende Konzepte oder Software-Komponenten überhaupt einbinden zu können.

Dieses Kapitel gliedert sich in vier Teile. Als erstes und wohl wichtigstes Konzept wird die Objektorientierung besprochen, und welchen Einfluß sie im entwickelten Prototyp nimmt. Der zweite Teil behandelt die Grundzüge von Datenbanksystemen, wie Mehrbenutzerfähigkeit, Transaktionsmanagement, Ausnahmebehandlung und Triggerkonzept. Der dritte Teil behandelt das bereits mehrmals angesprochene Rollenkonzept, und im viertenTeil wird die Integration von vorhandenen Systemen erläutert.

3.1 Objektorientierung

Die Grundaspekte des objektorientierten Konzepts werden als bekannt vorausgesetzt. Deshalb werden nur noch die zentralen Eigenschaften und deren Verwendung im Workflow System beschrieben. Ansonsten wird auf einschlägige Literatur wie etwa [Nier93] verwiesen.

Die *Kapselung von Daten und Funktionen* wird bei Objekten als selbstverständlich angesehen. Dennoch spielt diese, gerade bei der Verwendung des objektorientierten Datenbanksystems GemStone [Butt91, Serv91], eine große Rolle. Hier werden im Gegensatz zu den meisten anderen objektorientierten Datenbanksystemen nicht nur die Daten, sondern auch der Code der Methoden in die Datenbank abgelegt. Damit befinden sich die Objekte wirklich zur Gänze in der Datenbank, und nicht wie in anderen Systemen die Daten in der Datenbank und die Methoden in externen Codedateien. Dies hat den Vorteil, daß die Eigenschaften des Datenbanksystems auch für die Methoden eines Objekts gelten, wie etwa Fehlerbehebung und Wiederanlauf („Recovery").

Die *Vererbung* von Attributen und Funktionen bei der Klassendefinition ist eine der wichtigsten Eigenschaften der Objektorientierung. Die dabei entstehende Hierarchie wird auch als „subclass of"-Hierarchie bezeichnet. Sie wird unter anderem zur Spezialisierung von Akttypen verwendet. Die Akttypen werden dabei von einem „Standardakt" abgeleitet und erhalten damit dessen sämtliche Attribute und Funkionen. Auch die Gruppentyphierarchie (siehe Kapitel 4) wird durch Vererbung gebildet, jedoch nicht die Struktur der Abteilung einer Betriebsorganisation, sondern lediglich die Spezialisierung der Abteilungen. Diese werden von einer „Standardabteilung" abgeleitet, damit etwaige zusätzliche Attribute hinzugefügt werden können. Der Workflow System Prototyp benötigt vorerst keine Gruppentyphierarchie, sehr wohl aber die Abteilungsstruktur. Eine Gruppentyphierarchie kann aber z.B. für ein Betriebsinformationssystem von Nutzen sein.

Zum Thema Hierarchien muß gleich eines vorausgeschickt werden: Eine Rollentyphierarchie des Workflow System Prototyps entspricht nicht einem typischen Vererbungsbaum der objektorientierten Programmierung, da die Vererbung nicht auf Klassen- sondern auf Instanzebene durchgeführt wird. Dieser Umstand wird unter Punkt 4.1.2 näher erläutert.

Ein weiterer Punkt in der objektorientierten Programmierung ist die erleichterte *Wiederverwendbarkeit* bereits definierter Klassen [Gamm92]. Diese Eigenschaft findet zum Beispiel in der Spezialisierung der Akttypen Verwendung. Innerhalb eines Ablaufs darf nur ein Akttyp, dieser aber sehr wohl in mehreren Prozessen (Workflow Typen) verwendet werden. Analog dazu ist auch die Wiederverwendung bereits definierter Funktionen in der Rollentyphierarchie und Attributen in der Gruppentyphierarchie zu erwähnen.

3.2 Datenbanksysteme

Alle Daten, die für den Ablauf des Workflow Systems notwendig sind und die damit verbundenen Anwendungsdaten, werden in einer Datenbank gesammelt. Wie oben bereits erwähnt, werden aber nicht nur die Daten, sondern durch die Verwendung des objektorientierten Datenbanksystems GemStone auch die Funktionen in die Datenbank abgelegt. Mit Ausnahme der Benutzeroberflächen und der anderen externen Anwendungsprogramme liegt das gesamte Workflow System in einer Datenbank. Aus diesem Grund ist ein Schwerpunkt der Arbeit das Festlegen der Anforderungen an ein Datenbanksystem, um ein Workflow System vernünftig darauf aufbauen zu können. Im Folgenden werden die wichtigsten Eigenschaften eines Datenbanksystems auf ihre Tauglichkeit in Workflow Systemen untersucht und kurz die entstehenden Probleme aufgezeigt (siehe auch [Rein93, Voss91, Öszu91]).

Der Einsatz und die Einbindung verschiedener Komponenten in ein Workflow System verlangen nach der Verwendung von verteilten Datenbanksystemen. Die Anforderungen an ein Workflow System erfordern sogar ein „voll verteiltes Datenbanksystem". Voll verteilt sind Datenbanksysteme, wenn sie heterogen, physisch verteilt und autonom sind. Heterogen bedeutet, daß Systeme verschiedener Hersteller oder Systeme mit verschiedenen Konzepten miteinander verwendet werden. Ein Datenbanksystem ist physisch verteilt, wenn die Daten auf mehreren Knoten eines Netzes liegen, und autonom, wenn jedes System auf den Knoten über eigene Kontrolle verfügt. Zur Zeit wird kein Datenbanksystem allen diesen Anforderungen zugleich gerecht. Die jetzigen Systeme vernachlässigen die eine oder andere Anforderung. Diese Arbeit beschränkt sich auf die Verwendung eines zentralisierten Datenbanksystems, unter Punkt 3.4 wird aber nocheinmal allgemein auf die Verteilung von Systemen eingegangen.

3.2.1 Mehrbenutzerfähigkeit

Selbstverständlich sind ins Workflow Management viele Agenten involviert, und viele arbeiten zur gleichen Zeit mit dem System, indem sie ihre Arbeitsschritte ausführen oder Änderungen am System durchführen. Das Datenbanksystem muß die Aktivitäten der verschiedenen Benutzer ordnungsgemäß ausführen können. Alle damit verbundenen Maßnahmen, wie Sperren und Freigeben von Objekten, Kommunikation zwischen Agenten usw. müssen teils vom Anwendungsprogramm (hier: Workflow System) teils vom Datenbanksystem getroffen werden. Die Unterstützung mehrerer Benutzer wird allgemein als „Mehrbenutzerfähigkeit" bezeichnet und ist somit eine Grundvoraussetzung für den Aufbau eines Workflow Systems.

3.2.2 Transaktionsmanagement

Um ein Workflow System vor Datenverlust zu schützen, werden viele Daten persistent in der Datenbank gehalten. Die Garantie für die Aktualität und Richtigkeit von Daten muß das Datenbanksystem abgeben. Dazu wird ein Transaktionskonzept verwendet. Eine Transaktion ist eine Zusammenfassung von Operationen auf Daten, die von einem konsistenten Zustand der Datenbank ausgehend diese wieder konsistent hinterlassen. Eine Transaktion kann nicht unterbrochen, sondern nur vollendet oder ganz zurückgesetzt werden. Daraus resultieren die unter dem Namen ACID bekannten Eigenschaften von Transaktionen:

Atomarität (Atomicity): Die Transaktion wird entweder vollständig ausgeführt oder zurückgesetzt.

Konsistenz (Consistency): Die Datenbank befindet sich vor und nach Ausführung einer Transaktion in einem konsistenten Zustand.

Isolation (Isolation): Jedem Benutzer wird eine eigene Sicht auf die Daten zugeteilt. Die darauf ausgeführten Transaktionen laufen unabhängig von denen anderer Benutzer ab.

Dauerhaftigkeit (Durability): Nach Beendigung einer Transaktion stehen die Ergebnisse persistent in der Datenbank.

Die Isolation der Transaktion ist besonders für den zuvor beschriebenen Mehrbenutzerbetrieb wichtig. In diesem Zusammenhang werden in Datenbanksystemen verschiedene Protokolle zur Sperre und Freigabe von Objekten angegeben, z.B. das Zweiphasen-Sperrprotokoll. Das im Prototyp verwendete Datenbanksystem unterstützt kein solches Protokoll, Objekte können aber sehr wohl gesperrt und freigegeben werden (siehe dazu Kapitel 4). Atomarität, Konsistenz und Dauerhaftigkeit sind für die Fehlerbehandlung von großer Bedeutung. Wenn Systemfehler auftreten, soll sichergestellt sein, daß der Zustand vor dem Fehlerzeitpunkt wieder ganz hergestellt werden kann. Bei Datenbanksystemen fällt dies unter den Begriff Wiederanlauf („Recovery“).

Diese ACID-Transaktionen sind geeignet für kurze Verarbeitungseinheiten im Mehrbenutzerbetrieb. In Workflow Systemen stößt dieses Transaktionskonzept jedoch sehr schnell an seine Grenzen, wenn es darum geht, langdauernde Transaktionen — ein Workflow ist eine langdauernde Transaktion — geeignet zu unterstützen. Die Atomarität von Transaktionen stellt dabei ein großes Hindernis dar, weil ein Workflow in mehreren Transaktionsschichten ablaufen soll. Einerseits soll ein Workflow in einer Transaktion ablaufen, damit die Kontrollstrukturen Sequenz, Verzweigung und Zusammenführung in einer Transaktion zusammengefaßt werden können, um im Fehlerfall die korrekte Wiederherstellung des gesamten Ablaufs gewährleisten zu können. Andererseits dürfen innerhalb einer Transaktion keine konkurrierenden Operationen, wie sie durch eine Verzweigung entstehen, auftreten. Logisch gesehen ist jede Aktivität eines Aktivitätennetzes eine eigenständige Verarbeitungseinheit und muß somit in einer Transaktion ablaufen. Das herkömmliche ACID-Konzept unterstützt aber keine geschachtelten Transaktionen. Außerdem werden während der Bearbeitung einer Transaktion Ressourcen gesperrt. Bei langdauernden interaktiven Aktivitäten führt dies zu einer langen Blockierung anderer Transaktionen, weil der konkurrierende Zugriff auf gemeinsame Ressourcen durch die Isolation geregelt wird. Abhilfe kann hier wieder eine Aufspaltung in mehrere Transaktionen schaffen, wobei eine Aktivitätenabarbeitung in mehrere Einheiten zerlegt wird.

Ein herkömmliches Transaktionskonzept schränkt also die Flexibilität eines Workflow Systems zu sehr ein. Es müssen daher andere, erweiterte oder weniger strenge Konzepte verwendet werden, um eine sinnvolle Ausführung und geeignete Fehlerbehandlung zu gewährleisten. Wenigstens die Schachtelung von Transaktionen („Nested Transactions“) muß möglich sein. In der Literatur gibt es bereits mehrere Konzepte zu Behandlung von langdauernden Transaktionen, wie „Sagas“ [Garc87], „ConTracts“ [Wäch92] und „Transaktionsketten“ [Rein93]. Das Datenbanksystem des Prototyps stellt aber nur ein ACID-

Konzept zur Verfügung, was dazu führt, daß im Prototyp keine ausreichende Fehlerbehandlung durchgeführt werden kann. [Elma91], [Elma92] und [Geor93] enhalten zahlreiche Erweiterungsvorschläge in bezug auf das klassische ACID-Transaktionskonzept.

Weitere Probleme bringt der Einsatz von verteilten Systemen, womit auch ein verteiltes Transaktionskonzept erforderlich wird. Da verteilte Systeme nicht Gegenstand dieser Arbeit sind, sei auf weiterführende Literatur wie [Öszu91] verwiesen.

3.2.3 Ausnahmebehandlung

Jeder Fehler in einem Workflow System muß in geeigneter Weise erkannt und nach Möglichkeit behoben werden. Viele derartige Ausnahmen werden von Hard- oder Software-Fehlern hervorgerufen. Eine Behandlung ist nur mit einer Unterstützung durch das Transaktionsmanagement des Datenbanksystems möglich, weil diese Art von Fehlerbehandlung dem Wiederanlauf („Recovery“) von Datenbanksystemen entspricht. Workflows sind langdauernde Transaktionen, die im Fehlerfall natürlich nicht zur Gänze zurückgesetzt werden können. Für die geeignete Wiederherstellung ist die Schachtelung von Transaktionen unerläßlich. Ein Workflow wird innerhalb einer langen Transaktion geführt. Jede Aktivität und jeder Teilbaum soll auch in einer Transaktion ablaufen. Ja sogar eine Aktivität soll unter Umständen in mehrere Transaktionen zerlegt werden, um die Ressourcen nicht zu lange zu blockieren, und damit deren sinnvolle Auslastung zu ermöglichen. Damit im Fehlerfall möglichst wenig Schaden entsteht, muß der Ablauf wieder so weit wie möglich in den Zustand unmittelbar vor Auftreten des Fehlers gebracht werden. In [Rein93] werden drei Alternativen der Ausnahmebehandlung angeführt:

Unilaterale Ausnahmebehandlung: Die Aufgabe wird nochmals in die Arbeitsliste des Agenten gestellt, weil sich die Umgebung der Aufgabenverarbeitung geändert hat und somit der Anwendungsfehler nicht mehr auftreten wird.

Bilaterale Ausnahmebehandlung: Die Vorgängeraufgabe wird in die Ausnahmebehandlung einbezogen. Das heißt, auch die Vorgänger der Aktivität, in der ein Fehler aufgetreten ist, müssen nocheinmal ausgeführt werden.

Multilaterale Ausnahmebehandlung: Die Vorgängeraufgaben werden über mehrere Stufen hinweg in die Fehlerbehandlung einbezogen. Im schlimmsten Fall muß der Workflow noch einmal gestartet werden.

Diese Alternativen können natürlich nur durchgeführt werden, wenn das dem Workflow System zugrundeliegende Datenbanksystem ein erweitertes Transaktionskonzept zur Verfügung stellt.

Der Prototyp behandelt eine sehr einfache Form der Ausnahmebehandlung: jemand (z.B. der Workflow System Administrator) wird vom Auftreten eines bestimmten Fehlers informiert, und der Workflow wird sofort gestoppt. Das heißt, alle bereits in die Arbeitslisten der Agenten eingetragenen Aktivitäten, die zu diesem Workflow gehören, müssen wieder eingezogen werden, und die Aktivitäten, die gerade ausgeführt werden, dürfen nach der Bearbeitung nicht mehr verteilt werden.

Als weiterführende Literatur zum Thema Ausnahmebehandlung in Workflow Systemen können [Good75], [Karb90] und [Saas92] angeführt werden.

3.2.4 Triggerkonzept

„Herkömmliche" Datenbanksysteme haben die Aufgabe, auf Anfrage eines Benutzers oder einer Applikation, Abfragen durchzuführen und deren Ergebnisse zu präsentieren oder Änderungen durchzuführen. Diese Systeme werden deshalb auch als „passiv" bezeichnet. Es wäre aber wünschenswert, daß ein Datenbanksystem von sich aus tätig wird, wenn z.B. ein Lagerbestand zu niedrig wird oder der Status eines beliebigen Objekts verändert wird. Im ersten Beispiel handelt es sich um ein *aktives datengesteuertes Datenbanksystem*, da hier das Verhalten der Daten beobachtet wird und unter bestimmten Voraussetzungen („Lagerstand zu niedrig") eine weitere Aktion ausgeführt wird [Etzi93]. Das Pendant dazu ist das *aktive ereignisgesteuerte Datenbanksystem*, das auf das Eintreten bestimmter Ereignisse reagiert. Worauf das System reagieren und welche Reaktion stattfinden soll, das muß aber sehr wohl vom Anwendungsprogrammierer vorgegeben werden. Das heißt, ein aktives Datenbanksystem wird nicht von selbst aktiv, sondern reagiert lediglich auf gewisse Ereignisse oder Datenzustände. Aufgrund dieser Eigenschaft müßte eigentlich von einem „reaktiven" Datenbanksystem gesprochen werden. In der Folge wird das Konzept der aktiven Datenbanksysteme anhand der ereignisgesteuerten Datenbanksysteme erläutert [Jasp94, Appe93]. Die Grundlage bilden die *ECA-Regeln* (event-condition-action).

Zur Verwaltung dieser Regeln — auch Trigger genannt — besteht ein aktives Datenbanksystem aus den drei Komponenten Regelverwaltungssystem („rule management"), Ereigniserkennung („event detection") und einer Erklärungskomponente („explanation") (siehe Abbildung 3.1). Vor der Darstellung der Komponenten folgt eine detaillierte Beschreibung der ECA-Regeln.

3.2.4.1 Ereignissteuerung mit ECA-Regeln

Das bei weitem am meisten verwendete Konzept zur „Aktivierung" eines Datenbanksystems, ist das der ECA-Regeln. ECA steht dabei für *event-condition-action.* Das heißt, jede Regel ist aus einem Ereignis-, einem Bedingungs- und einem Aktionsteil aufgebaut.

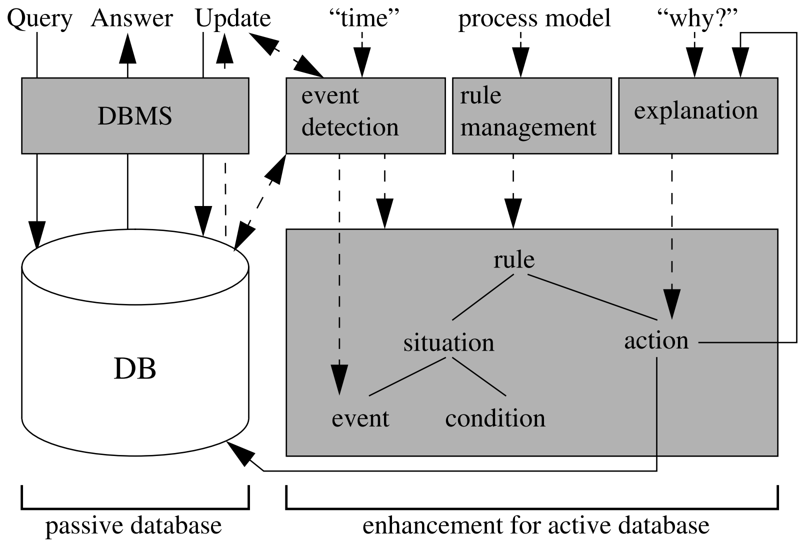


Abbildung 3.1: Komponenten eines aktiven Datenbanksystems [Jasp94]

Ereignisteil:
Die ECA-Regeln werden durch Ereignisse aktiviert. Diese Ereignisse können einerseits vom Anwender ausgelöst werden (z.B. Änderung eines Eintrags) oder auch zeitliche Ereignisse sein. Zeitereignisse können periodisch (z.B. jede Woche am Montag um 7.00 Uhr) oder auch einmalig (z.B. am 25.3.1994 um 6.55 Uhr) auftreten. Regeln können aber auch durch das Auftreten mehrerer bestimmter Ereignisse angestoßen werden. Hier spricht man dann von zusammengesetzten oder komplexen Ereignissen.

Bedingungsteil:
Nach der Auslösung des zur Regel gehörenden Ereignisses werden die hier angeführten Bedingungen überprüft. Nur wenn alle Bedingungen erfüllt sind, wird der Aktionsteil ausgeführt. Ist das Ergebnis der Bedingungsauswertung negativ, so entfällt der Aktionsteil und die Regel „wartet“ auf das nächste auslösende Ereignis.

Aktionsteil:
Im Aktionsteil werden jene Operationen angegeben, die bei positiver Bedingungsauswertung ausgeführt werden sollen. Man beachte, daß diese Operationen im Aktionsteil wiederum Ereignisse auslösen und damit auch neue ECA-Regeln anstoßen können. Dann spricht man von geschachtelten oder kaskadierten Regeln. Wenn zwei Regeln direkt oder indirekt — über mehrere andere Ereignisse hinweg — einander anstoßen, kann es zu einem ununterbrochenen Anstoßen von Ereignissen und damit zu einem „Livelock“[1] kommen, der tunlichst zu vermeiden ist und wenn möglich vom aktiven Datenbanksystem zu erkennen ist.

[1] Ein Livelock unterscheidet sich von einem Deadlock dadurch, daß etwa eine Transaktion nicht an der Ausführung, sondern an der Beendigung gehindert wird, weil sie meist wegen zu niedriger Priorität laufend abgebrochen wird [Bern87]. Hier wird die Ausführung der Regel

Die folgenden zwei Beispiele für ECA-Regeln sind frei erfunden und sollen nur ein Gefühl für die Verwendung solcher Regeln vermitteln.

Beispiel 3.1 *Das Ereignis wird jeweils am Montag um 0.00 Uhr ausgelöst. Wenn niemand am System angemeldet ist, so wird die Datenbank reorganisiert. Befindet sich jemand im System, so geschieht nichts und die Regel wartet auf den nächsten Montag.*

```
EVENT:     monday, 0.00h
CONDITION: nobody_logged_in
ACTION:    reorganize_database
```

Beispiel 3.2 *Hier wird durch das Auswählen einer Aufgabe durch einen Benutzer eines Workflow Systems die Regel gestartet. Die Aktion wird durch die immer wahre Bedingung „true" stets ausgeführt. Und zwar werden dem Texteditor Daten in einem Eingabetext zur Verfügung gestellt, der Editor aufgerufen, die Daten aus dem Ausgabetext wieder gelesen und dann neue Aufgaben verteilt.*

```
EVENT:     select_task_from_queue
CONDITION: true
ACTION:    store_data_into('textin.txt');
           execute('editor textin.txt textout.txt');
           load_data_from('textout.txt');
           distribute_tasks;
```

3.2.4.2 Triggersystem

Das Triggersystem ist die aktive Komponente des aktiven Datenbanksystems und besteht aus folgenden drei Komponenten:

- Regelverwaltungssystem
- Ereigniserkennung
- Erklärungskomponente

Das *Regelverwaltungssystem* bildet das Herz eines aktiven Datenbanksystems. Die Regeln werden hier definiert und verwaltet. Eine eigentlich aus drei Teilen bestehende ECA-Regel wird in zwei logische Teile (Ereignis-Aktions-Paar) zerlegt. Eine Regel setzt sich somit für das Triggersystem aus zwei Bestandteilen zusammen:

- dem Auslöser des Ereignisses (z.B. Erreichen einer bestimmten Tageszeit) und einer zusätzlich vorliegenden „inneren", meist auf die Datenbank bezogenen Bedingung.

verhindert, weil durch den Aktionsteil laufend neue Regeln angestoßen werden.

- einer Aktion, die wiederum aus mehreren Operationen bestehen kann, aber nur dann ausgeführt wird, wenn die Regel „gefeuert“[2] wird, d.h. sowohl das Ereignis eintritt als auch die Bedingung erfüllt ist.

Die Komponente *Ereigniserkennung* erkennt alle im Regelverwaltungssystem definierten Ereignisse, ob es sich nun um Änderungen in der Datenbank, einmalige Zeitereignisse, periodische Zeitereignisse oder auch um von „außen“ — etwa durch eine Applikation — ausgelöste Ereignisse handelt. Bei der anschließenden Bearbeitung der zum Ereignis definierten ECA-Regel können neue Ereignisse auftreten und so zu einer undurchsichtigen Reihe von Aktionen führen. Um das Systemverhalten klar zu machen, sollte deshalb das Auftreten jedes Ereignisses mitprotokolliert werden, was insbesondere bei Fehlerfällen, Systemabstürzen und ähnlichen Ausnahmesituationen von Bedeutung ist.

Damit der Benutzer nicht von den internen Aktionen der Regeln überrascht wird, muß er auf Wunsch informiert werden, was in der Datenbank wirklich vor sich geht. Die Benutzerinformation soll die *Erklärungskomponente* liefern. Bei passiven Datenbanksystemen steuert der Benutzer alle Aktionen und kennt meist deren Folgen. In aktiven Datenbanksystemen können aber durchaus Situationen entstehen, in denen ohne Hilfestellung durch das System die Aktionen nicht mehr einsichtig sind. Diese Komponente ist aber nicht in allen aktiven Datenbanksystemen vorhanden, obwohl besonders bei sehr zahlreichen und komplexen Regeln auf die Nachvollziehbarkeit der Datenbankoperationen sehr großer Wert gelegt werden müßte. Herkömmliche Log-Files reichen nicht aus, um festzustellen, welche Regel wann welche Operation ausgeführt hat.

3.2.4.3 Trigger System und Workflow System

Eine Art der Verwendung von Regeln in einem Workflow System wird auch in [Rein93] beschrieben. Und zwar wird hier die Struktur jedes Workflows durch ein Aktivitätennetz dargestellt. Die Ablaufkontrolle wird von einem getrennten Ablaufkontrollsystem durchgeführt. Zur Ausführung der einzelnen Arbeitsschritte werden Regeln verwendet. Wie ein Triggersystem in der vorliegenden Diplomarbeit eingesetzt wird, wird in Kapitel 4 beschrieben.

3.3 Rollenkonzept

Zur Abbildung der Organisation werden die Kompetenzen der Agenten in Form von Rollen festgelegt. Unter Kompetenz sind jene Aktionen zu verstehen, die ein Agent ausführen kann, wenn er in die entsprechende Rolle schlüpft. Zuvor müssen die Rollentypen angelegt werden, und der Agent bemächtigt werden, diese zu spielen. Die objektorientierte Realisierung dieses Konzepts ist nicht

[2] Der Begriff „feuern“ stammt aus dem Konzept der Petri-Netze [Pete81]. Dort werden die Transitionen (hier: Aktionen) gefeuert (hier: ausgeführt), wenn alle Eingangszustände (hier: Bedingungen) belegt sind.

ohne weiteres möglich, da in einem „herkömmlichen" objektorientierten System Objekte zur Laufzeit die Klassenzugehörigkeit nicht ändern können, so wie es für das „in eine Rolle Schlüpfen" von Agenten notwendig wäre.

Wenn man auf eine saubere Lösung dieses Problems achtet, ist die Verwendung eines eigenen Rollenkonzepts unumgänglich. „Saubere Lösung" bedeutet, daß eine Rollentyphierarchie aufgebaut werden kann, und die Funktionen gemäß dem objektorientierten Paradigma vererbt, aber auch überschrieben werden können, so wie es in Kapitel 2 beschrieben wurde.

3.4 Integration bereits vorhandener Systeme

Ein Paradigma von Workflow Systemen ist, vorhandene Software einzubinden und wiederzuverwenden. Voraussetzung dafür ist, daß die Programme eine möglichst offene Schnittstelle bieten, um mit dem Workflow System kommunizieren zu können. Dies erfordert eine einheitliche Verwaltung der im Netz verteilten Applikationen.

Ein *Distributed Object Management System* (DOMS) ist eine Verbindung von Betriebssystem und Entwicklungsumgebung (beide objektorientiert). Es dient zur systemweiten Verwaltung von Objekten inklusive deren Kommunikation, zur Verwaltung der Ressourcen (Speicherkapazität, Prozessoren etc.), und ist auch für die Fehlerbehandlung verantwortlich.

Die eindeutige Identifizierung der Objekte innerhalb des verteilten Systems ist die Grundvoraussetzung für weitere Funktionen, wie die Behandlung von konkurrierendem Zugriff auf Objekte, deren Persistenz und die Objektreplikation. Unter *Lokationstransparenz* versteht man die Referenzierung der Objekte unabhängig von dem Knoten im System, auf dem sie sich befinden.

Festgelegt ist das System durch die *Object Management Architecture*, welche eine einfache Klassifikation von Programmen und Objekten darstellt. An erster Stelle steht der *Object Request Broker*, der die Kommunikation mit Objekten bestimmt. Eine Standardisierung wurde durch die *Common Object Request Broker Architecture* (CORBA) erreicht [Desh93, Geih95].

Mit Hilfe eines Object Request Broker kann nun die Kommunikation mit anderen Applikationen und damit auch deren Einbindung ins Workflow System durchgeführt werden. Da die Entwicklung eines verteilten Workflow Systems nicht Ziel dieser Arbeit ist, wird der interessierte Leser auf weiterführende Literatur [Biro93, Geor93, Rein93] verwiesen.

Kapitel 4

TriGS$_{flow}$ — Prototyp eines Workflow Systems

Das in Kapitel 2 beschriebene Konzept von Workflow Systemen wurde im Rahmen der Entwicklung eines Prototyps erstellt. Das Konzept war demnach nicht von Anfang an vorhanden, sondern wurde nach und nach erweitert, Teile davon weggestrichen oder geändert. Da bei der Erstellung des Konzepts immer auf die Realisierung im Prototyp geachtet wurde, sind auch nur die elementarsten Teile eines Workflow Systems implementiert. Erweiterungen wie z.B. die Unterstützung von Ad-hoc Workflows oder ausreichende Ausnahmebehandlung wurden aus zwei Gründen nicht entwickelt. Einerseits sind für viele Erweiterungen die technologischen Voraussetzungen nicht gegeben (z.B.: Eine einwandfreie Fehlerbehandlung ist ohne geeignetes Transaktionskonzept nicht möglich). Und andererseits würde die Entwicklung eines Workflow Systems, das in der Praxis sofort eingesetzt werden kann, den Rahmen dieser Diplomarbeit sprengen.

Aus dem Workflow Konzept (Kapitel 2), den Basiskonzepten und Realisierungsgrundlagen (Kapitel 3) ist schließlich ein Grundsystem enstanden, das mit einer geeigneten Benutzeroberfläche die Grundkomponenten eines Workflow Systems enthält. Weiters wurde ein Katalog an Verbesserungs- und Erweiterungsvorschlägen erstellt (Kapitel 7, Anhang B und C), der für weitere Arbeiten in dieser Richtung verwendet werden kann.

Zentrum der Entwicklung ist eine Klassenbibliothek mit dem Namen TriGS$_{flow}$. Der Name wurde von der aktiven Komponente *TriGS* (= Triggersystem für GemStone [Kapp94a, Kapp94b]) des darunterliegenden Datenbanksystems *GemStone* abgeleitet.

Zunächst folgt die Beschreibung der Architektur des Prototyps und der darin verwendeten Komponenten (Abschnitt 4.1). Anschließend werden analog zu Kapitel 2 die Modellierung und der Ablauf in den Abschnitten 4.2 und 4.3 anhand zahlreicher Beispiele erläutert. Beide Abschnitte bauen auf den theoretischen Erkenntnissen des Kapitels 2 auf. Im Abschnitt 4.4 werden noch die wesentlichen Implementierungsaspekte betrachtet, wie etwa die Verwendung

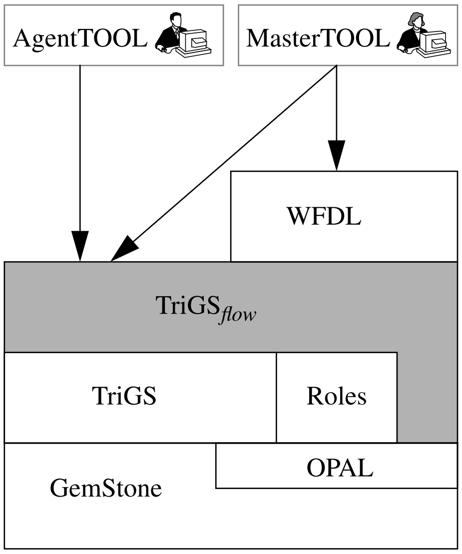


Abbildung 4.1: Gesamtarchitektur des Workflow System Prototyps

von ECA-Regeln.

4.1 Gesamtarchitektur

Viele Neuentwicklungen bauen auf bereits bestehende Konzepte und Systeme auf. So wurde auch hier versucht, bereits bestehende Konzepte oder Systeme einzubinden und im entstehenden System zu verwenden.

Das objektorientierte Datenbanksystem *GemStone* bildet zusammen mit der Datendefinitions- und -manipulationssprache *OPAL* das Fundament der Gesamtarchitektur. Darauf bauen die aktive Komponente *TriGS* und das Rollenkonzept *Roles* [Gott94] auf. Mithilfe von OPAL-Code werden die GemStone-Klassen angelegt und verwendet. Darüber liegt nun die Klassenbibliothek TriGS$_{flow}$ für Workflow Systeme. Sie wurde im Rahmen dieser Arbeit entwickelt. TriGS$_{flow}$ integriert sowohl GemStone als auch Roles und TriGS. Die Workflow Definition Language *WFDL* setzt wiederum die TriGS$_{flow}$-Architektur voraus. Eine solche Definitionssprache ermöglicht es, mit einfacher Grammatik Workflow Typen aufzubauen. Die als Text vorliegende Beschreibung wird einem Übersetzer übermittelt, der daraus den Workflow Typ in die Datenbank legt und zur Ausführung bereitstellt.

Diese Grundlagen werden von den Benutzeroberflächen eines Workflow Systems benötigt. Natürlich verwenden diese noch einige Systeme mehr, wie etwa jene Programmierumgebung, in der sie aufgebaut werden (z.B. VisualWorks/Smalltalk). Das *AgentTOOL* soll jene Funktionalität zur Verfügung stellen, die der „gewöhnliche“ Teilnehmer, also der Agent, zum alltäglichen Gebrauch benötigt. „Alltäglich“ bedeutet die Ausführung der übertragenen Aufgaben und den Start von Abläufen. Wie in anderen Systemen soll es auch in

Workflow Systemen Personal geben, das für den Aufbau, die Erweiterung und die Instandhaltung des Systems verantwortlich ist, einen sogenannten „Workflow System Administrator“. Alle seine Tätigkeiten im Zusammenhang mit dem Workflow System sollen durch das *MasterTOOL* unterstützt werden.

4.1.1 GemStone und OPAL

Das Fundament des Prototyps bildet das objektorientierte Datenbanksystem *GemStone* [Serv91, Butt91]. Eine Besonderheit von GemStone ist, daß Objekte zur Gänze in die Datenbank gespeichert werden. „Zur Gänze“ bedeutet, daß nicht nur die Daten des Objekts als Einheit, sondern auch das Verhalten (die Methoden) der Klassen in der Datenbank und nicht in separate Codedateien gespeichert wird. Viele andere objektorientierten Datenbanksysteme legen zwar die Daten in die Datenbank, den Code aber in eigene Codedateien.

Die Interaktionssprache *OPAL* dient zur Kommunikation mit dem Datenbanksystem. Es ist sowohl eine Datendefinitionssprache (DDL) als auch eine Datenmanipulationssprache (DML). Zur mengenorientierten Abfrage bzw. Selektion von Datensätzen aus Mengen stehen eigene Methoden und Konstrukte zur Verfügung. OPAL ist sehr stark an Smalltalk [Gold83] angelehnt, bis auf wenige Unterschiede sogar ident dazu. Grundsätzlich gilt für OPAL: Jedes Datum ist ein Objekt. Ob Zahlen, Zeichenketten, Mengen, Listen — sie alle sind Objekte.

4.1.2 Rollenkonzept in OPAL

In vielen klassenbasierten objektorientierten Systemen kann eine Instanz nur zu einer Klasse gehören. Wenn man die Klassenzugehörigkeit ändern will, so muß eine Instanz der neuen Klasse gebildet und die Attribute der alten Instanz übernommen werden. Wenn alle Rollentypen als neue Klassen in der herkömmlichen Klassenhierarchie abgebildet werden, so wächst die Hierarchie sehr schnell und wird zugleich unübersichtlich und schlecht wartbar. Für jede mögliche Kombination an Rollen, die ein Objekt annehmen kann, muß dabei eine neue Klasse definiert werden.

In [Gott94] wird eine Erweiterung solcher klassenbasierter objektorientierter Systeme um ein Rollenkonzept vorgestellt. Da das Konzept und die Implementierung auf Smalltalk basiert, kann sie leicht auf andere objektorientierte Systeme portiert werden. Deshalb ist dieses Konzept auch in OPAL verfügbar [Ober94], weil auch OPAL — wie schon zuvor erwähnt — auf Smalltalk aufbaut.

Verwendet werden die Rollen zur Darstellung der Kompetenzen eines Agenten. Dieser kann zur Laufzeit eine oder mehrere Rollen annehmen, sofern er vorher dazu ermächtigt wurde. Neben der Klassenhierarchie kann auch eine Rollentyphierarchie aufgebaut werden, was eine intuitive Abbildung der Realität in einem objektorientierten Modell ermöglicht. Ohne viel Aufwand kann dieses Rollenkonzept in das Workflow System eingebunden werden.

Man benötigt nur zwei Klassen, um eine Rollentyphierarchie aufzubauen: `ObjectWithRoles` und `RoleType`. Eine Instanz kann Rollen annehmen, wenn dessen Klasse eine direkte oder indirekte Subklasse von `ObjectWithRoles` ist. Dieses Objekt wird als Wurzel („root object") der Rollentyphierarchie angesehen. Die Rollentyphierarchie wird von der Klasse `RoleType` abgeleitet. Jede Subklasse von `RoleType` stellt einen Rollentyp dar. Die Hierarchie wird nicht wie eine Klassenhierarchie als Vererbungsbaum gebildet, sondern die Vererbung wird auf Instanzebene vorgenommen und über Klassenvariablen der Rollentypen realisiert. Die Hierarchie ist in Wirklichkeit eine „flache" Menge von Subklassen der Klasse `RoleType`. Die Vererbung der Methoden von Rollentypen wird über ein „Lookup" über die Klassenvariablen simuliert.

Um eine Rolle eines Agenten einer Gruppe zuteilen zu können, wird eine weitere Klasse verwendet: `QualifiedRoleType`. Sie ist nur eine Spezialisierung von `RoleType` und ermöglicht, die Rolle einem Objekt von zuvor spezifizierter Klasse zuzuordnen.

Das Rollenkonzept von [Gott94] wurde in dieser Arbeit um zwei Klassen erweitert, die die Bildung von neuen Rollentypen vereinfachen: `Role` für herkömmliche Rollentypen und `QualifiedRole` für Rollentypen, die Gruppen zugeordnet werden. Beide sind Subklassen der Klasse `Object` und enthalten nur eine Klassenmethode. Um einen „normalen" (nicht qualifizierten) Rollentyp zu erstellen, schickt man der Klasse `Role` die Methode `newRole:roleSuperType:`. Um einen qualifizierten (einer Gruppe zugeordneten) Rollentyp zu erzeugen, schickt man der Klasse `QualifiedRole` die Methode `newRole:roleSuperType:classOfQualifyingObj:` mit dem Rollennamen und der Klasse der Qualifikation als Parameter. Das Beispiel 4.1 veranschaulicht, wie in OPAL mit Hilfe der beiden Klassen neue Rollentypen angelegt werden können.

Beispiel 4.1 *Der Rollentyp* `Employee` *soll in der Rollentyphierarchie an der Spitze stehen, das heißt,* `HumanAgent` *als Superklasse haben.* `HumanAgent` *ist kein Rollentyp, sondern eine indirekte Subklasse von* `ObjectWithRoles`*. Damit ist dies die Wurzelklasse der darauf aufbauenden Rollentyphierarchie.*

```
Role
    newRole: 'Employee'
    roleSuperType: HumanAgent.
```

Druckertreiber sollen als Software Agenten angesprochen werden. Jeder Spooler muß einer Abteilung zugeordnet werden. Welche Art von Abteilung, ist dabei nicht festgelegt. Die Superklasse `Machine` *ist ein Rollentyp, weshalb der Rollentyp* `PrintSpooler` *keine Wurzelklasse einer Rollentyphierarchie darstellt.*

```
QualifiedRole
    newRole: 'PrintSpooler'
    roleSuperType: Machine
    classOfQualifyingObj: Department.
```

4.1.3 Aktive Komponente TriGS

TriGS (= Triggersystem für GemStone) ist eine aktive Erweiterung des objektorientierten Datenbanksystems GemStone. Grundlage von TriGS bilden die unter Punkt 3.2.4 eingeführten ECA-Regeln. Das Konzept von TriGS wird in [Kapp94a] und [Kapp94b] vorgestellt, die Implementierung wurde im Rahmen einer Diplomarbeit vorgenommen und in [Aich94] beschrieben.

Als *Ereignisse* können in TriGS zur Zeit nur Methodenaufrufe verwendet werden. Zeitereignisse, ob absolut (z.B. am 1.4.95 um 9.00 Uhr) oder periodisch (z.B. jeden Tag um 0.00 Uhr), und komplexe Ereignisse werden in dieser Implementierung noch nicht unterstützt. Komplexe Ereignisse sind — wie bei komplexen Organisationspolitiken — eine Verknüpfung einzelner Ereignisse mit logischen Operatoren zu neuen „zusammengesetzten“ Ereignissen. Bei einem Ereignis, das durch einen Methodenaufruf erzeugt wird, kann angegeben werden, ob es vor („pre“) oder nach („post“) der Ausführung der Methode eintreten soll.

In Bezug auf den Bedingungsteil und den Aktionsteil kann auch ein Ereignis, der sogenannte „Condition-Event-Selector“ bzw. „Action-Event-Selector“, angegeben werden, welches besagt, wann die Bedingung evaluiert bzw. der Aktionsteil ausgeführt werden soll. Die dadurch entstehenden Kombinationen der Zeitpunkte des Triggerns einer Regel und der Evaluierung der Bedingung bzw. der Ausführung der Aktion der getriggerten Regel werden als *Kopplungsarten* bezeichnet. Meist handelt es sich aber um dasselbe Ereignis, wie das der ganzen Regel, die vor oder nach der Ereignismethode zur Gänze ausgeführt wird. Diese Kopplungsart wird „immediate“ genannt. Im Aktionsteil kann eine Folge von OPAL-Methoden an Objekte geschickt werden.

In TriGS können globale und lokale Regeln definiert werden. *Globale Regeln* gelten für alle Objekte innerhalb des Datenbanksystems, die die im Ereignis spezifizierte Methode verstehen. *Lokale Regeln* werden nur auf eine bestimmte Klasse angewendet und unterstützen auch deren Vererbung: Wenn auf einer Klasse eine Regel definiert wird, so gilt sie auch für alle deren Unterklassen. Der Gültigkeitsbereich von lokalen Regeln kann in unterschiedlicher Granularität angegeben werden: auf Klassenebene oder auf Instanzebene. Eine Regel kann zwar global für eine ganze Klasse deaktiviert sein, für einige Instanzen aber durchaus aktiviert werden. Umgekehrt kann eine Regel auch für eine Klasse aktiviert, für einige Instanzen aber inaktiv sein. Der Vererbungshierarchie entlang vererbte Regeln können auch durch Angabe neuer Regeln für die Unterklassen überschrieben werden.

Das folgende Beispiel 4.2 wird in der üblichen TriGS-Notation angegeben. Das Schlüsselwort `rule_trgO` steht für das „rule triggering object“, also für jenes Objekt, das das Ereignis ausgelöst hat.

Beispiel 4.2 *Wenn eine Maschine in den Zustand „verfügbar“ übergeht, soll sie die nächste Operation aus ihrem Eingabepuffer lesen und ausführen:*

```
DEFINE RULE MachineRule_1
ON POST (Machine, changeState: newState) DO
ON POST (rule_trgO, changeState: newState)
   IF newState = 'available' THEN
ON POST (rule_trgO, changeState: newState)
   EXECUTE rule_trgO schedule: (rule_trgO getBuffer nextOp)
```

Hier werden sowohl für die Regel selbst als auch für den Bedingungs- und Aktionsteil die gleiche Methode und der gleiche Zeitpunkt als Ereignis angegeben. Damit ist diese Regel ein Beispiel für die Kopplungsart „immediate".

Für TriGS wurde auch eine Benutzeroberfläche entwickelt [Aich94], die aber wegen der automatischen Erstellung vieler Regeln bei der Definition von Workflow Typen wenig verwendet wird. Welche Probleme beim Einsatz von TriGS aufgetreten sind und wie sie beseitigt wurden, wird im Anhang B aufgezeigt. Welche Regeln konkret im Workflow System eingesetzt werden, wird später in 4.4.4 behandelt.

4.1.4 Prototyp TriGS$_{flow}$

TriGS$_{flow}$ besteht aus 17 vorgegeben Basisklassen. Sobald ein Workflow Typ erstellt wird, werden zusätzliche Klassen für Rollentypen, Gruppentypen und Akttypen angelegt. Wie dieser Prototyp verwendet wird, um einen Workflow Typ zu modellieren, anzulegen und danach auch ablaufen zu lassen, wird in 4.2 und 4.3 beschrieben. Die Klassenhierarchie dieses Prototyps wird im Anhang A vorgestellt, und alle seine Klassen werden dort genauer beschrieben.

4.1.5 Workflow Definition Language WFDL

Der Trend zur grafischen interaktiven Eingabe hat auch in Workflow Systemen Einzug gehalten. Dennoch sind auch Definitionssprachen immer noch aktuell, weil sie zumeist höheren Informationsgehalt haben, als mit grafischen Symbolen noch übersichtlich dargestellt werden kann. Deshalb haben viele Workflow Systeme eine Schnittstelle zu einer Definitionssprache und einem syntaxgesteuerten Editor zur Unterstützung der Eingabe. Eine gute grafische Oberfläche kann eine WFDL aber ohne weiteres ersetzen.

Eine WFDL, in BNF definiert, existiert in diesem Prototyp nicht. Anstatt Datentypen, Organisation und Aktivitäten mittels einer WFDL anzugeben, wird die Datendefinitions- und -manipulationssprache OPAL über den *GemStone Workspace* verwendet. Akttypen können auch über den *GemStone System Browser* angelegt werden.

4.1.6 Benutzeroberflächen AgentTOOL und MasterTOOL

Um das Workflow System möglichst einfach und komfortabel benutzen zu können, und jedem Teilnehmer (Agenten) den Gebrauch zu ermöglichen, muß eine dazu geeignete Benutzeroberfläche vorhanden sein. Das *AgentTOOL* soll die zur Ausführung von Abläufen notwendigen Funktionen erfüllen, indem es dem Agenten folgende Möglichkeiten bietet:

- Jedem Agenten muß seine Arbeitsliste präsentiert werden. Einträge aus der Liste können in beliebiger Reihenfolge ausgeführt werden, sofern keine Prioritäten oder Ausführungszeitbeschränkungen vorliegen.
- Weiters soll ähnlich einem Mail-Tool, die Kommunikation mit anderen Agenten oder dem Workflow System Administrator unterstützt werden.
- Alle Workflow Typen des Systems sollen aufgelistet und gestartet werden können. Es ist nicht möglich, nur jene anzuzeigen, die von dem betreffenden Agenten gestartet werden können, da diese gemäß der Startorganisationspolitik zur Laufzeit ausgewertet werden müssen. Eine Organisationspolitik kann nämlich kontextabhängig sein, oder der Agent inzwischen weitere Fähigkeiten (Rollen) erlangt haben.
- Der Agent soll auch die Möglichkeit haben, die Bearbeitung einer Aufgabe zurückweisen zu können, wenn die Ausführung, aus welchen Gründen auch immer, nicht möglich ist.
- Der Agent soll natürlich auch den Aktenlauf zurückverfolgen und weiter beobachten können. Inwieweit er dabei in den weiteren Verlauf eingreifen kann, ist schwer zu entscheiden deshalb noch nicht festgelegt.

Der Workflow System Administrator hat nun zusätzliche Aufgaben zu erfüllen. Auch diese sollen durch ein eigenes Werkzeug, dem *MasterTOOL*, unterstützt werden:

- Alle Anforderungen an das AgentTOOL sollen auch für das MasterTOOL Geltung haben.
- Weiters muß der gesamte Modellierungsprozeß unterstützt werden. Das heißt, es müssen Modellierung und Definition von Organisation, Aktivitäten und Daten ermöglicht werden. Wünschenswert ist natürlich sowohl eine grafische Eingabe als auch die Eingabe via WFDL. Außerdem muß eine Änderung der Workflow Typen jederzeit möglich sein, und von neu gestarteten Abläufen sofort verwendet werden. Die noch dem alten Schema entsprungenen Aktenläufe behalten die alte Version des Workflow Typs bei (vgl. Versionierung und Schemaevolution des objektorientierten Paradigmas).

- Ein sogenannter Monitor zur Überwachung (z.B. Prüfung der Auslastung von Agenten) und Kontrolle (z.B. Engpässe herausfinden, Ablaufhistorien) des Systems ist ebenfalls zu integrieren.
- Die An- und Abmeldung (Aktivierung und Deaktivierung) von Agenten überwacht auch der System Administrator (Deaktivierte Agenten können keine Aktivitäten mehr empfangen. Das Aktivieren eines Agenten kann einen gestoppten Workflow wieder „zum Leben erwecken"). Die vollständige Kontrolle der Software Agenten obliegt auch dem MasterTOOL.

Neben den angegebenen Funktionen soll der Workflow System Administrator auch Einblick in die Arbeitsliste von Agenten haben, damit Aufgaben auf andere Agenten verteilt werden können, wenn Engpässe auftreten.

Der Prototyp verfügt zur Zeit über keine geeignete grafische Oberfläche, weder über ein AgentTOOL noch über ein MasterTOOL. Eine Implementierung hätte den Rahmen dieser Arbeit gesprengt. Die Interaktion erfolgt deshalb noch mittels GemStone Workspace.

4.2 Modellierung mit TriGS$_{flow}$

Modelliert werden die drei Bereiche Organisation, Aktivitäten und Daten. Die Integration von Daten und Organisation wird bereits bei der Aktivitätendefinition vorgenommen, weshalb die Integration noch vor der Aktivitätenmodellierung besprochen wird. Die Beschreibung der Modellierung wird stets mit Beispielen unterstützt.

4.2.1 Organisationsmodellierung

Bevor die einzelnen Teile der Organisation modelliert werden, wird noch angeführt, in welchen Schritten sich die derzeitige Abbildung der Organisationsstruktur entwickelt hat.

4.2.1.1 Sukzessive Entwicklung von Konzepten zur geeigneten Abbildung von Organisationsstrukturen

Am Beginn der konzeptuellen Entwicklung des Workflow Systems war nur eine Rollenstruktur in Form eines Vererbungsbaumes in der objektorientierten Programmierung vorhanden (siehe Abbildung 4.2). Dabei wurden die Agenten und deren Rollen in einer Hierarchie zusammengestellt. Eine Gruppenzuteilung existierte nicht.

Mit dieser Vererbungshierarchie von Rollentypen, dargestellt in 4.2, lassen sich nur sehr begrenzt realitätsnahe Organisationsstrukturen beschreiben. Die

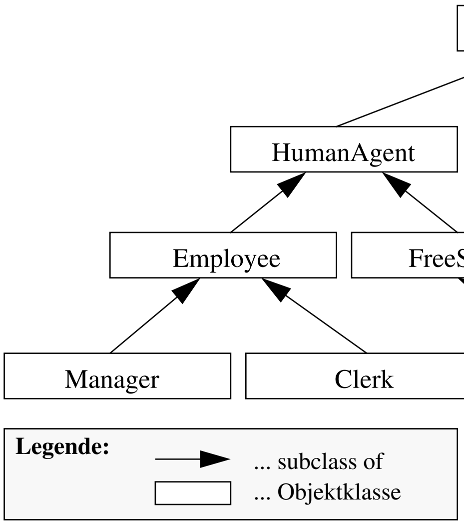


Abbildung 4.2: Bloße Vererbung der Rollentypen

Möglichkeit der Vererbung von Rollentypen ist aber sehr nützlich. Man verwendet sie, um die Spezialisierung von Rollentypen direkt implementieren zu können. Die Vererbung im objektorientierten Sinn bedeutet auch eine Spezialisierung von Eigenschaften und Verhalten. Hier ist vor allem Zweiteres von großem Nutzen, denn die Ausführung der Aufgaben wird in den Methoden der Agenten gespeichert. Die Agenten, die eine spezialisiertere Rolle spielen, können also auch alle Aufgaben der übergeordneten Rollen ausführen. Die objektorientierte Programmierung und damit auch dieses Modell scheitert aber, wenn ein und dasselbe Objekt seine Klassenzugehörigkeit bzw. Rolle ändern oder gar mehrere Rollen gleichzeitig spielen soll. Außerdem lassen sich hiermit keine Gruppenzugehörigkeiten darstellen.

Im zweiten — ebenfalls einfach zu realisierenden — Ansatz, dargestellt in Abbildung 4.3, wurden zu jedem Agenten seine Rollen und die Gruppen, denen er angehört, gespeichert. Rollen und Gruppen wurden nur durch ihre Namen definiert. Die Nachteile dieser Methode liegen auf der Hand: Zum einen fällt jegliche Vererbung innerhalb der Rollen weg, weil keine Rollentyphierarchie mehr vorhanden ist. Zum anderen kann auch keine Beziehung zwischen Rolle und Gruppe eines Agenten angegeben werden. Weiters kann auch keine Abteilungsstruktur dargestellt werden.

Beispiel 4.3 *Herr Maier ist Abteilungsleiter der Marketingabteilung und zugleich Projektmitarbeiter in einem Modernisierungsprojekt. Er spielt somit zwei Rollen („Abteilungsleiter“ und „Projektmitarbeiter“) und gehört zwei Gruppen („Marketingabteilung“ und „Modernisierungsprojekt“) an. Wenn keine Möglichkeit besteht, Rollen und Gruppen eines Agenten in Beziehung zu setzen, so kann das Workflow System Herrn Maier auch als Abteilungsleiter des Modernisierungsprojekts und Projektmitarbeiter der Marketingabteilung interpretieren.*

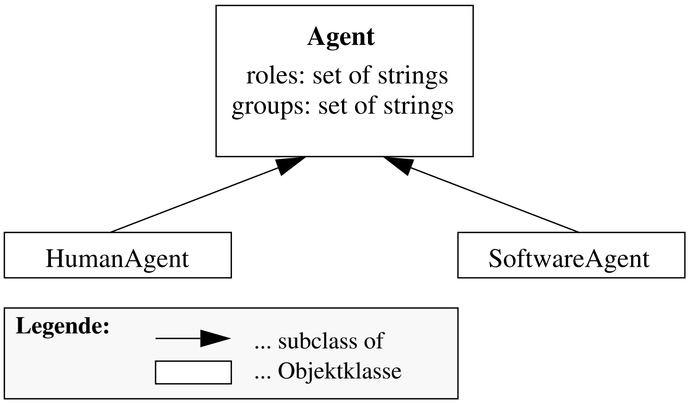


Abbildung 4.3: Agenten mit einfacher Rollen- und Gruppenzuordnung

Da der zweite Ansatz wegen fehlender Vererbung und fehlender Zuordnung von Rollen zu Gruppen für die realitätsnahe Abbildung einer Organisationsstruktur auch nicht geeignet ist, mußte das Konzept wieder geändert werden.

In 4.1.2 wurde bereits die Anwendung des Rollenkonzepts beschrieben. Es bleibt nur noch anzumerken, daß ein Objekt zur gleichen Zeit ein oder mehrere solcher Rollenobjekte annehmen und wieder freigeben kann. Als nächstes ist zu klären, ob sich dieses Konzept auch für die Verwendung von Gruppen eignet.

Die Gruppen- bzw. Abteilungsbildung wird in [Kies92] als mehrstufige Gliederung des Stellengefüges beschrieben. Das bedeutet, daß es auch eine Struktur innerhalb der Gruppen geben muß. Um das zu erreichen, gibt es eine spezielle Art von Rollentypen, nämlich *qualifizierte Rollentypen*. Als Qualifizierung kann eine Instanz jener Klasse (oder ein Subklasse davon) verwendet werden, die bei der Definition des Rollentyps als „Constraint" angegeben wird. Eine Gruppe des Betriebes qualifiziert damit die Rolle. Gruppen sollten auch verschiedene Attribute besitzen, um die Abteilungsstruktur auch für andere Zwecke als für das Workflow System weiterverwenden zu können. Ein Anwendungsgebiet wäre etwa ein Betriebsinformationssystem, das unter anderem genaue Daten über einzelne Abteilungen sammelt und für bestimmte Abteilungen verschiedene Attribute benötigt. Deshalb werden *Gruppentypen* eingeführt, die einer Klassenhierarchie entsprechend ihre Instanzvariablen vererben. Jede Gruppe ist eine Instanz eines Gruppentyps. Die *Gruppentyphierarchie* wird durch eine einfache Klassenhierarchie aufgebaut (siehe Abbildung 4.4) und dient lediglich zur Spezialisierung von Gruppentypen und stellt keine konkrete Ausprägung wie eine Abteilungsstruktur dar.

Abbildung 4.5 veranschaulicht die Verbindung der drei Hierarchien von Agenten, Rollentypen und Gruppentypen: Diese Verbindung bildet nun ein Konzept, das sich für die Abbildung der Organisationsstruktur der meisten Betriebe eignet. Besonderheiten, die mit diesem Konzept nicht dargestellt werden können, müssen entweder an dieses Konzept angepaßt werden, oder man erweitert das Rollen- und Gruppenkonzept, um auch diese Spezialfälle abbilden zu können.

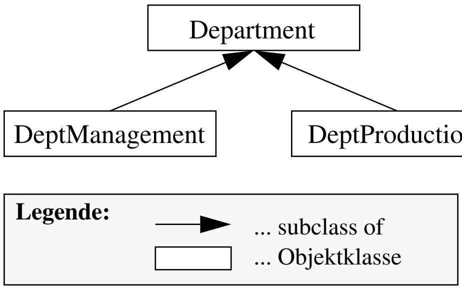


Abbildung 4.4: Gruppentyphierarchie

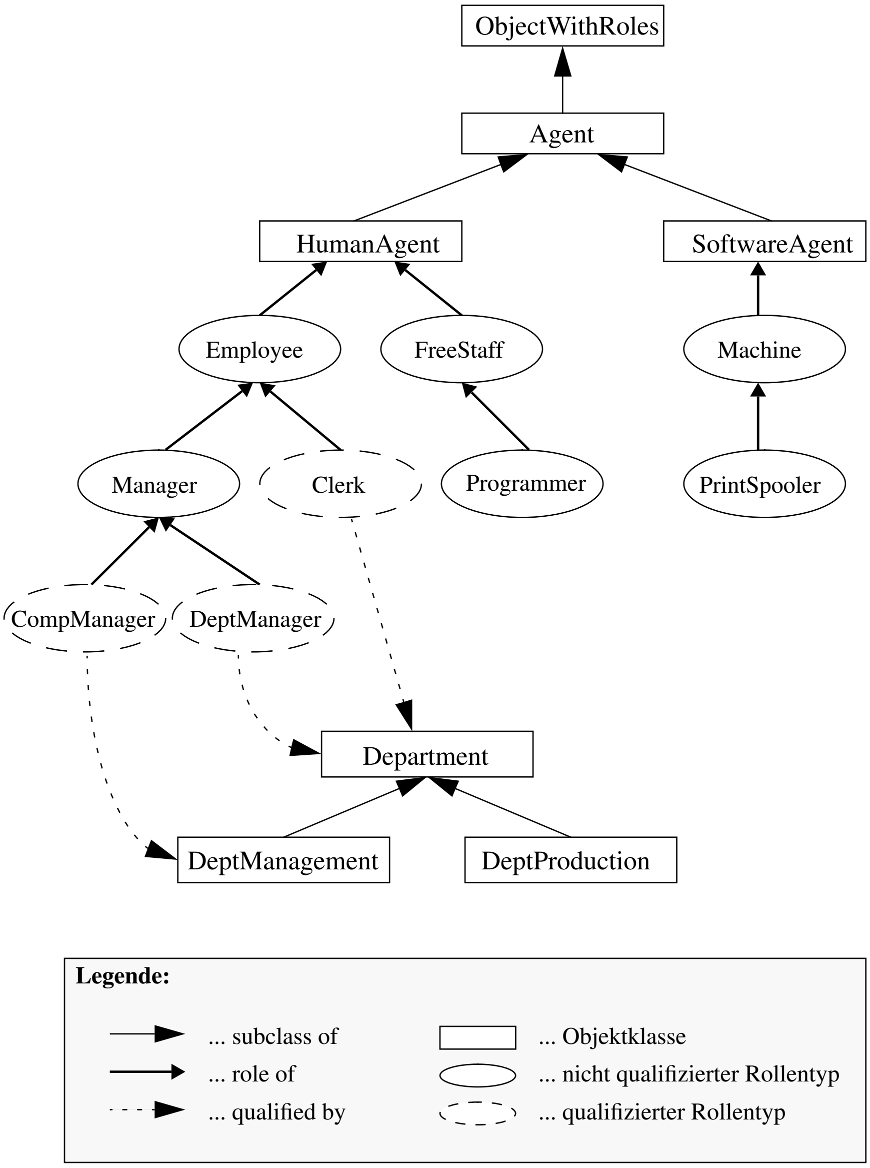


Abbildung 4.5: Verbindung von Agenten-, Rollentyp- und Gruppentyphierarchie

4.2.1.2 Definition der Rollentyphierarchie

Nicht qualifizierte Rollentypen werden angelegt, indem man der Klasse `Role` die Methode `newRole:roleSuperType:` schickt. Die Klasse `Role` erzeugt dann den entsprechenden Rollentyp. Die Rollentyphierarchie wird durch den `roleSuperType:` bestimmt. Es wird dabei eine neue Klasse mit dem entsprechenden Namen als Subklasse von `RoleType` angelegt. Es ist zu beachten, daß keine Rollentyphierarchie im Sinne einer Vererbung der Klassen aufgebaut wird, sondern die Vererbung auf Instanzebene über die Klassenvariable `roleSuperType` der Klasse `RoleType` vollzogen wird.

Beispiel 4.4 *Definition von Rollentypen, die keiner Gruppe (nicht qualifiziert) oder einer Gruppe (qualifiziert) zugeordnet sind:*

```
Role
    newRole: 'Manager'
    roleSuperType: Employee.

QualifiedRole
    newRole: 'CompManager'
    roleSuperType: Manager
    classOfQualifyingObj: DeptManagement.
```

Sobald die Rollentypen angelegt sind, können jederzeit Aktionsmethoden definiert werden. Diese Aktionsmethoden werden aufgerufen, wenn eine Aktivität ausgeführt wird. Die zu lesenden Daten werden über die Methodenparameter eingelesen. Wichtig ist, daß bei der Definition der Aktivität die zu lesenden Instanzvariablen des Aktes in der Reihenfolge angegeben werden, in der sie von der Aktionsmethode als Parameter eingelesen werden. Die zu schreibenden Instanzvariablen werden über ein Feld als Rückgabeobjekt geliefert. Die Reihenfolge der Ausgabeinstanzvariablen bei der Aktivität muß wieder der Reihenfolge der Elemente im Rückgabefeld entsprechen. An erster Stelle wird immer der Status angegeben, mit dem die Aktion beendet wird. Nur beim Status `#ok` erfolgt eine weitere Verarbeitung, andernfalls wird die Aktivität zurückgesetzt und verbleibt in der Arbeitsliste des Agenten. Gibt es keinen Rückgabewert, so muß unbedingt ein einelementiges Feld (z.B.: `(Array new) add: #ok`) zurückgegeben werden.

Die Methode wird gespeichert, wenn die Aktion beim Rollentyp übersetzt wird, entweder per Browser oder im GemStone Workspace mit der Methode `compileMethod:dictionaries:category:`. Die Aktionsmethoden werden am besten unter der Kategorie `Actions` bei der Klasse des Rollentyps abgelegt.

Beispiel 4.5 *Die folgende Aktionsmethode berechnet aus dem Eingabeparameter Geburtsdatum das Alter in Jahren. Zurückgegeben wird das Alter als einziges Element des Rückgabefeldes neben dem Status. Diese Berechnung kann vom Rollentyp „CompManager“ und allen seinen Subtypen durchgeführt werden.*

```
CompManager
     compileMethod: 'computeAge: aBirthdate
```

```
  | newArray age |
  age := ((DateTime now) subtractDate: aBirthdate)//365.
  newArray:= Array new:1.
  newArray
    at: 1 put: #ok
    at: 2 put: age.
  ^newArray'
dictionaries: (System myUserProfile symbolList)
category: #Actions.
```

Nur in wenigen Fällen beinhaltet eine Aktionsmethode derart simple Berechnungen. Im Normalfall werden hier Programme gestartet, die mit bestimmten Eingabedaten gespeist werden. Nach dem Beenden werden die Resultate in das Feld geschrieben und schließlich in den Akt übernommen. Die Einbindung fremder Applikationen wurde aber noch außer Acht gelassen, da sie über OPAL-Anweisungen nicht möglich ist, sondern nur über die C-Schnittstelle ermöglicht werden kann. Diese Möglichkeit wurde aber bisher noch nicht näher untersucht. Dazu kommt dann ein weiteres Problem: Eine Programmausführung kann synchron oder asynchron erfolgen. Synchron bedeutet, das aufgerufene Programm läuft innerhalb des aufrufenden ab, und das rufende wird erst nach Ende des gerufenen fortgesetzt. Bei asynchroner Ausführung läuft das rufende Programm weiter. Dies würde bedeuten, daß nach Ende des gerufenen, das rufende erneut gestartet und zur Übernahme der Daten angestoßen werden muß.

4.2.1.3 Definition der Gruppentyphierarchie und der Abteilungsstruktur

Die *Gruppentyphierarchie* muß zur Zeit noch im GemStone System Browser angelegt werden. Es werden dort die einzelnen Gruppentypen als Subklasse von `Department` definiert, sofern eigene Attribute überhaupt verwendet werden. In diesem Workflow System werden die Abteilungen nur zur Zuordnung einer Gruppe verwendet. Deshalb sind auch Zusatzattribute hier nicht von Bedeutung, und eine Spezialisierung der Gruppentypen kann entfallen. Will man die Gruppentyphierarchie aber auch anderweitig verwenden, so läßt sich diese an alle Anforderungen anpassen.

Die *Abteilungsstruktur* wird gebildet, indem die Hauptabteilungen auf der obersten Ebene eines Betriebes in Unterabteilungen gegliedert werden. Mit der Methode `addSubDept:` kann einer Abteilung eine Unterabteilung hinzugefügt werden.

Beispiel 4.6 *Definition der Gruppe (Abteilung) „Produktion“ und Festlegung der drei zuvor definierten Unterabteilungen.*

```
dept3 := (DeptProduction new)
         id: 'Dept. 3';
         name: 'Production Department';
         addSubDept: dept31;
         addSubDept: dept32;
         addSubDept: dept33.
```

Eine Abteilung wird also entweder nur als Instanz der Klasse `Department` verwendet, oder es wird dafür eine separate Subklasse von `Department` oder einer Unterklasse davon erzeugt. Für das Workflow System sind nur die Nummern (`id`) von Bedeutung, denn durch sie wird jede Abteilung eindeutig identifiziert. Diese Nummer wird in der Definition von Organisationspolitiken verwendet (siehe dazu 4.2.3.1). Die Abteilungsstruktur wird dann in den Betrieb (`Company`) integriert, indem bei diesem mit der Methode `addDept:withId:` die Hauptabteilungen eingetragen werden.

4.2.1.4 Definition von Agenten

Auch ein Agent muß im GemStone Workspace angelegt werden, und zwar durch Angabe seiner Nummer und seines Namens. Danach werden seine Rollen definiert und er selbst aktiviert, sodaß er vom System auch erkannt wird. Nur wenn ein Agent aktiviert ist, können ihm auch Aufgaben übertragen werden. Das Anlegen eines neuen oder die Aktivierung eines alten Agenten hat zur Folge, daß alle gespeicherten Workflow Typen durchsucht werden, ob nicht ein Workflow gestoppt wurde und vielleicht jetzt wieder fortgesetzt werden kann. (Diese Eigenschaft ist in ECA-Regeln abgelegt, die die Methode `Agent activate` beobachten.)

Wie die Rollen eines Agenten definiert werden, wurde bereits erklärt. Wie ein Agent in eine Rolle schlüpfen und diese wieder freigeben kann, wurde hingegen noch nicht beschrieben. Wenn man einem Agenten eine Rolle verpassen will, deren Typ nicht an der Spitze der Rollentyphierarchie steht, so müssen zuerst alle „Vaterrollen" definiert werden. Außerdem muß als Parameter immer der Agent mittels der Methode `as:` (bzw. `as:in:` für qualifizierte Rollen) in die übergeordnete Rolle schlüpfen. Mit der Methode `abandon` kann eine Rolle wieder entfernt werden.

Beispiel 4.7 *Anlegen eines Agenten, Definition seiner Rollen und Aktivierung; anschließende Entfernung der Rolle*

```
agent1 := (HumanAgent new)
          id: 'h001';
          name: 'Murthaler Josef'.

Employee newRoleOf: agent1.
Manager newRoleOf: (agent1 as: Employee).
CompManager newRoleOf: (agent1 as: Manager) qualifiedBy: dept10.
agent1 activate.

(agent1 as: Manager in: dept10) abandon.
```

Neu angelegte Agenten sind immer inaktiv und müssen daher, sobald sie Aufgaben erhalten sollen, aktiviert werden.

4.2.1.5 Zusammenfassung von Gruppen, Rollen und Agenten in einem Betrieb

Um die Gruppen, Agenten und ihre Rollen im Workflow System verwenden zu können, müssen sie zuerst in einem Betriebs-Objekt (`Company`) zusammengefaßt werden. Daß die Agenten nicht systemweit Geltung haben, hat den Vorteil, daß innerhalb eines Systems mehrere Organisationsstrukturen verwendet werden können. Vor allem für Testzwecke ist dies sehr günstig, weil damit verschiedene Organisationsstrukturen ausprobiert werden können.

Beispiel 4.8 *Erstellen eines Betriebes. Es wird angenommen, daß die Gruppen- und Agentenobjekte bereits definiert sind und durch die Variablen* `dept1` *bis* `dept5` *und* `agent1` *bis* `agent5` *referenziert werden. Alle Betriebe müssen in das Dictionary* `WFCompanyBase` *abgelegt werden.* `WFCompanyBase` *ist vom Typ* `CompanyBase`*, der bloß eine Sublasse der Klasse* `Dictionary` *darstellt und keinerlei Erweiterungen hat. Diese Klasse wurde nur entwicklelt, um konsistent mit den Worfklow Typen zu bleiben, die für Ihre Speicherung ebenfalls eine eigene Klasse haben. Diese Klasse hat aber zusätzliche Funktionalität (zum Fortsetzen gestoppter Workflows). Außerdem können die beiden Klassen für den späteren Gebrauch erweitert werden.*

```
aCompany := Company new.
aCompany id: 'C1';
         name: 'Team 99 Ltd.';
         address: '1929 Broadway - 10056 New York'.
WFCompanyBase at: (aCompany id) put: aCompany.

(WFCompanyBase at: 'C1')
   addDept: dept1 withId: 'Dept. 1';
   addDept: dept2 withId: 'Dept. 2';
   addDept: dept3 withId: 'Dept. 3';
   addDept: dept4 withId: 'Dept. 4';
   addDept: dept5 withId: 'Dept. 5'.

(WFCompanyBase at: 'C1')
   addAgent: agent1 withId: 'h001';
   addAgent: agent2 withId: 'h002';
   addAgent: agent3 withId: 'h003';
   addAgent: agent4 withId: 'a001';
   addAgent: agent5 withId: 'a002'.
```

In einem bestehenden Betrieb können Abteilungen und Agenten auch entfernt werden. Wenn für einen Betrieb alle Agenten neu definiert werden, so müssen alle alten zuerst herausgelöscht werden.

Beispiel 4.9 *Löschen aller Agenten und Abteilungen*

```
(WFCompanyBase at: 'C1')
   removeAllAgents;
   removeAllDepts.
```

Löschen einzelner Agenten oder Abteilungen

```
(WFCompanyBase at: 'C1')
   removeAgentWithId: 'h001';
   removeDeptWithId: 'Dept. 1'.
```

Alle so angelegten Betriebe stehen dem System zur Verfügung und können in den Organisationspolitiken verwendet werden.

4.2.2 Datenmodellierung

Jeder Akt (Folder) enthält folgende Informationen: eine eindeutige Nummer (`id`), den Ausführungsstatus (`status`), den Agenten, der den Workflow gestartet hat und eine Verarbeitungshistorie (`history`). In einem Workflow kann nur ein Akttyp behandelt werden. Er muß eine von der Klasse `Folder` abgeleitete Klasse sein. Die für den konkreten Workflow relevanten Daten müssen als Instanzvariablen der neuen Folder-Klasse definiert werden. Eine Klasse kann aber sehr wohl von mehreren Workflow Typen benutzt werden.

Die Definition kann direkt im GemStone System Browser oder im Workspace als Subklasse von `Folder` erfolgen.

Beispiel 4.10 *Folder für eine Kreditapplikation*

```
Folder subclass: 'CreditFolder'
  instVarNames: #('name' 'address' 'birthdate' 'amount')
  classVars: #()
  poolDictionaries: #[]
  inDictionary: WFObjects
  constraints: #[ #[ #name, String],
                  #[ #address, String],
                  #[ #birthdate, DateTime],
                  #[ #amount, Integer] ]
  isInvariant: false
```

Bei jedem Folder sollen zu einer Aktivität Kommentare angebracht werden können. Diese Kommentare können vor oder während der Bearbeitung einer Aktivität vom jeweiligen Agenten gelesen werden. Es handelt sich dabei z.B. um Informationen für die nachfolgenden Bearbeiter. Diese Funktionalität wurde aber aus zeitlichen Gründen noch nicht implementiert.

4.2.3 Integration

Bevor zur Aktivitätenmodellierung übergegangen wird, soll zuerst noch die Integration besprochen werden, weil begleitend zur Definition einer Aktivität auch die Organisationspolitik und die Datenflußbeschreibung angegeben werden kann bzw. üblicherweise angegeben wird.

4.2.3.1 Verbindung von Organisation und Aktivitäten: Organisationspolitiken

Die Auflösung der Organisationspolitik erfolgt dynamisch, also während des Programmablaufs, sodaß eine möglichst aktuelle Zuordnung von Aufgaben zu ausführenden Personen durchgeführt werden kann.

Die Aktionsmethoden sind bei den Rollentypen der Rollentyphierarchie definiert. Bevor eine solche Methode ausgeführt werden kann, muß der Agent zuvor in die entsprechende Rolle schlüpfen. Zum Zeitpunkt der *Agent Selection* steht in den meisten Fällen — Ausnahmen sind bestimmte Abfragekategorien — bereits fest, welche Rolle der Agent zu übernehmen hat, um die angegebene Aktion auszuführen. Das heißt, eine Organisationspolitik soll so eingeschränkt wie möglich (am besten Rolle und Gruppe) sein. Bei Mehrdeutigkeiten wird vom System das erste entsprechende Rollenobjekt verwendet.

Eine Organisationspolitik besteht immer aus einem Bedingungs- und Selektionsteil. Der *Bedingungsteil* gibt an, unter welchen Umständen die Selektion ausgewertet wird. Sie ist vor allem für die kontextabhängige Politik interessant. Da jede Politik als ECA-Regel abgelegt wird, ist die Bedingung so anzugeben, daß das Ergebnis der Bedingung als Rückgabewert deklariert wird. Der Grund dafür ist, daß TriGS eine Bedingung als Methode speichert, die bei der Evaluierung ausgeführt wird. Ist der Rückgabewert wahr (`true`), so ist die Bedingung erfüllt. Eine immer wahre Bedingung lautet somit `^true`. Die Rückgabe wird mit dem Symbol `^` angedeutet. Der *Selektionsteil* nimmt die Abfragekategorien auf. Das Ergebnis der Auswertung der Selektion sind Agenten. Sie befinden sich bereits in der zur Ausführung benötigten Rolle.

Die Organisationspolitik `(a playsRole:#ProjManager inGroup:#Project-Dept)` bestimmt Rolle und Gruppe eindeutig. Alle anderen Abfragefunktionen lassen dem System einen gewissen Spielraum in der Festlegung der Rolle und Gruppe eines Agenten.

Nummern-, workflow- und aktivitätenbezogene Abfragen geben keinen Aufschluß über Rolle oder Gruppe. Deshalb muß aus allen möglichen Rollen eine ausgewählt werden, die die Methode versteht. Gibt es keine solche Rolle, dann liegt ein Workflow Designfehler in den Organisationspolitiken dahingehend vor, daß ein Agent eine Methode ausführen soll, die er nicht versteht. Solche Fehler können zwar durch ein gutes Workflow Design Tool verringert werden, durch dynamische Änderung der Organisationsstruktur aber nicht völlig ausgeschlossen werden.

Ein anderer Fehler ergibt sich dadurch, daß überhaupt kein Agent selektiert wird. Der Grund dafür liegt aber in der dynamischen Änderung der Agentenbelegung (z.B. viele Mitarbeiter sind auf Urlaub), denn es wurden der Organisationspolitik nach keine passenden Agenten gefunden. Diese Unzulänglichkeiten können zwar vom Design Tool zum Modellierungszeitpunkt ausgeschlossen werden, wegen späterer dynamischer Änderungen aber durchaus vorkommen.

Das nächste Problem ist die Auswertung der Abfragefunktionen, da diese beliebig miteinander verknüpft werden können. Zur Zeit wird eine Rolle oder Gruppe aufgenommen, wenn die zugehörige Abfragefunktion als *wahr* evaluiert wird. Nur wenn der gesamte Ausdruck (Organisationspolitik) wahr ist, wird der Agent in die Auswahl der möglichen Ausführenden aufgenommen.

Wegen der möglichen Probleme mit Organisationspolitiken empfiehlt es sich, alle sorgfältig auszuwählen und vorher zu testen, ob sie überhaupt erfüllt werden können.

Abfragekategorien Die drei verschiedenen Kategorien von Abfragen in einer Politik können wie ein boolscher Ausdruck miteinander verknüpft werden. Jede Abfragefunktion wird in einem Klammernpaar *(...)* zusammengefaßt. Vor jeder Funktion muß noch der Buchstabe `a` als Abkürzung für *agent* stehen (Ausnahme: Hilfsfunktionen).

nummernbezogene Funktionen:

Wenn nicht unbedingt notwendig, soll die folgende Funktion nicht verwendet werden, da sie die Flexibilität des Workflow Systems sehr einschränkt. Es kann aber manchmal von Nutzen sein, einen Agenten direkt, das heißt über seine `id`, auszuwählen. Es ist aber immer günstig, Rolle und Gruppe zusätzlich zu spezifizieren, da bei mehreren gleichnamigen *Aktionen* — eine Aktion ist die Methode einer Aktivität — ein Ausführender vom System bestimmt wird, und diese Wahl nicht erwünscht ist. Die Zusammensetzung verschiedener Funktionen wird etwas später behandelt.

- `hasId: anId`

  Ein Agent wird durch seine innerhalb der gesamten Organisation eindeutige Nummer identifiziert und ausgewählt.

  Beispiel 4.11 *Der Agent mit der Nummer* `4711` *soll ausgewählt werden.*

  ```
  (a hasId: #4711)
  ```

rollen- und gruppenbezogene Funktionen:

Sie werden durch Funktionen realisiert, die auf das im Prototyp verwendete Rollen- und Gruppenkonzept aufbauen. Agenten werden durch Rolle und/oder Gruppe ausgewählt. Erfüllen mehrere Agenten die Politik, so wird vom System ein Ausführender bestimmt.

- `playsRole: aRole inGroup: aGroup`

  Durch Angabe von Rolle *und* Gruppe werden die Agenten am besten ausgewählt. Obwohl die Auswahl sehr beschränkt ist, verbleibt doch noch jene Flexibilität, die durch andere Funktionen wie etwa `hasId:` nicht gegeben ist. Es gibt auch keine Alternative in der Auswahl der Rolle bzw. Gruppe, wie das in `playsRole:` bzw. `isMemberOfGroup:` möglich ist.

Beispiel 4.12 *Auswahl eines Agenten, der die Rolle* `ProjectManager` *in der Abteilung* `DeptProject` *spielt:*

```
(a playsRole: #ProjectManager inGroup: #DeptProject)
```

- `playsRole: aRole`

  Es werden jene Agenten ausgewählt, die eine bestimmte Rolle spielen. Welcher Gruppe sie angehören, wird nicht berücksichtigt. Gibt es mehrere solche Agenten, so wird — egal in welcher Gruppe — vom System einer bestimmt.

  Beispiel 4.13 *Ein Agent mit der Rolle* `ProjectManager` *(gleichgültig in welcher Gruppe):*

  ```
  (a playsRole: #ProjectManager)
  ```

- `isMemberOfGroup: aGroup`

  Die Agenten einer bestimmten Gruppe werden ausgewählt. Analog zu `playsRole:` wird bei Mehrfachergebnissen ein Agent durch das System ausgewählt.

  Beispiel 4.14 *Ein Agent der Abteilung* `DeptProject` *(gleichgültig in welcher Rolle):*

  ```
  (a isMemberOfGroup: #DeptProject)
  ```

- `playsAllRoles: aSetOfRoles`

  Bestimmt die Agenten, die alle Rollen der angegebenen Menge spielen.

- `playsOneOfRoles: aSetOfRoles`

  Jeder Agent, der mindestens eine der angegebenen Rollen spielen kann, wird ausgewählt.

- `isMemberOfAllGroups: aSetOfGroups`

  Bestimmt die Agenten, die allen Gruppen der angegebenen Menge angehört.

- `isMemberOfOneOfGroups: aSetOfGroups`

  Jeder Agent, der zu mindestens einer der angegebenen Gruppen gehört, wird ausgewählt.

Die folgenden zwei Funktionen entsprechen den oben angegebenen mit der Erweiterung, daß auch jene Agenten dazugezählt werden, die mittels einer Rolle einer Unterabteilung zugeordnet sind:

- `playsRole: aRole inGroupOrSub: aGroup`
- `isMemberOfGroupOrSub: aGroup`

workflow- und aktivitätenbezogene Funktionen:

Sie dienen zur Auswahl von Agenten in Bezug auf Tätigkeiten im System. Das heißt es werden Agenten danach bestimmt, ob sie eine bestimmte Aktivität eines Workflows (und eventuell auch auf einem angegebenen Akt) schon einmal ausgeführt haben oder nicht. Auch bei diesen Funktionen ist eine Kombination mit Rollen und/oder Gruppen empfehlenswert, welche wieder später unter komplexen Politiken beschrieben wird.

```
hasPerformedActivity: anActId
          inWorkflow: aWfId

hasPerformedActivity: anActId
          inWorkflow: aWfId
          withFolder: aFolderId

hasInitiatedWorkflow: aWfId
          withFolder: aFolderId
```

Die erste Funktion bestimmt jene Agenten, die eine bestimmte Aktivität im angegebenen Workflow schon einmal ausgeführt haben. Die zweite Funktion bezieht einen bestimmten Folder mit ein. Die dritte wählt jenen aus, der den angegebenen Workflow mit dem eintreffenden Folder gestartet hat. Soll der Agent ausgewählt werden, der den aktuellen Workflow gestartet hat, so muß für den aktuellen Folder das Schlüsselwort `actFolder` verwendet werden. Um nun die Nummer für den Parameter `aFolderId` zu erhalten, schreibt man: `(actFolder id)`.

Beispiel 4.15 *Auswahl des Agenten, der die Aktivität mit der Nummer* `003` *im Workflow Typ* `STIP` *auf dem aktuellen Folder ausgeführt hat.*

```
(a hasPerformedActivity: #003 inWorkflow: #STIP
                              withFolder: (actFolder id))
```

Hilfsfunktionen Diese Funktionen sollen das Spektrum der Selektionsfunktionen erweitern. Sie liefern keine boolschen Werte, sondern Rollen, Gruppen, Aktivitäten oder Agenten und sind für die folgenden Funktionen als Parameter zu verwenden:

```
playsAllRoles: aSetOfRoles
playsOneOfRoles: aSetOfRoles
isMemberOfAllGroups: aSetOfGroups
isMemberOfOneOfGroups: aSetOfGroups
```

Da die Parameter immer eine Menge von Rollen oder Gruppen darstellen, werden dafür zwei Hilfsfunktionen vorgesehen, die die jeweilige Menge des Agenten mit der Nummer `anId` liefern:

```
rolesOfAgent: anId
groupsOfAgent: anId
```

Beispiel 4.16 *Um alle Agenten auszuwählen, die die gleiche Qualifikation wie ein anderer Agent haben, verwendet man etwa folgendes Statement:*

```
(a playsAllRoles: (rolesOfAgent: #4711))
```

Komplexe Politik Alle oben angeführten Abfragefunktionen mit boolschem Ergebnis können mit logischen Operatoren — wie etwa *and* und *or* — und den allgemein gültigen Verknüpfungs- und Vorranggesetzen zu komplexen Organisationspolitiken zusammengesetzt werden. Statt `and` und `or` werden in OPAL die Zeichen `&` und `|` verwendet.

Beispiel 4.17 *Es soll ein Agent ausgewählt werden, der die Rolle* `Manager` *und* `Clerk` *spielt, oder die Nummer* `4711` *hat.*

```
((a playsRole: #Manager)&(a playsRole: #Clerk)|(a hasId: #4711))
```

Kontextabhängige Politik Diese Politik ist die einzige, die den Bedingungsteil einer Organisationspolitik benützt. Alle anderen Politiken haben im allgemeinen die Bedingung `^true`. Der Kontextbezug zum aktuellen Folder wird durch das Schlüsselwort `actFolder` und dem darauf folgenden Attribut angegeben. Will man sich auf einen anderen Folder beziehen, so gibt man direkt seine Referenz in der Datenbank an, z.B. `WFWorkflows folders at: #fx003`. Wenn es mehrere Alternativen gibt, so müssen diese in einzelne Politiken aufgeteilt werden.

Beispiel 4.18 *Wenn der Wert des aktuellen Kreditantrages ÖS 500.000.- übersteigt, dann soll der Leiter der Bank über die Genehmigung entscheiden, ansonsten der involvierte Sachbearbeiter. Zwei Alternativen (*$> 500.000,-$ *bzw.* $<= 500.000,-$*) ergeben zwei Politiken:*

1. *Condition:* `^(actFolder amount > 500000)`
   Selection: `(a playsRole: #Manager)`
2. *Condition:* `^(actFolder amount <= 500000)`
   Selection: `(a playsRole: #Clerk)`

Wird eine Organisationspolitik von mehreren Agenten erfüllt, so wird vom System der erste gefundene Agent zur Ausführung zugeordnet, der die Aktion auch wirklich ausführen kann, d.h. eine Rolle spielen kann, die die Aktion versteht.

Die Organisationspolitik wird bei der Aktivitätendefinition mit der Methode `orgPol:cond:sel` mit der Nummer der Alternative, dem Bedingungs- und Selektionsteil als Parameter angegeben.

4.2.3.2 Verbindung von Aktivitäten und Daten: Datenflußbeschreibung

Aktivitäten und Akte (Folder) werden über die Namen der zu lesenden oder zu schreibenden Instanzvariablen verbunden. Die Eingangsparameter einer Aktionsmethode sind jene Daten des Aktes, die gelesen werden (Input-Variablen).

Die Rückgabewerte der Aktionsmethode (der Reihe nach in ein Feld geschrieben) werden in die Instanzvariablen des Aktes gespeichert, die bei der Aktivität zum Schreiben vorgesehen sind (Output-Variablen).

Konkret geht das so vor sich, daß bei einer Aktivität neben der Aktionsmethode (`action:`) auch die Ein- und die Ausgabevariablen (`inInstVars:`, `outInstVars:`) angegeben werden. Die anzugebenden Variablennamen sind die Instanzvariablen des zugehörigen Aktes.

Beispiel 4.19 *Die Aktionsmethode* `computeAge:` *verlangt als Eingabeparameter ein Datum und liefert als Ausgabeparameter das Alter als* `SmallInteger`*.*

```
act1 action: 'computeAge:';
     inInstVars: #('birthdate').
     outInstVars: #('age');
```

4.2.4 Aktivitätenmodellierung

In der *Aktivitätenmodellierung* werden nicht nur die Aktivitäten und deren Ausführungsreihenfolge bestimmt, sondern zugleich auch Organisation und Daten integriert.

4.2.4.1 Aktivitäten

Da es im Laufe der Bearbeitung eines Aktes mehrere Sonderfälle gibt, wurde für jeden dieser Fälle eine eigene Aktivität entworfen. Gemeinsam haben alle Arten, daß sie einem Workflow Typ zugeordnet werden müssen (`workflowType:`), und daß eine Organisationspolitik (`orgPolicy:cond:sel:`) und eine Zuordnungsart (`orgPolicyBind:`) definiert werden müssen.

Startaktivität:

Diese Aktivität ist die erste, die nach dem Start eines Workflows verteilt bzw. ausgeführt wird. Da sie ganz am Anfang des Netzes steht, kann sie keine Vorgänger haben. Nummer (`id:`) bzw. Name (`name:`) sind als `START` bzw. `Start` vordefiniert.

Beispiel 4.20 *Definition einer Startaktivität:*

```
act1 := StartActivity new.
act1 workflowType: wf;
     action: 'computeAge:';
     inInstVars: #('birthdate');
     outInstVars: #('age');
     orgPolicyBind: #less;
     orgPolicy: 1
        cond: '^true'
        sel:  '(a playsRole: #ProjectManager)'.
```

Endeaktivität:

Sie ist die letzte bei der Abarbeitung eines Workflows. Es werden wiederum keine Vorgänger angegeben, da diese aufgerufen wird, wenn keine Nachfolger einer inneren Aktivität mehr definiert sind. Die Endeaktivität selbst kann auch keine Nachfolger mehr haben. Analog zur Startaktivität sind auch hier Nummer und Name von vornherein mit `END` und `End` festgelegt. Da die Aktionsmethode als `confirmEnd` festgelegt ist, müssen weder Aktion noch Eingabe- und Ausgabevariablen festgelegt werden. Es kann aber jederzeit eine neue Aktion angegeben werden.

Beispiel 4.21 *Definition einer Endeaktivität:*

```
act5 := EndActivity new.
act5 workflowType: wf;
     orgPolicyBind: #less;
     orgPolicy: 1
     cond: '^true'
     sel:  '(a playsRole: #CompManager)'.
```

Ausnahmeaktivität:

Die Ausnahmeaktivität eines Workflow Typs wird dann ausgeführt, wenn zwar Nachfolger einer Aktivität definiert sind, aber deren Ausführungsbedingung nicht erfüllt ist. In diesem Fall soll lediglich jemand vom Auftreten des Problems informiert werden. Informieren heißt, daß der Agent die Ausnahmeaktivität wie jede andere in seine Arbeitsliste gestellt bekommt. Wie die Endeaktivität hat auch die Ausnahmeaktivität eine vordefinierte Aktion, nämlich `confirmException`, die auch nur die Information eines Agenten erlaubt. Sie kann ebenfalls keine Nachfolger haben.

Beispiel 4.22 *Definition einer Ausnahmeaktivität:*

```
act6 := ExceptionActivity new.
act6 workflowType: wf;
     orgPolicyBind: #less;
     orgPolicy: 1
        cond: '^true'
        sel:  '(a playsRole: #CompManager)'.
```

Innere Aktivität:

Diese Aktivität ist die flexibelste von allen. Sie kann sowohl Vorgänger als auch Nachfolger haben und wird deshalb zum Aufbau des Aktivitätennetzes verwendet. Sie wird durch eine Nummer (`id:`) eindeutig identifiziert. Weiters müssen der Name (`name:`), der Workflow Typ (`workflowType:`), die Aktionsmethode (`action:`), die Instanzvariablen zur Ein- und Ausgabe der Folderdaten (`inInstVars:`, `outInstVars:`), die Ausführungsbedingung (`execCondition:`), die Organisationspolitiken (`orgPolicy:-cond:sel:`) und die Zuordnungsart (`orgPolicyBind:`) angegeben werden. Es ist zu beachten, daß nur bei dieser Aktivität eine Ausführungsbedingung angegeben werden kann.

Beispiel 4.23 *Definition einer Inneren Aktivität:*

```
act4 := InnerActivity new.
act4 id: '004';
     name: 'Aktivitaet 4';
     workflowType: wf;
     action: 'getMachineName';
     inInstVars: #();
     outInstVars: #('address');
     execCondition: '^true';
     orgPolicyBind: #less;
     orgPolicy: 1
        cond: '^true'
        sel:  '(a playsRole: #Machine)'.
```

Die Gemeinsamkeiten und wesentlichen Unterschiede der gerade definierte Arten sollen nocheinmal kurz herausgehoben werden: Jede Art muß einem Workflow Typ zugeordnet sein und eine Organisationspolitik und Zuordnungsart enthalten, um ausführbar zu sein. Bei der Start-, Ende- und Ausnahmeaktivität sind Nummer und Name vordefiniert, können aber nachträglich geändert werden. Ende- und Ausnahmeaktivität haben zusätzlich eine vordefinierte Aktionsmethode, womit auch die Ein- und Ausgabeinstanzvariablen des Aktes vorbestimmt sind (nämlich keine!). Nur für eine Innere Aktivität kann eine Ausführungsbedingung definiert werden. Eine Startaktivität kann Nachfolger, eine Innere Aktivität muß Vorgänger und kann Nachfolger haben.

4.2.4.2 Organisationspolitiken: Agentenauswahl und -zuordnung

Die Organisationspolitiken werden anhand der Methode (`orgPolicy:cond:-sel:`) definiert, wobei der erste Parameter die Nummer der Politik darstellt. Wenn mehrere Alternativpolitiken angegeben werden, so müssen sie der Reihe nach numeriert werden. Durch Angabe dieser Nummer kann eine Politik mit der Methode `removeOrgPolicy:` wieder gelöscht werden. Wird bei der Definition zweimal die gleiche Nummer vergeben, so wird die zuerst definierte Politik überschrieben. Mit der Angabe der Organisationspolitik ist die Auswahl der Agenten definiert (*Agentenauswahl*).

Nun gibt es aber noch eine weitere Methode im Zusammenhang mit Organisationspolitiken: `orgPolicyBind:`. Diese legt fest, welche der ausgewählten Agenten zur Ausführung herangezogen werden (*Agentenzuordnung*). Dazu gibt es mehrere Möglichkeiten:

`#less` Damit wird jener Agent zur Ausführung herangezogen, dessen Arbeitsliste die wenigsten Aufgaben enthält. Diese Art wird am häufigsten verwendet.

`#random` Der Agent wird nach einem Zufallsverfahren ausgewählt.

Es gibt zwar noch weitere Möglichkeiten, von denen die einen wenig Sinn haben, die anderen nicht implementiert sind. Es hat etwa wenig Sinn, den ersten (`#first`) oder letzten (`#last`) der ausgewählten zu bestimmen. Es kann aber sehr wohl von Nutzen sein, mehreren Agenten die Aufgabe zu übertragen, und nur die Ergebnisse des ersten beendeten zu übernehmen. Im Prototyp ist aber eine Mehrfachvergabe von Aufgaben nicht vorgesehen.

Zu beachten ist, daß sowohl für die Agentenauswahl als auch für deren Zuordnung ECA-Regeln erzeugt werden. Deshalb ist es auch notwendig, die Organisationspolitiken über die vorgegebenen Methoden zu entfernen, damit alle Regeln mitgelöscht werden (siehe auch 4.4.4).

Es fehlen noch die Angaben über Vorgänger- und Nachfolgeaktivtiäten. Diese werden erst am Ende der Aktivitätendefinition angegeben, da dafür die Objekte bekannt sein müssen.

4.2.4.3 Aktivitätennetz: Vorgänger und Nachfolger

Es gibt zwei Möglichkeiten, ein Aktivitätennetz aufzustellen. Entweder definiert man die Nachfolger einer Aktivität oder man läßt eine Aktivität Nachfolger einer anderen werden. Das bedeutet, daß nicht nur eine Innere Aktivität Nachfolger definieren kann, sondern auch eine Startaktivität. Aus implementierungstechnischen Gründen wurde aber nur zugelassen, einer Inneren Aktivität zu sagen, sie solle Nachfolger einer anderen Inneren Aktivität oder auch der Startaktivität werden. Dazu wird bei der Nachfolgeaktivität die Methode `becomeSuccessorOf:` mit der Vorgängeraktivität als Parameter aufgerufen. Wie die Organisationspolitiken werden auch diese Nachfolgerrelationen in ECA-Regeln abgelegt (siehe 4.4.4). Eine solche Beziehung wird entfernt, indem dem Nachfolger die Methode `removeSuccessorTo:` wieder mit dem Vorgänger als Parameter geschickt wird.

Beispiel 4.24 *Festlegen der Nachfolger im Aktivitätennetz:*

```
act2 becomeSuccessorOf: act1.
act3 becomeSuccessorOf: act1.
act4 becomeSuccessorOf: act2.
act4 becomeSuccessorOf: act3.
...
```

Löschen einer solchen Beziehung:

```
act3 removeSuccessorTo: act1.
```

4.2.4.4 Workflow Typ

Der Workflow Typ faßt Organisation, Daten und Aktivitäten zusammen. Er wird durch die eindeutige Nummer (`id:`) gekennzeichnet. Zusätzlich müssen der

Name (`name:`), der Typ der Akte, die verarbeitet werden können (`folderType:`) und die Organisationspolitiken (`startOrgPolicy:cond:sel:`) für die Initiierung eines Durchlaufs angegeben werden. Denn es dürfen nur jene Agenten einen Workflow starten, die die Startorganisationspolitik erfüllen. Weiters werden die drei Sonderfälle der Aktivitäten angegeben: Startaktivität, Endeaktivität und Ausnahmeaktivität. Alle Workflow Typen müssen im Objekt `WFWorkflowBase` gespeichert werden. Das ist eine Instanz der Klasse `WorkflowBase`, die von `Dictionary` abgeleitet ist. Die Klasse verhält sich wie ein Dictionary, weist lediglich eine zusätzliche Methode auf, die zum Start gestoppter Workflows dient. Näheres dazu wird unter Punkt 4.4.4 beschrieben.

Beispiel 4.25 *Definition des Workflow Typs* `STIP`

```
wf := WorkflowType new.
wf id: 'STIP';
   name: 'Stipendienbehandlung';
   company: (WFCompanyBase at: 'C1');
   folderType: StudentFolder;
   startOrgPolicy: 1
      cond: '^(actFolder name first <= $P)'
      sel: '(a hasId: #h003)';
   startOrgPolicy: 2
      cond: '^(actFolder name first >= $Q)'
      sel: '(a hasId: #h004)'.

wf startActivity: act1;
   endActivity: act5;
   exceptionActivity: act6.

WFWorkflowBase at: (wf id) put: wf.
```

Die Organisationspolitiken bedeuten, daß der Agent mit Nummer `h003` *nur Stipendienanträge der Antragsteller von* `A-P`*, und jener mit Nummer* `h004` *nur Anträge von* `Q-Z` *starten kann.*

Das gesamte Aktivitätennetz ist implizit durch die Nachfolger der Startaktivität definiert und muß hier nicht separat mit einer Methode des Workflow Typs angelegt werden.

Der Workflow Typ enthält unter anderem eine sehr wichtige Methode: `removeAllDynamicRules`. Diese Methode entfernt alle mit dem Workflow Typ in Zusammenhang stehende Regeln. Das sind alle Regeln zur Agentenauswahl und -zuordnung bei den Aktivitäten, die Nachfolgerrelationen zwischen den Aktivitäten und auch die Startorganisationspolitik des Workflow Typs selbst. Es gilt also nicht wie bei herkömmlichen GemStone-Objekten, daß sie gelöscht sind, wenn sie nicht mehr referenziert werden. *Im Zusammenhang mit TriGS müssen alle Regeln explizit entfernt werden!*

4.3 Ablauf in TriGS$_{flow}$

Nachdem ein Workflow Typ angelegt ist, kann er ausgeführt werden. Ausführen bedeutet, mit einem neuen Akt (Folder) einen Workflow zu starten. Bevor detailliert auf den Durchlauf im Prototyp eingegangen wird, folgt eine kurze Wiederholung der Abarbeitung aus 2.2.

4.3.1 Kurzbeschreibung einer Aktivitätenabarbeitung

Analog zur Beschreibung im theoretischen Teil (siehe 2.2) werden hier die beiden Abschnitte *Ausführung* (*Execution*) und *Arbeitsverteilung* (*Work Distribution*) dargestellt. Bevor ein Agent eine Aktivität ausführen kann, muß sie ihm zuerst übermittelt werden. Übermitteln bedeutet, es werden alle Nachfolgeaktivitäten gesucht, jede einzelne verteilt und dabei dem entsprechenden Agenten zugesandt. Das wäre eigentlich die richtige Reihenfolge. Die Schritte sind aber besser zu erklären, wenn man von bereits verteilten Aufgaben ausgeht.

Ausführung (Execution):
Wählt ein Agent eine Aufgabe aus seiner Arbeitsliste zur Ausführung aus, so werden die Daten aus dem Akt gelesen (*Datenbereitstellung*), die Aktionsmethode vom entsprechenden Rollenobjekt ausgeführt (*Aktionsausführung*) und danach die Daten wieder in den Folder gespeichert (*Datenübernahme*). Anschließend folgt die Arbeitsverteilung, außer es wurde gerade eine Ende- oder Ausnahmeaktivität ausgeführt.

Arbeitsverteilung (Work Distribution):
Jede Nachfolgeaktivität wird nach Ende der Ausführung eines Arbeitsschrittes verteilt. Wenn der Durchlauf erst gestartet wird, so beginnt dieser mit der Verteilung der Startaktivität. Verteilen in diesem Sinne heißt, die Organisationspolitik auflösen (*Agentenauswahl, Agent Selection*), einen der ausgewählten Agenten zuordnen (*Agentenzuordnung, Agent Binding*) und diesem die Aufgabe in die Arbeitsliste stellen (*Agentenbenachrichtigung, Agent Notification*).

Es wird nicht jede Nachfolgeaktivität ausgeführt, sondern nur jene, deren Ausführungsbedingung erfüllt ist. Sind gar keine Nachfolger definiert, so wird die Endeaktivität ausgeführt. Sind zwar welche angegeben, sind aber alle deren Ausführungsbedingungen nicht erfüllt, so wird die Ausnahmeaktivität ausgeführt. Bei Zusammenführungen wird die Arbeitsverteilung der Nachfolgeaktivität solange verzögert, bis der letzte Pfad abgeschlossen ist.

Ende- und Ausnahmeaktivitäten werden grundsätzlich nicht mehr verteilt.

In diesem Prototyp wird die Abarbeitung nach dem obigen Schema durchgeführt. Die nun folgende Aufzählung von Objekten und deren Methoden entspricht der Abarbeitungsreihenfolge der wichtigsten Methoden im System. Vorerst wird eine fehlerfreie Ausführung der Aktion und die problemlose Arbeitsverteilung vorausgesetzt. Eventuell auftretende Fehler werden erst später behandelt.

4.3.2 Ausführung einer Aktivität

Die Ausführung beginnt damit, daß ein Teilnehmer eine Aktivität aus seiner Arbeitsliste auswählt. „Menschliche“ Teilnehmer (`HumanAgent`) können eine beliebige Aufgabe auswählen, sofern keine Ausführungszeitbeschränkungen oder Prioritäten vorliegen. Die automatische Auswahl der Aufgaben von Programmen (`SoftwareAgent`) muß nicht von einem Prozeß gesteuert werden, sondern kann durch den Einsatz von ECA-Regeln durchgeführt werden (siehe Abschnitt 4.4).

Um eine Aufgabe auswählen zu können, besitzt die Klasse `HumanAgent` die Methode `selectElem:`, der als Parameter die Nummer des Arbeitslisteneintrags mitgegeben wird. Der Klasse `SoftwareAgent` steht die Methode `selectElem` ohne Parameterübergabe zur Verfügung. Sie wählt als Voreinstellung den ersten Eintrag aus. Um einen Arbeitsschritt auszuführen genügt es also, einem `HumanAgent` die Methode `selectElem:` (siehe Beispiel 4.26) und einem `SoftwareAgent` die Methode `selectElem` (siehe Beispiel 4.27) zu schicken. Alles weitere wird vom System erledigt.

Beispiel 4.26 *Der menschliche Agent (aus dem Betrieb* `C1`*) mit der Nummer* `h001` *wählt den dritten Eintrag zur Bearbeitung aus:*

```
| aHumanAgent |
aHumanAgent := (WFCompanyBase at: 'C1') agents at: 'h001'.
aHumanAgent selectElem: 3.
```

Beispiel 4.27 *Der Software Agent (aus dem Betrieb* `C1`*) mit der Nummer* `a323` *soll einen (den ersten) Eintrag bearbeiten:*

```
| aSoftwareAgent |
aSoftwareAgent := (WFCompanyBase at: 'C1') agents at: 'a323'.
aSoftwareAgent selectElem.
```

Nachdem der entsprechende Eintrag ermittelt worden ist, wird die Methode `executeElem:` aufgerufen, und als Parameter wird der Eintrag vom Typ `WorklistItem` übergeben. Diese Methode regelt nun die gesamte Ausführung und stößt auch die Arbeitsverteilung an. Sie ist für beide Agententypen gleich, da sie bei der Superklasse `Agent` definiert ist.

4.3.2.1 Start eines Workflows

Bevor ein Akt durch das Aktivitätennetz „laufen“ kann, muß er zuerst eingeführt werden. Ein Durchlauf wird eröffnet, indem man dem Agenten die Methode `startWorkflow:withFolder:` mit der Nummer des Workflow Typen und einem gerade angelegten Folder als Parameter schickt. Das Beispiel 4.28 veranschaulicht diesen Vorgang.

Beispiel 4.28 *Der Agent (aus dem Betrieb* `C1`*) mit der Nummer* `h003` *versucht, im Workflow Typ* `STIP` *mit dem angegeben Akt einen Durchlauf (Workflow) zu starten.*

```
| aFolder anAgent  |

aFolder  := StudentFolder new.
aFolder  id: '9455345';
        name: 'Muster Karl';
        address: 'Kamingasse 34, 1136 Wien';
        birthdate: (DateTime newWithYear: 1972 month: 9 day: 20
                             hours: 0 minutes: 0 seconds: 0).

anAgent := (WFCompanyBase at: '1') agents at: 'h003'.
anAgent startWorkflow: 'STIP' withFolder: aFolder.
```

Der Durchlauf wird nur gewährt, wenn noch kein Akt mit derselben Nummer im Workflow Typ existiert.

4.4 Wichtige Implementierungsaspekte

4.4.1 Ausführung

Die Ausführung beinhaltet sowohl das Bearbeiten einer Aufgabe aus der Arbeitsliste als auch den Start eines Workflows. Beide Tätigkeiten werden anhand deren Methoden näher erläutert.

4.4.1.1 Ausführung und die Methode `executeElem:`

Nachdem eine Aufgabe aus der Arbeitsliste mit der Methode `selectElem` bzw. `selectElem:` ausgewählt ist, wird `executeElem:` mit dem Arbeitslisteneintrag als Parameter aufgerufen. In der folgenden Aufstellung wird die Methode in drei Hauptschritte zerlegt. Jeder Teil wird innerhalb einer Transaktion ausgeführt.

1. **Vorbereitung (Datenbereitstellung)**
   (a) Start einer Transaktion.
   (b) Schreibsperre auf Agenten selbst und Exklusivsperre des Aktes.
   (c) Die Argumente für die Aktionsmethode werden gemäß den Leseinstanzvariablen (`inInstVars`) — definiert bei der Aktivität — aus dem Folder herausgelesen und in ein Feld gestellt.
   (d) Ende der Transaktion.

(e) Löschen der Sperre des Agenten. Die Exklusivsperre des Aktes bleibt erhalten.

2. **Aktionsausführung und Datenübernahme**

   (a) Start einer Transaktion.

   (b) In die Historie wird der Ausführungsbeginn eingetragen. (Die ECA-Regel `R_History_BegExec` wird gefeuert.)

   (c) Ausführung der Aktion durch Aufruf der Methode `performAction:-withArguments:by:` mit den Leseargumenten, dem Methodennamen und dem Rollenobjekt, das die Methode ausführen soll, als Parameter.

   (d) In die Historie wird das Ausführungsende eingetragen. (Die ECA-Regel `R_History_EndExec` wird gefeuert.)

   (e) Wenn die Aktion erfolgreich abgeschlossen wurde, folgt die Übernahme der Daten aus dem Rückgabefeld in die entsprechenden Schreibinstanzvariablen (`outInstVars`) — definiert bei der Aktivität. Bei nicht erfolgreichem Abschluß (erstes Objekt im Rückgabefeld ungleich `#ok`) wird die Transaktion abgebrochen.

   (f) Ende der Transaktion.

3. **Nachbereitung (Arbeitsverteilung)**

   (a) Start einer Transaktion.

   (b) Schreibsperre des Agenten.

   (c) Verteilung der Nachfolger, wenn nicht gerade eine Ende- oder Ausnahmeaktivität ausgeführt wurde. Weiters wird eine Aktivitität nicht mehr verteilt, wenn inzwischen durch die Ausführung einer anderen Aktivität auf dem Folder eine Ausnahme aufgetreten ist. Noch bevor die Arbeitsverteilung in Angriff genommen wird, werden alle Nachfolgeaktivitäten gesperrt. Die Verteilung wird angestoßen, indem der gerade bearbeiteten Aktivität die Methode `distribute:` mit dem Folder als Parameter geschickt wird. Am Ende werden die Sperren auf den Nachfolgeaktivitäten wieder aufgehoben. Unter Punkt 4.4.2 wird die Arbeitsverteilung detailliert beschrieben.

   (d) Erst jetzt wird der bearbeitete Eintrag aus der Arbeitsliste entfernt.

   (e) Ende der Transaktion.

   (f) Löschen der Sperren des Agenten und des Folders.

4.4.1.2 Start eines Workflows mit der Methode `startWorkflow:withFolder:`

Anhand dieser Methode kann ein Workflow gestartet werden. Der Start wird nur gewährt, wenn noch kein Akt mit derselben Nummer im Workflow Typ existiert.

In der folgenden Aufzählung wird der Ablauf der Methode näher betrachtet. Die drei Punkte (...) hinter einem Regelnamen bedeuten, daß der Regelname ein Suffix hat und damit einer bestimmten Aktivität, Nachfolgerbeziehung oder einem Workflow Typ zugeordnet ist.

(a) Start einer Transaktion.

(b) Schreibsperre auf Workflow Typ und dessen Startaktivität.

(c) Durch Aufruf der Methode `isAllowedToStart:withFolder:` (beim Workflow Typ) wird die ECA-Regel `R_StartWFOrgPolicy_`... gefeuert. Sie wertet die Organisationspolitik für die Starterlaubnis aus.

(d) Die Methode `isAllowedToStart:withFolder:` prüft, ob der Agent in der Menge ist, wenn ja, darf er ausführen, wenn nein, so endet hier der Startversuch.

(e) Dem Workflow Typ wird die Methode `startBy:withFolder:` geschickt, worin versucht wird, den Folder aufzunehmen. Ist schon ein Akt mit gleicher Nummer vorhanden, so endet auch hier die Ausführung. Ansonsten wird der Akt aufgenommen, der initiierende Agent eingetragen und mit der Verteilung der Startaktivität fortgesetzt. Die Verteilung beginnt mit dem Aufruf der Methode `notifyAgent:` der Startaktivität. Das heißt, es fällt lediglich die Auswertung und Behandlung der Kontrollstrukturen weg.

(f) Nach der Verteilung erfolgt die Rückkehr zur Methode `startWorkflow:withFolder:`, wo wieder alle Sperren aufgehoben werden.

(g) Ende der Transaktion.

Kann ein Objekt nicht sofort gesperrt werden, so wird der Ablauf mit einer Warteschleife kurz verzögert und erneut versucht, die Sperrung durchzuführen. Arbeiten mehrere Teilnehmer gleichzeitig mit dem System, so kann es durchaus passieren, daß wegen einer von einem Agenten ausgelösten Arbeitsverteilung ein anderer nicht verteilen kann, da eines der betroffenen Objekte von ersterem gesperrt ist. Da diese Sperren ohnehin nur über wenige Sekunden aufrecht erhalten werden, müßte eine kleine Verzögerung des Prozesses ausreichen, um das Objekt zu sperren und somit erfolgreich fortsetzen zu können. Diese und ähnliche Probleme werden in 4.4.5 zusammengefaßt.

Anwendungsbedingte Fehler haben in der Ausführung einen Transaktionsabbruch und damit eine Wiederherstellung der Ausgangssituation vor der Aktivitätsausführung zur Folge. Ein solcher Fehler ist etwa das Einfügen eines Aktes unter einer Nummer, die schon vorhanden ist.

4.4.2 Arbeitsverteilung

Durch den Aufruf der Methode `distribute:` nach der Ausführung der Aktion, werden alle Nachfolger angestoßen. Die Nachfolgerbeziehung zweier Aktivitäten ist in einer Regel abgelegt. Welche Regeln verwendet werden, wird detailliert unter Punkt 4.4.4 dargstellt. Diese Regeln werden gefeuert, wenn der Vorgängeraktivitat die Methode `distribute:` geschickt wird. Was unmittelbar vor, innerhalb und nach dieser Methode geschieht, wird in der Folge dargestellt.

1. **Auswertung der Kontrollstrukturen**

   Noch vor Ausführung der Methode `distribute:` werden mittels der Methode `combine:fromActivity:` bei der Nachfolgeaktivität entsprechend der Kontrollstrukturen die Pfade bearbeitet. Weil für jede Vorgänger-Nachfolger-Beziehung eine Regel definiert ist, wird diese Methode genau so oft ausgeführt, wie Regeln definiert sind.

   (a) *Zusammenführung:*
   In `combine:fromActivity:` wird festgestellt, ob noch ein Pfad offen ist; wenn ja, wird nicht weiter gearbeitet und das System wartet auf das Ende eines anderen „Vorgängerpfades“. Wenn ja, so wurde der letzte Pfad beendet, und die Arbeitsverteilung kann mit der Auswertung der Organisationspolitik fortgesetzt werden, indem der Nachfolgeaktivität die Methode `notifyAgent:` — wieder mit dem Folder als Parameter — geschickt wird.

   (b) *Verzweigung:*
   Bei einer Verzweigung muß nicht gewartet werden, da nur eine Vorgängeraktivität vorhanden ist.

   (c) *Sequenz:*
   Eine Sequenz kann als Sonderfall einer Zusammenführung angesehen werden, weil nur ein Pfad kombiniert werden muß.

2. **Auswertung der Organisationspolitk**

   Bevor die Methode `notifyAgent:` zur Benachrichtigung tatsächlich ausgeführt wird, werden von TriGS einige Regeln gefeuert.

   (a) *Agentenauswahl (Agent Selection):*
   Als erstes wird die Regel `R_OrgPolicyInit` gefeuert, die die Auswahl der Agenten initialisiert. Anschließend werden alle Regeln mit dem Namen `R_OrgPolicy_...` ausgeführt, die die möglichen Alternativen der Politik auswerten.

   (b) *Agentenzuordnung (Agent Binding):*
   Nachdem die Agenten ausgewählt sind, wird einer von ihnen durch die Regel `R_OrgPolBinding_...` zugeordnet.

3. **Agentenbenachrichtigung (Agent Notification):**

   Jetzt kann der zugeordnete Agent informiert werden, d.h. die Methode `notifyAgent:` wird endlich ausgeführt. Konnte keiner zugeordnet werden, so muß der Durchlauf gestoppt werden.

   (a) Sperren des zu benachrichtigenden Agenten, sofern er nicht schon durch die Ausführung der Vorgängeraktivität gesperrt ist.

(b) Hinzufügen der Aufgabe durch die Methode `addToWorklist: folder:role:` mit der Aktivität selbst, dem Folder und dem zugeordneten Rollenobjekt als Parameter in die Arbeitsliste des zugeordneten Agenten.

(c) Nach der Ausführung der Methode wird die Regel `R_History_Notification` gefeuert, und damit ein neuer Eintrag in die Historie gemacht.

(d) Wenn der Agent zuvor gesperrt wurde, so wird die Sperre jetzt wieder aufgehoben.

Damit ist der Durchlauf beendet. Die Aktion wurde ausgeführt und alle Nachfolger sind verteilt worden, unter der Annahme, daß dabei keine Fehler aufgetreten sind. Welche Fehler auftreten können, wie und ob sie überhaupt behandelt werden, wird in 4.4.5 beschrieben. Besonderheiten der Arbeitsverteilung werden in 4.4.3 aufgezeigt.

4.4.3 Kontrollstruktur Zusammenführung

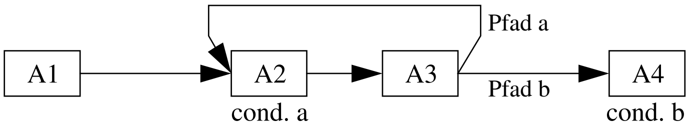
Einer genaueren Erklärung bedarf die Zusammenführung von Pfaden, da sie in Workflow Systemen von essentieller Bedeutung ist. Um eine korrekte Zusammenführung durchführen zu können, ist eine Graphanalyse notwendig. Das Aktivitätennetz wird durchsucht, in Teilgraphen aufgelöst und dann festgestellt, ob alle Teilgraphen fertig bearbeitet sind. Im theoretischen Teil wurde bereits erwähnt, daß im Prototyp keine derartige Analyse durchgeführt wird, sondern eine sehr simple Methode gefunden wurde, die aber nicht alle Aspekte einer vollständigen Graphanalyse berücksichtigt. Mit Hilfe der Historie wird festgestellt, ob alle Vorgänger bereits ausgeführt sind. Eine Aufgabe gilt als beendet, wenn im entsprechenden Historieneintrag der Zeitpunkt vom Ausführungsende gespeichert ist. Bei der Analyse wird folgendermaßen vorgegangen:

A. Suche alle definierten Vorgänger der Aktivität, bei der die Zusammenführung stattfindet.

B. Durchsuche alle Historieneinträge des Folders, die im Zusammenhang mit einer Vorgängeraktivität entstanden sind. Wird ein Eintrag gefunden, dessen Endzeit noch nicht gesetzt ist, so ist diese Aufgabe noch nicht beendet; damit kann bei der Zusammenführung noch nicht fortgesetzt werden. Teilpfade, die gar nicht ausgeführt wurden, bleiben unbeachtet, da sie nicht in der Historie aufscheinen.

Daß diese Methode einige gravierende Nachteile hat, liegt auf der Hand. Und doch stellt sie eine vorerst taugliche Lösung dar:

1. Die Bestimmung sämtlicher Vorgänger bei jeder Zusammenführung kostet sehr viel Zeit. Dieses Problem ließe sich durch laufende Weitergabe der Vorgänger an die Nachfolgeaktivität beschleunigen.



Die Ausführungsbedingungen a und b müssen für alle Belegungen komplementär sein, damit Pfad a und Pfad b nie gleichzeitig ausgeführt werden.

Abbildung 4.6: Unerlaubte Kontrollstruktur

2. Außerdem wird nur eine Art der Zusammenführung realisiert. Dabei wird auf alle Zweige gewartet, die tatsächlich ausgeführt wurden. Eine ODER-Zusammenführung ist nicht möglich, da das Transaktionskonzept nicht geeignet ist, um ganze Teilgraphen des Aktivitätennetzes zurückzusetzen.

3. Weiters verlangt diese Methode, daß das Aktivitätennetz ähnlich den Richtlinien der strukturierten Programmierung aufgebaut ist. D.h. es sind zwar Schleifen erlaubt, aber keine wahllosen Sprünge, die einem „GOTO“ entsprechen. Bei Schleifen muß gewährleistet sein, daß während sich der Aktenlauf in einer Schleife befindet, dieser bei der Abbruchbedingung nicht gespalten wird, und plötzlich zwei Aktenläufe mit ein und demselben Akt entstehen. Das bedeutet, daß die beiden Bedingungen, die für die Schleife und ihr Ende verantwortlich sind (Ausführungsbedingungen der Verzweigung am Schleifenende), komplementär zueinander sein müssen. Sind sie es nicht, und sind beide Bedingungen erfüllt, so wird der Durchlauf sowohl am Beginn der Schleife als auch nach der Schleife fortgesetzt, wodurch zwei Aktenläufe entstehen. Abbildung 4.6 veranschaulicht diesen Sachverhalt.

In [Rein93] wurde das Aktivitätennetz als bipartiter, gerichteter Graph definiert. Damit werden Schleifen, die der gerade erwähnten GOTO-Programmierung („Spaghetti-Programmierung“) entsprechen, von vornherein ausgeschaltet. In diesem Prototyp besteht aber das Problem, daß ohne Graphanalyse nicht festgestellt werden kann, ob das Netz den Richtlinien entspricht oder nicht. Man ist daher auf den richtigen Einsatz der Kontrollstrukturen durch den Anwender angewiesen.

4.4.4 ECA-Regeln im Workflow System

In diesem Abschnitt wird näher auf die Einsatzbereiche und den Nutzen von ECA-Regeln in diesem Prototyp eingegangen. Die Regeln werden schematisch anhand der in TriGS verwendeten Grammtik angeführt. Im Anhang D findet sich ein detaillierter Auszug aus der Regelbasis des Prototyps.

4.4.4.1 Mitprotokollierung von Historiendaten

In einem Workflow System müssen große Datenmengen sowohl während der Ausführung einer Aktivität als auch während des Ablaufs eines Workflows gesammelt werden. Diese können später — ähnlich den „Log-Files“ eines Datenbanksystems — dazu verwendet werden, Rückschlüsse zu ziehen, was im System, im speziellen im Zusammenhang mit der Bearbeitung eines Aktes, vorgegangen ist. In erster Linie sind das Informationen wie: wer hat was wann auf welchem Computer ausgeführt, und wie lange hat er dazu gebraucht.

Die Sammlung der arbeitsbezogenen Daten eines Teilnehmers birgt die Gefahr des Mißbrauchs in sich. An dieser Stelle soll aber nicht auf die ethische Verantwortung der Workflow Systembetreiber eingegangen werden.

Die Mitprotokollierung bestimmter Zeitpunkte in der Ausführung und die Information über die Ausführung der Vorgänge kann unabhängig von der eigentlichen Ausführung von Vorgängen durchgeführt werden. ECA-Regeln schaffen einen geeigneten Rahmen, um diese „nebensächlichen“ Vorgänge aus dem Programmcode zu entfernen und an anderer Stelle quasi als *beobachtende Einheit* anzusiedeln. Regeln *beobachten* die Programmabfolge und bekommen durch Ereignisse Meldung, wann was zu protokollieren ist.

Konkret werden im Prototyp bei einem Aktenlauf folgende Daten für jede Aktivität gesammelt:

bearbeitender Agent: Dies ist jener Agent, der nach der Zuordnung ausgewählt wurde, und dem dann die Aufgabe gesandt, d.h. in die Arbeitsliste gestellt wird.

Benachrichtigungszeitpunkt: Jener Zeitpunkt, zu dem dem Agenten die Aufgabe übermittelt wird.

Beginn der Aktionsausführung: Der Zeitpunkt, zu dem der Agent die Ausführung der mit der Aktivität verbundenen Aktion begonnen hat.

Ende der Aktionsausführung: Endzeitpunkt der Ausführung der Aktion.

Es gibt noch eine Unmenge anderer Daten die mitprotokolliert werden könnten, wie etwa der Ressourcenverbrauch durch den Start der Aktion, oder auf welchem Rechner die Aktion ausgeführt wird. Mit diesen Daten kann der Administrator dann auch Engpässe des Systems feststellen und Verbesserungen durchführen.

Zur Abdeckung der wichtigsten Protokolldaten werden drei Regeln benötigt, die im Gegensatz zu anderen workflowspezifischen (lokalen) Regeln nur einmal vorhanden sind und auch nie geändert werden müssen. Diese drei Regeln gelten also für alle Workflow Typen und deren Aktivitäten und werden in Zukunft auch als „statisch“ bezeichnet. Die Definition erfolgt in der für TriGS üblichen Notation.

1. **Benachrichtigungszeitpunkt**

```
DEFINE RULE R_History_notification
ON POST (Agent, addActivity: anAct
                       with: aFolder) DO
ON POST (rule_trgO, addActivity: anAct
                           with: aFolder)
   IF true THEN
ON POST (rule_trgO, addActivity: anAct
                           with: aFolder)
   EXECUTE rule_trgO addHistoryNotificationTo: aFolder
                                         with: anAct
```

Diese Regel feuert, nachdem die Methode `addActivity:with:` an einen Agenten geschickt worden ist. Der Aktionsteil wird unmittelbar ausgeführt, da die Bedingung immer wahr ist. Die Methode `addActivity:-with:` fügt der Arbeitsliste des Agenten, dem sie geschickt wird, einen Folder und die darauf anzuwendende Aktivität hinzu. Er wird also benachrichtigt. Die Aktivität, der Agent und der aktuelle Zeitpunkt werden protokolliert.

2. **Beginn der Aktionsausführung**

```
DEFINE RULE R_History_begExec
ON PRE (Agent, performAction: anAction
               withArguments: iArgs
                          by: aRole) DO
ON PRE (rule_trgO, performAction: anAction
                   withArguments: iArgs
                              by: aRole)
   IF true THEN
ON PRE (rule_trgO, performAction: anAction
                   withArguments: iArgs
                              by: aRole)
   EXECUTE rule_trgO addHistoryBegExecTo: aFolder
                                    with: anAct
```

Mittels der Methode `performAction:withArguments:by:` startet der Agent die Ausführung der Aktion. Die zu lesenden Daten und der Agent in der entsprechenden Rolle, die die Aktion versteht, werden dabei übergeben. Aufgrund des Schlüsselworts `PRE` wird der Aktionsteil der Regel vor dem Methodenaufruf ausgeführt. Im Aktionsteil wird das Attribut Ausführungsbeginn im bereits eingetragenen Historiensatz auf den aktuellen Zeitpunkt gesetzt. Ein Historiensatz wird durch das Objekt `HistoryItem` dargestellt (siehe dazu Anhang A).

3. **Ende der Aktionsausführung**

```
DEFINE RULE R_History_endExec
ON POST (Agent, performAction: anAction
                withArguments: iArgs
                           by: aRole) DO
ON POST (rule_trgO, performAction: anAction
                    withArguments: iArgs
```

```
                                by: aRole)
      IF true THEN
   ON POST (rule_trgO, performAction: anAction
                  withArguments: iArgs
                             by: aRole)
      EXECUTE rule_trgO addHistoryEndExecTo: aFolder
                                    with: anAct
```

Diese Regel ist analog zur Aufzeichnung des Aktionsbeginns aufgebaut. Der einzige Unterschied ist, daß sie wegen des Schlüsselworts POST erst nach dem Methodenaufruf gefeuert wird und damit das Ende aufzeichnet.

4.4.4.2 Fortsetzen gestoppter Workflows

Obwohl die Ursache für das Anhalten eines Aktenlaufs bereits mehrmals dargestellt wurde (zuletzt unter Punkt 4.4.2), wird noch einmal das für das Verständnis des Regelneinsatzes notwendige Umfeld geschildert.

Ein Workflow wird gestoppt, wenn bei der Auswertung der Organisationspolitik keine Agenten diese erfüllen, und damit auch keiner einer Aktivität zugeordnet werden kann. Grund dafür können einerseits falsche Organisationspolitiken (zu stark eingeschränkt oder überhaupt unerfüllbar) oder auch vorübergehende Abwesenheit (eventuell verursacht durch Krankheit oder Urlaub) von Agenten sein. Agenten werden als „abwesend“ erkannt, wenn sie vom System Administrator für „inaktiv“ erklärt werden.

Wenn ein neuer Agent angelegt wird oder ein bereits vorhandener wieder aktiviert wird, so kann es sein, daß die neue Konstellation an Agenten einen gestoppten Workflow wieder zum Laufen bringt. Deshalb wird das System nach gestoppten Workflows durchsucht und die gefundenen werden wieder angestoßen. Die Organisationspolitiken werden abermals ausgewertet, und wenn sie jetzt von jemandem erfüllt werden, kann der Workflow wieder fortgesetzt werden.

Damit die Überwachung des Anlegens oder Aktivierens eines Agenten nicht in den entsprechenden Methoden vorgenommen werden muß, sondern außerhalb definiert werden kann, kommen wieder ECA-Regeln zum Einsatz. Ereignisauslösend ist das Anlegen eines neuen Agenten und das Aktivieren eines Agenten. Da aber neu angelegte Agenten explizit aktiviert werden müssen, um im Workflow System Einfluß nehmen — sprich Aktivitäten bearbeiten — zu können, genügt es, die Methode `activate` der Klasse `Agent` durch eine Regel überwachen zu lassen. Die Methode `new` deaktiviert einen Agenten beim Anlegen und braucht somit nicht beachtet werden.

```
DEFINE RULE R_RestartWFOnAgentActivation
ON POST (Agent, activate) DO
ON POST (rule_trgO, activate)
   IF (at least one workflow is paused and its
       company is the same as that one of the
       activated agent (rule_trgO)) THEN
ON POST (rule_trgO, activate)
```

```
EXECUTE WFWorkflowBase restartWorkflows
```

Voraussetzung ist natürlich, daß solche gestoppte Aktenläufe überhaupt existieren. Nur dann wird nach diesen gesucht und deren Fortsetzung versucht. Ausführen muß den Aktionsteil jene Instanz, die alle Workflow Typen speichert, nämlich `WFWorkflowBase`, mittels der Instanzmethode `restartWorkflows`.

4.4.4.3 Nachfolgerbeziehungen und Ausführungsbedingungen von Aktivitäten

Ein typisches Beispiel für den Einsatz von ECA-Regeln in Workflow Systemen ist die Realisierung der Vorgänger-Nachfolger-Beziehung. Zudem kann die Gewichtung der Kanten durch Bedingungen in den Bedingungsteil der Regeln übernommen werden. Und so wird es auch in TriGS$_{flow}$ gehandhabt, mit der Einschränkung, daß als Bedingung die Ausführungsbedingung der Nachfolgeaktivität verwendet wird. Der Grund dafür wurde schon einmal erläutert.

Zur Wiederholung: Relevant ist dies nur bei Zusammenführungen. Denn hier können keine verschiedenen Bedingungen angegeben werden, sondern für alle Zusammenführungen nur die Ausführungsbedingung der gemeinsamen Nachfolgeaktivität. Bei verschiedenen Bedingungen könnte es passieren, daß ein Zweig nicht in die Zusammenführung einfließt. Hierauf müssen die Daten, die durch den zu verwerfenden Pfad geändert wurden, wieder auf den Stand vor Beginn des Pfades rückgeführt werden. Dies ist aber nur mittels eines geeigneten Transaktionskonzepts („Geschachtelte Transaktionen") möglich, das aber in GemStone nicht vorliegt. Die Gewichtung bei Zusammenführungen wird auf eine Bedingung — die der Nachfolgeaktivität — beschränkt, damit dieser Fall nicht eintreten kann.

Da die Vorgänger-Nachfolger-Beziehung vom jeweiligen Worflow Typ und den beiden betroffenen Aktivitäten abhängt und somit nicht für alle gültig ist, werden diese Regeln im Gegensatz zu den „statischen" von vorhin als „dynamisch" bezeichnet, gekennzeichnet durch ein Suffix (z.B: _STIP_001_002). Erzeugt werden die Regeln während der Definition der Nachfolgerbeziehung mittels der Methode `becomeSuccessorOf:`, die der Nachfolgeaktivität mit dem Vorgänger als Parameter geschickt wird. Da nur innere Aktivitäten Vorgänger haben können, ist diese Methode auch nur bei diesen definiert.

Jede Regel erhält zur Identifikation ein Namenssuffix, das aus der Nummer des Workflow Typs, der Nummer der Vorgänger- und der Nummer der Nachfolgeaktivität besteht. Die Regel ist nach folgendem Schema aufgebaut:

```
DEFINE RULE R_ActivitySuccession_<workflowtype id>_<predact id>
                                                   _<succact id>
ON PRE (Activity, distribute: aFolder) DO
ON PRE (rule_trgO, distribute: aFolder)
   IF (<succact execcond>) THEN
ON PRE (rule_trgO, distribute: aFolder)
   EXECUTE <succact> combine: aFolder fromActivity: <predact>
```

Die in Klammern eingefaßten Platzhalter (<...>) werden bei der Erzeugung durch die aktuellen Werte ersetzt:

`<workflowtype id>`	... Nummer des Workflow Typs
`<predact id>`	... Nummer der Vorgängeraktivität
`<succact>`	... Nachfolgeaktivität
`<succact id>`	... Nummer der Nachfolgeaktivität
`<succact execcond>`	... Ausführungsbedingung der Nachfolgeaktivität

Diese Regeln beobachten die Methode `distribute:` einer Aktivität. Wird diese Methode aufgerufen, so werden alle Regeln gefeuert, deren Vorgänger die betreffende Aktivität ist und deren Bedingungen wahr sind. Damit werden auch die Sonderfälle behandelt: Sind gar keine Regeln definiert, so wird in der Methode `distribute:` die Endeaktivität — falls vorhanden — ausgeführt. Gibt es zwar Regeln mit Nachfolgern, konnte aber keine Bedingung erfüllt werden, so liegt ein Ausnahmefall vor, und die Ausnahmeaktivität wird — wiederum falls definiert — ausgeführt. Das Beispiel 4.29 zeigt eine mögliche Ausprägung des Regelschemas.

Beispiel 4.29 *Die Aktivität mit der Nummer* `003` *ist als Nachfolger der Aktivität* `002` *definiert. Eine Nachfolge tritt nur dann ein, wenn die Ausführungsbedingung der Aktivität* `003` *(*`actFolder creditAmount > 500000`*) erfüllt ist.*

```
DEFINE RULE R_ActivitySuccession_STIP_002_003
ON PRE (Activity, distribute: aFolder) DO
ON PRE (rule_trg0, distribute: aFolder)
   IF (actFolder creditAmount > 500000) THEN
ON PRE (rule_trg0, distribute: aFolder)
   EXECUTE Activity-003 combine: aFolder fromActivity: Act.-002
```

4.4.4.4 Organisationspolitiken: Agentenauswahl und -zuordnung

Bevor ein Agent über die Ausführung einer Aufgabe benachrichtigt werden kann, muß dieser durch die Auswertung der Organisationspolitik der Aktivität zuerst ermittelt werden. Die Auswertung umfaßt bekanntlich zwei Schritte: *Agentenauswahl (Agent Selection)* und *Agentenzuordnung (Agent Binding)*. Da eine Organisationspolitik immer für eine Aktivität angegeben wird, sind die hier definierten Regeln auch *dynamisch*.

Die Organisationspolitik wird als Regel abgebildet. Da es mehrere Alternativen im Zusammenhang mit kontextabhängigen Politiken geben kann, können zu einer Aktivität mehrere „Politikregeln" existieren, für jede Alternative eine Regel. Die Kontextbedingung wird in den Bedingungsteil der Regel eingebracht. Das Schema einer solchen Regel hat folgendes Aussehen:

```
DEFINE RULE R_OrgPolicy_<workflowtype id>_<act id>_<orgpol nr>
ON PRE (Activity, notifyAgent: aFolder) DO
```

```
ON PRE (rule_trg0, notifyAgent: aFolder)
   IF (<orgpol contextcond>) THEN
ON PRE (rule_trg0, notifyAgent: aFolder)
   EXECUTE <act> selectAgents_<workflowtype id>_<act id>
                                              _<orgpol nr>
```

Die in Klammern eingefaßten Platzhalter (<...>) werden bei der Erzeugung wieder durch die aktuellen Werte ersetzt:

`<workflowtype id>`	... Nummer des Workflow Typs
`<act>`	... Aktivität
`<act id>`	... Nummer der Aktivität
`<orgpol nr>`	... Nummer der Organisationspolitikalternative
`<orgpol contextcond>`	... Kontextbedingung der Organisationspolitikalternative

Jede Alternative wird mit einer Nummer versehen, damit die Identifikation eindeutig ist. Das Ereignis stellt die Methode `notifyAgent:` dar. Noch vor deren Ausführung wird die Kontextbedingung ausgewertet und die Selektion durchgeführt. Die Selektion wird von einer Methode ausgeführt, die — wie die Regeln selbst — bei Definition der Politik mit der Methode `orgPol:cond:sel:` automatisch angelegt wird. Die Selektionsmethode wird mit dem gleichen Suffix wie das der Regel gekennzeichnet. Sie werden bei der Klasse der jeweiligen Aktivitätenart (Start-, Ende-, Ausnahme- oder Innere Aktivität) unter der Kategorie `AgentSelection` abgelegt.

Nachdem die Agenten ausgewählt und in ihren ausführbaren Rollen vorliegen, muß von diesen einer zugeordnet werden. Dies hat noch vor Ausführung der Methode `notifyAgent:` aber nach der Selektion zu geschehen. Da die Zuordnung für jede Aktivität individuell definiert werden kann (z.B. `#less` oder `#random`), muß auch hier für jede Aktivität eine Regel definiert werden. Es gibt zwar keine Alternativen wie bei der Selektion, aber dennoch gibt es eine Regel pro Aktivität.

```
DEFINE RULE R_OrgPolBinding_<workflowtype id>_<act id>
ON PRE (Activity, notifyAgent: aFolder) DO
ON PRE (rule_trg0, notifyAgent: aFolder)
   IF (true) THEN
ON PRE (rule_trg0, notifyAgent: aFolder)
   EXECUTE <act> <bindingmethod>
```

Die in Klammern eingefaßten Platzhalter (<...>) werden bei der Erzeugung wieder durch die aktuellen Werte ersetzt:

`<workflowtype id>`	... Nummer des Workflow Typs
`<act>`	... Aktivität
`<act id>`	... Nummer der Aktivität
`<bindingmethod>`	... Zuordnungsmethoden: `bindAgentWithLessWorklistSize`, `bindRandomAgent`

Je nach Art der Zuordnung wird die entsprechende Methode dafür in den Aktionsteil gespeichert. Die Zuordnungsmethoden sind bei der Klasse `Activity` in der Kategorie `AgentBinding` abgelegt.

Bevor die Agentenauswahl und -zuordnung funktioniert, müssen noch zwei Dinge erledigt werden:

1. Bei der Auswahl wird eine Menge an Rollenobjekten angelegt, die die Politik erfüllen. Damit die Menge vor der Auswertung der ersten Alternative leer ist, aber nicht vor jeder weiteren Alternative geleert wird, wird eine weitere Initialisierungsregel eingeführt, die vor der ersten Selektion abgearbeitet wird.

2. Die Reihenfolge der Ausführung der Regeln Initialisierung, Selektionen und Zuordnung muß über eine Prioritätensteuerung festgelegt werden.

Die Initialisierungsregel ist wieder statisch. Bei jeder Aktivität hat die Menge geleert zu werden. Die Regel hat folgenden Aufbau:

```
DEFINE RULE R_OrgPolicyInit
ON PRE (Activity, notifyAgent: aFolder) DO
ON PRE (rule_trgO, notifyAgent: aFolder)
   IF (true) THEN
ON PRE (rule_trgO, notifyAgent: aFolder)
   EXECUTE rule_trgO actAgentRoleSet: (Set new)
```

Damit die Regeln in der richtigen Reihenfolge ausgewertet und ausgeführt werden, werden sie mit Prioritäten versehen. Die Initialisierungsregel erhält die höchste Priorität 2, alle Selektionsregeln die nächstkleinere 1 und die Zuordnungsregel die Priorität 0. Damit ist gewährleistet, daß die Regeln in der richtigen Reihenfolge ausgeführt werden.

Nicht nur bei einer Aktivität wird eine Organisationspolitik definiert, sondern auch beim Workflow Typ. Dort dient sie zur Feststellung, ob ein Agent berechtigt ist, einen Aktenlauf zu starten. Da kein Agent zugeordnet werden muß, entfällt die Regel für die Zuordnung. Ansonsten gelten die gleichen Bestimmungen wie für die Politiken der Aktivitäten. Das heißt, es gibt eine Initialisierungsregel mit Priorität 2 und eine oder mehrere Regel(n) für die Selektionsalternativen mit Priorität 1. Als Ereignis dient die Methode `isAllowedToStartWithFolder:`. In der Folge werden nur noch die beiden Schemata für die Regeln angegeben.

```
DEFINE RULE R_StartWFOrgPolicy_<workflowtype id>_<orgpol nr>
ON PRE (WorkflowType, isAllowedToStartWithFolder: aFolder) DO
ON PRE (rule_trgO, isAllowedToStartWithFolder: aFolder)
   IF (<orgpol contextcondition>) THEN
ON PRE (rule_trgO, isAllowedToStartWithFolder: aFolder)
   EXECUTE <act> selectStartAgents_<workflowtype id>: actFolder
```

Die in Klammern eingefaßten Platzhalter (<...>) werden bei der Erzeugung wieder durch die aktuellen Werte ersetzt:

`<workflowtype id>`	... Nummer des Workflow Typs
`<orgpol nr>`	... Nummer der Organisationspolitikalternative

```
DEFINE RULE R_StartWFOrgPolicyInit
ON PRE (WorkflowType, isAllowedToStartWithFolder: aFolder) DO
ON PRE (rule_trgO, isAllowedToStartWithFolder: aFolder)
   IF (true) THEN
ON PRE (rule_trgO, isAllowedToStartWithFolder: aFolder)
   EXECUTE rule_trgO startAgentSet: (Set new)
```

4.4.4.5 Abarbeitung der Arbeitsliste bei Software Agenten

Ein Software Agent wählt seine Arbeitsschritte bekanntlich „automatisch" aus. Anstatt einen Prozeß zu starten, der die Arbeitsliste überwacht und die Ausführung anstößt, werden zwei neue Regeln eingeführt.

Nachdem ein Software Agent einen Eintrag aus der Arbeitsliste entfernt hat, und diese daraufhin nicht leer ist, wird der nächste Arbeitsschritt ausgeführt. Die Regel `R_StartOnRemove` wird nach dem Entfernen des Eintrags getriggert und abgearbeitet.

```
DEFINE RULE R_StartOnRemove
ON POST (SoftwareAgent, removeFromWorklist: anItem) DO
ON POST (rule_trgO, removeFromWorklist: anItem)
   IF (rule_trgO isEmpty not) THEN
ON POST (rule_trgO, removeFromWorklist: anItem)
   EXECUTE rule_trgO selectElem
```

Wird hingegen ein neuer Eintrag der Arbeitsliste hinzugefügt, wird dieser nach dem Hinzufügen ausgeführt, wenn die Liste zuvor leer war. Es ist zu beachten, daß die Bedingung bereits vor der triggernden Methode auszuwerten ist, die Aktion aber erst nachher ausgeführt wird. Nur diese Regel verwendet keinen „immediate"-Kopplungsmodus.

```
DEFINE RULE R_StartOnInsert
ON PRE (SoftwareAgent, addToWorklist: anActivity
                               folder: aFolder
                                 role: aRole) DO
ON PRE (rule_trgO, addToWorklist: anActivity
                           folder: aFolder
                             role: aRole)
   IF (rule_trgO isEmpty) THEN
ON POST (rule_trgO, addToWorklist: anActivity
                           folder: aFolder
                             role: aRole)
   EXECUTE rule_trgO selectElem
```

4.4.5 Probleme mit Transaktionsmanagement und Sperren

Zur Zeit erfolgt die Unterteilung eines Durchlaufs in drei Transaktionen. Alle in der jeweiligen Transaktion gelesenen oder geschriebenen Objekte werden bei Transaktionsbeginn entsprechend gesperrt (Schreib- oder Lesesperre) und am Ende wieder freigegeben. Damit wird eine Art des Zwei-Phasen-Sperrprotokolls realisiert, welches besagt, daß am Anfang alle Sperren angefordert und erst am Ende alle Sperren freigegeben werden.

4.4.5.1 Unterteilung in drei Transaktionen

Um ein erfolgreiches Wiederaufsetzen im Falle von abgebrochenen Workflows gewährleisten zu können, müßte ein anderes Transaktionskonzept als jenes in GemStone zur Verfügung stehen. In GemStone können Transaktionen nur auf einer Ebene ablaufen. In einer Sitzung können Transaktionen nur sequentiell auftreten. Das heißt, wenn eine Transaktion endet, so kann der Start einer zweiten Transaktion veranlaßt werden, sofern nicht automatisch durch das Ende einer Transaktion eine neue gestartet wird (Voreinstellung). Das Transaktionskonzept von GemStone wird in [Serv91] und [Butt91] ausführlich beschrieben. Um Transaktionen zu beginnen oder zu beenden, stellt die Klasse `System` folgende drei Methoden zur Verfügung:

`beginTransaction` Wird diese Methode während einer laufenden Transaktion ausgeführt, dann wird diese mit einem *commit* beendet und eine neue gestartet.

`commitTransaction` Beendet die Transaktion mit dem Speichern der Änderungen und beginnt eine neue, sofern der automatische Transaktionsmodus eingestellt ist.

`abortTransaction` Beendet eine Transaktion ohne Speicherung der Änderungen und beginnt eine neue, sofern der automatische Transaktionsmodus eingestellt ist.

TriGS baut auf das GemStone-Transaktionskonzept auf. Verschiedene Kopplungsmodi lassen es zu, daß eine Regel vor Ausführung der triggernden Methode gefeuert, der Aktionsteil der Regel aber erst nach der Abarbeitung des Methodenrumpfes ausgeführt wird. Wird aber eine Transaktion zwischen Regeltriggerung und Aktionsausführung beendet oder abgebrochen, so wird die nachfolgende Ausführung der Aktion nicht durchgeführt.

TriGS stellt die Klasse `MySystem` zur Verfügung, die ebenfalls die drei Methoden zum Beginnen, Beenden und Abbrechen von Transaktionen enthält. Da TriGS Ereignisse nur mit umhüllten Methoden erkennt, werden auch diese drei Methoden umhüllt. Die Klasse `System` kann nicht umhüllt werden, da eine

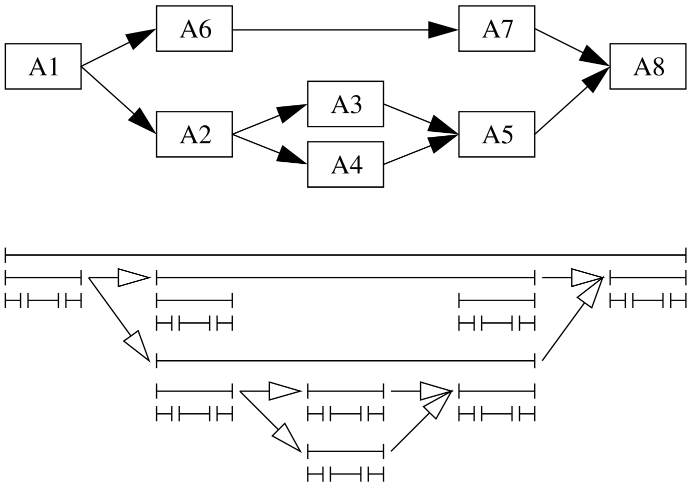


Abbildung 4.7: Transaktionsebenen im Workflow

Veränderung der GemStone-Basisklassen nicht erlaubt ist, weshalb die Klasse `MySystem` eingeführt wurde. Transaktionen sollen bei der Verwendung von TriGS nur über die Klasse `MySystem` bedient werden.

Ein Workflow System verlangt eigentlich nach einem erweiterten Transaktionskonzept. Jede Aktivität sollte in einer eigenen Transaktion ablaufen. Die zu einem Aktivitätennetz zusammengefaßten Aktivitäten bilden wieder eine eigene Einheit, die auch in einer Transaktion ablaufen soll. Um eine ODER-Zusammenführung realisieren zu können, muß jeder Teilgraph in einer Transaktion ablaufen. Diese Transaktion beginnt bei der Verzweigung und endet bei der Zusammenführung. Soll ein Zweig nicht übernommen werden, so endet die Transaktion mit einem *abort* ansonsten mit einem *commit*. Der gesamte Ablauf eines Workflows läuft also auf verschiedenen Transaktionsebenen ab, zu deren Ausführung *geschachtelte Transaktionen* („Nested Transactions") [Bern87, Voss91] notwendig sind. Abbildung 4.7 veranschaulicht diesen Sachverhalt. Der obere Teil der Abbildung stellt ein Aktivitätennetz mit mehreren Verzweigungen dar. Im unteren Teil werden die dazu passenden Transaktionsebenen angeführt, wobei das gesamte Netz, jeder Teilgraph und auch jede Aktivität in einer Transaktion abläuft. Die Ausführung einer Aktivität wird in drei Transaktionen zerlegt. Die „leeren" Pfeile bedeuten, daß die beiden betroffenen Transaktionen in der gleichen Ebene (parallel) ablaufen.

Tritt in einer Ebene ein unvorhergesehener Transaktionsabbruch auf, so besteht die Möglichkeit, diese Transaktion erneut zu starten. Erfolgt erneut ein Abbruch der Transaktion, so wird die nächsthöhere Transaktion in der Transaktionshierarchie abgebrochen und erneut gestartet und so fort. Ein Abbruch einer höheren Transaktion veranlaßt den Abbruch aller darunterliegenden Transaktionen.

Da GemStone die Schachtelung von Transaktionen nicht unterstützt, ist eine korrekte Abarbeitung der Abläufe insbesondere bei Fehlern nicht immer möglich. Die Ausführung einer Aktivität wird in drei Transaktionen zerlegt.

Jeweils eine Transaktion für die Datenbereitstellung, eine für die Aktionsausführung und Datenübernahme und eine für die Arbeitsverteilung. Dieser Ablauf kann nicht in einer Transaktion stattfinden, da z.B. auch die Ergebnisse der Aktionsausführung verworfen werden, wenn in der Arbeitsverteilung ein Fehler auftritt. Die Unterteilung in drei Transaktionen beseitigt aber auch nicht alle Probleme. Tritt nach der Aktionsausführung ein Fehler auf, so wird die Arbeitsverteilung zurückgesetzt, die Aktionsausführung bleibt bestehen und der Eintrag wird nicht aus der Arbeitsliste des Agenten gelöscht. Bei einem neuerlichen Start der Aufgabe werden bereits veränderte Daten wieder verwendet.

Neben geschachtelten Transaktionen gibt es zahlreiche andere Transaktionsmechanismen, die für die Verwendung in Workflow Systemen geeignet sind. Leider existieren sie meist nur als Konzept und sind daher in der Praxis nicht verfügbar. Der Artikel [Geor93] befaßt sich zum Beispiel mit einem erweiterten Transaktionskonzept für Workflow Systeme im Kontext verteilter Systeme. Weitere Ansätze sind in [Elma91] und [Elma92] zu finden.

4.4.5.2 Behandlung von Fehlern

Bei der Ausführung können prinzipiell zwei Arten von Fehlern auftreten. Die einen sind durch die Anwendung bedingt, wie z.B. das Starten eines Workflows mit einem Akt, unter dessen Nummer schon einer eingetragen ist. Diese Fehler führen korrekterweise zu einem Transaktionsabbruch und zu einer Rückkehr aus der Bearbeitung. Die Ergebnisse werden in diesem Fall nicht in den Akt übernommen.

Die zweite Fehlerart ist ganz anderer Natur. Fehler können nämlich auch auftreten, wenn ein Objekt nicht gesperrt werden kann. Während der Abarbeitung einer Aufgabe werden viele Objekte gesperrt, unter anderem der Agent, die Aktivität, der Akt und schließlich auch die Nachfolgeaktivitäten und die zu benachrichtigenden Agenten. Obwohl außer dem Akt die Objekte nur für wenige Sekunden gesperrt sind, können Konflikte mit Prozessen anderer Workflow Teilnehmer auftreten. Kommt es zu so einem Fehler, wird zwar versucht, diese Sperre einige Sekunden später nocheinmal anzufordern, gelingt es abermals nicht, erfolgt ein Transaktionsabbruch und die Rückkehr aus der Abarbeitung. Wie oben schon erwähnt, ist es nicht sinnvoll, aufgrund solcher meist kurzzeitiger Verzögerungen, auch die Aktion zurückzusetzen, weshalb diese mit der Datenübernahme in einer eigenen Transaktion abläuft und dadurch bereits gespeichert ist. Das Problem ist, wo jetzt wieder aufgesetzt werden soll. Es müßte dazu nämlich vermerkt werden, daß die Aktion bereits erfolgreich ausgeführt worden ist und nur die Arbeitsverteilung Probleme verursacht hat. Also könnte von Hand oder durch die Steuerung von selbständigen Prozessen die Arbeitsverteilung einige Zeit später nachgeholt werden.

Aus diesen Problemen läßt sich ersehen, daß ohne eine Erweiterung des Transaktionskonzepts von GemStone in Richtung „geschachtelte Transaktionen“ eine ausreichende Fehlerbehandlung nicht möglich ist.

4.4.5.3 Sperren des Aktes während der Aktionsausführung

Der Akt muß während der Ausführung einer Aktivität gesperrt werden. Wie schon im theoretischen Teil der Arbeit beschrieben, würde es ausreichen, bloß jene Teile des Aktes zu sperren, die in der Aktivität verwendet werden. Eine Lesesperre auf die Input-Instanzvariablen und eine exklusive Sperre auf die Output-Instanzvariablen erfüllen diesen Zweck. Dadurch können parallele Aktivitäten (auf verschiedenen Pfaden im Aktivitätennetz befindlich) tatsächlich gleichzeitig ausgeführt werden. Die Konsistenz der Daten wird durch diese Sperren gesichert. Außerdem sollten bereits bei der Netzdefinition durch ein geeignetes Werkzeug mögliche derartige Konflikte erkannt und nicht zugelassen werden.

In diesem Prototyp gibt es jedoch wieder ein Problem. In GemStone kann ein Objekt nur zur Gänze gesperrt werden und nicht einzelne Teile (Instanzvariablen)[1] davon. Während der Ausführung muß deshalb der gesamte Akt gesperrt werden, womit eine parallele Ausführung zweier Aktivitäten verhindert wird. Fazit: Vielleicht bietet GemStone in einer neueren Version die Möglichkeit der Referenzsperre, oder man entwickelt selbst einen deratigen Mechanismus.

Eine andere Möglichkeit, um eine parallele Ausführung zu ermöglichen, besteht darin, daß für jeden Teilgraph eine Kopie des Aktes vergeben wird. Die bei der Verzweigung erzeugten Kopien sollen bei der Zusammenführung wieder vereint werden. Es gibt zwei Arten, ein Objekt zu kopieren. Eine *shallow copy* eines Objekts kopiert nur auf der obersten Ebene, das heißt, es wird zwar ein neues Objekt angelegt, die Referenzen auf andere Objekte bleiben jedoch dieselben. Damit ist das Problem der Zugriffe auf ein und denselben Akt nicht gelöst. Eine *deep copy* kopiert hingegen alle Objekte bis in die unterste Ebene. Es werden also alle referenzierten Objekte neu angelegt. Eine solche Kopie anzulegen ist eine sehr zeit- und speicherplatzraubendes Unterfangen, weshalb GemStone diese Art der Kopie nicht zur Verfügung stellt.

Obwohl eine „tiefe Kopie“ relativ einfach zu implementieren ist, hat es aus folgenden Überlegungen wenig Sinn, diese einzusetzen:

1. Der Kopiervorgang kostet sehr viel Zeit und Speicherplatz.

2. Die Zusammenführung („merge“) der Daten am Ende der Teilbäume kostet wieder Zeit. Wenn nicht festgelegt ist, welche Daten von den parallelen Aktivitäten geändert werden dürfen, so können beim Zusammenspielen der Kopien einzelne Daten überschrieben werden. Wird darauf geachtet, daß sich die Schreibinstanzvariablen „wechselseitig ausschließen“, so werden die Kopien überflüssig, weil es dann ausreicht, einzelne Instanzvariablen (Referenzen) eines Aktes zu sperren, sofern dies möglich ist.

3. Zusätzlich stellt sich die Frage, ob Aktivitäten mit Zugriff auf dieselben Daten überhaupt in parallelen Zweigen auftreten dürfen. In einem sol-

[1] Instanzvariablen sind i.a. Referenzen auf andere Objekte. Bei einer Sperre einer Referenz soll nur der Zugriff auf das Objekt, nicht das referenzierte Objekt selbst gesperrt werden.

chen Fall liegt möglicherweise ein Designfehler der Aktivitätenmodellierung vor.

Daraus läßt sich der Schluß ziehen, daß die Sperre einzelner Referenzen die einzig vernünftige Lösung darstellt. Eine Referenzsperre könnte auch selbst implementiert werden, indem jede Referenz eines Aktes um ein „Sperrflag“ ergänzt wird, anhand dessen Sperren gesetzt und gelöscht werden.

Kapitel 5

Modellierung und Ablauf in TriGS$_{flow}$

Anhand der in den vorangegangenen Kapiteln erworbenen Kenntnisse über den Prototyp wird in der Folge ein konkretes Beispiel auf das Workflow System abgebildet. Zunächst folgt noch einmal eine kurze Beschreibung des Anwendungsbeispiels aus der Einleitung. Anschließend wird es vollständig modelliert, anhand der OPAL-Anweisungen in das Datenbank- bzw. Workflow System übertragen, und es werden jene Anweisungen vorgestellt, die zum Start und zur Ausführung eines Workflows notwendig sind. Am Ende des Kapitels folgen noch einige Bemerkungen zu den Problemen bei der Modellierung und Ablauf.

5.1 Kurzbeschreibung des Anwendungsbeispiels

Produktionsbetriebe sind damit konfrontiert, auf Anfragen potentieller Kunden über eines ihrer Produkte möglichst schnell ein Angebot zu erstellen. In vielen Fällen werden die damit zusammenhängenden Kalkulationen noch von Hand durchgeführt oder das Angebot noch von Hand mit einigen wenigen vorgefertigten Textbausteinen erstellt. In [Sche94] wird der Vorgang der Angebotsbearbeitung als Vorgangskette dargestellt und festgelegt, wer welchen Schritt zu bearbeiten hat. Das Beispiel gliedert sich dabei grob in folgende Schritte:

1. Wenn eine Kundenanfrage eintrifft, wird ein Formular angelegt und die Kundendaten eingegeben. Ist der Kunde nicht gespeichert, werden seine Daten entsprechend aufgenommen.

2. Die Anfrage bezieht sich auf einen oder auf mehrere Artikel in der gewünschten Anzahl. Die Artikelnummer und die Anzahl werden in den Antrag aufgenommen.

3. Die Konstruktionsabteilung ist dafür verantwortlich, die benötigten Kalkulationsdaten wie Teile (Mengen und Kosten), Herstellungszeiten und

Kostensätze der Betriebsmittel zur Verfügung zu stellen.

4. Nachdem die Daten ermittelt worden sind, kann die Kalkulation automatisch erfolgen. Das Ergebnis wird wieder in den bereits angelegten Akt gespeichert.

5. War die Kalkulation erfolgreich, so wird jetzt ein Angebot erstellt bzw. ausgedruckt. Kann aufgrund des Kalkulationsergebnisses keine Produktion oder Auslieferung erfolgen, so wird die Angebotsbearbeitung abgebrochen und für nicht erfüllbar erklärt. Sind hingegen die Daten nicht ausreichend spezifiziert, so wird die Bearbeitung bei der Ermittlung der Kalkulationsdaten wieder fortgeführt.

Im nächsten Abschnitt wird versucht, dieses Anwendungsbeispiel schrittweise zu modellieren, und die Ergebnisse im Workflow System abzulegen. Die Abbildung erfolgt mit OPAL-Anweisungen in einem GemStone Workspace. Grundkenntnisse in der Bedienung von GemStone werden vorausgesezt.

5.2 Modellierung

Die Vorgehensweise gleicht der im vorhergehenden Kapitel. Aus praktischen Gründen wird hier auch die Integration (Datenflußbeschreibung und Organisationspolitiken) vor der Aktivitätenmodellierung behandelt. Die Reihenfolge der Definition der Objekte ist wie folgt:

(1) Betrieb

(2) Gruppentyphierarchie und Abteilungsstruktur

(3) Rollentyphierarchien

(4) Aktionen der Rollentypen (Funktionalität)

(5) Agenten und deren Rollen

(6) Aktivitäten

(7) Datenstruktur (Akttyp)

(8) Datenfluß

(9) Organisationspolitiken

(10) Workflow Typ

(11) Integration von Datenfluß und Organisationspolitik

(12) Ausführungsreihenfolge

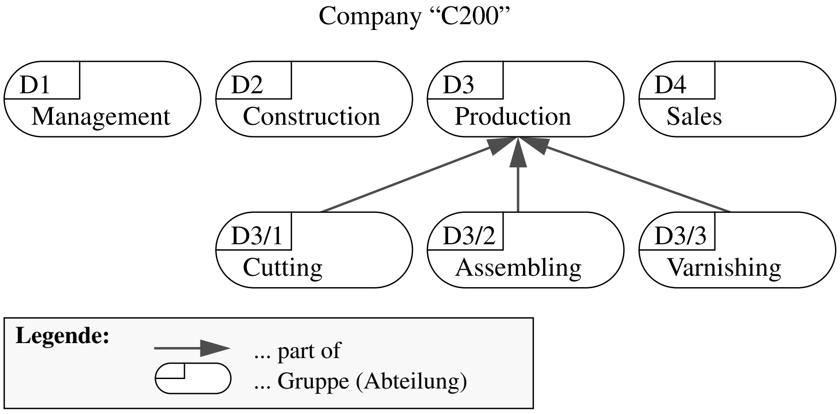


Abbildung 5.1: Abteilungsstruktur

Die Reihenfolge dieser Schritte ist keinesfalls als obligatorisch anzusehen. Der Datenfluß und die Organisationspolitiken werden begleitend zu den vorhergehenden Schritten entwickelt, obwohl deren endgültige Bestimmung erst kurz vor der bzw. bei der Aktivitätendefinition erfolgt. Bei der Aktivitätendefinition müssen aber in vielen Fällen noch Änderungen an den Schnittstellen und Politiken vorgenommen werden. Die letzten Ungereimtheiten scheinen erst bei der Ausführung auf und werden durch nachträgliche Änderungen beseitigt. In der folgenden Modellierung werden die einzelnen Schritte am Rand durch die Nummern der obigen Aufzählung gekennzeichnet.

(1) Die Modellierung des konkreten Beispiels beginnt mit dem *Betriebsobjekt*. Der Betrieb selbst ist eine Großtischlerei, die Möbel im Rahmen von Großaufträgen herstellt. wobei als Daten nur Name und Adresse neben der Identifikationsnummer von Bedeutung sind. Im Betriebsobjekt werden später die Abteilungsstruktur und die Agenten zusammengefaßt. Der Betrieb wird in der `WFCompanyBase` unter seiner Nummer abgelegt.

```
" *** create company ***  "

| comp |

comp := Company new.
comp id: 'C200';
     name: 'Berenson Ltd.';
     address: 'Morganroad 387, 21229 SomeWhere'.
WFCompanyBase at: (comp id) put: comp.
```

(2) Im nächsten Schritt wird der Aufbau der Organisation festgelegt. Die *Abteilungsstruktur* ist in Abbildung 5.1 dargestellt. Die vier Hauptabteilungen der obersten Ebene sind `Management`, `Construction`, `Production` und die Vertriebsabteilung `Sales`. Die Produktionsabteilung besteht aus drei Unterabteilungen, dem Zuschnitt (`Cutting`), dem Zusammenbau (`Assembling`) und der Lackiererei (`Varnishing`).

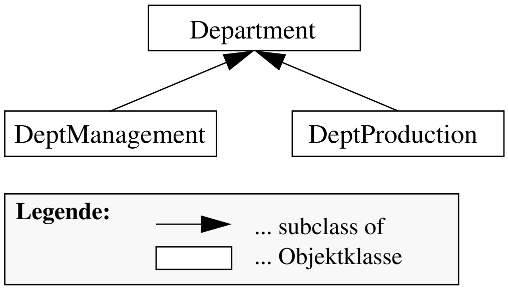


Abbildung 5.2: Gruppentyphierarchie

Die folgenden OPAL-Anweisungen veranschaulichen die Definition der Abteilungsstruktur. Dabei ist zu beachten, daß alle Abteilungen, mit Ausnahme der Firmenleitung und der Produktion, Instanzen der Klasse `Department` sind. Diese beiden Abteilungen enthalten weitere Zusatzattribute, die hier nicht verwendet werden, aber etwa für ein anderes betriebsinternes Informationssystem von Bedeutung sein können. Die Klassen `DeptManagement` und `DeptProduction` wurden dazu im GemStone System Browser als Subklassen von `Department` angelegt, und bilden damit die *Gruppentyphierarchie*, die in Abbildung 5.2 dargestellt wird. Am Ende der Definition der Abteilungen werden diese dem Betriebsobjekt hinzugefügt.

```
" *** define department structure ***  "

| dept1 dept2 dept3 dept31 dept32 dept33 dept4 |

dept1 :=  (DeptManagement new)
           id: 'D1'; name: 'Department Management'.

dept2 :=  (Department new)
           id: 'D2'; name: 'Department Construction'.

dept31 := (Department new)
           id: 'D3/1'; name: 'Department Cutting'.

dept32 := (Department new)
           id: 'D3/2'; name: 'Department Assembling'.

dept33 := (Department new)
           id: 'D3/3'; name: 'Department Varnishing'.

dept3 :=  (DeptProduction new)
   id: 'D3'; name: 'Department Production';
           addSubDept: dept31;
           addSubDept: dept32;
           addSubDept: dept33.

dept4 :=  (Department new)
           id: 'D4'; name: 'Department Sales'.

(WFCompanyBase at: 'C200')
   removeAllDepts;
   addDept: dept1 withId: (dept1 id);
```

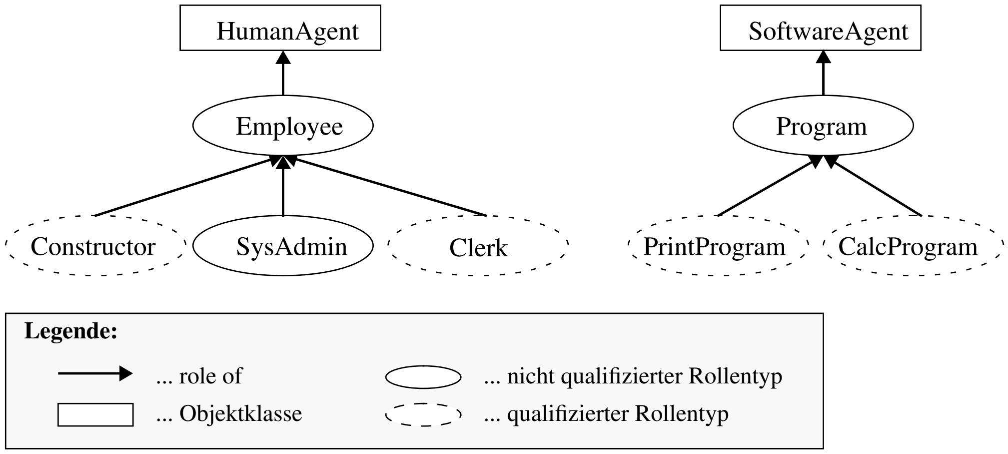


Abbildung 5.3: Rollentyphierarchien für menschliche Agenten und Software Agenten

```
addDept: dept2 withId: (dept2 id);
addDept: dept3 withId: (dept3 id);
addDept: dept4 withId: (dept4 id).
```

Die Methode `removeAllDepts` kann verwendet werden, wenn die Abteilungsstruktur völlig neu definiert wird, um nicht alle Abteilungen einzeln entfernen zu müssen.

(3) Von den im ganzen Workflow System notwendigen *Rollentypen* werden nur jene angeführt, die für dieses Beispiel relevant sind. Prinzipiell teilen sich die Rollentypen wieder in zwei Rollentyphierarchien, wobei eine für menschliche Agenten und die andere für Software Agenten vorgesehen ist. Die Klassen `HumanAgent` und `SoftwareAgent` bilden daher die Wurzeln der beiden Rollentyphierarchien. Die Abbildung 5.3 zeigt beide Rollentyphierarchien. In diesem Beispiel steht der Rollentyp `Employee` „ganz oben" in der Rollentyphierarchie der menschlichen Agenten, da diese Rolle jeder Beschäftigte annehmen muß. Davon werden alle anderen Rollentypen abgeleitet. Bei den Software Agenten wird als allgemeinster Rollentyp `Program` gewählt, der die gemeinsame Funktionalität der beiden Subrollentypen `PrintProgram` (ein Programm zum Drucken) und `CalcProgram` (ein Programm zum Kalkulieren der Angebotsdaten) realisiert.

```
" *** role type hierarchy for human agents *** "

Role
   newRole: 'Employee'
   roleSuperType: HumanAgent.

Role
   newRole: 'SysAdmin'
   roleSuperType: Employee.

QualifiedRole
   newRole: 'Clerk'
```

```
      roleSuperType: Employee
      classOfQualifyingObj: Department.

   QualifiedRole
      newRole: 'Constructor'
      roleSuperType: Employee
      classOfQualifyingObj: Department.

   " *** role type hierarchy for software agents *** "

   Role
      newRole: 'Program'
      roleSuperType: SoftwareAgent.

   QualifiedRole
      newRole: 'CalcProgram'
      roleSuperType: Program
      classOfQualifyingObj: Department.

   QualifiedRole
      newRole: 'PrintProgram'
      roleSuperType: Program
      classOfQualifyingObj: Department.
```

(4) Als nächstes muß definiert werden, mit welchen *Aktionen* die Rollentypen ausgestattet, und wie die Aktionen implementiert werden. Externe Programme können zur Zeit in TriGS$_{flow}$ nicht aufgerufen werden. Diese Aufrufe werden simuliert, indem vorgefertigte Daten zurückgeliefert werden. Es ist noch in Erinnerung zu rufen, daß das erste Datenobjekt des Rückgabefeldes (hier: `#ok`) dem Workflow System anzeigt, ob die Ausführung erfolgreich war oder nicht. In allen folgenden Fällen wird immer angenommen, daß die Verarbeitung mit Erfolg abgeschlossen werden konnte.

Jeder Beschäftigte soll das Ende eines Workflows bestätigen können, weshalb dem Rollentyp `Employee` die Methode `confirmEnd` zur Verfügung gestellt wird. Der Workflow System Administrator kann Ausnahmen behandeln. Der gleichnamige Rollentyp `SysAdmin` erhält daher die Methode `confirmException`. Beide Aktionen haben keinen eigentlichen Rumpf und dienen deshalb nur zur Information über Ende oder Ausnahme, indem — wie später festgelegt — der Antrageinbringer vom Ende, und der Systemadministrator von einer Ausnahme informiert wird. Informiert werden sie über die ihnen in die Arbeitsliste gestellte Ende- oder Ausnahmeaktivität.

```
   Employee
      compileMethod: 'confirmEnd

         " action for executing the end activity  "

         ^(Array new: 1)
            at: 1 put: #ok.'
      dictionaries: (System myUserProfile symbolList)
      category: #Actions.
```

```
SysAdmin
   compileMethod: 'confirmException

      " action for executing the exception activity "

      ^(Array new: 1)
         at: 1 put: #ok.'
   dictionaries: (System myUserProfile symbolList)
   category: #Actions.
```

Ein Sachbearbeiter (`Clerk`) sammelt alle kundenspezifischen Daten und nimmt die Nummer und Anzahl des Artikels auf, von dem ein Angebot gemacht werden soll. Der Sachbearbeiter erhält somit eine Methode zur Vervollständigung der Angebotsdaten.

```
Clerk
   compileMethod: 'completeCustomerData: custId

      " add following data: name, address, partno, count "

      ^(Array new)
         at: 1 put: #ok;
         at: 2 put: #Buxbaum_Karl;
         at: 3 put: #Karlsplatz_67/2b_1090_Wien;
         at: 4 put: #a00125;
         at: 5 put: 3.'
   dictionaries: (System myUserProfile symbolList)
   category: #Actions.
```

Wie bereits erwähnt, ist die Konstruktionsabteilung für die Ermittlung aller zur Kalkulation benötigten Daten verantwortlich. Deshalb wird jeder Konstrukteur mit dieser Aufgabe anhand der folgenden Aktion betraut:

```
Constructor
   compileMethod: 'gatherCalcData: partno

      " gather all calculation relevant data "

      | parts times costs |

      " each article consists of these 3 components "
      parts := (Array new) add: #no1; add: #no2; add: #no3.
      times := (Array new) add: 4; add: 2; add: 7.
      costs := (Array new) add: 35; add: 48; add: 51.

      ^(Array new: 4 )
         at: 1 put: #ok;
         at: 2 put: parts;
         at: 3 put: times;
         at: 4 put: costs.'
   dictionaries: (System myUserProfile symbolList)
   category: #Actions.
```

Die darauf folgende Kalkulation und der Ausdruck der Ergebnisse wird von zwei Programmen übernommen. Ersteres (`CalcProgram`) erledigt die automatische Kalkulation, zweiteres (`PrintProgram`) stellt die drei Methoden für den Druck zur Verfügung. Die Methode zum Drucken des Angebotes wird nur in ganz vereinfachter Form implementiert. Der auszudruckende Text wird zwar erezugt, aber an keinen Drucker geschickt.

```
CalcProgram
   compileMethod: 'calculatePart: partno count: count
                                         times: times
                                         costs: costs
      " calculate the offer "

      | amount i |

      i:=1. amount := 0.
      [i<=(times size)] whileTrue:
         [ amount := amount + (times at: i)*(costs at: i).
           i:=i+1 ].
      amount := count * amount.

      ^(Array new: 2 )
         at: 1 put: #ok;
         at: 2 put: amount;
         at: 3 put: #ok.'
      dictionaries: (System myUserProfile symbolList)
      category: #Actions.

PrintProgram
   compileMethod: 'printOffer: id name: name address: address
                    partno: partno count: count amount: amount
      " print offer "

      | str |
      str := String new.
      str add: #custid + (Character tab) + id + (Character lf).
      str add: name + (Character lf) + address + (Character lf).
      str add: #partno + (Character tab) + partno + (Character lf).
      str add: #count + (Character tab) +
              (count asString) + (Character lf).
      str add: #amount + (Character tab) + (amount asString).

      " send str to printer! "

      ^(Array new: 1 )
         at: 1 put: #ok.'
   dictionaries: (System myUserProfile symbolList)
   category: #Actions;
```

Die Aktionen zum Ausdrucken eines nicht zustande gekommenen Angebotes (`rejection`) und zum Druck unzureichend spezifizierter Daten (`insufficiency`) werden analog zum Angebotsdruck mit leicht verändertem Text definiert.

```
PrintProgram
```

```
compileMethod: 'printRejection: custId
                          name: name
                       address: address
                        partno: partno
...
compileMethod: 'printInsufficiency: custId
                              name: name
                           address: address
                            partno: partno
...
```

(5) Wenn der Betrieb, seine Abteilungsstruktur und die Rollentyphierarchien definiert sind, dann können auch die *Agenten* und deren *Rollen* angelegt werden. Im folgenden Beispiel werden sechs Agenten samt Rollen definiert. Am Ende werden sie dem Betrieb hinzugefügt und aktiviert, damit sie einen Arbeitsschritt in Empfang nehmen und ausführen können. Die Aktivierung beansprucht etwas Zeit, da alle definierten Workflow Typen des gleichen Betriebs nach gestoppten Workflows durchsucht werden, denn es kann sein, daß durch die neue Belegung ein Workflow wieder fortgesetzt werden kann.

```
" *** define agents and let them play their roles *** "

| agent1 agent2 agent3 agent4 agent5 agent6
  dept1 dept2 dept3 dept31 dept32 dept33 dept4
  comp |

comp   := WFCompanyBase at: 'C200'.

dept1  := comp allDepartments at: 'D1'.
dept2  := comp allDepartments at: 'D2'.
dept3  := comp allDepartments at: 'D3'.
dept31 := comp allDepartments at: 'D3/1'.
dept32 := comp allDepartments at: 'D3/2'.
dept33 := comp allDepartments at: 'D3/3'.
dept4  := comp allDepartments at: 'D4'.

agent1 := (HumanAgent new)
           id: 'h001'; name: 'Langthaler Johann'.
Employee newRoleOf: agent1.
SysAdmin newRoleOf: (agent1 as: Employee).

agent2 := (HumanAgent new)
           id: 'h002'; name: 'Brandstaetter Martin'.
Employee newRoleOf: agent2.
Constructor newRoleOf: (agent2 as: Employee) qualifiedBy: dept2.

agent3 := (HumanAgent new)
           id: 'h003'; name: 'Mayr Heidelinde'.
Employee newRoleOf: agent3.
Clerk newRoleOf: (agent3 as: Employee) qualifiedBy: dept32.
Clerk newRoleOf: (agent3 as: Employee) qualifiedBy: dept33.

agent4 := (HumanAgent new)
           id: 'h004'; name: 'Proell Claudia'.
Employee newRoleOf: agent4.
```

```
Clerk newRoleOf: (agent4 as: Employee) qualifiedBy: dept4.

agent5 := (SoftwareAgent new)
           id: 'a002'; name: 'Offer Calculating Program'.
Program newRoleOf: agent5.
CalcProgram newRoleOf: (agent5 as: Program) qualifiedBy: dept2.

agent6 := (SoftwareAgent new)
           id: 'a001'; name: 'Offer Printing Program'.
Program newRoleOf: agent6.
PrintProgram newRoleOf: (agent6 as: Program) qualifiedBy: dept4.

comp
   removeAllAgents;
   addAgent: agent1 withId: (agent1 id);
   addAgent: agent2 withId: (agent2 id);
   addAgent: agent3 withId: (agent3 id);
   addAgent: agent4 withId: (agent4 id);
   addAgent: agent5 withId: (agent5 id);
   addAgent: agent6 withId: (agent6 id).

agent1 activate.
agent2 activate.
agent3 activate.
agent4 activate.
agent5 activate.
agent6 activate.
```

Die Methode `removeAllAgents` kann wieder verwendet werden, wenn bei einer späteren Neudefinition aller Agenten die alten auf einmal gelöscht werden sollen.

(6) Wenn die Organisationsstruktur aufgebaut ist, kann mit der Modellierung der *Aktivitäten* begonnen werden. Die Abbildung 5.4 stellt den im vorigen Abschnitt kurz beschriebenen Ablauf der Angebotsbearbeitung grafisch dar.

(7) In Kapitel 4 wurde bereits die Modellierungsreihenfolge Organisation – Aktivitäten – Daten – Integration verändert. Auch hier soll die Datenmodellierung noch vor der tatsächlichen Abbildung der Aktivitäten, des Aktivitätennetzes und des Workflow Typs erfolgen, da dort bereits die Rollen, Gruppen und die Datenstruktur verwendet werden. Die *Datenstruktur* `OfferFolder` ist eine Subklasse von `Folder` und beinhaltet alle Daten, die für die Angebotsbearbeitung notwendig sind.

```
Folder subclass: 'OfferFolder'
  instVarNames: #( 'custId' 'name' 'address'
                   'partno' 'count' 'parts' 'times'
                   'costs' 'amount' 'calcStatus')
  classVars: #()
  poolDictionaries: #[]
  inDictionary: WFFolder
  constraints: #[ #[ #custId, String],
                  #[ #name, String],
                  #[ #address, String],
                  #[ #partno, String],
```

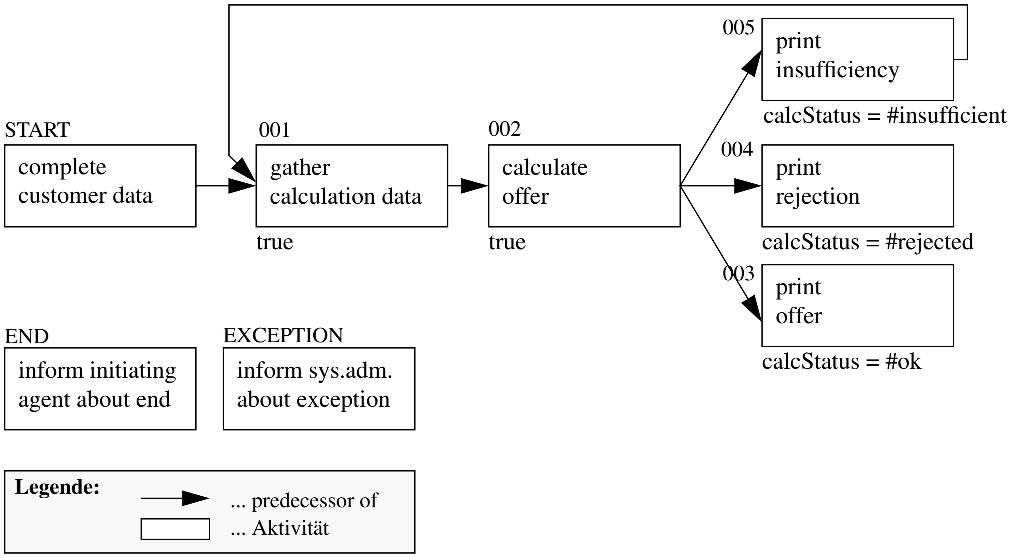


Abbildung 5.4: Aktivitätennetz zur Angebotsbearbeitung

```
                    #[ #count, Integer],
                    #[ #parts, Array],
                    #[ #times, Array],
                    #[ #costs, Array],
                    #[ #amount, Float],
                    #[ #calcStatus, Symbol] ]
    isInvariant: false
```

(8) Nun sind alle Voraussetzungen getroffen, um endlich das Aktivitätennetz und den Workflow Typ abbilden zu können. Die *Datenflußbeschreibung* wurde bereits durch die Definition der Aktionsmethoden festgelegt.

(9) Die *Organisationspolitiken* müssen nicht unbedingt detailliert erklärt werden, da sie ohnehin dem nachfolgenden Beispiel zu entnehmen sind.

(10) Der Workflow Typ `OFFER` bestimmt sowohl das Betriebsobjekt (`company:`) als auch die zu verwendende Datenstruktur (`folderType:`). Die Datenflußbeschreibung wird durch die beiden Methoden `inInstVars:` und `outInstVars:` realisiert. Mit der Methode `orgPolicy:cond:sel:` werden die Organisationspolitiken definiert.

```
" *** definition of the workflow type and its activities **** "

| wf actstart act1 act2 act3 act4 act5 actend actex |

wf := WorkflowType new.
wf id: 'OFFER';
   name: 'Offer Treatment';
   company: (WFCompanyBase at: 'C200');
   folderType: OfferFolder;
   startOrgPolicy: 1
      cond: '^true'
```

```
      sel: '(a playsRole: #Clerk)'.
```

(11)

```
actstart  := StartActivity new.
actstart
   workflowType: wf;
   action: 'completeCustomerData:';
   inInstVars: #('custId');
   outInstVars: #('name' 'address' 'partno' 'count');
   orgPolicyBind: #less;
   orgPolicy: 1
      cond: '^true'
      sel: '(a playsRole: #Clerk)'.

act1 := InnerActivity new.
act1 id: '001';
   name: 'Gather Calculation Data';
   workflowType: wf;
   action: 'gatherCalcData:';
   inInstVars: #('partno');
   outInstVars: #('parts' 'times' 'costs');
   execCondition: '^true';
   orgPolicyBind: #less;
      orgPolicy: 1
      cond: '^true'
      sel: '(a playsRole: #Constructor inGroup: #D2)'.

act2 := InnerActivity new.
act2 id: '002';
   name: 'Calculate Offer';
   workflowType: wf;
   action: 'calculatePart:count:times:costs:';
   inInstVars: #('partno' 'count' 'times' 'costs');
   outInstVars: #('amount' 'calcStatus');
   execCondition: '^true';
   orgPolicyBind: #less;
      orgPolicy: 1
      cond: '^true'
      sel: '(a playsRole: #CalcProgram inGroup: #D2)'.

act3 := InnerActivity new.
act3 id: '003';
   name: 'Print Offer';
   workflowType: wf;
   action: 'printOffer:name:address:partno:count:amount:';
   inInstVars: #('custId' 'name' 'address' 'partno' 'count' 'amount');
   outInstVars: #();
   execCondition: '^(actFolder calcStatus = #ok)';
   orgPolicyBind: #less;
      orgPolicy: 1
      cond: '^true'
      sel: '(a playsRole: #PrintProgram inGroup: #D4)'.

act4 := InnerActivity new.
act4 id: '004';
   name: 'Print Rejection';
   workflowType: wf;
   action: 'printRejection:name:address:partno:';
```

```
    inInstVars: #('custId' 'name' 'address' 'partno');
    outInstVars: #();
    execCondition: '^(actFolder calcStatus = #rejected)';
    orgPolicyBind: #less;
    orgPolicy: 1
       cond: '^true'
       sel: '(a playsRole: #PrintProgram inGroup: #D4)'.

 act5 := InnerActivity new.
 act5 id: '005';
    name: 'Print Insufficiency';
    workflowType: wf;
    action: 'printInsufficiency:name:address:partno:';
    inInstVars: #('custId' 'name' 'address' 'partno');
    outInstVars: #();
    execCondition: '^(actFolder calcStatus = #insufficient)';
    orgPolicyBind: #less;
    orgPolicy: 1
       cond: '^true'
       sel: '(a playsRole: #PrintProgram inGroup: #D4)'.

 actend := EndActivity new.
 actend workflowType: wf;
    orgPolicyBind: #less;
    orgPolicy: 1
       cond: '^true'
       sel: '(a hasInitiatedWorkflow: #OFFER withFolder: (actFolder id))'.

 actex := ExceptionActivity new.
 actex workflowType: wf;
    orgPolicyBind: #less;
    orgPolicy: 1
    cond: '^true'
    sel: '(a playsRole: #SysAdmin)'.
```

(12)

```
 act1 becomeSuccessorOf: actstart.
 act2 becomeSuccessorOf: act1.
 act3 becomeSuccessorOf: act2.
 act4 becomeSuccessorOf: act2.
 act5 becomeSuccessorOf: act2.
 act1 becomeSuccessorOf: act5.

 wf startActivity:     actstart;
    endActivity:       actend;
    exceptionActivity: actex.

 WFWorkflowBase at: (wf id) put: wf.
```

Am Ende der Definition werden die Nachfolgebeziehungen aufgebaut und die Aktivitäten dem Workflow Typ zugeordnet. Das gesamte Aktivitätennetz ist implizit durch die Nachfolger der Startaktivität gegeben. Als letzter Schritt wird der Workflow Typ unter seiner Nummer `OFFER` in die „Workflow Basis" `WFWorkflowBase` gespeichert.

Damit ist ein Workflow Typ im Workflow System spezifiziert. Mit verschiedenen Akten können jetzt Durchläufe gestartet und weiterverarbeitet werden.

5.3 Ablauf

Ein Workflow wird gestartet, indem man einen Folder erzeugt, ihn mit einer Nummer und eventuell anderen notwendigen Daten versieht, und ein Agent die Methode `startWorkflow:withFolder:` aufruft. Der Agent muß in der Auswertung der Startorganisationspolitik des Workflow Typs enthalten sein, ansonsten erhält er als Rückgabewert den Fehlercode `#err_notAllowedToStart`. Im folgenden Beispiel startet der Agent `h004` mit dem Akt mit der Nummer `103` einen neuen Durchlauf im Workflow Typ `OFFER`.

```
" *** start workflow OFFER *** "

| fol anAgent  |
fol := OfferFolder new.
fol id: '103';
    custId: '3499'.
anAgent := (WFCompanyBase at: 'C200') agents at: 'h004'.
anAgent startWorkflow: 'OFFER' withFolder: fol.
```

Damit wird die Startaktivität verteilt und liegt als Eintrag in der Arbeitsliste eines Agenten, natürlich unter der Voraussetzung, daß kein Fehler aufgetreten ist.

Will man nun die einzelnen Schritte abarbeiten, so muß der jeweils betroffene menschliche Agent einen Eintrag mit der Methode `selectElem:` ausführen. Die automatische Abarbeitung der Arbeitslisten von Software Agenten wird durch Regeln gesteuert, weshalb die Methode `selectElem` bei diesen im Normalfall nicht ausgeführt werden muß.

Die Agenten werden im Folgenden kurz mit der Methode zur Ausführung eines Eintrags aufgelistet. Im GemStone Workspace können diese OPAL-Anweisungen mit „GS-inspect“ oder „GS-do it“ ausgeführt werden. Die Software Agenten werden nur der Vollständigkeit halber erwähnt. Deren Ausführung wird, wie bereits erwähnt, von ECA-Regeln automatisch angestoßen.

```
" *** execute working steps *** "

((WFCompanyBase at: 'C200') agents at: 'h001') selectElem: 1.

((WFCompanyBase at: 'C200') agents at: 'h002') selectElem: 1.

((WFCompanyBase at: 'C200') agents at: 'h003') selectElem: 1.

((WFCompanyBase at: 'C200') agents at: 'h004') selectElem: 1.

((WFCompanyBase at: 'C200') agents at: 'a001') selectElem.

((WFCompanyBase at: 'C200') agents at: 'a002') selectElem.
```

Die Arbeitsschritte müssen natürlich nicht in *einer* GemStone-Sitzung („session“) ausgeführt werden. In der Praxis hat jeder Agent eine eigene Verbindung zur Datenbank und führt in seiner eigenen Umgebung die Arbeitsschritte aus.

Zur Beobachtung der einzelnen Arbeitsschritte und deren Ergebnisse kann der Folder, seine Daten und die Historie mittels folgender Anweisung untersucht werden:

```
" *** have a look on the folder *** "

(WFWorkflowBase at: 'OFFER') folders at: '103' asString.
```

5.4 Bemerkungen zu Modellierung und Ablauf

Schon die Modellierung eines solch kleinen Beispiels verlangt nach Werkzeugunterstützung. Die bloße Eingabe der Rollen, Gruppen, Datenstrukturen, Aktivitäten etc. verschlingt sehr viel Zeit. Kommen noch Fehler hinzu, die erst bei der Ausführung bemerkt werden, so verlängert sich die Definitionszeit enorm.

Viele Fehler treten dabei in der Definition der Aktionen bzw. des damit zusammenhängenden Datenflusses auf. Stimmt etwa nur ein Variablenname nicht, so wird der Einfachheit halber der gesamte Workflow Typ neu definiert, weil dabei lediglich der Name in der Aktivitätendefinition geändert, und die gesamte Workflow Typdefinition selektiert und erneut ausgeführt werden muß. Denn die Datenschnittstelle einer einzelnen Aktivität umzudefinieren, ist vom GemStone Workspace aus leider nicht sehr einfach, da zuerst die Aktivität referenziert werden muß. Das Beispiel 5.1 zeigt die Anweisungsfolge, die zur Änderung bloß einer Datenschnittstelle im GemStone Workspace einzugeben und auszuführen ist.

Beispiel 5.1 *Änderung der Leseinstanzvaraiablen der Aktivität* `003` *aus dem Workflow Typ* `OFFER`*:*

```
((WFWorkflowBase at: #OFFER) getActivityWithId: '003')
    inInstVars: #('custId' 'name' 'address' 'partno'
                 'count' 'amount' 'newVar').
```

Die völlige Neudefinition eines Workflow Typs verschlingt aber ebenfalls einiges an Zeit, da alle Regeln erneut erzeugt werden müssen. Bei der Neudefinition ist zu beachten, daß die Regeln des alten Workflow Typs nur dann überschrieben bzw. geändert werden, wenn sie den gleichen Namen besitzen. Wird bei der Neudefinition z.B. die Nummer einer Regel geändert und die alte nicht mehr verwendet, so bleiben die alten Regeln in der Regelbasis bestehen. Sie stören zwar den Ablauf nicht, weil sie nicht mehr gefeuert werden können, sie verbrauchen aber unnötigen Speicherplatz. Noch dazu werden die Objekte, die von ihnen referenziert werden, für den „garbage collector“ nicht freigegeben. Bevor ein Workflow Typ neu definiert wird, und sich dabei die Nummern von Aktivitäten ändern, oder Alternativen von Organisationspolitiken nicht mehr verwendet werden, empfiehlt es sich, dem alten Workflow Typ zuvor noch die

Methode `removeAllDynamicRules` zu senden. Daraufhin werden alle Regeln entfernt, die im Zusammenhang mit dem Workflow Typ angelegt worden sind.

Auch die Ausführung läßt eine komfortable Benutzung missen. Gerade Werkzeuge zur Überwachung der Ausführung wären für einen Test der Modellierungsergebnisse wichtig. Ein Simulationswerkzeug würde natürlich eine ideale Ergänzung darstellen.

Weiters ist zu bemerken, daß die Entwicklung dieses Beispiels nie ganz nach dem angegebenen Schema durchgeführt werden kann. Wie im Software Engineering nicht nur „top-down", sondern auch „bottom-up" vorgegangen wird, so kann auch hier teilweise „von hinten nach vorne" modelliert werden. Ein geeignetes Werkzeug kann die sonst übliche Vorgangsweise „trial and error" beseitigen.

Kapitel 6

Analyse eines kommerziellen Workflow Systems

In dieser Arbeit wurde ein Prototyp eines Workflow Systems entwickelt, bei dessen konzeptueller Gestaltung bisher wenig Augenmerk auf kommerziell verfügbare Workflow Systeme gelegt wurde. Wissenschaftliche Ansätze und Prototypen wurden aber sehr wohl einbezogen. In diesem Kapitel wird TriGS$_{flow}$ mit einem kommerziellen Workflow System vergleichen. In Hinblick auf die Benutzerfreundlichkeit können jedoch kaum Vergleiche angestellt werden, da für TriGS$_{flow}$ zur Zeit keine geeignete Benutzeroberfläche vorhanden ist. Der Vergleich beruht daher auf den aus der Benutzung abgeleiteten Konzepten.

Als kommerzielles Produkt wird *WorkFlow* von C.S.E. Systems herangezogen [CSES94]. Bevor Vergleiche gezogen werden, wird das Programmpaket *WorkFlow* auf der Basis einiger wichtiger Aspekte im Abschnitt 6.1 analysiert. Beim darauffolgenden Vergleich im Abschnitt 6.2 wird sich herausstellen, inwieweit *WorkFlow* den in Kapitel 1 angeführten Anforderungen der Workflow Management Coalition, einer Vereinigung zahlreicher Hersteller und wissenschaftlicher Institute im Bereich Workflow Managment [WFMC94], entspricht. C.S.E Systems ist Mitglied dieser Vereinigung. Abschnitt 6.3 faßt allgemeine Bemerkungen in bezug auf *WorkFlow* und TriGS$_{flow}$ zusammen.

6.1 Analyse von *WorkFlow*

Das Anwendungsgebiet von WorkFlow beschränkt sich auch auf den Büro- und Verwaltungsbereich. Einerseits kann es in der Privatwirtschaft als Büroinformationssystem (BIS) zur Geschäftsfallabwicklung verwendet werden, andererseits als Kanzleiinformationssystem (KIS) in der Verwaltung. Eine Erweiterung zur unternehmensweiten Unterstützung, damit auch für Produktionsprozesse, wird nicht erwähnt. Es unterstützt aber jegliche Mischform von reinem Papier-Geschäftsfall bis zu reinem elektronischen Geschäftsfall.

Neben den Hard- und Software-Voraussetzungen wird in der Folge die Funktionalität der einzelnen Programmkomponenten erläutert. Die zwei Grundkomponenten sind:

WorkFlow WorkDesk: Arbeitsumgebung zur Geschäftsfallbearbeitung

WorkFlow ConAd: Konfiguration und Administration des Systems

Die Informationen über das Programmpaket stammen aus der Arbeit mit dem Programmpaket *WorkFlow* selbst und den mitgelieferten Benutzer- und Installationshandbüchern [CSES94].

6.1.1 Hardware- und Software-Voraussetzungen

Der WorkFlow-Server ist auf einer IBM RS6000 mit einem UNIX-Betriebssystem AIX 3.2 einzurichten. Die WorkFlow Clients sind auf PC-Basis verfügbar. An Speicherplatz sind am Server ca. 30 MB für die WorkFlow-Applikation selbst und ca. 150–200 MB für den Datenbereich zu veranschlagen.

An Software muß am Server das Betriebssystem AIX 3.2 und das Datenbanksystem ORACLE 7.0 eingesetzt werden. Für ORACLE sind am Server in etwa 130 MB Speicherplatz zur Verfügung zu stellen. Die Datenübertragung erfolgt mittels TCP/IP.

Die PC-Clients können unter DOS/Windows 3.x oder OS2/WINOS 3.x laufen, wobei DOS ab der Version 3.3 und OS2 ab Version 2.0 vorliegen muß. Da die Datenübertragung vom PC zum Server über TCP/IP erfolgt, muß auch am PC die entsprechende Software installiert sein.

6.1.2 WorkFlow WorkDesk

Dieses Programm stellt den Arbeitsplatz des Benutzers dar und dient zur Ababarbeitung der Geschäftsfälle. Für die Bearbeitung von Geschäftsfällen sind im wesentlichen drei Module zuständig:

- Eingangsstelle
- Arbeitskorb
- Globale Suche

Im Folgenden werden diese drei Funktionen und noch weitere näher erläutert.

6.1.2.1 Eingangsstelle

Die Eingangsstelle enthält zwei Listen: Eingangsstücke (EST) und Geschäftsfälle (GF). EST führen im Normalfall zum Anlegen eines GF. In der Eingangsstelle können EST angelegt, bearbeitet, an andere Benutzer verschickt oder einem GF zugeordnet werden. EST werden durch verschiedene Typen klassifiziert (Papier, WinWord, Notiz, ...). GF können ebenfalls bearbeitet, angelegt und weitergeleitet werden. Wird ein EST einem GF zugeordnet, so scheint es nur mehr in dessen Schriftstückliste auf (siehe Arbeitskorb).

6.1.2.2 Arbeitskorb

Der Arbeitskorb ist das Kernstück der Applikation. Die GF werden zwar auch in der Eingangsstelle aufgelistet, es gibt aber weniger Funktionen zu deren Bearbeitung. Die Synchronisation mit den begleitenden Unterlagen eines Papier-GF erfolgt durch Setzen eines dafür bestimmten Attributes. Ein im Arbeitskorb angezeigter GF kann auf einen späteren Zeitpunkt verlegt werden, womit dieser aus dem Arbeitskorb verschwindet und erst wieder nach Ablauf der Wiedervorlagezeit erscheint (Wiedervorlage). Ferner kann ein GF einem anderen untergeordnet, oder von diesem wieder freigestellt werden. Unter Leitweginformation versteht man die Information über den Benutzer, der den GF weitergeleitet (zuletzt bearbeitet) hat, oder allgemein die Information über den Verarbeitungsweg eines Geschäftsstückes. Leitwege und Metadaten eines GF können auch geändert werden. Der Leitweg ist die Reihenfolge der Verarbeitungsschritte bzw. die Abfolge der Statusübergänge. Metadaten beschreiben nicht den Aufbau, sondern sind allgemeine Daten eines Geschäftsstückes.

Einem GF können Schlagwörter zugeteilt werden, um die Suche zu erleichtern. Der Schlagwortkatalog ist hierarchisch gegliedert und wird im Konfigurationsprogramm ConAd gewartet. Ein GF besitzt auch Ausführungsfristen; Referenzen auf andere GF sind ebenfalls möglich.

GF, die einem anderen untergeordnet werden, werden Subgeschäftsfälle genannt, der übergeordnete wird zum Bündelgeschäftsfall. Zu beachten ist, daß die Unterordnung nur einstufig erfolgen kann. Das bedeutet, daß ein Bündel keinem anderen GF untergeordnet werden kann. GF dürfen nur *einem* Bündel zugeordnet werden. Jede Unterordnung kann wieder aufgelöst werden.

Nach der Bearbeitung eines Geschäftsfalles wird dieser weitergeleitet. Der Leitweg wird in der Konfiguration vorgegeben, kann aber beeinflußt werden. Jedem Folgebenutzer können Notizen mitgegeben werden.

Ein GF beinhaltet zwei Arten von Dokumenten (Geschäftsstücken): *Eingangsstücke* (EST) und *Schriftstücke* (SST). Während EST sowohl in elektronischer als auch in Papierform vorliegen können, existieren SST nur in elektronischer Form und innerhalb eines GF. EST und SST haben zwei getrennte Datenbereiche: die Metadaten und ein Dokument. Metadaten beschreiben allgemeine Attribute, wie z.B. Erstellungsdatum. Ein Dokument ist eine Datei, die den

Inhalt eines EST oder SST repräsentiert. Von einem in Papierform vorliegenden EST existieren nur die Metadaten in elektronischer Form. Ein Dokument wird mittels Aufruf einer Methode (z.B. WinWord) bearbeitet. Methoden sind externe Programmaufrufe. SST können zusätzlich versioniert, importiert und exportiert werden.

Üblicherweise kann ein GF von einem anderen Benutzer übernommen, approbiert (genehmigt), storniert, hinterlegt oder abgetreten werden. Die Ausführung dieser Funktionen ist an vordefinierte, aber konfigurierbare Benutzerrechte gebunden.

6.1.2.3 Globale Suche

Die Suche erfolgt nach Geschäftsfällen oder Dokumenten. Bei der Dokumentensuche wird zwischen EST und SST unterschieden. In die Dokumente bzw. GF, die den Suchkriterien entsprechen, kann Einsicht genommen werden. Das Suchergebnis wird aber immer den Verschlußrechten des Benutzers angepaßt. Dokumente bzw. GF mit höheren Verschlußrechten werden bei der Suche nicht berücksichtigt.

6.1.2.4 Adressenverwaltung

Der WorkDesk-Arbeitsplatz beinhaltet auch eine vollständige Adressenverwaltung. Die Adressen werden in private Adressen, Abteilungsadressen und systemweite Adressen unterteilt. Jedem SST können solche Adressen in Form von Adressaten zugeordnet werden.

6.1.2.5 Druckauftrag

Die SST können in Form einer Einzelerledigung oder eines Summenerledigungsentwurfes ausgedruckt werden. Dabei werden die SST einzeln oder gesammelt für die Adressaten ausgegeben.

6.1.2.6 Funktionen außerhalb der Geschäftsfallbearbeitung

Neben der eigentlichen GF-Bearbeitung wird noch weitere Unterstützung angeboten:

Auswertungen: Es gibt verschiedene vordefinierte Reporte (z.B. Abgeschlossene GF, Neue GF). Sie werden im Hintergrund ausgeführt und in Dateien gespeichert, die dann weiter bearbeitet und/oder ausgedruckt werden können. Mit Hilfe von *WorkFlow ConAd* können weitere Reporte definiert werden.

Post: WorkDesk ermöglicht es, innerhalb des Workflow Systems Mitteilungen an andere Benutzer zu senden. Der Post können auch Dateien angehängt werden. Posteingang und Postausgang erledigen den Empfang bzw. das Versenden der Mitteilungen. Ist das Workflow System auf mehrere Knoten verteilt, so werden die Mitteilungen auch außerhalb des lokalen Netzes verschickt. Mails anderer E-Mail-Systeme können auch verarbeitet werden. Zusätzlich kann eine Option angegeben werden, die den Empfänger zur Quittierung des Empfangs veranlaßt.

Vertretung: Jeder Benutzer kann einen anderen vertreten. Er übernimmt damit die Daten des Mitarbeiters in der Eingangsstelle, im Arbeitskorb usw. Die Vertretung kann jederzeit wieder aufgelöst werden.

6.1.3 Konfiguration von WorkDesk

Die Konfiguration eines WorkDesk-Arbeitsplatzes wird durch drei Dateien gesteuert:

`WF.INI` Definition der WorkFlow-Server-Verbindung, Pfade und Dateinamen, Betriebsarten, login, mitzustartende externe Programme, ...

`METHODEN.INI` Definition von arbeitsplatzbezogenen Daten, wie die Zuordnung von Produkten zu Methoden; Pfade zu den Programmen

`PRODUKTE.INI` Definition der vom Arbeitsplatz abhängigen Daten für die externen Produkte, wie Aufrufparameter oder DDE-Befehle (Dynamic Data Exchange)

Die Bearbeitung dieser Konfigurationsdateien kann entweder von Hand über einen Texteditor oder mit dem Programm *WorkFlow WDSetUp* durchgeführt werden. Neben den oben aufgezählten Definitionen können auch Anwendergruppen mit unterschiedlichen Berechtigungen zur Konfigurationsänderung angelegt werden.

6.1.4 WorkFlow ConAd

Der Name *ConAd* steht für „Configuration & Administration“. Es dient zur Einrichtung des Workflow Systems. Die Konfiguration kann eine unterschiedliche Reichweite im System haben: benutzerspezifisch, abteilungsspezifisch, systemweit (LAN) oder sogar netzweit (WAN). Im Folgenden werden die einzelnen Module des Programmes beschrieben. Zu jedem Modul wird beschrieben, welche Komponenten des Workflow Systems darin verwaltet werden.

6.1.4.1 Aufbauorganisation

Organisationseinheiten: Die Organisationseinheiten werden hierarchisch in einer Abteilungsstruktur gegliedert. Jede Organisationseinheit wird einem Knoten im WorkFlow-WAN zugeordnet.

Benutzer: Alle benutzerspezifischen Daten, wie Kennzeichen, Namen, (Amts-) Titel und (Amts-) Funktion, werden hier verwaltet. Jeder Benutzer erhält weiters eine Telefon-Durchwahl, ein Paßwort, einen Stellvertreter und wird einer oder mehreren Organisationseinheiten zugeordnet, wobei eine als primär angenommen wird.

Abteilungsleiteroptionen: Für jede Organisationseinheit kann ein Benutzer als Abteilungsleiter, Empfänger und Versender festgelegt werden.

eigene Stellvertreter: Der konfigurierende Benutzer kann hiermit seine eigenen Stellvertreter bestimmen.

Benutzerrechte: Die Rechte von einem oder mehreren Benutzern werden hier definiert. Benutzerrechte können nicht an eine ganze Abteilung vergeben werden, es wird aber die Rechtedefinition aller Benutzer einer Abteilung unterstützt. Rechte können direkt vergeben werden, oder als Rechteprofil zusammengefaßt und dann verteilt werden. Dabei gibt es vier sogenannte Rechteklassen: Leitwegerechte, Masken-/Funktionsrechte, Verschluß-/Datenrechte und allgemeine Rechte.

6.1.4.2 Aktenplan/Dokumentstruktur

Sachgebiete: Sachgebiete entsprechen Geschäftsstücktypen. Sie können hierarchisch gegliedert werden. Zu einem Sachgebiet können auch ein oder mehrere Betrefftexte definiert werden.

Schlagworte: Der Schlagwortkatalog wird ebenfalls hierarchisch gegliedert und ist vor allem für die globale Suche von Dokumenten und Geschäftsfällen von Bedeutung.

Personal: Jeder Personaleintrag kann zusätzlich zu den üblichen Daten (Personalnummer, Namen, Adresse usw.) auch Bezüge zu Sachgebieten oder deren Gruppen und zu Vorgängen herstellen. Damit können Personen bestimmt werden, die sich mit einem vorgegebenen Bereich beschäftigen.

6.1.4.3 Ressourcen

Adressaten: Ein Adressat kann entweder zu den privaten Adressen, Abteilungsadressen oder systemweiten Adressen gezählt werden.

Verteiler: Ein Verteiler steht für mehrere Adressaten; wenn ein Schriftstück einen Verteiler als Adressaten hat, so wird z.B. beim Drucken das Schriftstück für alle Adressaten des Verteilers ausgegeben.

Vorlagen: Vorlagen können für jegliche Art von Geschäftsstücken verwendet werden., z.B. für Bestellung, Eingang usw. Sie enthalten meist Textbausteine und werden in abteilungsweite und systemweite Vorlagen getrennt. Unter „abteilungsweit“ versteht man jene Organisationseinheit, der der konfigurierende Benutzer primär zugeordnet ist.

Drucksorten: Eine Drucksorte beschreibt das Layout von Geschäftspapieren.

6.1.4.4 Ablauforganisation

Vorgänge: Hier werden nur die Bezeichnungen von Vorgängen verwaltet. Der Ablauf (die Ausführungsreihenfolge) wird erst durch den Leitweg bestimmt. Der Leitweg reiht mögliche Zustandsübergänge aneinander. Es ist zu beachten, daß nur noch nicht gesicherte Vorgänge geändert oder gelöscht werden können.

Stati: Stati sind Zustände, in denen sich ein GF befinden kann. Sie bestehen aus einer Kurzbezeichnung und einer Beschreibung. Ein Status kann wiederum nur dann gelöscht werden, wenn er noch nicht gesichert ist.

Zustandsgraph: Für jeden Vorgang kann ein Zustandsgraph angelegt werden. Dieser Graph sollte für die meisten Benutzer wenigstens zur Einsicht zur Verfügung stehen. Er besteht aus einer Menge von Zuständen und deren Übergängen. Jeder Übergang wird durch folgende Daten näher bestimmt:

- Methode (Programmaufruf, wird von WorkFlow initiiert)
- Trigger (ruft Programme oder begleitende Aktionen am WorkFlow-Server auf)
- externe Server-Programme (GF weiterleiten)
- externe Server-Programme (GF öffnen)
- externe WorkDesk-Programme (GF weiterleiten)
- externe WorkDesk-Programme (GF öffnen)
- Ikone (für jeden Zustandsübergang kann eine Ikone gewählt werden, mit der ein Benutzer den Übergang einleiten kann.
- Übergangsbezeichnung
- Leitwegerecht, erforderlich für jeden Übergang

Unter *Methode* vesteht man eine Aktion des Programmes WorkFlow, die bei der Durchführung eines bestimmten Statusübergangs ausgeführt wird. In WorkFlow gibt es fünf vordefinierte Methoden. Mit jedem Statusübergang muß eine dieser fünf ausgeführt werden: Standardmethode, Approbation (Genehmigung), Hinterlegen, Stornieren, Bearbeiten.

Unter *Trigger* versteht man eine Aktion, die bei einem Vorgang vom WorkFlow-Server-Programm ausgeführt wird. Zur Zeit sind drei Trigger definiert: Standardtrigger 0 (keine zusätzliche Aktion am Server), Trigger

1 (Erweiterung der Geschäftszahl um die Kurzbezeichnung der Organisationseinheit des Bearbeiters) und Trigger 2 (Genehmigungsdatum scheint am Aktendeckel auf).

Standardleitwege: Jedem Vorgang wird der Standardleitweg zugeordnet. Dieser Leitweg kann aber durch den Benutzer bei der Konfiguration geändert werden, indem z.B. zusätzliche Zustände eingefügt, andere herausgenommen werden, oder die Reihenfolge geändert wird. Bei im Zustandsgraphen definierten Übergängen werden dann die entsprechenden Methoden ausgeführt. Jeder Position kann eine Organisationseinheit oder ein Benutzer zur Ausführung zugeordnet werden. Findet keine Zuordnung statt, so muß der weiterleitende Bearbeiter einen Nachfolger auswählen.

Metadaten: Dabei handelt es sich nicht um den Aufbau von Dokumenten, sondern um Daten, wie z.B. Erstellungszeit, die für einen ganzen Dokumenttyp verwendbar sind. Es gibt drei verschiedene Arten von Metadaten: Metadaten für Aktendeckel, Schriftstücke und Schriftstücklisten. Die fix vorgegebenen Metadaten können für einzelne Zustände gesperrt werden, damit sie nicht geändert werden können.

6.1.4.5 Systemkonfiguration

(Amts-)Titel: z.B. Amtsrat, Hofrat

(Amts-)Funktionen: z.B. Einkäufer, Sachbearbeiter

Methoden: Eine Methode definiert eine auszuführende Aktion. Die Aktion entspricht im Normalfall einem Aufruf eines externen Programmes. Die Methode „WINWORD“ startet etwa das Programm WinWord von Microsoft und versorgt es mit den entsprechenden Parametern.

Skartierungen/Aussonderungen: Hier werden die Jahre der Skartierung bzw. Aussonderung festgelegt.

Knoten: Das Workflow System kann auf mehreren Knoten in einem WAN verteilt sein. Hier werden die Bezeichnung und die Identifikationsnummer der Knoten verwaltet. Ein angelegter Knoten kann nicht wieder gelöscht werden!

Bezugstypen: Bezugstypen setzen Akte oder Dokumente untereinander in Beziehung (z.B. Vorakt-Nachakt). Bezugstypen selbst können wieder verbunden werden.

Adreßimport: Adressen können von externen Dateien mit bestimmtem Format in das Workflow System eingebracht werden.

Fristentypen: Jedem GF kann eine Frist gegeben werden. Fristen werden in Typen unterteilt, die hier definiert werden können.

Export der Weisungssammlung: Darunter versteht man den Export der Daten jener Akten, die für einen Export gekennzeichnet sind. Die exportierten Daten können z.B. auf Diskette gespeichert und somit auf Rechnern ohne Serververbindung eingesehen werden.

6.1.5 WorkFlow PDI (Program & Data Interface)

Anhand des PDI kann das Workflow System an Benutzerwünsche angepaßt werden. Es werden sowohl Schnittstellen zur Weiterverarbeitung der Daten, zur Einbindung von externen Applikationen, als auch zur Definition weiterer Reporte angeboten.

PDI/IDF (Interface Data Files):
Acht IDF-Dateien beschreiben die Schnittstelle zu externen Programmen, indem sie stets die aktuellen Daten in vordefinierter Struktur beinhalten.

PDI/WPI (WorkDesk Programming Interface):
Hiermit wird die Einbindung externer Applikationen definiert, unter anderem durch ihre Aufrufparameter. Die Applikationen können *manuell* (aus WorkDesk heraus) oder *automatisch* (bei Statusübergängen als Methoden) aufgerufen werden.

PDI/SPI (Server Programming Interface):
Die bei Statusübergängen durch Trigger am Server ausgeführten Programme werden hier eingebunden.

PDI/RPI (Report Pogramming Interface):
Die Auswertungen (Reporte) werden hier definiert. Die vorgegeben Reporte können so um weitere ergänzt werden.

6.1.6 Weitere Programmodule

Um das WorkFlow System auch in einem globalen Netz betreiben zu können, werden zwei Module zur Verfügung gestellt.

WorkFlow Distributed ermöglicht den Zusammenschluß mehrerer lokaler Workflow Systeme (LAN) zu einem globalen (WAN). Jeder Knoten im WAN wird als lokales Workflow System betrieben, das aus einem Server und mehreren Clients besteht. Einer der Server wird als Master auserkoren. Alle anderen Server (*Slaves*) übernehmen dessen Konfiguration in bezug auf Knoten, Aufbau- und Ablauforganisation etc., damit sich auf allen Knoten dieselben Definitionen befinden. Eine Konfigurationsänderung am Master wird an alle Slaves weitergegeben. Die Verteilung ist für den Benutzer prinzipiell transparent. Nur bei der Konfiguration sind mehrere Knoten sichtbar. Die globale Suche, die im Normalfall im lokalen Netz abläuft, kann vom Benutzer auf das globale Netz ausgedehnt werden. Es ist zu beachten, daß ein einmal angelegter Knoten nicht mehr gelöscht werden kann, was vermutlich mit der aufwendigen und komplizierten Aufrechterhaltung der Konsistenz der Daten zusammenhängt.

WorkFlow Bridge dient zur Erweiterung der Postverarbeitung. Damit können auch Benutzer auf verschiedenen Knoten über das integrierte Postsystem kommunizieren. Der Benutzer wird dabei lediglich aus der Organisationsübersicht ausgewählt. Die Eingabe einer E-Mail-Adresse entfällt somit. Außerdem kann mit diesem Modul Post von anderen E-Mail-Systemen empfangen und an Benutzer anderer E-Mail-Systeme geschickt werden. Die Verwendung eines zweiten E-Mail-Programmes zur Kommunikation mit Benutzern außerhalb des Workflow Systems kann daher entfallen.

6.2 Vergleich mit TriGS$_{flow}$

Im vorhergehenden Abschnitt wurde die gesamte Funktionalität des Programmpaketes *WorkFlow* beschrieben. In diesem Abschnitt wird diese Funktionalität auf konzeptueller und soweit möglich auch auf implementierungstechnischer Ebene mit TriGS$_{flow}$ verglichen. Ein Vergleich der Benutzerschnittstelle kann nicht erfolgen, da TriGS$_{flow}$ noch keine grafische Benutzerschnittstelle zur Verfügung stellt. Neben Vergleichen werden an den entsprechenden Stellen auch ergänzende Bemerkungen angebracht und Hinweise auf Unzulänglichkeiten des WorkFlow-Konzepts gegeben.

6.2.1 Anwendungsbereich

An erster Stelle ist festzustellen, daß sich auch das Produkt WorkFlow auf den Büro- und Verwaltungsbereich beschränkt. Eine mögliche Erweiterung des Anwendungsbereiches um die Unterstützung aller unternehmensweiten Prozesse inklusive Produktionsprozesse wird nicht erwähnt.

TriGS$_{flow}$ beschränkt sich ebenfalls auf den Büro- und Verwaltungsbereich. Die Ausweitung zur Unterstützung von Produktionsprozessen wurde zwar diskutiert, aber vorerst nicht realisiert, wenngleich Software Agenten einen ersten Vorstoß in diese Richtung darstellen.

6.2.2 Implementierung

WorkFlow verwendet wie TriGS$_{flow}$ ein Datenbanksystem als Grundlage der Datenhaltung. Während TriGS$_{flow}$ auf das objektorientierte Datenbanksystem GemStone aufsetzt, verwendet WorkFlow das relationale Datenbanksystem ORACLE 7.0. Inwieweit aktive Elemente eingesetzt wurden, kann nicht beurteilt werden, da nur die Benutzer- und Installationshandbücher, aber keine Systemdokumentation vorliegen. Jedenfalls enthält ORACLE 7.0 auch ein

Triggerkonzept. Der Zugriff auf Daten stellt etwa ein Ereignis dar (z.B. ON UPDATE OF ... DO ...), in dessen Verbindung auch Aktionen ausgeführt werden können.

Die Hardware- und Software-Voraussetzungen entsprechen in etwa denen von TriGS$_{flow}$. WorkFlow verwendet als Server eine IBM RS6000 mit dem UNIX-Betriebssystem AIX. Die WorkFlow-Clients werden auf PC installiert, die entweder mit Windows oder OS2 ausgestattet sind. In TriGS$_{flow}$ ist dies ähnlich. GemStone läuft ebenfalls auf einer Sun Workstation mit UNIX als Betriebssystem. GemStone erlaubt auch die Installation von Clients auf PCs.

Der Verteilungsaspekt wird in WorkFlow berücksichtigt. Mit den Modulen *WorkFlow Distributed* und *WorkFlow Bridge* kann das System auf ein globales Netz (WAN) ausgedehnt werden. Der Endbenutzer bemerkt das nur in der Tatsache, daß in diesem Fall die globale Suche auf das gesamte Netz ausgedehnt werden kann. In TriGS$_{flow}$ wurde bis jetzt der Verteilungsaspekt außer Acht gelassen.

Die Einbindung externer Applikationen wird von WorkFlow unterstützt, wenngleich diese Einbindung sehr komplex ist. Es müssen nicht nur der Programmname und der Pfad bestimmt werden, sondern jedes Programm verlangt auch bestimmte Aufrufparameter, um die Daten des Workflow Systems übernehmen zu können. In Windows ist die Kommunikation unter anderem mittels DDE (Dynamic Data Exchange) möglich, wodurch weitere Definitionen erforderlich sind. Die Einbindung einer Applikation ist auch hier ein aufwendiges Vorgehen. In TriGS$_{flow}$ wurde dies bisher nur simuliert, indem die Aktionen entweder leer oder mit einer einfachen Berechnung versehen waren. Ein anderes Programm wurde nicht aufgerufen. Unter Verwendung der C-Schnittstelle von GemStone kann aber die Verwendung externer Programme ermöglicht werden.

6.2.3 Modellierung

Ähnlich wie in TriGS$_{flow}$ wird die Organisation auch in WorkFlow in Organisationseinheiten (Abteilungen) unterteilt. Jeder Abteilung können dann Benutzer zugeordnet werden. Ein Benutzer kann auch mehreren Abteilungen angehören, von denen eine aber als primär gilt. Rollen kommen nur insofern ins Spiel, als daß für eine Abteilung ein Leiter, ein Empfänger und ein Versender definiert werden. Jeder Benutzer kann zwar mit einer Funktion und einem Titel ausgestattet werden, diese haben aber keinen Einfluß auf die Auswahl zur Bearbeitung.

Ein Akt wird in WorkFlow in unterschiedliche Arten eingeteilt. So gibt es Eingangsstücke (in elektronischer oder Papierform) und Schriftstücke. Erstere veranlassen im Normalfall das Anlegen eines Geschäftsfalles, zweitere existieren nur innerhalb dieses GF. Ein Geschäftsfall ist mit einem Workflow in TriGS$_{flow}$ gleichzusetzen. In TriGS$_{flow}$ existiert nur ein Akttyp innerhalb eines Workflows. Er kann aber alle verwendeten Schriftstücke umfassen.

Ein Geschäftsfall wird über Sachgebiete einem Vorgang zugeordnet und erhält somit dessen Ablaufstruktur. Einem Vorgang wird ein Statusgraph *und* ein Leitweg zugeordnet. Der Graph beschreibt die möglichen Statusübergänge und die dabei verwendeten Aktionen. Im Leitweg werden die Statusübergänge in eine Reihenfolge gebracht, und die Weiterleitung an eine andere Abteilung oder einen anderen Benutzer kann bei jedem Übergang definiert werden. Diese Definition muß nicht erfolgen, da bei der aktuellen Weiterleitung nachgefragt wird, wer als nächster Bearbeiter in Frage kommt. Ist eine — in TriGS$_{flow}$ als Organisationspolitk bezeichnete — Zuordnung definiert, so kann der nachfolgende Benutzer immer noch geändert werden.

Daraus ist zu erkennen, daß bei der Formulierung der Zuordnung von Organisation und Aktivitäten einige Unterschiede bestehen. Obwohl die Benutzung durch den manuellen Eingriff bei der Weiterleitung sehr komfortabel ist, werden die Aktivitäten dennoch direkt einem Benutzer, bestenfalls einer Organisationseinheit zugeteilt. TriGS$_{flow}$ verwendet hingegen das Konzept der Organisationspolitiken [Bußl94], die nicht nur in der Organisationshierarchie eine flexible Zuordnung ermöglichen, sondern auch kontextbezogene Politiken enthalten, was in WorkFlow nicht möglich ist. Die Wahl eines anderen Benutzers aufgrund spezieller Datenkonstellationen kann in WorkFlow nur über manuelle Auswahl des Nachfolgers erfolgen.

Zuletzt ist noch die Flexibilität in der Vergabe von Rechten zu erwähnen. WorkFlow erlaubt für nahezu alle Tätigkeiten im System die Vergabe von Rechten. TriGS$_{flow}$ unterstützt zur Zeit Zugriffs- und Ausführungsbeschränkungen über Organisationspolitiken, die für jede Aktivität, aber auch für den Start eines Workflows definiert werden können. WorkFlow erweitert die Rechtevergabe um Maskenrechte, die bei TriGS$_{flow}$ wegen fehlender Benutzeroberfläche natürlich noch nicht vorgesehen sind.

6.3 Resümee der Gegenüberstellung

WorkFlow kann gewiß in Betrieben zur Unterstützung der Geschäftsfallbearbeitung oder Verwaltung eingesetzt werden. Der Einsatz eines solchen Systems erfordert aber große Sorgfalt in der Definition der Komponenten, denn die Vielfalt und Flexibilität kann leicht verwirren. So gibt es Zustandsgraphen und Leitwege, die alle mit der Abarbeitungsreihenfolge zu tun haben. Die Vielfalt der Rechtevergabe an Benutzer läßt auch großen Spielraum. Zusätze, wie ein integriertes Mailingsystem, erweitern auch den Einsatzbereich. WorkFlow kann auch über mehrere Knoten in einem WAN verteilt sein. Ein einmal angelegter Knoten kann aber nie mehr gelöscht werden, weil die Verteilung hier noch einige Probleme offen läßt. Vorgänge und Stati dürfen auch nur dann entfernt werden, wenn sie noch nicht gesichert sind. Damit geht man großen Schwierigkeiten aus dem Weg.

WorkFlow bietet zwar umfangreiche Benutzer- und Installationshandbücher an, aber eine Anleitung zu methodischem Vorgehen beim Einsatz des Systems ist leider nicht vorhanden. Dieser Umstand erschwert es, das Workflow System zu bedienen, denn die Angaben und Hinweise zur Modellierung sind immer in der Beschreibung der Benutzeroberfläche versteckt oder überhaupt nicht vorhanden. Eine optimale Einrichtung bzw. Anpassung des Systems ohne Vorkenntnisse auf dem Gebiet des Workflow Managements ist deshalb schwierig.

Eine zentrale Forderung der Workflow Management Coalition ist die Kommunikation von Workflow Systemen verschiedener Hersteller. In WorkFlow werden zwar Schnittstellen durch die PDI-Module (Program & Data Interface) angeboten, sie zielen aber auf die Kommunikation mit externen Applikationen ab und nicht mit anderen Workflow Systemen. Die Einbindung eines anderen Workflow Systems wird nicht erwähnt. Daraus läßt sich ersehen, daß auch die Umsetzung der vorgegebenen Anforderungen der Workflow Management Coalition noch einige Zeit in Anspruch nehmen wird.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Bevor nocheinmal ein kurzer Überblick über diese Arbeit gegeben wird, soll zunächst das am Anfang gesteckte Ziel noch einmal in Erinnerung gerufen werden. Es sollte ein Prototyp eines Workflow Systems auf Basis eines aktiven objektorientierten Datenbanksystems entwickelt werden. Neben dem Studium bestehender Ansätze und der Aufstellung der Grundanforderungen in bezug auf Modellierung und Ablauf sollten die Stärken und Schwächen herausgearbeitet werden. Der Anwendungsbereich wurde dabei auf den Büro- und Verwaltungsbereich beschränkt.

Der Einsatz von *ECA-Regeln* der aktiven Komponente TriGS im Workflow System bildet den Hauptbestandteil der Arbeit, was auch im Namen des Prototyps, TriGS$_{flow}$, ausgedrückt wird. Wenngleich deren Einbindung mit einigen Problemen verbunden war (siehe Anhang B), ermöglichten sie eine relativ flexible Gestaltung des Systems, die wahrscheinlich ohne Regeln nicht in der Form möglich gewesen wäre.

Die Einbindung eines *Rollenkonzepts* ist eine Erweiterung des Prototyps. Neben den ECA-Regeln stellt das Rollenkonzept einen weiteren Schwerpunkt des Prototyps dar. Denn ohne Rollen und Gruppen wäre die jetzt gegebene Flexibilität in der Zuordnung von Agenten und Aktivitäten nicht möglich gewesen.

Die konzeptuelle Gestaltung des Prototyps teilt sich in zwei Teile: Modellierung und Ablauf. Die *Modellierung* umfaßt die Definition von Organisation, Aktivitäten und Daten und deren Integration mittels Organisationspolitiken und Datenflußbeschreibung. Der *Ablauf*, auch „Enactment“ genannt, umfaßt im wesentlichen zwei Phasen: die Ausführung der Aktion und die anschließende Arbeitsverteilung auf die Nachfolgeaktivitäten, die sich wiederum in Auswahl, Zuordnung und Benachrichtigung der Agenten unterteilt. Die wesentlichen Komponenten des Workflow System Prototyps sind, als Teile der Organisation, die Agenten (menschliche Agenten und Software Agenten), Rollen und Gruppen (Abteilungen), zur Datenhaltung die Akte bzw. Akttypen und zur Steuerung des Ablaufs die Aktivitäten. Der Workflow Typ faßt alle drei Grundkomponenten zusammen. Ein Workflow ist eine konkrete Instanz eines Workflow Typs. Die Umsetzung des Ablaufs von Aktivitäten wird durch die Kontrollstrukturen ermöglicht, deren Grundlage die Ausführungsbedingungen der Aktivitäten bilden.

Nachdem ein Workflow System auf größtmögliche Flexibilität ausgerichtet ist, darf aber eines nicht außer Acht gelassen werden: die elektronische Unterstützung von Arbeitsabläufen ergibt eine Unmenge an anfallenden, elektronisch verarbeitbaren Daten. Dabei handelt es sich nicht nur um sachbezogene Daten, sondern auch um Daten rund um die Arbeit des Personals. Es werden Daten protokolliert, die jederzeit Rückschlüsse auf den Arbeitsrhythmus, die Geschwindigkeit, und die Qualität der Arbeit eines Mitarbeiters zulassen. Der Umgang mit diesen Daten sollte äußerst vorsichtig gehandhabt werden. Vor allem sollte die Frage geklärt werden, welche Daten überhaupt gesammelt werden dürfen, um die Wahrscheinlichkeit eines Mißbrauchs gering zu halten.

Zur Arbeit selbst muß noch hinzugefügt werden, daß viele der Anregungen und Erkenntnisse aus den im Literaturverzeichnis angeführten Quellen stammen. Das Literaturstudium hat vor allem am Anfang der konzeptuellen Entwicklung viel Zeit in Anspruch genommen, viele neue interessante Aspekte erzeugt, aber auch viele Unklarheiten und (noch) nicht lösbare Probleme mit sich gebracht. Im Anschluß werden die noch nicht gelösten Probleme und die notwendigen Erweiterungen des Prototyps besprochen.

7.2 Ausblick

Einige Teile des konzeptuellen Entwurfes des Workflow Systems konnten bereits erfolgreich umgesetzt werden. Viele Aspekte mußten jedoch teils aus zeitlichen Gründen, teils aber auch aus rein technischen Gründen weggelassen werden. Die Implementierung aller vorhandenen bzw. konzipierten Ideen hätte den Rahmen der Arbeit gesprengt, manche ließen sich unter den gegebenen Vorraussetzungen (Hard- bzw. Software) nicht umsetzen. Übrig bleibt eine lange Liste an Verbesserungsvorschlägen und Erweiterungen.

Der Auslöser für viele Probleme ist das dem Datenbanksystem GemStone zugrundeliegende Transaktionskonzept mit „flachen“ Transaktionen. Eine Erweiterung um geschachtelte und parallele Transaktionen ermöglicht eine angemessenere Ausnahmebehandlung (Exception Handling) und die Erweiterung der Kontrollstrukturen um die Oder-Zusammenführung. Damit könnten alle Kanten gewichtet werden, und es müßten nicht mehr die Ausführungsbedingungen der Nachfolgeaktivitäten dazu herangezogen werden. Mit einem geeigneten Transaktionskonzept wäre auch ein angemessenes Recovery durchzuführen. Auch die Agentenzuordnung (Agent Binding) könnte flexibler gestaltet werden, weil auch mehrere Agenten ein und dieselbe Aufgabe ausführen können, und nur die Ergebnisse jenes Agenten übernommen werden, der als erster fertig wird. Dabei müssen die Änderungen aller anderen Agenten verworfen werden, was nur möglich ist, wenn sie in verschiedenen Transaktionen ablaufen.

Mit einer Graphanalyse kann das Aktivitätennetz auf seine Gesetzmäßigkeit und Deadlockfreiheit überprüft werden, was insbesondere auf die Abbruchbedingungen von Schleifen zutrifft.

Eine Erweiterung der Ablaufsteuerung ist ebenfalls notwendig. So muß der Empfang und das Weiterleiten von Aufgaben flexibler gestaltet werden. Ein Benutzer soll den Empfang einer Aufgabe ablehnen und die Weitergabe beeinflussen können. Ein weiterer Punkt ist, daß der Agent eine Aufgabe für eine bestimmte Zeit hinterlegen kann, und nach Ablauf der Zeitspanne diese Aufgabe wieder in seiner Arbeitsliste erscheint. Dazu ist eine Erweiterung der Ereignisarten von TriGS um Zeitereignisse von Vorteil. Zeitereignisse können auch dazu verwendet werden, um die Reihenfolge der Abarbeitung in der Arbeitsliste zu verändern. Die Vergabe von Prioritäten oder die Beschränkung der Ausführungszeit verbessert die Einsatzmöglichkeiten. Eine zeitliche Beschränkung kann verschiedene Formen haben: Eine Möglichkeit wäre, einen Zeitpunkt festzusetzen, an dem die Arbeit spätestens begonnen oder beendet werden muß (absolut); eine andere Variante läge darin, die maximale Dauer der Bearbeitung zu begrenzen (relativ). Von Vorteil wäre auch die Information des Agenten über den bisherigen und weiteren Verlauf eines Aktes. Er soll vor der Weitergabe den Empfänger des Aktes kennen und diesem auch Notizen mitgeben können. Weiters ist die im Programmpaket WorkFlow eingeführte Vertretung eines anderen Benutzers sinnvoll. Die Vertretung führt aber zu einer dynamischen Änderung der Arbeitslisten zweier Agenten, da die Aufgaben des einen zum anderen transferiert werden müssen.

Eine kleine Verbesserung bei der Rollendefinition in der Organisation wäre auch noch wünschenswert. Sie müßte aber am Rollenkonzept selbst vorgenommen werden. Es stellte sich heraus, daß vor Anlegen einer Rolle immer alle „Superrollen“ definiert werden müssen, und auch bei der Vergabe einer Rolle ein Objekt immer zuerst in die Superrolle schlüpfen muß. Es wäre komfortabel, wenn durch Anlegen einer Rolle für ein Objekt automatisch alle noch nicht vorhandenen Superrollen erzeugt werden würden. Der Bereich der Organisationspolitiken kann erweitert werden, indem man neue Zuordnungsalgorithmen (Agent Binding) entwirft. So soll eine Aufgabe gleich mehreren Agenten geschickt werden, und entweder nur die Ergebnisse des Agenten übernommen werden, der als erster fertig ist, oder nur jener arbeiten, der als erster die Aufgabe auswählt. Die Aufgabe muß dabei von den Arbeitslisten der anderen Agenten eingezogen werden.

Der Zugriff auf die Daten eines Aktes erfordert zur Zeit die Sperre des ganzen Aktes, weil GemStone zwar das Sperren eines Objekts erlaubt, aber nicht das Sperren einzelner Referenzen. Es besteht zur Zeit nur die Möglichkeit, die Referenzsperre selbst zu implementieren, um schon jetzt Aktivitäten verschiedener

Zweige des Aktivitätennetzes parallel ausführen zu können.

Die Einbindung externer Applikationen konnte in diesem Prototyp noch nicht verwirklicht werden, da GemStone keine geeigneten Sprachkonstrukte in OPAL dafür vorsieht. Die einzige vorstellbare Lösung liegt in der Verwendung der C-Schnittstelle. Diese Alternative wurde aber noch nicht untersucht.

Als Ergänzung ist auch die Einführung von Zugriffsrechten aufzunehmen. In WorkFlow wurden bereits verschiedene Kategorien von Zugriffsrechten (Verschlußrechte, Maskenrechte usw.) in verschiedenen Stufen eingesetzt, um den Mißbrauch der Daten in Schranken zu halten.

Ein weiterer nicht berücksichtigter Einflußfaktor im Workflow System ist die Performanz. Der Anwendungsbereich wurde von vornherein auf den Büro- und Verwaltungsbereich beschränkt, da bei der Einbindung von Produktionsprozessen die Performanz eine wesentlich größere Rolle spielt. Zusätzliche Probleme wie Echtzeitverarbeitung mußten daher nicht berücksichtigt werden. Eine Erweiterung des Anwendungsgebietes stellt natürlich ein große Herausforderung dar. Außerdem wird von der Workflow Management Coalition die Kooperation mit anderen Workflow Systemen und im besonderen auch die Verteilung des Systems auf mehrere Knoten in einem globalen Netz gefordert.

Zu guter Letzt verbleibt noch der wohl wichtigste Aspekt neben der Erweiterung des Transaktionskonzepts und der vielen kleineren Ergänzungen: die Schaffung der Benutzeroberflächen AgentTOOL und MasterTOOL. Wie bei der Modellierung des Beispiels in den vorangegangenen Kapiteln schon zu erkennen war, ist eine Verwendung des Prototyps ohne diese Oberflächen unzumutbar. Nachdem das Konzept des Prototyps überarbeitet und die wichtigsten Änderungen gemacht sind, ist die Entwicklung einer Oberfläche die wohl dringendste Weiterentwicklung. Werkzeuge zur Simulation von Workflows und die Entwicklung einer Workflow Definition Language gelten als weitere nützliche Erweiterungen.

Anmerkungen und Verbesserungsvorschläge zur aktiven Komponente TriGS und zum Rollenkonzept befinden sich in den Anhängen B und C.

Anhang A

Dokumentation der Objektklassen in TriGS$_{flow}$

A.1 Vorraussetzungen

Ein wichtiger Teil sind die Klassen des Rollenkonzepts. Die Ergänzungen `Role` und `QualifiedRole` erleichtern das Anlegen der Rollentypen.

TriGS selbst besteht aus einer großen Menge von Klassen. Bei deren Verwendung ist zu beachten, daß die vorhandenen Parametermethoden der Klasse `MethodCall` und die Query-Methoden der Klasse `Condition` von den installierten Regeln (Aktionen und Bedingungen) abhängen.

A.2 Zusammenfassung der Klassen in Kategorien

Die Klassen und Objekte der TriGS$_{flow}$-Umgebung und die TriGS$_{flow}$-Klassen selbst werden in folgende Kategorien eingeteilt:

`Roles`	Klassen des Rollenkonzepts inklusive `Role` und `QualifiedRole`
`TriGS`	TriGS-Klassen
`WFBases`	Speicherung der Betriebe in der Instanz `WFCompanyBase` der Klasse `CompanyBase` sowie der Workflow Typen in der Instanz `WFWorkflowBase` der Klasse `WorkflowBase`
`WFRules`	Regelbasen `WFActionBase`, `WFConditionBase`, `WFEventSelectorBase`, `WFRuleBase` und `WFRuleScheduler`
`WFGroups`	anwendungsspezifische Gruppenklassen
`WFRoles`	anwendungsspezifische Rollenklassen

`WFObjects` TriGS$_{flow}$-Basisklassen

Die Einteilung in diese Kategorien ist zwar nicht zwingend, wird aber wegen der besseren Überschaubarkeit empfohlen.

A.3 Übersicht über die Klassen

Die Klassen werden in einer Klassenhierarchie und zugleich in einer Klassenkompositionshierarchie dargestellt. Um eine Überladung der ohnehin immer noch sehr komplexen Darstellung zu vermeiden, wird diese in zwei Teile geteilt. Die Abbildung A.1 zeigt die TriGS$_{flow}$-Struktur ohne das Rollenkonzept. Die Verbindung von Rollentypen und TriGS$_{flow}$ wird erst in der Abbildung A.2 dargestellt. Die grau unterlegten Klassen sind anwendungsspezifische Klassen. In beiden Grafiken werden außerdem nur die wichtigsten Instanzvariablen angezeigt. Sie sollen die Zusammenhänge der Klassen veranschaulichen. Auf Attribute wie ID oder Name sowie auf Methoden wird aus Gründen der Übersichtlichkeit verzichtet.

In keiner der angegebenen Hierarchien scheint eine TriGS-Klasse auf, weil sie von den übrigen Klassen völlig abgekoppelt sind, und außerdem keine neuen Klassen zu TriGS hinzugefügt wurden.

Die Abbildung A.3 zeigt noch einmal die gesamte Hierarchie der entwickelten Klassen in einer Darstellungsform, wie sie in GemStone üblich ist. Die in normaler Schrift abgebildeten Klassen bezeichnen die GemStone Basisklassen, erweitert um die rollenspezifischen Klassen, und die **fett** gedruckten Klassenbezeichner die entwickelten TriGS$_{flow}$-Klassen. Die *kursiv* dargestellten Klassen bezeichnen anwendungsspezifische Klassen.

Der hierarchischen Gliederung der Aktivitätenklassen liegen folgende Überlegungen zugrunde: Obwohl die Information über Nachfolgerbeziehungen vollständig in Regeln abgelegt wird, müssen gewisse Vorsichtsmaßnahmen getroffen werden. Es soll verhindert werden, daß eine Endeaktivität und eine Ausnahmeaktivität Nachfolger haben können. Deshalb haben nur die Startaktivität und die Innere Aktivität die Möglichkeit, Nachfolger anzulegen und diese zu suchen. Als abstrakte Superklasse der beiden letztgenannten Aktivitäten fungiert `ActivityWithSucc`. Die Klasse `InnerActivity` hat zusätzlich noch die Möglichkeit, Vorgänger zu bestimmen, d.h. sowohl die übliche Vorgänger-Nachfolger-Beziehung anzulegen, als auch Vorgänger zu suchen. Die Start-, Ende- und Ausnahmeaktivitäten setzen ID und Namen einer Instanz bereits bei deren Erzeugung auf die Standardwerte.

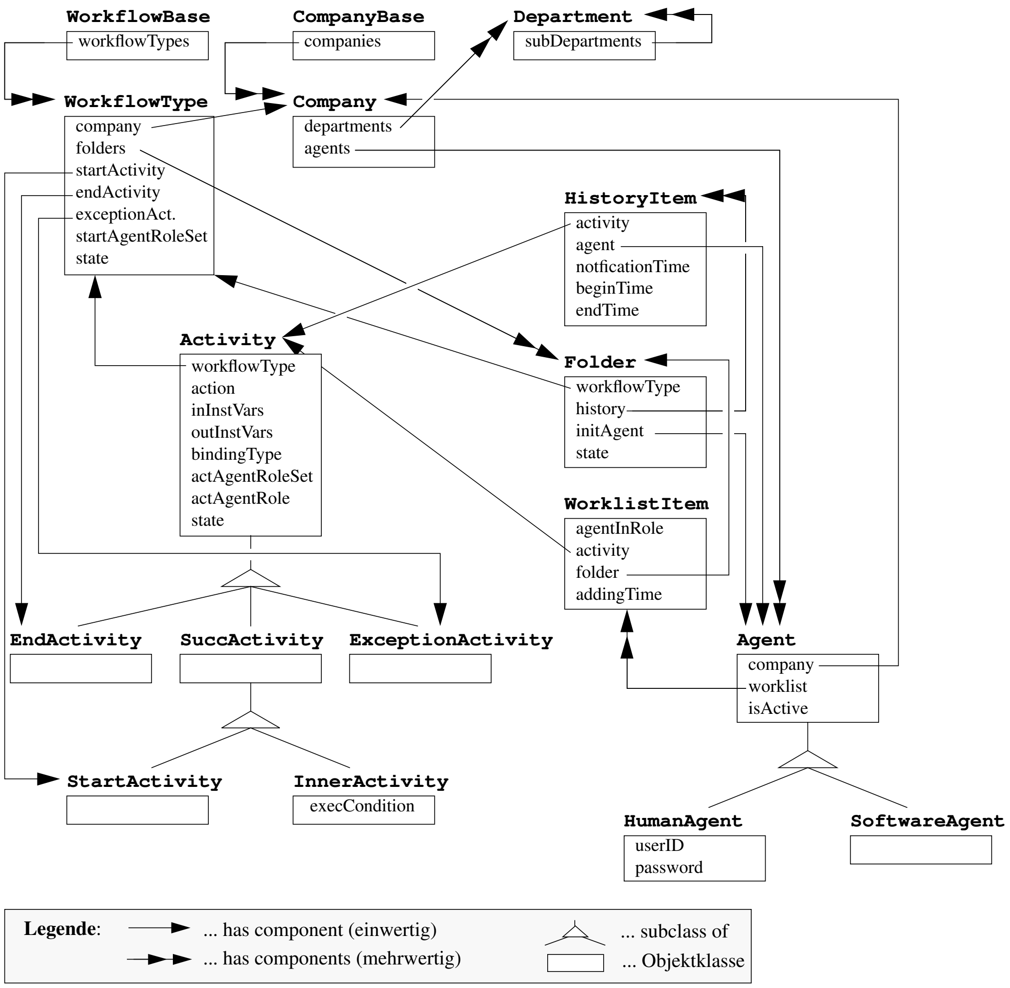


Abbildung A.1: Klassenhierarchie und Klassenkompositionshierarchie von TriGS$_{flow}$ ohne Rollentypen

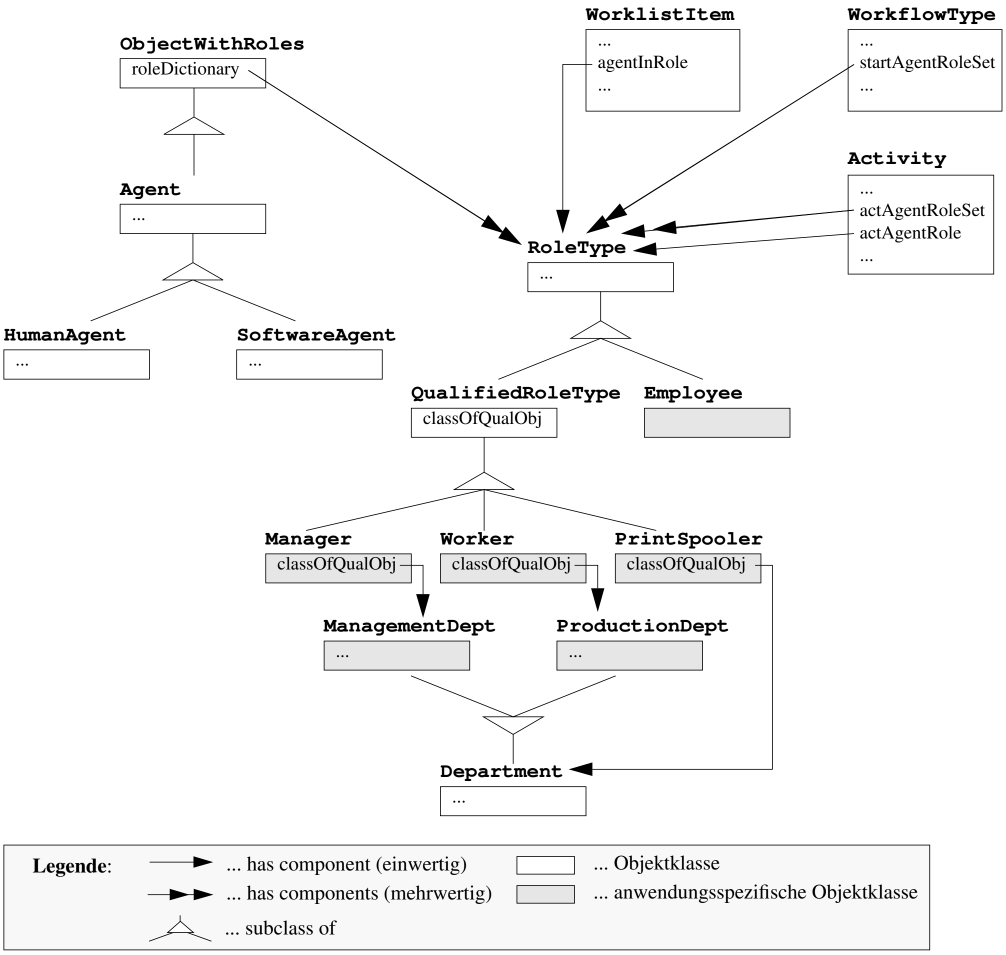


Abbildung A.2: Rollentypen und TriGS$_{flow}$

Object
    Activity
        EndActivity
        ExceptionActivity
        SuccActivity
            InnerActivity
            StartActivity
    Company
    Department
        ManagementDept
    Dictionary
        CompanyBase
        WorkflowBase
    Folder
        StudentFolder
    HistoryItem
    ObjectWithRoles
        Agent
            HumanAgent
            SoftwareAgent
    OrgPolicy
    RoleType
        Employee
        QualifiedRoleType
            Worker
    WorkflowType
    WorklistItem

Abbildung A.3: Klassenhierarchie von TriGS$_{flow}$

A.4 Beschreibung der wichtigsten Instanzvariablen und Methoden

Aus Gründen der Übersichtlichkeit werden an dieser Stelle nicht alle Instanzvariablen und Methoden aller Klassen von TriGS$_{flow}$ vorgestellt, sondern es werden hier nur die für das Verständnis der Ausführung relevanten Teile angeführt. Die Bedeutung der anderen Methoden geht aus der Prototypbeschreibung in Kapitel 4 und der Beispielmodellierung in Kapitel 5 hervor. Die Beziehungen der Instanzvariablen zu den Klassen sind in den Abbildungen A.1 und A.2 dargestellt.

A.4.1 Klasse `Agent`

Instanzvariablen

`worklist`
: Arbeitsliste, enthält Einträge vom Typ `WorklistItem`;

`isActive`
: besagt, ob der Agent aktiviert ist;

Methoden

`executeElem: anElem`
: führt den Eintrag `anElem` aus und initiiert anschließend die Verteilung;

`startWorkflow: anId WithFolder: aFolder`
: versucht einen Workflow vom Typ `anId` mit dem Akt `aFolder` zu starten;

`performAction: theAction withArguments: inputArgs by: aRole`
: Die mitgegebene Rolle wird zur Ausführung der angegebenen Aktion aufgefordert. Zwei der Historienregeln sind darauf definiert.

`addToWorklist: anActivity folder: aFolder role: aRole`
: fügt der Arbeitsliste einen Eintrag hinzu; eine Historienregel (Benachrichtigung) hat diese Methode als Ereignis.

`removeFromWorklist: anItem`
: entfernt einen Eintrag aus der Arbeitsliste und ist zugleich das Ereignis zur Ausführung eines weiteren Eintrags bei einem Software Agenten;

A.4.2 Klasse `HumanAgent`

Methoden

`selectElem: aNr`	wählt den Eintrag mit der Nummer `aNr` aus der Arbeitsliste aus und bringt ihn mit `executeElem:` zur Ausführung;

A.4.3 Klasse `SoftwareAgent`

Methoden

`selectElem`	wählt den ersten Eintrag aus der Arbeitsliste aus und bringt ihn mit `executeElem:` zur Ausführung;

A.4.4 Klasse `Activity`

Instanzvariablen

`action`	Name einer Methode, die bei einem Rollentyp definiert ist und in Verbindung mit der Aktivität auszuführen ist
`inInstVars`	Attribute des Aktes, die zu Beginn der Aktionsausführung gelesen werden
`outInstVars`	Attribute des Aktes, die am Ende der Aktionsausführung geschrieben werden
`bindingType`	Art der Zuordnung der Agenten (z.B. `#less` für jenen Agenten, der am wenigsten Einträge in der Arbeitsliste hat)
`actAgentRoleSet`	enthält alle Agentenrollen, die die Organisationspolitik erfüllen; wird für jede Auswertung neu gesetzt, nicht aber für jede Alternative;
`actAgentRole`	jener Agent, der entsprechend dem `bindingType` zugeordnet wird
`state`	enhält `#executable` im Normalfall, `#paused` wenn gestoppt, `#exception` bei einer Ausnahme;

Methoden

`distribute: aFolder` auslösendes Ereignis für die Verteilung; die Methode selbst ruft nur die Ende- oder Ausnahmeaktivität auf;

`notifyAgent: aFolder` benachrichtigt den zugeordneten Agenten mit dem Akt `aFolder`;

`restart` startet erneut die pausierte Aktivität;

`bindAgentWithLessWorklistSize`
ordnet von den Agenten, die die Organisationspolitik erfüllen, jenen der Ausführung zu, dessen Arbeitsliste die wenigsten Einträge hat;

A.4.5 Klasse `ActivityWithSucc`

Methoden

`succActivities` liefert nur die direkten (unmittelbaren) Nachfolgeaktivitäten;

`allSuccActivities` liefert alle Nachfolger, d.h. sowohl direkte als auch indirekte Nachfolgeaktivitäten;

A.4.6 Klasse `InnerActivity`

Instanzvariablen

`execCondition` enthält die Ausführungsbedingung, die bei der Regelerzeugung für die Nachfolgerbeziehungen bei allen Vorgängern verwendet wird;

Methoden

`combine: aFolder fromActivity: anActivity`
sammelt alle Vorgängerpfade und wartet solange, bis der letzte beendet ist (d.h. diese Methode aufruft), und benachrichtigt anschließend den zugeordneten Agenten mit `notifyAgent: aFolder`;

`predActivities` liefert nur die direkten (unmittelbaren) Vorgängeraktivitäten;

`allPredActivities` liefert alle Vorgänger, d.h. sowohl direkte als auch indirekte;

A.4.7 Klasse `WorkflowType`

Instanzvariablen

`folderType`	beschreibt den Typ, den ein zu verarbeitender Akt haben muß
`folders`	enthält alle bereits aufgenommenen Akte
`startActivity`	Startaktivität
`endActivity`	Endeaktivität
`exceptionActivity`	Ausnahmeaktivität
`startAgentRoleSet`	enthält alle Agentenrollen, die die Organisationspolitik zur Starterlaubnis erfüllen; wird für jeden Start neu belegt;
`state`	enhält den Ausführungszustand:

`#started`	nach dem Start,
`#processed`	während des Durchlaufs,
`#paused`	wenn gestoppt,
`#exception`	bei einer Ausnahme;

Methoden

`isAllowedToStart: anAgent withFolder: aFolder`
stellt fest, ob der Agent die Erlaubnis hat, den Workflow zu starten, indem die Organisationspolitik durch eine davor ausgeführte Regel ausgewertet wird;

`startBy: anAgent withFolder: aFolder`
startet einen Durchlauf, wenn der Agent die Erlaubnis hat, und noch kein Akt mit derselben ID vorhanden ist;

A.4.8 Klasse `WorklistItem`

Instanzvariablen

`agentInRole`	in eine Rolle geschlüpfter Agent, der die Aktivität zu bearbeiten hat
`activity`	auszuführende Aktivität
`folder`	der dabei zu bearbeitende Akt

`addingTime`	Zeitpunkt, zu dem der Eintrag zur Arbeitsliste hinzugefügt wird

A.4.9 Klasse `Folder`

Instanzvariablen

`history`	Menge aller Historieneinträge
`initAgent`	Agent, der den Workflow mit diesem Folder gestartet hat

A.4.10 Klasse `HistoryItem`

Instanzvariablen

`activity`	bearbeitete Aktivität
`agent`	Agent, der die Aktivität bearbeitet hat
`notificationTime`	Zeitpunkt der Benachrichtigung
`beginTime`	Zeitpunkt des Ausführungbeginns
`endTime`	Zeitpunkt des Ausführungendes

A.5 Klassen und Methoden, die TriGS verwenden

Um einen Überblick zu schaffen, welche Methoden in TriGS$_{flow}$ konkret mit TriGS interagieren, werden anschließend die Klassen mit deren Methoden aufgelistet. Methoden, die Ereignisse für Regeln darstellen, sind nicht enthalten. Regeln, deren Namen mit einem Suffix enden, also aktivitäten- oder workflowspezifisch sind, werden mit drei Punkten am Ende des Namens gekennzeichnet.

Klasse `Activity`

`orgPolicy:cond:sel:`	erzeugt eine Regel `R_OrgPolicy_...`
`orgPolicyBind:`	erzeugt eine Regel `R_OrgPolBinding_...`
`removeOrgPolicy:`	entfernt eine Regel `R_OrgPolicy_...`
`removeOrgPolicyBind`	entfernt eine Regel `R_OrgPolBinding_...`

Klasse `ActivityWithSucc`

`succActivities`	ermittelt anhand der Nachfolger-Regeln `R_ActivitySuccession_...` die unmittelbaren Nachfolger

`newSuccessor:`	erzeugt eine Nachfolger-Regel `R_Activity-Succession_...`
`removeSuccessor:`	entfernt eine Nachfolger-Regel `R_Activity-Succession_...`

Klasse `InnerActivity`

`predActivities`	ermittelt anhand der Nachfolger-Regeln `R_-ActivitySuccession_...` die unmittelbaren Vorgänger
`becomeSuccessorOf:`	erzeugt eine Nachfolger-Regel `R_Activity-Succession_...`
`removeSuccessorTo:`	entfernt eine Nachfolger-Regel `R_Activity-Succession_...`

Klasse `WorkflowType`

`startOrgPolicy:cond:sel`	erzeugt eine Regel `R_WFStartOrgPolicy_...` zur Auswertung der Organisationspolitik für den Workflow Start
`removeStartOrgPolicy:`	entfernt dieselbe

Diese Methoden greifen direkt auf die TriGS-Komponenten zu. Einige andere Methoden (vor allem zum Löschen der Regeln) bedienen sich dieser angegebenen Methoden.

A.6 Methoden, die Ereignisse für TriGS darstellen

Hier werden Methoden aufgelistet, die in TriGS$_{flow}$ umhüllt sind und damit für TriGS Ereignisse darstellen. Bei jeder Methode sind die Namen jener Regeln angeführt, die auf diese Ereignisse reagieren. Der Triggerzeitpunkt (*pre* oder *post*) wird ebenfalls angegeben. Regeln, deren Namen mit einem Suffix enden, also aktivitäten- oder workflowspezifisch sind, werden mit drei Punkten am Ende des Namens gekennzeichnet.

Klasse `Activity`

`distribute:`	pre `R_ActivitySuccession_...`
`notifyAgent:`	pre `R_OrgPolicyInit` pre `R_OrgPolBinding_...` pre `R_OrgPolicy_...`

Klasse `Agent`

`performAction:withArguments:by:`	pre `R_HistoryBeginExec` post `R_HistoryEndExec`

`addToWorklist:folder:role:`	
	post `R_HistoryNotification`
`activate`	post `R_RestartWFOnAgentActivation`

Klasse `SoftwareAgent`

`addToWorklist:folder:role:`	
	pre `R_StartOnInsert` (Bedingungsauswertung und Aktionsausführung *post*!)
`removeFromWorklist:`	post `R_StartOnRemove`

Klasse `WorkflowType`

`isAllowedToStart:withFolder:`	
	pre `R_StartWFOrgPolicyInit`
	pre `R_StartWFOrgPolicy`

Anhang B

TriGS — Anmerkungen, Änderungen, Erweiterungen

Die aktive Erweiterung des objektorientierten Datenbanksystems GemStone wurde im Rahmen eines ESPRIT-Projektes entwickelt. Die Konzepte werden in [Kapp94a] und [Kapp94b] vorgestellt. Die erste Implementierung wurde im Rahmen einer Diplomarbeit durchgeführt und ist in [Aich94] beschrieben. Die Benutzung von TriGS hat zum Großteil fehlerfrei funktioniert. Dennoch sind bei der Einbindung und Verwendung einige Probleme aufgetreten, die in der Folge kurz angeführt werden. Wo es möglich erschien, wurde auch ein Lösungsansatz angegeben. Änderungen an TriGS und der Benutzerschnittstelle werden in B.2 und B.3 dargestellt. Einerseits handelt es sich dabei um einige wenige Fehlerbehebungen, andererseits um Abänderung oder Hinzufügung einzelner Methoden, um die in B.4 aufgezeigten nützlichen Erweiterungen zu ermöglichen.

B.1 Anmerkungen

Umhüllen („Wrapping“) und RuleScheduler

Es ist nicht möglich, ein Event für mehrere RuleScheduler zu verwenden, ohne die Event-Messages (umhüllte Originalmethoden) nachträglich, d.h. nach der Umhüllung durch die Wrap-Funktion der Benutzerschnittstelle, zu ändern. Denn diese Funktion der Benutzerschnittstelle schickt `raiseEvent:` immer nur einem bestimmten RuleScheduler. Will man mehreren RuleSchedulern das Event schicken, so müssen diese von Hand in die entsprechende Funktion eingebracht und diesen die Methode `raiseEvent:` geschickt werden. Abhilfe würde ein Verwaltungsobjekt schaffen, das alle RuleScheduler kennt, und allen oder bestimmten daraus das anstehende Ereignis schickt.

Klasse `MySystem`

Die Klasse `MySystem` enthält die Methoden `commitTransaction` und `abortTransaction`. Leider sind diese Methoden nicht von vornherein umhüllt. Die Ursache dafür ist, daß mithilfe der Wrap-Funktion der Benutzerschnittstelle das Ereignis nur einem RuleScheduler geschickt werden kann, der explizit anzugeben ist. Da immer nur ein bestimmter RuleScheduler angesprochen wird, kann man diesen nicht im vorhinein festlegen. Die Lösung ist die gleiche wie oben beschrieben.

Fehler in Wrap-Funktion

Beim Wrappen der Klassenmethode `abortTransaction` der Klasse `MySystem` wird in der Wrapper-Methode der Returnwert der Methode `abortTransaction` als boolscher Wert behandelt. Tatsache ist, daß `abortTransaction` keinen boolschen Wert, sondern die Klasse `System` liefert. Abfrage auf `true` ist daher in der gewrappten Methode nicht möglich.

Erzeugung der Query- und Parameter-Methoden

Die bei Bedingungen angeführten `QueryPointer` werden als Methoden des Objekts `Condition` in die Kategorie `Queries` gespeichert. Da jede Bedingung als eine Methode abgelegt wird, gibt es sehr viele Methoden mit gleichem Methodenrumpf, wie z.B. für die immer wahre Bedingung `^true`.

Um die Parameter einer im Aktionsteil einer Regel als `MethodCall` definierten Methode verwenden zu können, müssen dafür von Hand Parameter-Methoden geschrieben werden. Diese werden in der Kategorie `parameterCode` des Objekts `MethodCall` abgelegt. Auch hier kann es wieder zu einer Unmenge von redundanten Parametermethoden kommen.

Beide Methodenspeicherungen lassen sich aber anscheinend nicht vermeiden, da es in OPAL keine Laufzeitauswertungen von Programm-Statements gibt.

Rekompilieren von Klassen

Wenn Klassen, auf deren Methoden Regeln bzw. Events definiert sind, im *GemStone System Browser* neu übersetzt werden, dann müssen auch die triggernden Klassen (`trgClass`) der Ereignisselektoren (`RuleEventSelector`) und auch die Empfänger (`receiver`) der MethodCalls der Aktionen, sofern es sich um Klassen handelt, neu eingegeben werden. Das Problem ist nämlich, daß eine Klasse als neues Objekt gespeichert wird, falls sie rekompiliert wird. Die Instanzvariablen `trgClass` und `receiver` verweisen dann auf die alten Klassen mit dem Prefix `old`.

Einige kurze OPAL-Statements beseitigen dieses Problem, indem sie alle Klassen der Events und die Receiver der Aktionen erneuern, falls es sich um Klassen handelt. Näher beschrieben wird dieses kleine Werkzeug in B.4.

Unwrapping

Es gibt leider keine Möglichkeit in der Benutzeroberfläche, Methoden „unzuwrappen“, d.h. die Umhüllung von Methoden zum Zwecke der Ereigniserkennung rückgängig zu machen. Es dürfte allerdings nicht allzuviel Aufwand darstellen, eine geeignete Methode dafür zu entwickeln, da lediglich die geänderte Methode gelöscht, und von der ursprünglichen Methode das Suffix `_ORIGINAL` entfernt werden muß.

Aktivierung und Deaktivierung einer Regel für bestimmte Objekte

Die Instanzvariablen `activatedFor` und `deactivatedFor` der Klasse `Rule` enthalten jene Objekte, für die eine Regel aktiviert bzw. deaktiviert ist. Objekte können mit den Methoden `addToActivatedFor:` bzw. `addToDeactivatedFor:` problemlos hinzugefügt werden. Auch zum Löschen sind die entsprechenden Methoden `removeFromActivatedFor:` bzw. `removeFromDeactivatedFor:` vorhanden. Dennoch kann ein Objekt, das man nicht kennt, nicht herausgelöscht werden, weil es nicht möglich ist, auf den Inhalt der Instanzvariablen zuzugreifen und somit die bereits gespeicherten herauszulesen. Um diesen Umstand zu beseitigen, müssen zwei Methoden geschaffen werden, die auf den Inhalt der Instanzvariablen zugreifen.

Definitionssprache

Die meisten Regelsysteme bieten eigene Sprachkonstrukte zur Erzeugung von Regeln an. Eine *RDL* (Rule Definition Language) hat den Vorteil, daß die Regeln in übersichtlicher Form aufgeschrieben und als Batch-Job auf einmal in die Datenbank eingebracht werden können, während das Erstellen von Regeln über die Benutzerschnittstelle einen erheblichen Aufwand darstellt. Ein weiterer Vorteil ist, daß sich die Regeln dann leicht auf andere Datenbanken transferieren lassen, wenn alle Regeln in Form einer Definitionssprache vorliegen. Falls in der Datenbank bereits geänderte Regeln in eine andere transferiert oder bloß extern gesichert werden sollen, dann empfiehlt es sich, eine Möglichkeit zu schaffen, um einen Auszug der Regeln in der Datenbank in Form der Definitionssprache machen zu können.

Da in TriGS ohnehin eine in der Backus-Naur-Form angegebene Grammatik der Regelbeschreibung vorliegt, muß „nur“ ein Übersetzer implementiert werden. Nach den Richtlinien des Übersetzerbaus vorgegangen, dürfte dieses Vorhaben keinen allzugroßen Aufwand darstellen. Denn es gibt Compiler-Compiler, die für eine angegebene Grammatik einen Compiler erzeugen. Es muß nicht notwendigerweise ein Smalltalk- oder OPAL-basierter Compiler sein. Da GemStone auch eine C-Schnittstelle anbietet, können GemStone-Objekte auch von einem in C geschriebenen Programm aus angelegt werden.

Auswertung der Regelbasis

Die Auswertung der Regeln in der Benutzerschnittstelle läßt etwas zu wünschen übrig, weil die dabei erzeugte Liste an Regeln unübersichtlich, klein und vor allem nicht sortiert ist. Regeln sind daher in der Übersicht nur schwer zu finden. Weiters ist der Ausdruck der Regelübersicht auf Papier wünschenswert. Ein kleines selbstgeschriebenes Hilfswerkzeug ermöglicht es nun, diese Übersicht in sortierter und etwas ansprechenderer Form auszugeben oder auch in eine Datei abzulegen. Kurz beschrieben wird das Werkzeug unter Punkt B.4.

Inkonsistenzen in der Namensgebung und Funktionalität

Zur weiteren Verbesserung der TriGS-Implementierung sollen auch die ohnehin nur selten auftretenden Inkonsistenzen in der Bezeichnerwahl von Methoden oder Instanzvariablen angeführt werden. Beispiele dafür sind sowohl in der Benutzerschnittstelle — also in VisualWorks/Smalltalk — als auch in den OPAL-Objekten zu finden.

Bsp. Visualworks/Smalltalk:

In den Klassen `Action`, `Condition` und `Event` wird die Methode zum Anzeigen der Übersicht `info` genannt. In der Klasse `Rule` hingegen wird diese mit `getInfo` bezeichnet.

Die Klassen `Action`, `Condition` und `Rule` sehen Methoden zur Änderung bereits gespeicherter Objekte vor. Die Klassen `EventSelector`, `CondActEventSelector` und `RuleEventSelector` hingegen lassen diese Methoden vermissen, wodurch auch keine Ereignisse geändert werden können.

Bsp. OPAL:

In den Klassen `Condition`, `Action`, `RuleEventSelector` und `CondActEventSelector` wird die Methode zur Erzeugung der Übersicht mit `makeStringArray` bezeichnet. Nur in der Klasse `Rule` heißt diese Methode `makeString`.

Umbenennung von Ereignissen und Regeln

Es ist schon schwierig und aufwendig genug, bei Bedingungen (`Condition`) und Aktionen (`Action`) den Namen zu ändern, bei Ereignissen `Event` und Regeln `Rule` ist dies aber gar nicht möglich, da keine Methoden dafür vorhanden sind. Es hat sich als geeignet erwiesen, Änderungen teils von der Benutzeroberfläche aus und teils von Hand durchzuführen. Von Hand bedeutet, über den OPAL

System Browser die Basen für Regeln, Ereignisselektoren, Bedingungen und Aktionen mit *inspect* durchzusehen. Ereignisse können von der Benutzeroberfläche aus leider überhaupt nicht geändert werden.

Prioritäten

Gleichzeitig auftretende Regeln werden ihrer Priorität nach abgearbeitet. Im Konzept von TriGS [Kapp94a] wird 0 als die höchste Prioritätsstufe angesehen, d.h. die Priorität fällt mit steigender Zahl. Die Implementierung geht aber genau umgekehrt vor: Je größer die Prioritätszahl, desto größer die Priorität. Aus [Aich94] geht nicht hervor, ob die Prioritäten auf- oder absteigend angesehen werden.

Aktionsteil bei negativer Bedingung

Manchmal ist es wünschenswert, statt nur einem Aktionsteil zwei zu haben, wobei der zweite dann ausgeführt wird, wenn die Bedingung nicht wahr ist. Zwar kann dies auch mit den herkömmlichen ECA-Regeln simuliert werden, indem einfach eine neue Regel mit der negativen Bedingung erstellt wird, doch scheint es einfacher zu sein, einen alternativen Aktionsteil zu führen.

Transaktionsgrenzen bei Regeldefinition

Mit der Methode `accept:` in der Kategorie `termination` aller Basen (`RuleBase`, `EventSelectorBase`,...) werden Änderungen oder neue Regeln in die Basen übernommen. Dabei wird auch immer ein „commit" durchgeführt. Wenn man, so wie in TriGS$_{flow}$, viele Regeln bei der Definition eines Workflow Typs automatisch anlegt, wird bei jedem erzeugten Regelteil in die Datenbank gespeichert. Dieser Umstand kostet sehr viel Zeit. Das Problem kann nur dadurch gelöst werden, daß die Aufrufe `System commitTransaction` entfernt werden.

Zeitereignisse und komplexe Ereignisse

Zur Zeit gelten nur Methodenaufrufe als Ereignisauslöser. Eine Erweiterung des Workflow System Prototyps in Richtung flexiblere Ablaufsteuerung machen aber auch eine Erweiterung der Ereignisse in TriGS notwendig. Es kann nützlich sein, eine Aufgabe für eine gewisse Zeit zu hinterlegen und erst nach Ablauf der Zeitspanne wieder vorzulegen. Diese Eigenschaft kann ganz bequem mit Regeln gelöst werden, wenn auch Zeitereignisse zur Verfügung stehen. Ansonsten müssen dazu eigene Prozesse gestartet werden, die die hinterlegten Aufgaben ständig überwachen. Die Erweiterung der Ereignisarten um komplexe Ereignisse würde die Flexibilität in der Verwendung von Regeln ebenfalls erhöhen.

B.2 Änderungen in OPAL

Neue Methoden für die Klasse `Condition`

Zur Reorganisation der `queryPointer` aller Bedingungen werden Methoden zum Lesen und Schreiben dieser Pointer benötigt. Neue Methoden:

```
Condition  Accessing  queryPointer
Condition  Updating   queryPointer:
```

Neue Methode für die Klasse `MySystem`: `beginTransaction`

Um die Konsistenz zu wahren, wurde die Klassenmethode `beginTransaction` hinzugefügt.

Neue Methoden und Klassen für sortierte Ausgabe

Um einen geeigneten Überblick über alle definierten Regeln, Ereignisselektoren, Bedingungen und Aktionen zu erhalten und diese eventuell auch in eine Datei abspeichern zu können, wurde ein großer String mit allen Informationen erzeugt. Als Vorlage für die neuen Methoden dienten `getInfo` und `makeStringArray` der Basen `RuleBase`, `EventSelectorBase`, `ConditionBase` und `ActionBase`. Analog zu `makeStringArray` gibt es jetzt zwei neue Methoden: `asString` und `asStringForFile`. Letztere unterscheidet sich nur in der Anzahl der Tabulatoren, da diese für die Ausgabe in eine Datei verwendet wird. Die Info-Methoden lauten analog: `getFormattedInfo` und `getFormattedInfoForFile`.

Die neuen Methoden werden wie folgt in den angeführten Klassen verwendet:

`getFormattedInfo` **und** `getFormattedInfoForFile`:
: `RuleBase`, `EventSelectorBase`, `ConditionBase`, `ActionBase`

`asString` **und** `asStringForFile`:
: `Rule`, `RuleEventSelector` , `CondActEventSelector`, `Condition`, `Action`, `MethodCall`

Außerdem mußten dafür die Klassen `EventSelectorSet`, `ConditionSet` und `ActionSet` angelegt werden. Die Klasse `RuleSet` war bereits in TriGS vorhanden.

B.3 Änderungen in VisualWorks/Smalltalk

In der Benutzeroberfläche wurde folgender kleiner Fehler ausgebessert: Die Klasse `ActionInterface` enthält unter anderem die zwei Methoden `changing`:

und `adding:`. Beim Aufruf dieser Methoden trat ein Konflikt mit der Klasse `OrderedCollection` auf. Die Parameter-Methoden eines Methodenaufrufes (`MethodCall`) einer Aktion werden in der als `OrderedCollection` definierten temporären Variable `paramList` abgelegt.

Die Klasse `OrderedCollection` ist nicht in der Standardklassenbibliothek von GemStone enthalten. Da die GemStone-Klasse `MethodCall` für die Parameterliste jedoch die Klasse `Array` und nicht `OrderedCollection` verwendet, muß die Parameterliste von Smalltalk auf einen `Array` in GemStone abgebildet werden. `OrderedCollection` in Smalltalk und `Array` in GemStone werden in der Klassenhierarchie von verschiedenen Klassen abgeleitet, weshalb es wahrscheinlich zu den Problemen kam.

Inwieweit die Probleme auch mit der aus dem Software-Entwicklungswerkzeug *GeODE* kommenden GemStone-Klasse `OrderedCollection` zusammenhängen, konnte nicht festgestellt werden.

Möglicherweise haben beide Ursachen zur falschen Abbildung der Parametermethoden geführt. Denn sie wurden in die Datenbank stets falsch abgelegt, indem die ersten zwei Speicherplätze des Feldes immer frei blieben. Erst bei der Ausführung der Aktionen trat der Fehler auf, da an der ersten erwarteten Stelle kein Eintrag erfolgt war.

Da die GemStone-Klasse `MethodCall` für die Parameterliste die Klasse `Array` und nicht `OrderedCollection` verwendet, wurde auch in der Benutzerschnittstelle an entsprechender Stelle die Klasse `Array` verwendet.

B.4 Hilfswerkzeuge für TriGS

Da die Benutzeroberfläche nur sehr eingeschränkt verwendbar ist und die Funktionen, die sie zur Verfügung stellt, nicht alle Anforderungen abdecken, mußten eigene kleine Hilfswerkzeuge geschrieben werden. Alle Werkzeuge wurden nur als Code-Fragmente im *Gemstone Workspace* der OPAL-Programmierumgebung realisiert.

Regelübersicht

Im Abschnitt B.1 wurde bereits die Notwendigkeit einer sortierten Ausgabe der Regeln aufgezeigt. Diese Funktion wertet alle Basen (`RuleBase`, `EventSelectorBase` , `ConditionBase` und `ActionBase`) aus und gibt die Daten in übersichtlicher Form als String zurück.

```
('ACTIVE RULE DUMP'+(DateTime now asString)+'
================
RULES:
------
```

```
' + WFRuleBase getFormattedInfo + '
EVENTS:
-------

' + WFEventSelectorBase getFormattedInfo + '
CONDITIONS:
-----------

' + WFConditionBase getFormattedInfo + '
ACTIONS:
--------

' + WFActionBase getFormattedInfo).
```

Verwendet man statt den herkömmlichen Info-Methoden die Methoden mit dem Suffix „forFile“, und schickt man dem resultierenden String die Methode `toServerTextFile:` einschließlich Pfad und Dateinamen, so kann der String auch abgespeichert werden.

```
(...) toServerTextFile: '~/gsi/ruledump.txt'
```

Hilfe bei Rekompilierung von Klassen

Wenn Klassen im GemStone System Browser rekompiliert werden, so entstehen neue und keine modifizierten alten Klassen. Waren auf diesen lokale Regeln definiert, so verweisen deren Ereignisselektoren (Klasse `RuleEventSelector`) danach über die Instanzvariable `trgClass` auf die alten Klassen. Wenn ein `MethodCall` einer Aktion die Methode an eine Klasse schickt, so befinden sich auch hier Referenzen auf die alten Klassen. Deshalb muß auch die Instanzvariable `receiver` der Methodenaufrufe `MethodCall`, sofern sie auf eine Klasse verweist, erneuert werden.

```
" *** renew references of events and action receivers  *** "

| events actions |

events := WFEventSelectorBase eventSelectorDictionary.
events doValues:
  [ :e | (e class == RuleEventSelector)
           ifTrue: [ e trgClass: (System myUserProfile
                     resolveSymbol: (e trgClass name)) value ]
  ].

actions := WFActionBase actionDictionary.
actions doValues:
  [ :a | a methodCalls do:
          [ :mc | (mc receiver isClass)
                  ifFalse: [ mc receiver: (System myUserProfile
                             resolveSymbol: (mc receiver name)) value ]
          ]
  ].
```

Umbenennung von Ereignissen, Bedingungen und Aktionen

Zur schnelleren Bearbeitung von Namensänderungen werden zuerst von Hand (*inspect*) die Namen der einzelnen Objekte geändert. Dann müssen die Verzeichniseinträge (`key`) in den Verzeichnissen (`eventDictionary`, `conditionDictionary`, `actionDictionary`) der Basen gemäß dem Namen der Objekte angepaßt werden.

```
" *** name dictionary-keys with name of value-objects *** "

WFEventSelectorBase eventSelectorDictionary do:
    [ :x | x key: (x value name) ]

WFConditionBase conditionDictionary do:
    [ :x | x key: (x value name) ]

WFActionBase actionDictionary do:
    [ :x | x key: (x value name) ]
```

Entfernen nicht mehr verwendeter Query-Methoden

Da bei manuellen Änderungen der Bedingungen, d.h. nicht über die Benutzerschnittstelle, die beim Objekt `Condition` gespeicherten Query-Methoden übrig bleiben, wird hier die ConditionBase durchsucht. Alle nicht mehr verwendeten Methoden werden gelöscht.

```
" *** remove unused query methods *** "

| querySet selectors |

querySet := Set new.
WFConditionBase conditionDictionary doValues:
    [ :c | querySet add: (c queryPointer) ].

selectors := Condition selectorsIn: #Queries.
selectors do:
    [ :s | (querySet includes: s)
              ifFalse: [ Condition removeSelector: s ] ].
```

Anhang C

Roles V3.0 — Anmerkungen, Änderungen, Erweiterungen

Das Rollenkonzept hat den Organisationspolitiken in TriGS$_{flow}$ erst die Flexibilität verliehen, durch die es sich jetzt auszeichnet. Bei der Einbindung des Konzepts waren jedoch einige Änderungen und Ergänzungen notwendig, die im Folgenden beschrieben werden.

C.1 Anmerkungen

Anlegen von Rollen

Beim Erzeugen eines neuen Rollenobjekts (nicht eines Rollentyps) muß das Objekt, dem die Rolle zugeordnet wird, immer in der übergeordneten Rolle vorliegen. Bei der Modellierung des Beispiels war dies oft der Fall:

```
Employee newRoleOf: agent2.
Constructor newRoleOf: (agent2 as: Employee) qualifiedBy: dept2.
```

Das Anlegen ginge wesentlich schneller und bequemer vor sich, wenn die übergeordneten Rollen automatisch angelegt würden und ein Agent bei der Annahme einer untergeordneten Rolle nicht unbedingt mit `as:` in die Vaterrolle schlüpfen müßte.

C.2 Änderungen

Fehlerbehebung in der Methode `doesNotUnderstand:`

Die Methode `doestNotUnderstand:` der Klasse `RoleType` enthielt einen kleinen Fehler, der sich aber leicht beheben ließ.

```
doesNotUnderstand: aMessage
   " If an instance of RoleType does not understand a
     message, the massage is sent to the 'father - instance'
     in the role-hierarchy "

   " OLD METHOD: ^roleOf perform: (aMessage selector)
                  withArguments: (aMessage arguments) "

   ^roleOf perform: (aMessage at: 1) withArguments: (aMessage at: 2)
```

Wie leicht ersichtlich ist, mußte lediglich die Methode `selector` durch `at: 1` und `arguments` durch `at: 2` ersetzt werden. Wahrscheinlich liegt die Ursache dieses Fehlers in der Konvertierung von Smalltalk nach OPAL.

C.3 Erweiterungen

Neue Methode `canUnderstand:`

Bei der Auswahl der Agenten in den Methoden `selectAgents` der Klasse `Activity` oder einer ihrer Unterklassen wird zusätzlich geprüft, ob ein Agent, der die spezifische Politik erfüllt, die Aktionsmethode ausführen kann. Dazu wird getestet, ob der Rollentyp (die Klasse des Rollenobjekts) die Methode versteht, was üblicherweise mit der Methode `canUnderstand:` erledigt wird. Da das Rollenkonzept aber die Vererbung auf Instanzebene (über Klassenvariablen realisiert) durchführt, mußte die Methode `canUnderstand:` ähnlich der bereits neu definierten Methode `doesNotUnderstand:` neu implementiert werden.

```
canUnderstand: aSelector

   " All role types are subclasses of RoleType and don't establish a
     role hierarchy. So they don't inherit methods. If a rolesubclass
     doesn't understand a selector, the method is transmitted to the
     roleSuperType (class variable). The method 'canUnderstand:' has
     to be adapted to this feature! Searching up in the role hierarchy
     ends on reaching the top (RoleType) and continues within class
     hierarchy. "

   (methodDict includes: aSelector)
    ifTrue:  [ ^true]
    ifFalse: [ (self name = 'RoleType')
                ifTrue:  [ ^super canUnderstand: aSelector ]
                ifFalse: [ ^self roleSuperType
                                 canUnderstand: aSelector ]]
```

Neue Klassen `Role` und `QualifiedRole`

Zur einfacheren Definition von Rollentypen wurden diese neuen Klassen geschaffen. Beide Klassen sind keine Unterklassen von `RoleType` oder `QualifiedRoleType`, sondern von der Klasse `Object`. Sie bestimmen lediglich jene Parameter, die nie verändert werden und rufen damit die Originalmethoden der Klassen `RoleType` und `QualifiedRoleType` auf. Erstere der beiden folgenden Methoden ist eine Klassenmethode der Klasse `Role`, zweitere ist eine Klassenmethode der Klasse `QualifiedRole`.

```
newRole: aRoleName roleSuperType: aSuperType

   ^RoleType roleSubclass: aRoleName
      instVarNames: #()
      classVars: #()
      poolDictionaries: #[]
      inDictionary: WFRoles
      constraints: #[]
      instancesInvariant: false
      isModifiable: false
      roleSuperType: aSuperType

newRole: aRoleName roleSuperType: aSuperType classOfQualifyingObj: aClass

   ^QualifiedRoleType qualifiedRoleSubclass: aRoleName
      instVarNames: #()
      classVars: #()
      poolDictionaries: #[]
      inDictionary: WFRoles
      constraints: #[]
      instancesInvariant: false
      isModifiable: false
      roleSuperType: aSuperType
      classOfQualifyingObj: aClass
```

Anhang D

Regeln in TriGS$_{flow}$

Dieser Anhang zeigt einen Auszug aus den Regelbasen von TriGS$_{flow}$. Die Regeln wurden mit dem in Abschnitt B.4 beschriebenen Werkzeug formatiert ausgedruckt. Enthalten sind sowohl Regeln, die für alle Workflow Typen gelten, als auch Regeln, die für das in Kapitel 5 beschriebene Beispiel angelegt wurden. Der Anhang unterteilt sich in vier Teile: Regeln, Ereignisse, Bedingungen und Aktionen. Regelnamen mit einem Suffix (z.B. `_OFFER_002_003`) bezeichnen workflowspezifische Regeln, also Regeln, die nur für einen bestimmten Workflow Typ oder eine bestimmte Aktivität gelten. Alle anderen Regeln gelten für alle Workflow Typen.

```
ACTIVE WORKFLOW DUMP: 05/02/1995 20:34:03
====================

RULES:
------

Rule name:         R_ActivitySuccession_OFFER_001_002
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_001_002
Action:            A_ActivitySuccession_OFFER_001_002
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 001;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_002_003
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_002_003
Action:            A_ActivitySuccession_OFFER_002_003
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 002;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_002_004
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_002_004
Action:            A_ActivitySuccession_OFFER_002_004
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 002;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_002_005
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_002_005
Action:            A_ActivitySuccession_OFFER_002_005
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 002;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_005_001
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_005_001
Action:            A_ActivitySuccession_OFFER_005_001
Rule invariant:    false
Priority:          1
Activation status: false
```

```
Activated for:     InnerActivity-Object: 005;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_START_001
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_START_001
Action:            A_ActivitySuccession_OFFER_START_001
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     StartActivity-Object: START;
Deactivated for:


Rule name:         R_ActivitySuccession_OFFER_START_002
Event selector:    E_ActivitySuccession
Condition:         C_ActivitySuccession_OFFER_START_002
Action:            A_ActivitySuccession_OFFER_START_002
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     StartActivity-Object: START;
Deactivated for:


Rule name:         R_HistoryBeginExec
Event selector:    E_HistoryBeginExec
Condition:         C_HistoryBeginExec
Action:            A_HistoryBeginExec
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_HistoryEndExec
Event selector:    E_HistoryEndExec
Condition:         C_HistoryEndExec
Action:            A_HistoryEndExec
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_HistoryNotification
Event selector:    E_HistoryNotification
Condition:         C_HistoryNotification
Action:            A_HistoryNotification
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:
```

```
Rule name:         R_OrgPolBinding_OFFER_001
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_001
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     InnerActivity-Object: 001;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_002
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_002
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     InnerActivity-Object: 002;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_003
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_003
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     InnerActivity-Object: 003;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_004
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_004
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     InnerActivity-Object: 004;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_005
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_005
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     InnerActivity-Object: 005;
Deactivated for:
```

```
Rule name:         R_OrgPolBinding_OFFER_END
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_END
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     EndActivity-Object: END;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_EXCEPTION
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_EXCEPTION
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     ExceptionActivity-Object: EXCEPTION;
Deactivated for:


Rule name:         R_OrgPolBinding_OFFER_START
Event selector:    E_OrgPolicy
Condition:         C_OrgPolBinding
Action:            A_OrgPolBinding_OFFER_START
Rule invariant:    false
Priority:          0
Activation status: false
Activated for:     StartActivity-Object: START;
Deactivated for:


Rule name:         R_OrgPolicyInit
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicyInit
Action:            A_OrgPolicyInit
Rule invariant:    false
Priority:          2
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_001_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_001_1
Action:            A_OrgPolicy_OFFER_001_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 001;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_002_1
Event selector:    E_OrgPolicy
```

```
Condition:         C_OrgPolicy_OFFER_002_1
Action:            A_OrgPolicy_OFFER_002_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 002;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_003_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_003_1
Action:            A_OrgPolicy_OFFER_003_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 003;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_004_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_004_1
Action:            A_OrgPolicy_OFFER_004_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 004;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_005_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_005_1
Action:            A_OrgPolicy_OFFER_005_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     InnerActivity-Object: 005;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_END_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_END_1
Action:            A_OrgPolicy_OFFER_END_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     EndActivity-Object: END;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_EXCEPTION_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_EXCEPTION_1
Action:            A_OrgPolicy_OFFER_EXCEPTION_1
```

```
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     ExceptionActivity-Object: EXCEPTION;
Deactivated for:


Rule name:         R_OrgPolicy_OFFER_START_1
Event selector:    E_OrgPolicy
Condition:         C_OrgPolicy_OFFER_START_1
Action:            A_OrgPolicy_OFFER_START_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     StartActivity-Object: START;
Deactivated for:


Rule name:         R_RestartWFOnAgentActivation
Event selector:    E_RestartWFOnAgentActivation
Condition:         C_RestartWFOnAgentActivation
Action:            A_RestartWFOnAgentActivation
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_StartOnInsert
Event selector:    E_StartOnInsert
Condition:         C_StartOnInsert
Action:            A_StartOnInsert
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_StartOnRemove
Event selector:    E_StartOnRemove
Condition:         C_StartOnRemove
Action:            A_StartOnRemove
Rule invariant:    false
Priority:          1
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_StartWFOrgPolicyInit
Event selector:    E_StartWFOrgPolicy
Condition:         C_StartWFOrgPolicyInit
Action:            A_StartWFOrgPolicyInit
Rule invariant:    false
Priority:          2
```

```
Activation status: true
Activated for:
Deactivated for:


Rule name:         R_StartWFOrgPolicy_OFFER_1
Event selector:    E_StartWFOrgPolicy
Condition:         C_StartWFOrgPolicy_OFFER_1
Action:            A_StartWFOrgPolicy_OFFER_1
Rule invariant:    false
Priority:          1
Activation status: false
Activated for:     WorkflowType-Object: OFFER;
Deactivated for:




EVENTS:
-------

Event selector name:    AES_StartOnInsert
Triggering method:      addToWorklist:folder:role:
Triggering method type: instance method
Execution type:         post
Triggering object:      self


Event selector name:    CES_ActivitySuccession
Triggering method:      distribute:
Triggering method type: instance method
Execution type:         pre
Triggering object:      self


Event selector name:    CES_HistoryBeginExec
Triggering method:      performAction:withArguments:by:
Triggering method type: instance method
Execution type:         pre
Triggering object:      self


Event selector name:    CES_HistoryEndExec
Triggering method:      performAction:withArguments:by:
Triggering method type: instance method
Execution type:         post
Triggering object:      self


Event selector name:    CES_HistoryNotification
Triggering method:      addToWorklist:folder:role:
Triggering method type: instance method
Execution type:         post
Triggering object:      self


Event selector name:    CES_OrgPolicy
```

```
Triggering method:      notifyAgent:
Triggering method type: instance method
Execution type:         pre
Triggering object:      self


Event selector name:    CES_RestartWFOnAgentActivation
Triggering method:      activate
Triggering method type: instance method
Execution type:         post
Triggering object:      self


Event selector name:    CES_StartOnInsert
Triggering method:      addToWorklist:folder:role:
Triggering method type: instance method
Execution type:         pre
Triggering object:      self


Event selector name:    CES_StartOnRemove
Triggering method:      removeFromWorklist:
Triggering method type: instance method
Execution type:         post
Triggering object:      self


Event selector name:    CES_StartWFOrgPolicy
Triggering method:      isAllowedToStart:withFolder:
Triggering method type: instance method
Execution type:         pre
Triggering object:      self


Event selector name:    E_ActivitySuccession
Triggering method:      distribute:
Triggering method type: instance method
Execution type:         pre
Triggering class:       Activity


Event selector name:    E_HistoryBeginExec
Triggering method:      performAction:withArguments:by:
Triggering method type: instance method
Execution type:         pre
Triggering class:       Agent


Event selector name:    E_HistoryEndExec
Triggering method:      performAction:withArguments:by:
Triggering method type: instance method
Execution type:         post
Triggering class:       Agent


Event selector name:    E_HistoryNotification
Triggering method:      addToWorklist:folder:role:
```

```
Triggering method type: instance method
Execution type:         post
Triggering class:       Agent


Event selector name:    E_OrgPolicy
Triggering method:      notifyAgent:
Triggering method type: instance method
Execution type:         pre
Triggering class:       Activity


Event selector name:    E_RestartWFOnAgentActivation
Triggering method:      activate
Triggering method type: instance method
Execution type:         post
Triggering class:       Agent


Event selector name:    E_StartOnInsert
Triggering method:      addToWorklist:folder:role:
Triggering method type: instance method
Execution type:         pre
Triggering class:       SoftwareAgent


Event selector name:    E_StartOnRemove
Triggering method:      removeFromWorklist:
Triggering method type: instance method
Execution type:         post
Triggering class:       SoftwareAgent


Event selector name:    E_StartWFOrgPolicy
Triggering method:      isAllowedToStart:withFolder:
Triggering method type: instance method
Execution type:         pre
Triggering class:       WorkflowType




CONDITIONS:
-----------

Condition name: C_ActivitySuccession_OFFER_001_002
Event selector: CES_ActivitySuccession
Query:          query183

^true


Condition name: C_ActivitySuccession_OFFER_002_003
Event selector: CES_ActivitySuccession
Query:          query178

^((ruleTrgMethodParamList at: 1) calcStatus=#ok)
```

```
Condition name: C_ActivitySuccession_OFFER_002_004
Event selector: CES_ActivitySuccession
Query:          query179

^((ruleTrgMethodParamList at: 1) calcStatus=#rejected)


Condition name: C_ActivitySuccession_OFFER_002_005
Event selector: CES_ActivitySuccession
Query:          query180

^((ruleTrgMethodParamList at: 1) calcStatus=#insufficient)


Condition name: C_ActivitySuccession_OFFER_005_001
Event selector: CES_ActivitySuccession
Query:          query184

^true


Condition name: C_ActivitySuccession_OFFER_START_001
Event selector: CES_ActivitySuccession
Query:          query182

^true


Condition name: C_ActivitySuccession_OFFER_START_002
Event selector: CES_ActivitySuccession
Query:          query177

^true


Condition name: C_HistoryBeginExec
Event selector: CES_HistoryBeginExec
Query:          query6

^true


Condition name: C_HistoryEndExec
Event selector: CES_HistoryEndExec
Query:          query7

^true


Condition name: C_HistoryNotification
Event selector: CES_HistoryNotification
Query:          query5

^true
```

```
Condition name: C_OrgPolBinding
Event selector: CES_OrgPolicy
Query:          query137

^true


Condition name: C_OrgPolicyInit
Event selector: CES_OrgPolicy
Query:          query163

^true


Condition name: C_OrgPolicy_OFFER_001_1
Event selector: CES_OrgPolicy
Query:          query181

^true


Condition name: C_OrgPolicy_OFFER_002_1
Event selector: CES_OrgPolicy
Query:          query171

^true


Condition name: C_OrgPolicy_OFFER_003_1
Event selector: CES_OrgPolicy
Query:          query172

^true


Condition name: C_OrgPolicy_OFFER_004_1
Event selector: CES_OrgPolicy
Query:          query173

^true


Condition name: C_OrgPolicy_OFFER_005_1
Event selector: CES_OrgPolicy
Query:          query174

^true


Condition name: C_OrgPolicy_OFFER_END_1
Event selector: CES_OrgPolicy
Query:          query175

^true
```

```
Condition name: C_OrgPolicy_OFFER_EXCEPTION_1
Event selector: CES_OrgPolicy
Query:          query176

^true


Condition name: C_OrgPolicy_OFFER_START_1
Event selector: CES_OrgPolicy
Query:          query170

^true


Condition name: C_RestartWFOnAgentActivation
Event selector: CES_RestartWFOnAgentActivation
Query:          query1


" if there's at least one paused workflow, try to restart
  (an agent is the rule triggering object) "

WFWorkflowBase doValues: [ :wf | (wf status=#paused)&
                                 (wf company == ruleTrgObject company)
                                     ifTrue: [ ^true ] ].

^false



Condition name: C_StartOnInsert
Event selector: CES_StartOnInsert
Query:          query185

^(ruleTrgObject worklist isEmpty)


Condition name: C_StartOnRemove
Event selector: CES_StartOnRemove
Query:          query168

^(ruleTrgObject worklist isEmpty not)



Condition name: C_StartWFOrgPolicyInit
Event selector: CES_StartWFOrgPolicy
Query:          query164

^true


Condition name: C_StartWFOrgPolicy_OFFER_1
Event selector: CES_StartWFOrgPolicy
Query:          query169

^true
```

```
ACTIONS:
--------

Action name:    A_ActivitySuccession_OFFER_001_002
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        raiseException:
   Methods for parameters: pFalse,

   Method selector:        combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_002_003
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        raiseException:
   Methods for parameters: pFalse,

   Method selector:        combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_002_004
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        raiseException:
   Methods for parameters: pFalse,

   Method selector:        combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_002_005
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        raiseException:
   Methods for parameters: pFalse,

   Method selector:        combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_005_001
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        raiseException:
   Methods for parameters: pFalse,
```

```
   Method selector:       combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_START_001
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       raiseException:
   Methods for parameters: pFalse,

   Method selector:       combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_ActivitySuccession_OFFER_START_002
Event selector: CES_ActivitySuccession
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       raiseException:
   Methods for parameters: pFalse,

   Method selector:       combine:fromActivity:
   Methods for parameters: pActFolder, pPredActivity,


Action name:    A_HistoryBeginExec
Event selector: CES_HistoryBeginExec
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       addHistoryBegExecTo:with:
   Methods for parameters: pHistExecFolder, pHistExecActivity,


Action name:    A_HistoryEndExec
Event selector: CES_HistoryEndExec
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       addHistoryEndExecTo:with:
   Methods for parameters: pHistExecFolder, pHistExecActivity,


Action name:    A_HistoryNotification
Event selector: CES_HistoryNotification
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       addHistoryNotificationTo:with:
   Methods for parameters: pHistNotFolder, pHistNotActivity,


Action name:    A_OrgPolBinding_OFFER_001
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:              ruleTrgObject
   Method selector:       bindAgentWithLessWorklistSize
```

```
Action name:    A_OrgPolBinding_OFFER_002
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_003
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_004
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_005
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_END
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_EXCEPTION
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolBinding_OFFER_START
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        bindAgentWithLessWorklistSize


Action name:    A_OrgPolicyInit
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        actAgentRoleSet:
   Methods for parameters: pSetNew,
```

```
Action name:    A_OrgPolicy_OFFER_001_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_001_1


Action name:    A_OrgPolicy_OFFER_002_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_002_1


Action name:    A_OrgPolicy_OFFER_003_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_003_1


Action name:    A_OrgPolicy_OFFER_004_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_004_1


Action name:    A_OrgPolicy_OFFER_005_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_005_1


Action name:    A_OrgPolicy_OFFER_END_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_END_1


Action name:    A_OrgPolicy_OFFER_EXCEPTION_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_EXCEPTION_1


Action name:    A_OrgPolicy_OFFER_START_1
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectAgents_OFFER_START_1
```

```
Action name:    A_RestartWFOnAgentActivation
Event selector: CES_RestartWFOnAgentActivation
MethodCalls:
   Receiver:               WFWorkflowBase
   Method selector:        restartWorkflows


Action name:    A_StartOnInsert
Event selector: AES_StartOnInsert
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectElem


Action name:    A_StartOnRemove
Event selector: CES_StartOnRemove
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectElem


Action name:    A_StartWFOrgPolicyInit
Event selector: CES_OrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        startAgentSet:
   Methods for parameters: pSetNew,


Action name:    A_StartWFOrgPolicy_OFFER_1
Event selector: CES_StartWFOrgPolicy
MethodCalls:
   Receiver:               ruleTrgObject
   Method selector:        selectStartAgents_OFFER_1:
   Methods for parameters: pStartActFolder,
```

Anhang E

Glossar

Hier werden die wichtigsten, der in der Arbeit verwendeten Begriffe noch einmal zusammengefaßt und kurz erläutert. Die Zusammenhänge der Begriffe werden ausführlicher im theoretischen Teil der Arbeit (Kapitel 1–3) beschrieben.

Ablauf (Enactment) Der Ablauf regelt die Ausführung eines Workflow Typs. Ein Workflow Typ wird instanziiert und diese Instanz, d.h. der Workflow, hält den konkreten Durchlauf durch das Aktivitätennetz fest. Der Ablauf einer einzelnen Aktivität besteht aus Ausführung (Execution) der Aktion und der anschließenden Arbeitsverteilung (Work Distribution).

Agent Ein Agent stellt einen Teilnehmer des Workflow Systems dar. Er kann menschlicher Natur (HumanAgent) oder ein Programm sein (Software-Agent). Der Hauptunterschied besteht in der Abarbeitungsreihenfolge der Aufgaben in der Arbeitsliste.

Agentenauswahl (Agent Selection) Die Agentenauswahl gehört zur Arbeitsverteilung (Work Distribution) einer Aktivität. Die angegebene Organisationspolitik wird ausgewertet und alle Agenten, die sie erfüllen, vorgemerkt.

Agentenbenachrichtigung (Agent Notification) Wenn ein oder mehrere Agenten zur Ausführung bestimmt sind, müssen diese benachrichtigt werden. Die Aktivität wird ihnen dabei in die Arbeitsliste gestellt.

Agentenzuordnung (Agent Binding) Nachdem die Agentenauswahl stattgefunden hat, müssen jene bestimmt werden, die die Aktivität konkret ausführen sollen. Üblicherweise wird nur einer bestimmt, nämlich jener, dessen Arbeitsliste am wenigsten Aufgaben enthält.

Akt (Folder) Der Akt ist die elektronische Umlaufmappe, die von Agent zu Agent gereicht wird und innerhalb eines Workflows verwendet und bearbeitet wird. Der Akt enthält alle Informationen, die während des Durchlaufs durch das Aktivitätennetz benötigt werden.

Aktion(-smethode) Sie wird für einen Rollentyp definiert und ist für alle Subtypen des Rollentyps gültig. Bei der Ausführung einer Aktivität wird deren Aktionsmethode ausgeführt. Das kann z.B. ein Programmaufruf oder eine Berechnung sein.

Aktives Datenbanksystem Eine aktive Komponente erweitert das Datenbanksystem um die Möglichkeit, auf bestimmte vordefinierte Ereignisse zu reagieren. Das Zentrum bilden meist ECA-Regeln, die das Verhalten der aktiven Komponente bestimmen.

Aktivität (Activity) Die Aktivität beschreibt, welche Aktion von einem Agenten ausgeführt werden muß, und welche Daten hierbei gelesen und geschrieben werden. Eine Aktivität wird ausgeführt, sofern ihre Ausführungsbedingung wahr ist. Wer sie auszuführen hat, wird durch die zugeordnete Organisationspolitik bestimmt.

Aktivitätennetz Das Aktivitätennetz beschreibt die Ausführungsreihenfolge der Aktivitäten eines Workflow Typs. Es wird durch Vorgänger- und Nachfolgerbeziehungen zwischen den Aktivitäten festgelegt.

Akttyp (Folder Type) Der Akttyp beschreibt den Aufbau der Daten, die in einem Workflow Typ bearbeitet werden. In einem bestimmten Workflow Typ kann nur ein bestimmter Akttyp verwendet werden.

Arbeitsverteilung (Work Distribution) Die Arbeitsverteilung einer Aktivität erfolgt nach der Ausführung der Vorgängeraktivität(en). Zuerst werden jene Agenten ausgewählt, die die Organisationspolitik erfüllen (Agentauswahl, Agent Selection). Dann werden die Ausführenden daraus bestimmt (Agentenzuordnung, Agent Binding), und schließlich werden die zugeordneten Agenten benachrichtigt (Agentenbenachrichtigung, Agent Notification).

Ausführung (Execution) Die Ausführung ist jene Phase des Ablaufs (Enactment), in der die Daten vom Akt (Folder) bereitgestellt, die Benutzeraktion ausgeführt, und schließlich die Daten wieder in den Akt übernommen werden. Anschließend folgt die Arbeitsverteilung (Work Distribution).

Ausführungsbedingung (Execution Condition) Hiermit wird für eine Aktivität festgelegt, ob sie unter den gegebenen Umständen ausgeführt wird. Von großer Bedeutung ist diese bei der Auswertung von Verzweigungen, indem einzelne Pfade bei negativer Auswertung nicht ausgeführt werden.

Datenfluß Welche Daten eines Aktes (Folder) in die Aktivitätsbearbeitung involviert sind, wird durch den Datenfluß beschrieben. Eine Beschreibung aller Aktivitäten eines Workflow Typs legt somit auch den Datenfluß innerhalb des Workflows fest.

ECA-Regel Eine Event-Condition-Action-Regel besteht aus einem Ereignis, einer Bedingung und einer Aktion. Tritt das Ereignis ein, so wird, wenn die Bedingung erfüllt ist, die Aktion ausgeführt. Ist die Bedingung nicht erfüllt, geschieht nichts. ECA-Regeln sind meist das zentrale Konzept aktiver Datenbanksysteme.

Geschachtelte Transaktion (Nested Transaction) Eine geschachtelte Transaktion läuft in mehreren Ebenen ab. Sie kann in mehrere Teiltransaktionen zerlegt werden, von denen jede einzelne zurückgesetzt werden kann. Wird eine übergeordnete Transaktion zurückgesetzt, werden auch die untergeordneten Transaktionen abgebrochen und zurückgesetzt. Wenn alle Subtransaktionen erfolgreich abgeschlossen sind, so kann auch die übergeordnete Transaktion beendet werden.

Geschäftsprozeß Vorgang (Workflow) im Büro- und Verwaltungsbereich.

Gruppe Eine Gruppe ist eine konkrete Ausprägung eines Gruppentyps. Die Agenten werden in Abteilungen zusammengefaßt. Eine Abteilung entspricht einer Gruppe. Die Abteilungsstruktur eines Betriebes wird aus Gruppen gebildet, indem sie in einer „part of"-Hierarchie angeordnet werden.

Gruppentyp Ein Gruppentyp beschreibt die Attribute einer Abteilung. Durch Anordnung in einer Gruppentyphierarchie können Attribute eines Gruppentyps geerbt und vererbt werden. Gruppentypen dienen zur Erweiterung der Organisationsstruktur, sind aber im Workflow System Prototyp zur Zeit nicht relevant.

Integration Im Zusammenhang mit dem Begriff der Modellierung, wie er in dieser Arbeit eingeführt wurde, bezeichnet die Integration die Verbindung der drei Modellierungsschichten Organisations-, Aktivitäten- und Datenmodellierung durch Organisationspolitiken und Datenflußbeschreibungen.

Komplexe Organisationspolitik Diese Organisationspolitik ist lediglich eine Verknüpfung einzelner Politiken mit logischen Operatoren.

Kontextabhängige Organisationspolitik Die Organisationspolitik bezieht sich auf den aktuellen Workflow, d.h. auf die Daten des aktuellen Aktes. Durch diesen Bezug kann die Auswahl der Agenten entscheidend beeinflußt werden.

Kontrollfluß Die Anordnung von Aktivitäten zusammen mit den Ausführungsbedingungen bestimmen den Kontrollfluß. Zur Beschreibung gibt es drei Kontrollstrukturen, nämlich Sequenz, Verzweigung und Zusammenführung.

Kontrollstrukturen Das Aktivitätennetz wird durch folgende drei Graphelemente gebildet: Sequenz (ein Nachfolger), Verzweigung (mehrere Nachfolger) und Zusammenführung (mehrere Vorgänger).

Organisationspolitik Ist die Integration von Organisation und Aktivitäten, die besagt, wer welche Aufgabe ausführen soll.

Produktionsprozeß Vorgang (Workflow) im Produktionsbereich.

Prozeß entspricht einem *Workflow*

Rolle Eine Rolle ist eine konkrete Ausprägung eines Rollentyps für einen Agenten. Das Verhalten (das Kompetenzprofil, die ausführbaren Aktionen) eines Agenten wird durch die Gesamtheit seiner Rollen beschrieben. Wenn ein Agent eine Aktion ausführen will, so muß er zuerst „in jene Rolle schlüpfen", die diese Aktion versteht.

Rollentyp Ein Rollentyp gibt die Aktionen an, die ein Agent ausführen kann, wenn die entsprechende Rolle spielt. Rollentypen können in Rollentyphierarchien die Aktionen vererben.

Sequenz Die Sequenz ist jene Kontrollstruktur, bei der eine Aktivität nach der anderen ausgeführt wird (Hintereinanderausführung zweier Aktivitäten).

Triggersystem Die aktive Komponente eines Datenbanksystems wird auch als Triggersystem bezeichnet. Es ist für die Ausführung der definierten ECA-Regeln verantwortlich.

Verzweigung Bei der Kontrollstruktur Verzweigung sind zu einer bestimmten Aktivität mehrere Nachfolger definiert. Es werden aber nur jene Nachfolgeaktivitäten ausgeführt, deren Ausführungsbedingung wahr ist.

Vorgang entspricht einem *Workflow*

Workflow Ein Workflow ist eine konkrete Ausprägung eines Workflow Typs. Er stellt die aktuellen Daten eines bestimmten Aktes dar und hält auch den tatsächlichen Durchlauf des Aktes durch das Aktivitätennetz fest.

Workflow Bank Sowohl Anwendungsdaten als auch workflowspezifische Daten werden in einer Datenbank abgelegt, die in Analogie dazu „Workflow Bank" bezeichnet werden kann.

Workflow Management System Die Gesamtheit der Softwareprogramme, die die Definition, Ausführung und Überwachung von Workflow Typen bzw. Workflows ermöglicht.

Workflow System Ein Workflow System setzt sich aus den Daten (Workflow Bank) und der Software (Workflow Management System) zusammen

Workflow Typ Der Workflow Typ integriert alle drei Modellierungsschichten Organisation, Aktivitäten und Daten. In einem Workflow Typ ist also die gesamte Organisationsstruktur, das Aktivitätennetz und die Aktstruktur einschließlich der Integration durch Organisationspolitik und Datenflußbeschreibung enthalten.

Zusammenführung Die Kontrollstruktur Zusammenführung verbindet mehrere Vorgängerpfade zu einem Pfad. Meist wird solange zusammengewartet, bis alle vorhergehenden Pfade fertig ausgeführt sind, ehe mit der weiteren Arbeitsverteilung fortgesetzt wird.

Literaturverzeichnis

[Aich94] B. Aichinger. *Realisierung aktiver Datenbankfunktionalität auf Basis des objektorientierten Datenbanksystems GemStone.* Diplomarbeit, Universität Linz, 1994

[Appe93] H.-J. Appelrath, H. Behrends, H. Jasper, H. Ortleb. *Die Entwicklung aktiver Datenbanken am Beispiel der Krebsforschung.* Informatik aktuell, Datenbanksysteme in Büro, Technik und Wissenschaft, 1993, pp 75–93

[Bern87] P.A. Bernstein, V. Hadzilacos, N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987

[Biro93] M. Biró, A. Micsik, T. Remzső. *Distributed Object Management, a Survey.* CON'93: The Challenge of Networking — connecting equipment, humans, institutions, R.Oldenbourg, Wien München, 1993, pp 15–26

[Brei93] Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, G. Weikum. *Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multisystem Workflows.* SIGMOD Record, Vol. 22, No. 3, September 1993

[Bußl93] C. Bußler. *Enterprise Process Integration Model and Infrastructure.* Proceedings of the IPSE/IFIP WG 5.3 International Workshop on The Design of Information Infrastructure Systems for Manufacturing, Tokyo, Japan, November 8–10, 1993

[Bußl94] C. Bußler, S. Jablonski. *An Approach to Integrate Workflow Modeling and Organisation Modeling in an Enterprise.* 3rd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE), Morgantown, West Virginia, April 1994

[Butt91] P. Butterworth, A. Otis, J. Stein. *The GemStone Object Datebase Management System.* Communications of the ACM, Volume 34, Number 10, October 1991, pp 64–77

[Chro94] G. Chroust, A. Benczúr (ed.). *Workflow Management — Challenges, Paradigms and Products.* CON'94, R. Oldenbourg, Wien München, 1994

[CSES94] C.S.E. Systems. *WorkFlow.* Version 2.2, Benutzer- und Installationshandbücher, 1994

[Desh93] S. Deshpande. *Developing Distributed Object-Oriented Applications Using CORBA.* Tutorial No. 9 of the European Conference on Object-Oriented Programming, 1993

[Dörf77] W. Dörfler. *Mathematik für Informatiker. Band 1: Finite Methoden und Algebra.* Carl Hanser Verlag, München Wien, 1977

[Eder86] J. Eder, G. Kappel, H. Werthner. *Evaluation of Scheduling Mechanisms Of A Dynamic Data Model By Simulation.* Proceedings of the IASTED International Symposium, Measurement, Signal Processing And Control - MECO'86, Acta Press, Anaheim Calgary Zürich, 1986, pp 86–91

[Eder87] J. Eder, G. Kappel, A. Tjoa, R. Wagner. *BIER — The Behaviour Integrated Entity Relationship Approach.* Elsevier Science Publishers B. V. (North-Holland), 1987

[Elli94] C. Ellis. *Workflow — Tutorial.* CON'94, Linz, Austria, October 19, 1994

[Elma91] A. Elmagarmid (ed.). *Special Issue on Unconventional Transaction Management.* Bulletin of the IEEE Computer Society Technological Committee on Data Engineering, Volume 14, Number 1, 1991

[Elma92] A. Elmagarmid (ed.). *Database Transaction Models for Advanced Applications.* San Mateo (CA), Morgan Kaufmann Publishers, 1992

[Etzi93] O. Etzion. *PARDES — A Data-Driven Oriented Active Database Model.* SIGMOD Record, Volume 22, Number 1, March 1993, pp 7–14

[Gamm92] E. Gamma. *Objektorientierte Software-Entwicklung am Beispiel von ET++: Design-Muster, Klassenbibliothek, Werkzeuge.* Springer-Verlag, Berlin, 1992

[Garc87] H. Garcia-Molina, K. Salem. *Sagas.* Proceedings of the ACM SIGMOD (San Francisco), 1987, pp 249–259

[Geih95] K. Geihs. *Client/Server-Systeme: Grundlagen und Architekturen.* (TAT: Thomson's aktuelle Tutorien; 6), International Thomson Publishing, Bonn, 1995

[Geor93] D. Georgakopoulos, M.F. Hornick, F. Manola, M.L. Brodie, S. Heiler, F. Nayeri, B. Hurwitz. *An Extended Transaction Environment for Workflows in Distributed Object Computing.* IEEE Bulletin of the Technical Commitee on Data Engineering, Volume 16, Number 2, June 1993

[Gold83] A. Goldberg, D. Robson. *Smalltalk-80: The Language And Its Implementation.* Addison-Wesley, May 1983

[Good75] J. Goodenough. *Exception Handling: Issues and a Proposed Notation.* Communications of the ACM, Volume 18, Number 12, December 1975, pp 683–696

[Gott94] G. Gottlob, M. Schrefl, B. Röck. *Extending Object-Oriented Systems with Roles.* Zur Publikation angenommen: ACM Transactions on Information Systems (TOIS), 1994

[Gruh94] V. Gruhn. *Communication Support in the Workflow Management Environment LEU.* CONnectivity 94: Workflow Management — Challenges, Paradigms and Products, R.Oldenbourg, Wien München, 1994, pp 187–200

[Hopc79] J.E. Hopcroft, J.D. Ullman. *Introduction To Automata Theory, Languages And Computation.* Addison-Wesley, 1979

[Jabl94a] S. Jablonski, C. Bußler. *Implementing Agent Coordination for Workflow Management Systems Using Active Database Systems.* Proceedings of the 4th International Workshops in Data Engineering, Houston, Texas, February 14–15, 1994, pp 53–59

[Jabl94b] S. Jablonski. *Workflow Management Systems — Modeling and Architecture.* CAiSE*94 - Tutorial

[Jasp94] H. Jasper. *Active Databases for Active Repositories.* Proceedings of the 10th International Conference on Data Engineering, Houston, Texas, February 14–18, 1994, pp 375–384

[Kapp87] G. Kappel, M. Schrefl. *A Behaviour Integrated Entity-Relationship Approach for the Design of Object-Oriented Databases.* Elsevier Science Publishers B. V. (North-Holland), 1987, pp 311–328

[Kapp89a] G. Kappel. *Proposed Reference Example for the TWG in Ithaca.* ITHACA Tools Working Group Interim Report, 1989

[Kapp89b] G. Kappel, J. Vitek, O. Nierstrasz, B. Junod, M. Stadelmann. *Scripting Applications in the Public Administration Domain.* SIGOIS Bulletin, Volume 10, Number 4, December 1989, pp 21–32

[Kapp91] G. Kappel, M. Schrefl. *Object/Behaviour Diagrams.* 7th IEEE International Conference on Data Engineering, Kobe, Japan, April 8–12, 1991

[Kapp94a] G. Kappel, S. Rausch-Schott, W. Retschitzegger, S. Vieweg. *TriGS — Making a Passive Object-Oriented Database System Active.* Journal of Object-Oriented Programming (JOOP), Juli/August, 1994

[Kapp94b] G. Kappel, S. Rausch-Schott, W. Retschitzegger. *Beyond Coupling Modes — Implementing Active Concepts on Top of a Commercial ooDBMS.* E.Bertino and S.Urban, International Symposium on Object-Oriented Methodologies and Systems (ISOOMS), Springer LNCS 858, 1994

[Kapp95a] G. Kappel, B. Pröll, S. Rausch-Schott, W. Retschitzegger. *$TriGS_{flow}$ — Active Object-Oriented Workflow Management.* Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95), January, 1995

[Kapp95b] G. Kappel, P. Lang, S. Rausch-Schott, W. Retschitzegger. *Workflow Management based on Objects, Rules, and Roles.* IEEE Bulletin of the Technical Commitee of Data Engineering, Volume 18, Number 1, March, 1995

[Karb90] B.H. Karbe, N.G. Ramsberger. *Influence of Exception Handling on the support of Cooperative Office Work.* ACM SIGOIS, Volume 11, Number 4, December 1990, pp 2–15

[Kies92] A. Kieser, H. Kubicek. *Organisation.* 3. Auflage, de Gruyter, Berlin/New York, 1992

[McGr92] J.D. McGregor, D.A. Sykes. *Object-Oriented Software Development: Engineering Software for Reuse.* Van Nostrand Reinhold, New York, 1992

[Nier93] O.M. Nierstraß (ed.). *ECOOP'93 — Object-Oriented Programming.* Proceedings of the 7th European Conference, Kaiserslautern, Germany, July 1993

[Ober94] G. Oberlik, S. Preishuber, A.M. Tjoa, R.R. Wagner, K. Wolfmayr. *VOMDraw — A Tool for Visual Object Modeling.* Proceedings of the CAST'94, Ottawa, Canada, May 1994

[Öszu91] M.T. Öszu, P. Valduriez. *Principles Of Distributed Database Systems.* Prentice Hall, 1991

[Pete81] J. L. Peterson. *Petri net theory and the modeling of systems.* Prentice Hall, 1981

[Rein93] B. Reinwald. *Workflow Management in verteilten Systemen.* Teubner, Leipzig, 1993

[Saas92] H.T. Saastamoinen, V.V. Savolainen. *Exception Handling in Office Information Systems.* Proceedings of the Third International Conference on Dynamic Modeling of Information Systems, Noordwijkerhout, 1992, pp 345–363

[Sche94] A.-W. Scheer. *Wirtschaftsinformatik — Referenzmodelle für industrielle Geschäftsprozesse.* 4. Auflage, Springer-Verlag, Berlin Heidelberg NewYork Tokio, 1994

[Serv91] Servio Corporation. *Programming in OPAL* GemStone Version 2.5, August 1991

[Tsic82] D. Tsichritzis. *Form Management.* Communications of the ACM, Number 7, Volume 25, July 1982, pp 453–478

[Tsic85] D. Tsichritzis. *Office Automation.* Springer-Verlag, Berlin Heidelberg, 1985

[Voss91] G. Vossen. *Data Models, Database Languages And Database Management Systems.* Addison-Wesley, 1991

[Wäch92] H. Wächter, H. Reuter. *The ConTract Model.* Database Transaction Models for Advanced Applications, San Mateo (CA), Morgan Kaufmann Publishers, 1992, pp 219–263

[WFMC94] Workflow Management Coalition. *Information Pack — The Work Of The Coalition.* Workflow Management Coalition, September 1994

[Zism77] M. Zisman. *Representation, Specification and Automation of Office Procedures.* University of Pennsylvania, Ph.D., 1977