EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Diplomarbeit im Fach Informatik

Portierung einer Workflow-Engine auf Applikationsserver unter Verwendung EJB 2.0

Vorgelegt von
Eugen Resch

SS 2003

Betreuer: Prof. Dr. W. Spruth

Lehrstuhl für Technische Informatik
Wilhelm-Schickard-Institut für Informatik
Eberhard Karls Universität Tübingen

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbst verfasst und nur die angegebenen Quellen benutzt habe. Ferner habe ich die Arbeit noch keinem anderen Prüfungsamt vorgelegt.

Tübingen, den 10.07.2003

Danksagung

Diese Diplomarbeit entstand im Sommersemester 2003 an der Universität Tübingen in Kooperation mit der Firma iT media Consult GmbH.

Mein besonderer Dank gilt Prof. Dr. Wilhelm G. Spruth für seine engagierte wissenschaftliche Betreuung sowie für das Ermöglichen der Zusammenarbeit mit der Firma iT media Consult GmbH .

Der Firma iT media Consult GmbH, besonders Herrn Fritz Fausel, danke ich für die gute Zusammenarbeit und die materielle Unterstützung bei der Entwicklung von Software-Komponenten.

Dr. Klaus Beschorner danke ich dafür, dass er mich auf die Idee dieser Diplomarbeit gebracht hat. Für die kompetente fachliche Unterstützung sowie zahlreiche wertvolle Diskussionen und Anregungen bedanke ich mich ganz herzlich bei meinem Betreuer Dr. Lothar Ludwig. Vielen Dank auch an Torsten Zey, der mir bei Fragen hilfsbereit und fachkundig zur Seite stand.

Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem Thema *Workflow-Management-Systeme* (WfMS). Das Konzept der WfMS bildet eine wesentliche Grundlage für die Enterprise Application Integration (EAI). WfMS werden zur Unterstützung unternehmensweiter Geschäftsprozesse in verteilten heterogenen Systemumgebungen eingesetzt und bilden somit das Fundament für prozessunterstützende Software in vielen Unternehmensbereichen. Es wird eine bestehende WfMS-Implementierung (*iHub-WfMS*), die bei der Firma **iT media Consult GmbH** realisiert wurde und als WfMS-Prototyp verwendet wird, mit dem standardisierten Referenzmodell der *Workflow Management Coalition* (WfMC) verglichen. Die Untersuchung ergibt, dass der Prototyp in einigen Aspekten dem Standardisierungs-Vorschlag der WfMC abweicht. Allerdings ist die Funktionalität der spezifizierten Komponenten des Referenzmodells nicht weiter definiert.

Eine Workflow-Engine stellt den Kern eines jeden Workflow-Management-Systems dar. Der praktische Teil der Arbeit beschäftigt sich mit der Integration der Workflow-Engine des Prototyps in eine bestehende E-Commerce-Applikation (*eCCo*). Neben der Realisierung der Integrations-Komponenten wird eine Prozessdefinition der Applikations-Vorgänge erstellt sowie die Business-Logik der Applikation in die ausführenden Aktivitätseinheiten (TaskExecutables) eingebettet.

Der Schwerpunkt der Diplomarbeit liegt jedoch in der Untersuchung der Portierbarkeit des iHub-WfMS auf die J2EE-Plattform. Als ein Standard zur Entwicklung von Multi-Tier Enterprise Applikationen, gewährleistet der J2EE Applikationsserver hohe Verfügbarkeit, Skalierbarkeit und Ausfallsicherheit. Die Verwendung eines standardisierten serverseitigen Komponentenmodells verspricht mehrere Vorteile wie zum Beispiel Transaktionsverwaltung, Sicherheit und Persistenz der Geschäftsobjekte. Diese Aspekte werden bei der Planung des neuen Entwurfskonzepts *iHubEJB* einbezogen. In der Analysephase erweist sich die EJB-Technologie in vielerlei Hinsicht auch als Einschränkung, vor allem wegen der Restriktion des Multi-Threading. Das führt dazu, dass die Kommunikation zwischen der Workflow-Engine und ihrem Client auf den JMS-Standard (*Java Messaging Service*) umgestellt wird. Dafür werden Message Driven Beans der neuen EJB 2.0 verwendet. Darüber hinaus wird auf weitere Einschränkungen eingegangen, welche die EJB-Architektur mit sich bringt, sowie die Lösungen für diese Probleme beschrieben.

Der enorme Aufwand, der durch die Bean-Verwaltung und die Netzwerklast des verteilten EJB-Komponentensystems erzeugt wird, verzeichnet geringe Einbußen bei der Geschwindigkeit der iHubEJB-Applikation gegenüber des iHub-WfMS. Ein durchgeführter Geschwindigkeitsvergleich ergibt, dass die iHubEJB fast um das Vierfache langsamer ist als das iHub-WfMS.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACID	Atomicity Consistency Isolation Durability
Ant	Another Neat Tool
API	Application Programming Interface
AWT	Abstract Window Toolkit
B2B	Business-to-Business
BMP	Bean Managed Persistence
BPR	Business Process Reengineering
CMP	Container Managed Persistence
DAO	Data Access Objects
DBMS	Database Management System
DV-Systeme	Datenverarbeitungs-Systeme
EAI	Enterprise Application Integration
eCCo	Electronic Commerce Concept
EDV	Elektronische Datenverarbeitung
EIS	Enterprise Information System
EJB	Enterprise JavaBeans
ERP	Enterprise Resource Planning
HTTP	Hyper Text Transfer Protocol
ID	Identification
IDE	Integrated Development Environment
iHub	integration Hub
IIOP	Internet Inter-ORB Protocol
IT	Information Technology
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
Java VM	Java Virtual Machine
JCA	Java Connector Architecture
JDBC	Java DataBase Connectivity
JMS	Java Messaging Service
JMX	Java Management Extension
JNDI	Java Naming and Directory Interface

JSP	Java Server Pages
JTA	Java Transaction API
JTS	Java Transaction Service
MCA	Message Channel Agent
MDB	Message Driven Bean
MOM	Message Oriented Middleware
MQI	Message Queue Interface
MQM	Message Queue Manager
OODBMS	Object Oriented DBMS
ORB	Object Request Broker
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SMS	Short Message Service
SNA	System Network Architecture
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/ Internet Protocol
TP-Monitor	Transaction Processing Monitor
UML	Unified Modeling Language
VO	Value Object
WAP	Wireless Application Protocol
WF-Engine	Workflow-Engine
WfM	Workflow-Management
WF-Manger	Workflow-Manager
WfMC	Workflow Management Coalition
WfMS	Workflow-Management-System
XML	eXtensible Markup Language

Kapitel 1

Einleitung

Um die Wettbewerbsfähigkeit eines Unternehmens zu erhalten, muss flexibel auf neue Herausforderungen reagiert werden können. Aus betriebswirtschaftlicher Sicht werden Unternehmen durch kürzere Produkt- und Prozesslebenszyklen, zunehmende Produktdifferenzierung und enormen Kostendruck zum schnellen Handeln gezwungen. Zur Steigerung der Wettbewerbsfähigkeit kommt oft der Aufbau neuer internationaler Absatz- und Beschaffungsmärkte hinzu. Aus technologischer Sicht ist diese Steigerung erst durch eine große Integrationstiefe bestehender Anwendungen und wirtschaftliche Integration neuer Anwendungen möglich. Mit *Enterprise Application Integration* (EAI)-Tools kann dies durch ihre Fähigkeit zur Integration auf Prozessebenen und durch eine standardisierte Anbindung der Anwendungen erreicht werden. Viele Anwendungsfälle wie *Business Process Reengineering* (BPR), *Business-to-Business* (B2B)-Integration oder die Verlagerung von Service auf billigere Kanäle (z.B. Kundenservicecenter, Business-to-Employee Portale) demonstrieren beispielhaft, dass für ihren Einsatz alte und neue Anwendungen integriert und existierende Datenverarbeitungs (DV)-Systeme zugänglich gemacht werden müssen.

Als eine mögliche EAI-Lösung bietet sich eine Geschäftsprozessintegration mit Workflow an. Das Thema *Workflow-Management-Systeme* (WfMS) gewinnt zunehmend an Bedeutung und bildet ein Fundament für prozessunterstützende Software in vielen Unternehmensbereichen. WfMS werden zur Modellierung, Optimierung und Durchführung der Arbeitsvorgänge in Organisationen verwendet, dabei wird eine große Bedeutung auf leichte und schnelle Anpassbarkeit eines Geschäftsprozesses gelegt. Die primäre Aufgabe der WfMS ist die Unterstützung unternehmensweiter und -übergreifender Geschäftsprozesse, welche in heterogenen Systemumgebungen durch Integration unterschiedlichster Applikationen erreicht wird.

Die zunehmende Verbreitung verschiedener Workflow-Management-Systeme und der Mangel einer einheitlichen Beschreibung der WfMS hat Anbieter und Anwender sol-

cher Systeme im Jahre 1993 dazu veranlasst, die *Workflow Management Coalition* (WfMC) zu gründen. Die WfMC ist eine Organisation, die sich mit der Standardisierung von WfMS beschäftigt. Ihre bisherigen Ergebnisse umfassen ein *Referenzmodell* für die Architektur von WfMS und den Entwurf für eine Programmierschnittstelle.

Als Grundlage zur Realisierung des von der WfMC entworfenen Konzepts eignet sich die J2EE-Plattform (*Java 2 Enterprise Edition*). Die J2EE Technologie liefert einen Standard für die Entwicklung von verteilten Anwendungen in Applikationsservern. Die J2EE-Architektur stellt neben zahlreichen standardisierten Diensten wie JDBC (Java DataBase Connectivity), JMS (Java Messaging Service), JTA (Java Transaction API) auch ein Komponentenmodell mit Enterprise JavaBeans (EJB) zur Verfügung. Der *Java Messaging Service* liefert einen Zugriff auf die Welt der Message Oriented Middleware (MOM). JMS ermöglicht eine Kommunikation verteilter Anwendungen, welche unabhängig von proprietären Messaging APIs sind.

1.1 Aufgabenstellung

Ziel dieser Arbeit ist es zum einen, eine Bewertung eines vorhandenen Workflow-Management-Systems aus dem Projekt **iHub** der Firma iT media Consult GmbH bezüglich der Standardisierung der WfMC zu geben. Dabei soll überprüft werden, inwiefern die Implementierung mit dem Referenzmodell der WfMC konform ist. Weiterhin soll eine Anwendungsintegration des iHub-WfMS überprüft werden. Für diesen Zweck wird ein bestehender E-Commerce-Prototyp herangezogen.

Zum anderen soll die Workflow-Engine auf der Basis der EJB-Technologie realisiert werden. Dazu ist eine Analyse der Systementwicklung notwendig. Im Anschluss an die Analysephase sollen die identifizierten Komponenten und deren Interaktionen realisiert werden.

Kapitel 2

Workflow-Management-Systeme

Durch technologischen Fortschritt und anhaltenden Preisverfall für Rechner und Netzkomponenten ist es in den letzten Jahrzehnten zu einem verstärkten Einsatz von IT-Systemen gekommen, die das gemeinsame und verteilte Bearbeiten von Daten ermöglichen und unterstützen sollen. Ein weiterer Trend ist das Entstehen globaler Märkte im Zeitalter der sogenannten „New Economy“, der zum erhöhten Wettbewerb unter den Unternehmen geführt hat.

Diese Entwicklungen haben zur Folge, dass die Arbeitsabläufe innerhalb der Firmen gestrafft und optimiert werden. Dabei gewinnt der Entwicklungsprozess von Produkten an flexibleren und kürzeren Einführungsphasen. So kann z. B. durch verstärkte Orientierung an den Wünschen der Kunden oder als Reaktion auf Produkte der Konkurrenz das eigene Produkt noch vor Beginn der Serienfertigung nochmals verändert werden. Ein weiterer Trend ist die zunehmende Verbreitung von Unternehmen und ihrer Zulieferer in der ganzen Welt, sei es wegen Standortvorteilen, wie z. B. geringere Lohn- und Arbeitskosten, aber auch um zeitliche Vorteile bei der simultanen Entwicklung von Produkten zu ermöglichen. Neue Konzepte wie Lean Management[1], Total Quality Management[2] oder Business Process Reengineering[3] wurden entwickelt, um die Wettbewerbsfähigkeit der Unternehmen durch Reorganisation zu steigern.

Diese Reorganisation kann durch IT-Systeme unterstützt werden, die eine Ausführung von Geschäftsprozessen in einer verteilten Systemumgebung ermöglichen. Die Ent-

[1] **Lean Management** wird in Betrieben eingeführt, um durch die Beteiligung der Beschäftigten an innerbetrieblichen Entscheidungen deren Motivation und Bindung zu erhöhen, und die Effizienz der Entscheidungs- und Produktionsprozesse zu steigern [Sy].

[2] **Total Quality Management** ist ein strategisches Management-Konzept, das die Organisation von umfassender und durchgehender Qualität beinhaltet [Ra].

[3] **Business Process Reengineering** bedeutet soviel wie Neuorganisation und Optimierung der Geschäftsprozesse. Dadurch kann man in sehr vielen Unternehmen Kosteneffekte erzielen, indem man bestehende Geschäftsprozesse vereinfacht oder mit Hilfe von Technologieeinsatz komplett neu erfindet.

wicklung von Workflow-Management-Systemen hat sich der Herausforderung zur Unterstützung unternehmensweiter und –übergreifender Geschäftsprozesse gestellt. Nach [Ve95] wird davon ausgegangen, dass nur etwa 20 Prozent der Durchlaufzeit eines Prozesses zu seiner produktiven Bearbeitung genutzt werden, da die Arbeitsteilung und ihre Ablauflogik oft ineffektiv sind. Es sind zwei Entwicklungsrichtungen zu beobachten. Zum einen greifen betriebliche Standardanwendungen wie etwa SAP R/3 zunehmend Konzepte und Ideen von Workflow-Management-Systemen auf, zum anderen integrieren Workflow-Management-Systeme unterschiedliche betriebliche Arbeitsabläufe, die durch verschiedene Softwareprodukte realisiert oder auch manuell durchgeführt werden [Ph97]. Workflow-Management-Systeme werden als Werkzeuge für eine effiziente Abwicklung der Arbeitsvorgänge bei der Organisation angesehen, die eine ständige Weiterentwicklung der Prozesse erlaubt. Der entscheidende Vorteil solcher Systeme besteht darin, dass sie helfen, große vorgangsorientierte Anwendungssysteme überschaubar zu gestalten und in heterogenen Systemumgebungen die unterschiedlichsten Applikationen miteinander zu integrieren.

2.1 Entstehungsgeschichte

Workflow-Management-Systeme, auch Vorgangssteuerungssysteme oder Bürovorgangssysteme [Hase87] genannt, entstanden als solche erst in den 90er Jahren. Ihre Ursprünge können aber bereits in den späten 70er und in den frühen 80er Jahren gefunden werden als Büroautomationssysteme auf den Markt kamen [Bu97]. Die grundlegende Idee dieser Systeme war die Spezifizierung von Arbeitsschritten sowie Arbeitsabläufen mit Hilfe formaler Sprachen, die von geeigneter Software ausgeführt werden sollte. Der Mangel an günstigen, leistungsfähigen Rechner hat den Einsatz der Büroautomationssysteme im Wesentlichen verhindert. Auch die Softwaretechnik war noch nicht so fortgeschritten, als dass Netzwerk-Kommunikation oder auch Speichern von großen Datenmengen problemlos gewesen wäre.

Im Jahre 1991 wurde erstmals der Begriff „Workflow Management“ von Hales und Lavery [HaLa91] beschrieben. Es begann die Entwicklung von konfigurierbaren Workflow-Management-Systemen, die sehr schnell zu einer ganzen Reihe von Produkten führte [Bu97]. Im Grunde genommen verfolgten die WfMS dieselbe Idee wie die Büroautomationssysteme, jedoch war der Unterschied auf technischer Ebene gewaltig. Hard- und Software wurde zu diesem Zeitpunkt entwickelt. Anwendungen sind in der Regel konsistenzerhaltend und verteilt. Der Gedanke des Workflow-Managements wurde für so wichtig erachtet, dass dafür 1993 ein Standardisierungsgremium, die *Workflow Management Coalition* (WfMC), gegründet wurde.

Die ersten Implementierungen von WfMS wurden entweder als solche entworfen oder sind aus anderen Programmen entstanden. Solche „Ursprungsprogramme“ sind Dokumentenverwaltungssysteme, E-Mailsysteme, Textverarbeitungssysteme, Groupware-Produkte und persönliche Informationssysteme [Er93]. Die Tabelle 2.1 zeigt eine Gegenüberstellung der Systemherkunft mit typischen Workflowfunktionen.

	Ursprungs-Quelle	Einsatzgebiete	Beispiel
Gruppe A	operative Anwendungen	- stark standardisierte Arbeitsabläufe - konstante Aufgaben/Vorgänge, die keiner Dynamik unterliegen - hohe Fallzahlen	- SAP Business Workflow
	Dokumenten-management-systeme	- Vorgänge, die stark dokumentenorientiert sind - Dokumente, deren Originalabbild der Informationsträger ist und die u.U. auch rechtlich erforderlich sind - Vorgänge, deren Durchlaufzeit derzeit hauptsächlich durch die fehlende Möglichkeit des parallelen Zugriffs auf Dokumente beeinträchtigt wird	- Business Flow - Keyfile - Workflow Business System
	E-Mailsysteme/ Bürokommunikations-Systeme	- Vorgänge mit hoher Arbeitsteilung - Vorgänge, die über mehrere Standorte verteilt sind - Kommunikationsintensive Vorgänge über viele Mitarbeiter und unterschiedlichen Rechnersystemen hinweg, die nicht über ein Vorgangssteuerungssystem integriert werden können	- Serie/M - TeamWorks
	Textverarbeitungs-Systeme	- Vorgänge mit hohem Textbe- beziehungsweise -verarbeitungsanteil - Vorgänge mit einem hohen Individualitätsgrad der Vorgangsergebnisse	- Word Perfect Office
	Groupware-produkte	- Offene Vorgänge, bei denen hauptsächlich Standardsoftware-Werkzeuge eingesetzt werden - unstrukturierte gruppenorientierte Vorgänge	- LinkWorks - Lotus Notes
	persönliche Informations-Systeme	- als Prototypingumgebung - Vorgänge mit starkem „Einplatzbezug“ - Vorgangsablauf ohne besondere Anforderungen an den Kommunikationsbedarf mit Hintergrundsystemen - offene Vorgänge, bei denen hauptsächlich Standardsoftware-Werkzeuge eingesetzt werden	
	Imaging-Systeme	- Vorgänge mit wenigen Stufen - Vorgänge, die durch gescannte Dokumente vollständig abgearbeitet werden können und die keine andere Software benötigen	- OPEN/ Workflow
Gruppe B	Originäre Vorgangssteuerungs-Systeme	- standardisierte Arbeitsabläufe - Konstante Aufgaben/Vorgänge, die einer Marktdynamik unterliegen - hohe Fallzahlen - permanente Anpassung und Weiterentwicklung der Abläufe	- FlowMark - Flowpath - ProMInanD - Staffware - ViewStar - Workflow

Tabelle 2.1: Strukturierung der Vorgangssteuerungssysteme [We96]

Die Gruppe A fasst derivative Vorgangssteuerungssysteme mit Zusatzfunktionen aus ihren ursprünglichen Anwendungen zusammen. Unter Zusatzfunktionen sind beispielsweise Versionsverwaltungen von Dokumenten, Gruppenterminkalender oder Archivierungsfunktionen für Dokumente zu verstehen. Die Gruppe B enthält reine Vorgangssteuerungssysteme, die für diesen Zweck erstellt wurden. Für die Systeme dieser Gruppe sind umfangreiche Transport- und Schnittstellenfunktionen charakteristisch, um Abläufe abbilden und Fremdprogramme einbinden zu können. Beispielsweise bie-

ten diese Systeme Terminfunktionen[4], parallele Schrittausführung sowie Event-Möglichkeiten, d.h. das Warten eines Vorgangs auf ein bestimmtes Ereignis und die Möglichkeit des Vorgangsanstoßes durch dieses Ereignis, an.

Unter Berücksichtigung dieser Klassen können in Anlehnung an [Teu95] folgende Generationen unterschieden werden:

- erste Generation der derivativen Vorgangssteuerungssysteme
- zweite Generation der originären Vorgangssteuerungssysteme
- dritte Generation der aktuellen Workflow-Management-Systeme

In der ersten Generation stellen WfMS einen funktionellen Teil anderer Applikationen wie z.B. Dokumentenmanagement- oder Textverarbeitungssysteme dar. Es liegen statische Workflow-Definitionen vor und dabei handelt es sich um proprietäre Applikationen.

In der zweiten Generation sind es dagegen eigenständige Applikationen, die mit Hilfe von Script-Sprachen anpassbar sind und Werkzeuge zur Erstellung von Workflow durch Drittanbieter besitzen.

WfMS der dritten Generation bieten generische Dienste an. Der Zugriff auf WfMS-Funktionen kann über API erfolgen, was das Erstellen eigener Applikationen für das WfMS ermöglicht. Dadurch ist auch eine volle Integration von Werkzeugen von Drittanbietern möglich. Für die Workflow-Definition steht eine graphische Benutzeroberfläche zur Verfügung. Bestehende Workflows können mit dieser Technik zum einen anschaulich dargestellt und zum anderen leicht verändert werden.

2.2 Begriffserläuterungen

Dieser Abschnitt klärt die wichtigsten Begriffe im Zusammenhang mit Workflow-Management. Für den Begriff Workflow-Management bzw. Workflow-Management-Systeme werden viele Definitionen verwendet. Das liegt wohl daran, dass die Herkunft von Workflow-Management-Systemen aus verschiedenen Bereichen wie Dokumentenverwaltung, Produktionsplanung und -steuerung oder Groupware stammt. Hier werden einige Definitionsvorschläge für den Begriff *Workflow-Management* aufgeführt.

> *„Workflow management software is a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It coordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines. The co-ordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfill their*

[4] Berechnungen von absoluten und relativen Terminen mit oder ohne Bedingungen

> *required contributions, taking default actions when necessary.“* [HaLa91]

> *„The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“* [WfMC99]

Die zentrale Idee des *Workflow-Managements* (WfM) ist die EDV-unterstützte Automatisierung von Geschäftsprozessen und in diesem Zusammenhang das Ausführen von Handlungen wie Organisierung, Planung, Entscheidung, Kontrolle, Steuerung und Führung.

Unter *Geschäftsprozessen* versteht man größere durch abgegrenzte Themenstellungen definierte Einheiten des Geschehens in einer Organisation, die nach einem strategischem Aktionsprogramm durchgeführt werden und damit zum Geschäftserfolg des Unternehmens beitragen. Nach [Or97] wird ein Geschäftsprozess als folgendermaßen definiert:

> *„Ein Geschäftsprozess ist ein Vorgang in Wirtschaftseinheiten (Unternehmen, Verwaltung), der einen wesentlichen Beitrag zu einem nicht notwendigerweise ökonomischen Unternehmenserfolg leistet. Dabei läuft er in der Regel funktions-, hierarchie- und standortübergreifend ab, kann die Unternehmensgrenzen überschreiten und erzeugt einen messbaren, direkten Kundennutzen.“*

Ein *Workflow-Management-System* (WfMS) ist ein aus mehreren Werkzeugen bestehendes System, das die Aufgaben des WfM durch die Ausführung von Software unterstützt [Te95]. WfMS wird von der Workflow Management Coalition wie folgt charakterisiert:

> *„A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.“*[WfMC99]

Workflow-Management-Systeme unterstützen mit ihren Komponenten sowohl die Entwicklung (**Modellierungskomponente**) von Workflow-Management-Anwendungen als auch die Steuerung und Ausführung (**Laufzeitkomponente**) von Workflows, deren Reihenfolge von einer computergestützten Workflow-Logik bestimmt wird. Workflow-Management-Systeme bauen auf einem Modell des Workflows auf, das die einzelne Schritte des modellierten Geschäftsprozesses abbildet. Einzelne Workflow-Instanzen (im Sinne von Ausprägung) sind dann für die Ausführung der definierten Schritte verantwortlich. Dabei können sie auf Daten zugreifen, Applikationen integrieren und auf menschliche Ressourcen zurückgreifen.

Die Implementierung eines Workflow-Management-Systems wird als *Workflow-Management-Anwendung* bezeichnet. Im Gegenteil zu gewöhnlichen Anwendungssystemen, die zum Teil Sprachprodukte sind, unterscheiden sich die Workflow-Management-Anwendungen dadurch, dass sie neben den Softwareanteilen noch Organisationseinheiten wie zum Beispiel Mitarbeiter, Maschinen und vorhandene Applikationen enthalten.

Als *Workflow*, Geschäftsprozess oder Vorgang wird nach [Heil94] ein abgrenzbarer, meist arbeitsteiliger Prozess bezeichnet, der zur Erstellung oder Verwertung betrieblicher Leistungen führt. Das Workflow-Konzept hängt eng mit Business Process Reengineering und Automatisierung der Geschäftsprozesse zusammen, da ein Workflow Aktivitäten, die sich auf Teile eines Geschäftsprozesses beziehen, auf konzeptueller Ebene beschreibt. Besondere Aufmerksamkeit gilt bei Workflows auch der Integration von Daten und Applikationen über Organisationseinheiten hinweg.

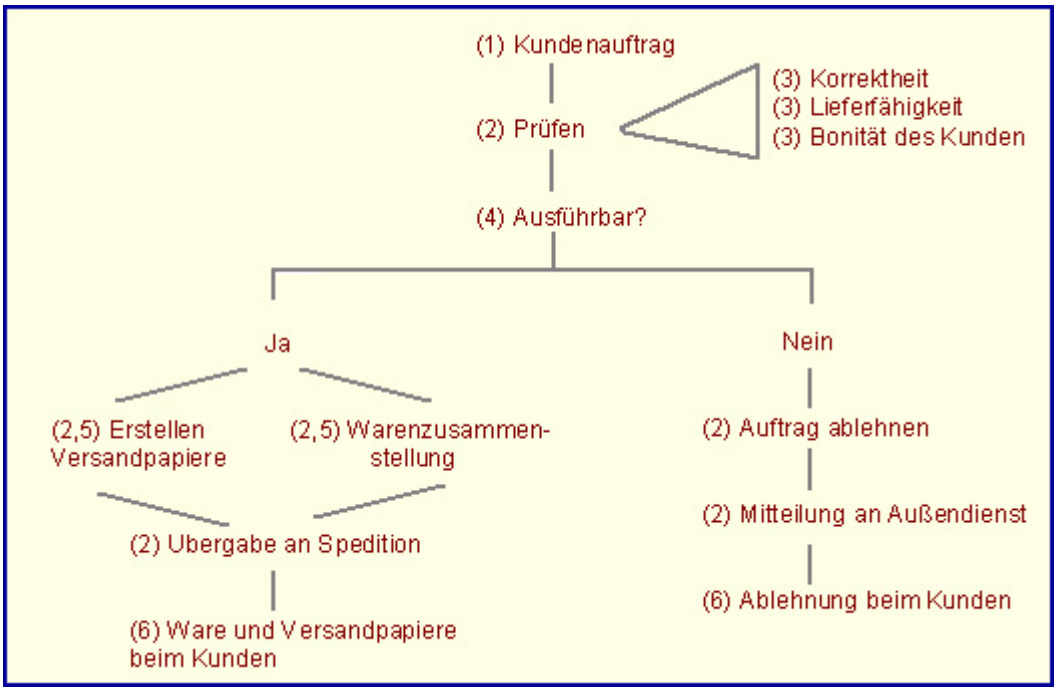

Abbildung 2.1: Abwicklung eines Kundenauftrages [Heil94]

Zur Verdeutlichung wird an dieser Stelle ein vereinfachtes Beispiel für einen Workflow besprochen. Im folgenden wird im Wesentlichen [Heil94] zitiert. Abbildung 2.1 zeigt eine Abwicklung eines Kundenauftrages ab Lager mit durchnummerierten Bezugspunkten.

- Ein Workflow entsteht durch einen Auslöser, auch Trigger (1) genannt. Im Allgemeinen handelt es sich dabei um einen Vorfall oder um das Erreichen oder das Ende eines Zeitpunkts. Es kann aber auch die Auslösung durch einen Menschen oder durch ein IT-Sys-

tem in Betracht kommen. Das Beispiel beschreibt hier den Eingang eines Kundenauftrags.

- Ein Workflow besteht aus verschiedenen Aktivitäten oder Aktionen (2), die gegebenenfalls auf mehreren Ebenen weiter zerlegt werden können (3).

- Ein Workflow kann, meist abhängig von Bedingungen, ganz oder in Teilen alternativ (4), aber auch parallel (5) ausgeführte Aktionen neben sequentiell ablaufenden enthalten.

- Ein Workflow hat einen eindeutigen Abschluss, der durch ein Ergebnis oder Ereignis gekennzeichnet ist. Auch mehrere Abschlussergebnisse sind ebenso wie ein Abschluss durch Abbruch denkbar. Im Beispiel endet der Workflow mit einem alternativen Ergebnis.

In der Literatur werden meist drei verschiedene Arten von Workflows unterschieden. Die Unterscheidung bezieht sich auf Klassifizierung des Formalisierungsgrads eines Workflows [Le98]. Es gibt:

- stark strukturierte Workflows,

- teilweise strukturierte Workflows und

- nicht strukturierte Workflows (Ad-hoc-Workflows).

WfMS unterstützen die Abarbeitung der verschiedenen Workflow-Arten in unterschiedlichem Umfang. Stark strukturierte Workflows sind für die Steuerung durch ein Workflow-Management-System gut geeignet. Die Abläufe eines stark strukturierten Workflows sind standardisiert und durch klar definierte Strukturen gekennzeichnet, die auch auf lange Sicht weitgehend stabil und planbar bleiben. Die Workflows werden ähnlich wie Transaktionen ausgeführt und erfordern deshalb keine flexible Steuerung.

Bei teilweise strukturierten Workflows liegt noch eine kontrollierbare Struktur und Komplexität vor, diese wird jedoch häufig durch individuelle Eingriffe der Benutzer verändert. Die Abläufe sind nicht mehr standardisiert, aber in der Regel bestimmbar. Teilweise strukturierte Workflows können insbesondere durch Dokumenten-Retrieval, E-Mail, Routineprüfungen und -anfragen unterstützt werden [Pi95]. Zur umfassenden Unterstützung teilstrukturierter Workflows wird die Entwicklung flexibler Workflow-Management-Systeme gefordert.

Der Einsatz von nicht strukturierten Workflows macht nur wenig Sinn, da sowohl die Aufgaben als auch der Ablauf des Prozesses kaum geplant werden können [Pi95]. Die Benutzung von WfMS zur Unterstützung dieser Workflow-Art ist umstritten [Le98]. Des öfteren werden passive Groupware zur Kommunikation und Kooperation der Aufgabenträger zur Verfügung gestellt, die die Steuerung den beteiligten Menschen über-

lassen. Ein WfMS kann lediglich Informationsressourcen und Kommunikationsunterstützung zur Verfügung stellen [Pi95].

2.3 Architektur

Workflow-Management hat als solches keinen eindeutigen Ursprung. Über die Jahre hinweg entstand ein Entwurf eines Systems zur Automatisierung von Geschäftsprozessen, das bei den Unternehmen als vielversprechend galt. Dabei gab es zu Beginn nur eine vage Vorstellung von den Funktionalitäten des Systems, also „was“ das System genau machen sollte. Die Vorstellung über den Inhalt und den Aufbau eines Workflow-Management-Systems nimmt in Diskussionen eine beinahe beliebige Bandbreite an. In kurzer Zeit ist eine Menge an kommerziellen Workflow-Management-Systemen (z.B. FlowMark von IBM, WorkParty von Siemens Nixdorf) und Forschungsprototypen entstanden, welche mehr einer spezifischen punktuellen Problemlösung folgen, als sich an einer fundierten theoretischen Basis zu orientieren.

2.3.1 Perspektiven des Architekturbegriffs

In [Ja97] wird ein allgemeines Konzept einer Basisarchitektur vorgestellt. Aus dem Architekturbegriff können drei Perspektiven in Bezug auf ein Workflow-Management-System abgeleitet werden:

- Systeminfrastruktur,
- Benutzer,
- Implementierung.

Aus der **Sicht einer Systeminfrastruktur** entspricht ein Workflow-Management-System einem Framework. Abbildung 2.2 zeigt dazu einen graphischen Überblick der Framework-Architektur. Ein *Framework* wird als eine Softwareumgebung definiert [Be96], die dazu dient, Anwendungen effektiv und effizient entwickeln zu können. Es stellt ein halbfertiges System dar, mit definierten Schnittstellen (API) und einer Menge von Werkzeugen. Frameworks werden auf Middleware aufgesetzt und können deren allgemeine Dienste in Anspruch nehmen. Darüber hinaus können Frameworks spezialisierte Middleware-Dienste zur Verfügung stellen.

Middleware-Dienste sind gemäß [Be96] solche Dienste, die zwischen der Systemplattform und den Anwendungen lokalisiert sind. Middleware-Dienste sind charakterisiert durch eine Systemplattform- und Anwendungs-Unabhängigkeit. Ihr Einsatzgebiet liegt in der verteilten Systemumgebung. Der Ressource-Manager für Dateisysteme oder relationale Datenbanken, TP-Monitore, welche Transaktionsdienste zur Verfügung stellen, Objekt-Broker-Dienste und Kommunikationsdienste wie z.B. RPC sind Middleware-Dienste. Auch Workflow-Management kann als ein Framework angesehen wer

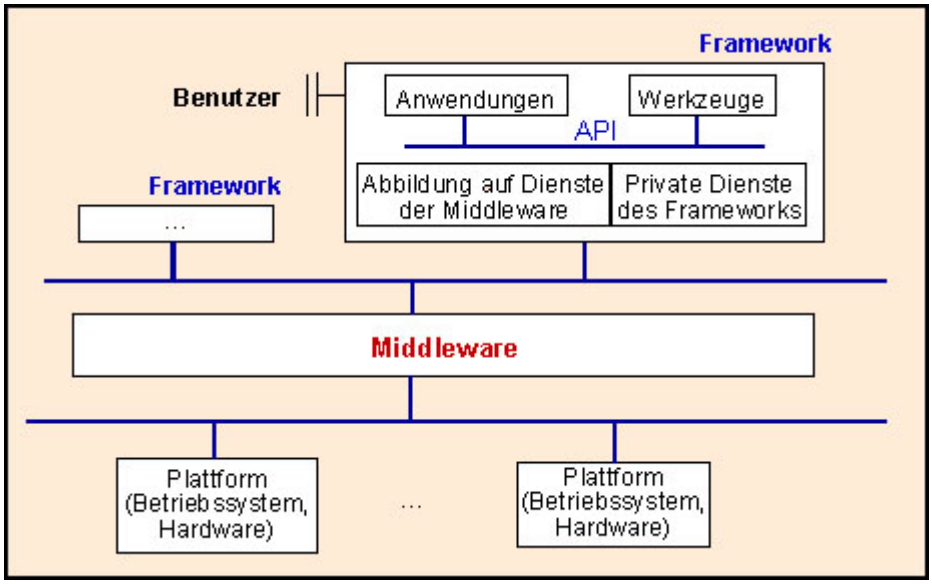

Abbildung 2.2: Architektur eines Frameworks [Be96]

den, das sich dieser Dienste bedienen kann. Darüber hinaus können von einem Workflow-Management-System weitere Dienste aufgebaut werden, wie zum Beispiel „Kontrollflusssteuerung“, „Datenflusssteuerung“, „Auflösung organisatorischer Zuordnungsstrukturen“, „Historienverwaltung“ oder „Applikationsintegration“.

Aus der **Perspektive des Benutzers** erscheint die Architektur eines Workflow-Management-Systems als eine Anordnung von Werkzeugen, welche ihm zur Modellierung und Ausführung von Workflows angeboten werden. Es werden verschiedene Gruppen von Benutzern unterschieden. Da gibt es Programmierer, Endanwender, Administratoren und Systemverwalter. Jeder dieser Benutzergruppen stehen Werkzeuge zur Verfügung. Tabelle 2.2 fasst die wesentlichen Werkzeuge für diese Benutzergruppen zusammen. In der Systembenutzung werden zwei Phasen unterschieden: die **Modellierungs**- und die **Ausführungsphase**.

	Programmierer	**Endanwender**	**Administrator**	**Systemverwalter**
Modellierungsphase	Editor Browser Übersetzer Debugger	-	Simulator Animator Typ-Organisator	Konfigurator
Ausführungsphase	Ablaufmonitor	Arbeitsliste		Ablaufmonitor Konfigurator

Tabelle 2.2: Werkzeugkasten für die Benutzergruppen [Ja97]

Während der Modellierungsphase werden *Workflow-Schemata* erstellt, wobei ein Workflow-Schema gemäß [Or97] eine zielgerichtete Anordnung von Begriffen zur Beschreibung, Ausführung und Steuerung von Workflows auf sprachlicher Ebene bzw. Zeichenebene darstellt. Ein solches Schema wird in der Ausführungsphase instantiiert,

um konkrete Workflows, also Abläufe, zu realisieren. Workflows werden von Programmierern definiert und von Endanwendern ausgeführt. Administratoren überwachen die Ausführung von Workflows, analysieren und optimieren die Abläufe. Zu den Aufgaben der Systemverwalter gehört die Installation und Konfiguration des Workflow-Management-Systems.

Jeder Benutzergruppe werden bestimmte Werkzeuge zugeordnet, die bei der Ausführung der jeweiligen Tätigkeit als Hilfsmittel eingesetzt werden können. Allgemein gesprochen, entspricht ein Werkzeug einer bestimmten Konfiguration, Anordnung oder auch Bündelung von Funktionen am API eines Workflow-Management-Systems. Im Zusammenhang mit der Systeminfrastruktur handelt es sich hier um eine frameworkcharakteristische Schnittstelle, durch die die Funktionalität des Frameworks angeboten wird.

Aus der **Sicht der Implementierung** eines Workflow-Management-Systems stellt sich ein solches als eine Menge von *Modulen* (Komponenten) mit wechselseitigen Beziehungen dar. Die Funktionalität eines Workflow-Management-Systems wird durch die Gesamtheit aller Module realisiert. Jedes dieser Module beziehungsweise eine Gruppe von Modulen implementiert ein Strukturelement des *Strukturmodells*, das die wichtigsten Bestandteile eines Workflows kategorisiert.

Das allgemeine Strukturmodell orientiert sich an einzelnen *Aspekten*, welche in ihrer Gesamtheit einen Workflow darstellen. In [Ja97] werden einige Aspekte beispielhaft vorgestellt (vgl. Abbildung 2.3). Der funktionale Aspekt beschreibt die funktionalen Ausführungseinheiten eines Workflows, welche zugleich die Rahmenstruktur bilden. Der verhaltensbezogene Aspekt eines Workflows bildet den Kontrollfluss zwischen den einzelnen Workflows. Der zwischen den Workflows zu realisierende Datenfluss wird im datenbezogenen Aspekt eingerichtet. Der organisatorische Aspekt ist für organisatorische Zuordnungen zuständig, beispielsweise Zuweisungen von Personen, die einen Workflow ausführen sollen. Die Einbindung von Applikationen in einen Workflow wird unter dem operationalen Aspekt diskutiert.

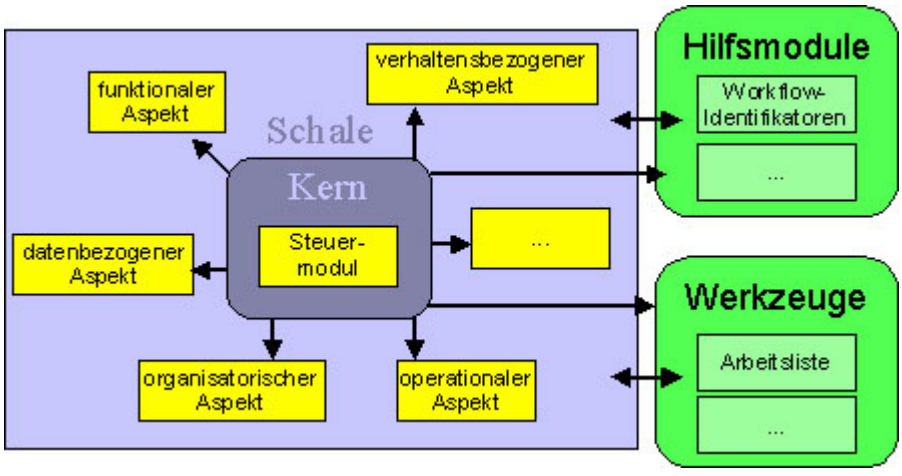

Abbildung 2.3: Architektur aus Implementierungssicht [Ja97]

Die Architektur eines Workflow-Management-Systems aus der Implementierungs-Perspektive kann in **Software-** sowie **Implementierungsarchitektur** unterteilt werden. Die *Softwarearchitektur* umfasst die wesentlichen logischen Komponenten (Module) eines Softwaresystems einschließlich ihrer gegenseitigen Beziehungen. Die Softwarearchitektur beschreibt letztendlich, wie die Funktionalität des Frameworks Workflow-Management-System erbracht wird. Die API des Frameworks ist Voraussetzung für die Realisierung der Werkzeuge, welche von den Benutzergruppen bei der Ausführung ihrer Tätigkeiten verwendet werden. Die Komponenten werden während der Analyse der Problemstellung, die eine Lösung im Ansatz schildert, identifiziert. Diese Systementwicklung vollzieht sich in den Analyse- beziehungsweise in den Entwurfsphasen. Im Anschluss erfolgt die *Implementierungsphase*, die die identifizierten Komponenten (Module) und deren Interaktionen realisiert. Darunter kann beispielsweise verstanden werden, dass zwei Komponenten in einer Client/Server-Beziehung zueinander stehen sollen. Darüber hinaus kann festgelegt werden, dass die dabei zu realisierende Kommunikationsverbindung asynchron über ein „hochverfügbares“ Medium erfolgen soll. Eine solche Analyse in der Implementierungsphase ergibt beispielsweise die Auswahl einer persistenten Warteschlange als Kommunikationskonzept zwischen den Komponenten. Das Ergebnis dieser Implementierungsphase ist die *Implementierungsarchitektur*.

Somit ist eine dritte Interpretation des Architekturbegriffs eingeführt, welche anhand von Abbildung 2.3 verdeutlicht wird. Die zentrale Bedeutung kommt hier einem Steuermodul zu, das für die Ausführung eines Workflows verantwortlich ist. Darüber hinaus koordiniert dieses Modul die Auswertung einzelner Aspekte bei der Bearbeitung eines Workflows. Dafür werden folgende Schritte durchgeführt.

- Aufruf der Komponente „funktionaler Aspekt“, welche die Ausführung eines Workflows beschreibt.
- Aufruf der Komponente „verhaltensbezogener Aspekt“, welche die Menge der als nächstes auszuführenden Subworkflows bestimmt.
- Aufruf der Komponente „datenbezogener Aspekt“, welche die Menge der auszuführenden Workflows mit Eingabeparameter versorgt.
- Aufruf der Komponente „organisatorischer Aspekt“, welche die Menge der organisatorischen Elemente beschreibt, die auszuführende Workflows abarbeiten dürfen.
- Aufruf der Komponente „operationaler Aspekt“, wenn ein Applikationsprogramm innerhalb eines elementaren Workflows auszuführen ist.

Das Steuermodul bildet den **Kern** eines Workflow-Management-Systems. Die ihn umgebende **Schale** enthält die verschiedenen Module, welche sämtlichen Aspekte des Strukturmodells beschreiben. Die Module des Kerns und der Schale realisieren die Basisfunktionalität eines Workflow-Management-Systems, sie werden daher *Basismo-*

dule genannt. Sie dienen zum einen der Definition von Werkzeugen und zum anderen der Realisierung von Anwendungen, die innerhalb des Frameworks Workflow-Management-System anzusiedeln sind und mit dessen Schnittstellen (API) verknüpft werden.

Neben den Basismodulen stehen noch spezielle Module zur Umsetzung zusätzlicher Funktionen zur Verfügung. In der Modulgruppe „Werkzeuge“ werden Funktionen realisiert, die Funktionalitäten der Basismodule in Bezug auf Werkzeuge der Benutzergruppen erweitern. Zum Beispiel wird im Werkzeug „Arbeitsliste“, das die Schnittstelle eines Endbenutzers zum Workflow-Management-System darstellt, eine Funktion zum Aktualisieren seiner Einträge benötigt. Die Modulgruppe „Hilfsmodule“ realisiert Hilfsfunktionen, wie beispielsweise eine Funktion zur Vergabe von eindeutigen Workflow-Identifikatoren. Die in der Abbildung 2.3 gezeigten Pfeile bezeichnen eine „benutzt“-Beziehung. Alle Module der Schale kommunizieren miteinander lediglich über das Steuermodul. So muss bei der Änderung eines Moduls nur die Verbindung zwischen dem Modul und Steuermodul aktualisiert werden.

2.3.2 Schichtenarchitektur

Beim Softwareentwurf großer Anwendungssysteme herrscht in der Fachwelt über die generelle Vorgehensweise und insbesondere über die Umsetzung bewährter Prinzipien des Software Engineering weitgehende Übereinstimmung. Zur Reduktion der Komplexität sollte der Entwurf auf eine durch Schichten von Abstraktionen gekennzeichnete Architektur mit klar definierten Spezifikationen führen. Diese Architektur erfüllt die Voraussetzungen für wesentliche Systemeigenschaften wie Modularität, Anpassbarkeit, Portabilität und Erweiterbarkeit.

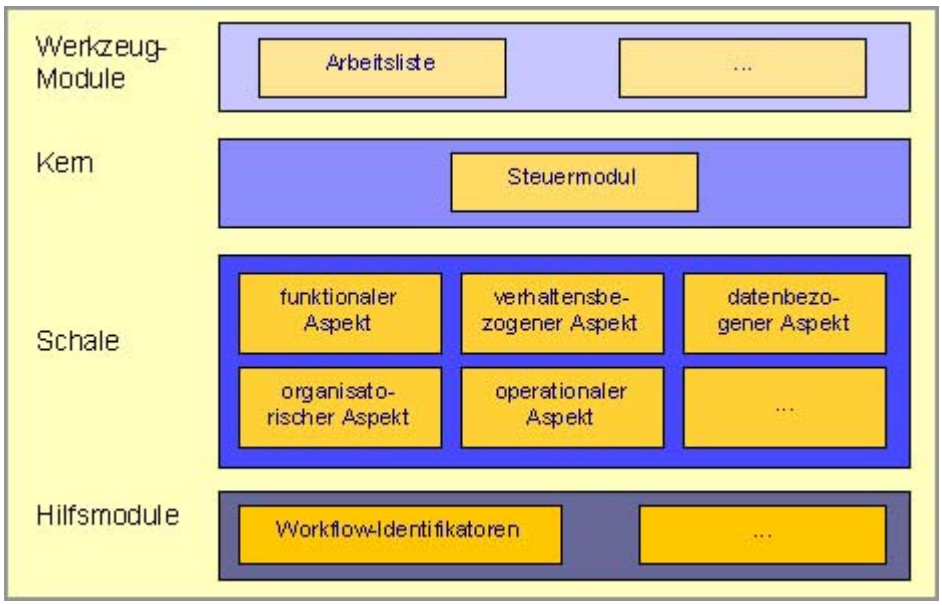

Abbildung 2.4: Schichtenarchitektur eines WfMS [Ja97]

Im Bereich Software Engineering findet man allgemeine Konzepte und Empfehlungen wie das Geheimnisprinzip[5] oder die hierarchische Strukturierung[6] zur Entwicklung einer Systemarchitektur. Daraus lassen sich wichtige Erkenntnisse herleiten, wie große Software-Systeme auf einer Schichtenarchitektur aufzubauen sind. Eine Schichtenarchitektur zeichnet sich durch mehrere hierarchisch angeordnete Schichten aus. Dabei „benutzt" die Schicht *i+1* die Operatoren und Datenobjekte, die Schicht *i* „realisiert". Somit stellt eine Schichtenarchitektur eine Veredelungshierarchie dar, welche sukzessive Objekte mit Funktionalität anreichert.

Ordnet man die Module des Workflow-Management-Systems in eine Schichtenarchitektur ein, wie es in Abbildung 2.4 zu sehen ist, so wird die unterste Schicht durch die sogenannten Hilfsmodule gebildet. Die Hilfsmodule wurden eingeführt, um grundlegende Funktionen für alle anderen Module zu realisieren (vgl. Abbildung 2.3). Die Schale bildet die zweite Schicht. Sie benutzt die primitiven Operationen der Hilfsmodule, um Operationen zu konstruieren, die die einzelnen Workflow-Aspekte bearbeiten. Zum Beispiel wird eine Operation angeboten, welche den Kontrollfluss eines Workflows realisiert. So eine Operation kann unter anderem die Reihenfolge der Workflows berechnen. Auf der nächsthöheren Schicht ist der Kern mit dem Steuermodul angesiedelt. Er fasst Aspekt-Operationen zu Workflow-Operationen zusammen. Eine Workflow-Operation führt beispielsweise einen Workflow aus oder versetzt einen Workflow in einen Wartezustand. Die letzte Schicht enthält Werkzeugmodule und bietet Operationen auf Workflows an. So können mit Hilfe des Werkzeugs „Ablaufmonitor" alle gegenwärtig beschäftigten Workflows aufgezeigt werden.

Eine Schichtenarchitektur weist eine Reihe von Vorteilen auf, die sich als Konsequenz aus der Nutzung von hierarchischen Strukturen und durch die „benutzt"-Beziehung ergeben. Werden Änderungen auf höheren Schichten vorgenommen, so nehmen sie keinen Einfluss auf tiefere Schichten. Das Umgekehrte gilt jedoch nicht, d.h. wenn die Komponenten einer unteren Schicht modifiziert werden, so sind Änderungen von höheren Modulen sehr wahrscheinlich. Komponenten der gleichen Hierarchieebene beeinflussen sich lediglich dann, wenn sie gegenseitig die Realisierung bestimmter Funktionen übernehmen würden. Die Komplexität der höheren Ebenen nimmt ab, wenn sie die Funktionalitäten der tiefen Ebenen benutzen können. Darüber hinaus lassen sich höhere Schichten ohne Aufwand abtrennen und die tieferen Schichten bleiben trotzdem funktionsfähig.

2.4 Anforderungen

Workflow-Management-Systeme sind spezielle Anwendungssysteme, die der Automatisierung von Geschäftsprozessen in Unternehmen dienen. Damit werden die von den Unternehmen zu erfüllenden Anforderungen unterstützt. Beispielsweise werden hier einige wichtige Anforderungen nach [Bu97] aufgeführt:

[5] Unter dem **Geheimnisprinzip** (information hiding) versteht Parnas [Pa72], Entwurfsentscheidungen, die sich ändern können, vor dem Rest des Systems zu verbergen.
[6] Parnas [Pa72] empfiehlt eine hierarchische Zerlegung eines Programms in Module, wobei er auf das THE-System von Dijkstra verweist, das aus hierarchischen Schichten aufgebaut ist.

- **Flexibilität**: schnelle Anpassung von Unternehmen an geänderte Rahmenbedingungen wie Rohstoffkosten, Gesetzgebung, Änderungen auf dem Weltmarkt.

- **Konkurrenzfähigkeit**: sehr effiziente und effektive Arbeitsweise, um die Produktion schneller, kostengünstiger und/oder qualitativ hochwertiger gestalten zu können.

- **Zuverlässigkeit**: sichere und zuverlässige Arbeitsabläufe trotz hoher Komplexität der Vorgänge und Verwaltungsvorschriften.

Zwischen den Unternehmensanforderungen und den Anforderungen der Workflow-Management-Systeme lassen sich Parallelen aufzeigen. WfMS zeichnen sich durch ihre **Flexibilität** aus, da sie in verschiedenen Anwendungsgebieten eingesetzt werden können. Dies wird bereits dadurch erkennbar, dass das Workflow-Management zur Middleware gerechnet wird [Ja97].

Eines der wesentlichen Charakteristika einer Middleware-Technologie ist ein breites Einsatzspektrum. Aufgrund der hohen Anzahl von Systemen mit unterschiedlicher fachlicher Ausdrucksfähigkeit werden WfMS in vielen Gebieten eingesetzt. Das hat zur Folge, dass die Funktionalität eines Workflow-Management-Systems erweitert werden muss. Genauer gesagt werden dem System neue Komponenten hinzugefügt beziehungsweise werden bestehende Komponente ausgetauscht. Das führt zu einer weiteren Anforderung an das WfMS, nämlich die **Erweiterbarkeit** des Workflow-Modells. Oft müssen zur Ausführung eines Workflow-Ablaufs sogenannte Altsysteme (legacy systems) integriert werden. Daher ist **Offenheit** des Workflow-Management-Systems notwendig, um dieser Anforderung zu genügen. WfMS eignen sich besonders für solche Anwendungsgebiete, die durch einfach strukturierte Routineprozesse gekennzeichnet sind. Diese Prozessen besitzen einen hohen Repetitionsgrad und sie bleiben auf lange Sicht stabil. Da die Anzahl der parallel auszuführenden Workflow-Instanzen einen Wert von mehreren Hundert oder sogar Tausende aufweisen kann, ist dessen **Skalierbarkeit** ein weiteres Kriterium.

Zu den systembezogenen Anforderungen zählt die Beherrschung einer **verteilten** und **heterogenen Plattform**, was sich auf den Einsatz von Workflow-Management-Systemen an verteilten Standorten von Unternehmen und öffentlichen Verwaltungen [Re94] zurückführen lässt. Darüber hinaus spielt die **Portabilität** der Implementierung eines WfMS eine weitere Rolle, um das Anwendungssystem in einem möglichst breiten Feld einsatzfähig zu machen. Dies sind zwei weitere Gründe, warum das Workflow-Management im Sinne eines Frameworks auf allgemeine Middleware-Dienste basiert. Die Ausführungszeit eines Workflows kann je nach Vorgabe unterschiedlich lang sein. Sie kann im Bereich von Stunden, Tagen oder Wochen liegen. Daher sind Fehlerfälle oder ein Systemversagen bei der Workflow-Ausführung leicht möglich. Eine einfache ACID[7]-Transaktionssemantik, bei der alle in Arbeit befindlichen Tätigkeiten abgebro-

[7] ACID steht für **A**tomicity – eine Transaktion wird entweder vollständig ausgeführt oder überhaupt nicht, **C**onsistency – eine Transaktion überführt das System von einem konsistenten Zustand in einen anderen konsistenten Zustand, **I**solation – die Auswirkungen einer Transaktion werden erst nach ihrer

chen und eventuell bereits stattgefundene Datenänderungen automatisch rückgängig gemacht werden, ist nicht anwendbar, da hier der Arbeitsaufwand in keinem Verhältnis zum Ergebnis stehen würde. Diese Überlegung führt zu einer weiteren wichtigen Anforderung, die sich auf die Zuverlässigkeit des Ausführungssystems bezieht. Hier sind daher semantisch reichere Fehlerbehandlungsmodelle gefragt.

2.5 Workflow-Modellierung

In den letzten Jahren ist die Tendenz zur Verwendung von Anwendungssystemen unter Einsatz von Basissystemen wie Datenbank-Management-Systemen (DBMS) oder Workflow-Management-Systemen (WfMS) für Anwender-Unternehmen erheblich gestiegen und die sogenannte „freie Anwendungsentwicklung“ in einem Programmiersystem ohne Basissystem deutlich gesunken. So entwickelt sich der objektorientierte Ansatz (z.B. objektorientierte Datenbank-Management-Systeme), der - als er aktuell war - zu neuen Programmiersprachen geführt hat, immer mehr zu einer basissystemorientierten Anwendungsentwicklung.

Mit **Basissystemen**, die die Implementierungsgrundlage für die Anwendungssysteme bilden, ist immer ein bestimmter Lösungsansatz der Anwendungssystementwicklung verbunden. So sind daten-, objekt- und prozessorientierte Ansätze, neben dem traditionellen funktionsorientierten Ansatz aus den 60er Jahren, auf die Entwicklung von Basissystemen wie zum Beispiel DBMS, OODBMS oder WfMS zurückzuführen. Bei der Entwicklung von anspruchsvollen Anwendungssystemen, zu denen auch die Workflow-Management-Anwendung zählt, müssen heutzutage nahezu alle Lösungsansätze beherrscht werden.

Für die Entwicklung der Workflow-Management-Systeme werden Ergebnisse aus fast allen Teilbereichen der Informatik herangezogen. Die Anwendung der WfMS beeinflusst nicht nur die Beziehung zwischen Geschäftspartnern auf nationaler und internationaler Ebene, sondern führt zu einer grundlegenden Veränderung der Organisation und Steuerung von Arbeitsabläufen in Unternehmen und Behörden. Dieser Sachverhalt erklärt die Notwendigkeit das Workflow-Management unter Gesichtspunkten weiterer Disziplinen wie Betriebswirtschaftslehre (insbesondere die Organisationslehre), Arbeitswissenschaft und Rechtswissenschaft zu betrachten [Le98]. Die Organisationslehre beschäftigt sich mit der Effizienz der zu gestaltenden Arbeitsabläufe und des Einsatzes von WfMS; die Rechtswissenschaft befasst sich in diesem Zusammenhang mit Fragen der Einhaltung von denjenigen Normen, die über technische Bedingungen und organisatorische Festlegungen hinausgehen, wie zum Beispiel die Anforderungen bezüglich der internen Revision an WfMS [Ph97]. Die Arbeitswissenschaft beschäftigt sich mit berufs- und beschäftigungsbedingten Einflussfaktoren auf den Menschen und der Gestaltung von Arbeitssystemen.

Eine interdisziplinäre Betrachtung der Modellierung von Workflows kommt besonders dann zum Tragen, wenn diese Modellierung nicht isoliert von anderen Maßnahmen zur

erfolgreichen Beendigung für andere Transaktionen sichtbar und **D**urability – die Auswirkungen einer erfolgreich beendeten Transaktion gehen nicht verloren.

Organisationsentwicklung, die wiederum Forschungsgegenstände benachbarter Disziplinen, etwa der Betriebswirtschaftslehre und hier insbesondere der Organisationslehre, sind, erfolgen soll.

Die Entwicklung von Workflow-Management-Anwendungen umfasst vor der eigentlichen Workflow-Modellierung nach [Le98] die Gebiete „Unternehmensgestaltung“ und „Organisationsmodellierung“, vgl. Abbildung 2.5. Aus der *Organisationsmodellierung* (z.B. eine Modellierung der Arbeitsabläufe) entwickelt sich die spezifische Aufgabe der *Informationssystementwicklung*, in die die Modellierung von Workflows einzuordnen ist. Die implementierte und eingeführte Workflow-Management-Anwendung wird als Lösung in die Aufbau- und Ablauforganisation eines Unternehmens integriert. Zu den wesentlichen Aufgaben der Organisationsmodellierung gehört die Bildung von Organisationseinheiten, die Zuordnung von Stellen (Aufbauorganisation) und im Detail festgelegte Arbeitsabläufe. Die Aufbau- und Ablauforganisation wird nach [Ko76] in der Arbeitsanalyse entwickelt, die als zentrales Instrument zur Gestaltung der Ablauforganisation verwendet wird. Einzelne Arbeitsvorgänge wie „Mahnung schreiben“ oder „Rechnung prüfen“ bilden das Ergebnis der Arbeitsanalyse und werden im Rahmen der Arbeitssynthese zu Arbeitsabläufen zusammengefasst. Aus den Arbeitsvorgängen lässt sich in der Aufgabenanalyse eine Aufgabenstruktur ableiten. Die Aufgabenstruktur ist dann für die Stellenbildung, die Bildung von Arbeitsgruppen und Organisationseinheiten und damit für die Aufbauorganisation (Aufgabensynthese) relevant.

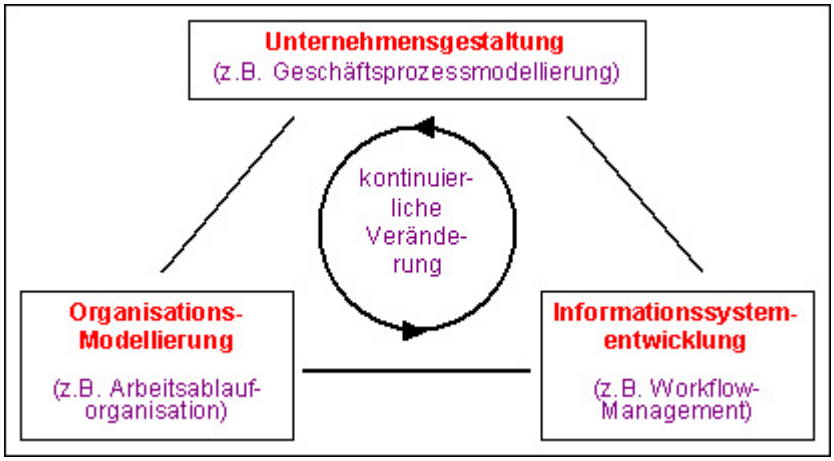

Abbildung 2.5: Modellierungsprozess der Organisationsstrukturen [Le98]

Allerdings wird aus heutiger Sicht der Geschäftsprozessmodellierung eine primäre Rolle zugeschrieben. Daher erfolgt sie vor der Organisationsmodellierung. Geschäftsprozessmodellierung gehört zum Aufgabegebiet der *Unternehmensgestaltung* und ist der Kern von Business Engineering[8] [Ha93]. Geschäftsprozesse sind im Allgemeinen nicht in der detaillierten Form festgelegt, so dass sie in Teilbereichen des Unternehmens oder ganz durch ein Workflow-Management-System unterstützt werden können. Deshalb ist an dokumentierten Spezifikationen der Geschäftsprozesse wie z.B. „Wert-

[8] Business Engineering steht für eine systematische Entwicklung neuer Geschäftsprozesse. Dabei zerlegt sie die Transformation von Unternehmen in beherrschbare Organisationseinheiten, gibt Anleitung zur Bearbeitung dieser Einheiten und verbindet diese in Vorgehensmodellen für Projekte.

papierhandel einer Bank“ noch nicht erkennbar, ob ein WfMS zum Einsatz kommen kann oder ob eine weitergehende Modellierung des Geschehens in dem betreffenden Bereich überhaupt möglich ist. Dies wird in der Regel erst in der globalen Phase „Organisationsmodellierung“ erkannt.

Der in Abbildung 2.5 gezeigte Modellierungsprozess besteht nicht aus einer sequentieller Folge von Aktivitäten, sondern basiert auf Rücksprüngen und gegenseitiger Beeinflussung der Teilgebiete „Unternehmensgestaltung“, „Organisationsmodellierung“ und „Informationssystementwicklung“. Während der Geschäftsprozess- und der Organisationsmodellierung treten Situationen auf, in denen über den Einsatz technischer Hilfsmittel zur Zielerreichung (z.B. Steuerung von Arbeitsprozessen) entschieden wird. Die Bereitstellung der technischen Mitteln liegt im Aufgabengebiet der Informationssystementwicklung. Umgekehrt wirken sich die Möglichkeiten des Einsatzes technischer Mittel auf die Organisation von Arbeitsprozessen und die Geschäftsprozessmodellierung aus.

Die *Modellierung* von Workflows dient unter anderem der Beschreibung beziehungsweise der Verdeutlichung des Geschehens in der Realität. Die Modellbildung dient weiterhin der Kommunikation, beispielsweise um anderen Personen das Ergebnis dessen, was der Modellierer sich vorgestellt hat, zu vermitteln. Das Ergebnis der Modellbildung wird als *Workflow-Schemata* bezeichnet [Bö97]. Workflow-Schemata werden von Workflow-Entwicklern erstellt und von Workflow-Management-Systemen verwendet. Auf diese Art und Weise entsteht eine Kommunikation zwischen dem Produzenten (Mensch) und dem Konsumenten (Maschine), die vorwiegend in eine Richtung geht. Das Ziel der Workflow-Modellierung besteht darin, bestimmte Aspekte und Inhalte von Arbeitsvorgängen in einer vom WfMS ausführbaren Form zu beschreiben. Die Modellierung ist somit sehr eng an das WfMS gebunden und kann nur im Rahmen dieses Kontextes stattfinden. Die Systeme enthalten bereits Workflow-Sprachen, die die Möglichkeiten bei der Modellierung von Workflows ganz unmittelbar bestimmen können.

Wie bereits erwähnt, beschreibt ein Workflow eine Gesamtheit von Aktivitäten, die sich auf Teile eines Geschäftsprozesses oder auf andere organisatorische Vorgänge beziehen. Ein Workflow ist wie ein Vorgang ein Geschehen in der Realität, das in der Vergangenheit stattgefunden hat, gerade abläuft oder für die Zukunft geplant ist. Dies kann anhand eines einfachen Beispiels „Genehmigungsvorgang in einer Bank“ veranschaulicht werden.

> Der Genehmigungsvorgang der Bank für den Kredit eines Kunden bleibt so lange ein (zu bearbeitender) Vorgang, bis die ablaufenden Arbeitsschritte (Kundeninformation einholen, kreditbezogene Daten zusammenstellen, Risiko einschätzen usw.) durch ein Workflow-Management-System ganz oder teilweise unterstützt werden.

Anhand dieser Bestimmung wird ein generelles Verfahren dargestellt, dessen „Idee“ die Vorgehensweise bei Kreditanträgen in dieser Bank beschreibt. Das Workflow-Management-System soll das Verfahren eigenständig unter Beachtung aller Geschäftsbedingungen, Vorschriften und Gesetze durchführen. Das gilt dann für alle Kunden der

Bank. Die Idee hinter so einem Verfahren, das durch ein Workflow-Management-System unterstützt werden soll, wird als *Workflow-Schema-Modell* bezeichnet [Bö97]. Zur Formulierung eines Workflow-Schema-Modells gibt es Darstellungsmittel (Workflow-Sprache). Der folgende Text stellt einen Teil eines Workflow-Schema-Modells mit einer Gebrauchssprache[9] (abstrakte Beschreibung) dar, die jedoch nicht mit der tatsächlichen Darstellung verwechselt werden darf.

> „Die Genehmigung eines Kreditantrags beginnt mit der Zusammenstellung von persönlichen Daten des Antragsstellers (Name, Adresse, Beruf, Alter) und kreditbezogenen Daten (Kredithöhe, Laufzeit, Zinssatz, Sicherheiten usw.). Der aufnehmende Kundenberater hat die Angaben in der zentralen Kundendatenbank zu erfassen und als nächsten Schritt eine Vorabschätzung des Risikos einzuleiten..."

Eine *Workflow-Sprache* muss zum einen auf bestimmte Inhalte reduziert sein und zum anderen in einer speziellen Form vorliegen. Eine gewöhnliche Gebrauchssprache oder sogar eine Normsprache, die eine absolut präzise Beschreibung des Sachverhalts liefert, ist ungeeignet, weil kein Workflow-Management-System mit solchen Formen der Beschreibung effizient umgehen kann. Mit einer Workflow-Sprache wird das Workflow-Schema-Modell beschrieben. Ist die Beschreibung vollständig, korrekt, eindeutig und widerspruchsfrei, so ergibt sich ein Workflow-Schema, mit der ein Workflow-Management-System nun Workflows steuern kann. Ein Workflow-Schema besteht aus einer strukturierten Anordnung und einer Darstellung von speziellen Begriffen (es kann auch in Form einer Zeichnung bzw. einer Graphik sein). Die Begriffe sind exakt auf die Anforderungen eines WfMS zugeschnitten und erlauben es beispielsweise Ausführungsreihenfolgen von Aktivitäten, Zuordnungen von Akteuren und Anwendungsprogrammen zu Aktivitäten usw. präzise zu beschreiben.

Die Workflow-Schemata können untereinander ganz unterschiedlich aussehen und trotzdem auf verschiedene Arten den Inhalt des gleichen Workflow-Schema-Modells beschreiben. Ein Workflow-Schema ist ein Sprachprodukt, das heißt, dass jedes Workflow-Schema zwangsläufig in einer konkreten Sprache formuliert wird. Die folgenden Code-Fragmente sind Teile von Workflow-Schemata für das Workflow-Schema-Modell „Kreditantrag", die dem [Bö97] entnommen sind. Das erste Beispiel zeigt einen Code in der Workflow-Sprache des WfMS FlowMark (IBM).

```
PROCESS ´CreditRequest´
       (´PersonInfo´, ´Default Data Structure´)
  DESCRIPTION ´Credit request for %FirstName% %LastName%´
  STAFF_INHERITED

  PROGRAM_ACTIVITY ´CollectCreditInformation´
      (´PersonInfo´, ´CreditInfo´)
  BEGIN
  END
  ...
 END ´CreditRequest´
```

[9] In diesem Zusammenhang wird auf die Bezeichnung „natürliche Sprache" absichtlich verzichtet und statt dessen der Begriff „Gebrauchssprache" verwendet, um die in einem Anwendungsgebiet gebräuchliche Sprache zu benennen [Le97].

Das nächste Code-Beispiel zeigt, unter Verwendung der Workflow-Sprache des WfMS MOBILE, ein weiteres Workflow-Schema für das Workflow-Schema-Modell „Kreditantrag“.

```
WORKFLOW_TYPE CreditRequest
       (IN PersonInfo: CreditRequestor)
 SUBWORKFLOWS
       ...
       CoCrIn: CollectCreditInformation;
       ...
 END_SUBWORKFLOWS
 WORKFLOW_DATA CreditInfo: c
 END_WORKFLOW_DATA
 ...
END WORKFLOW_TYPE
```

Nach der Modellierung eines Workflow-Schemas wird dieses dem WfMS zugänglich gemacht und von ihm interpretiert. Mittels der Workflow-Schemata, die ein Workflow-Schema-Modell beschreibt, ist es möglich, eine *Workflow-Instanz* zu erhalten. Eine Workflow-Instanz (im Sinne von Ausprägung) ist die Beschreibung eines konkreten (singulären) Workflows [Or97]. Sie bezeichnet Gegenstände auf der Zeichenebene (sprachliche Ebene), während ein Workflow Geschehnisse im Anwendungsbereich beschreibt. Die Workflow-Instanz wird vom WfMS erzeugt bzw. „instanziiert“, um für die Ausführung eines Workflows relevante Daten zu verwalten. Das folgende Beispiel zeigt eine Workflow-Instanz für das oben angeführte Workflow-Schema-Modell „Kreditantrag“. Sie enthält Angaben über einen Kunden, „Herrn Mustermann“.

(„Mustermann“, „Musterstrasse 1, 12345 Musterstadt“,
„Sachbearbeiter“, 45, 10.000)

Die Instanz stellt ein Objekt auf Zeichenebene dar und ist, anders als ein Workflow, kein Geschehen in der Realität. Die Angaben zu einer Workflow-Instanz werden vom Workflow-Schema festgelegt und mit Hilfe von definierten Datenstrukturen verwendet.

2.6 Workflow Management Coalition

Workflow-Management-Systeme sind in ihrem Funktionsumfang nicht einheitlich. Sie erfüllen unterschiedliche Aspekte ihres heterogenen Aufgabenfeldes. Dieser unterschiedliche Leistungsumfang veranlasste Anbieter, Anwender und Berater solcher Systeme ein Standardisierungsgremium, nämlich die *Workflow Management Coalition* (WfMC), im August 1993 zu gründen. Die WfMC ist eine internationale Non-Profit-Organisation mit rund 280 Mitgliedern (Stand Februar 2003). Dazu zählen Firmen wie Agilent, BEA, FileNET, Fujitsu, IBM, Microsoft, Oracle, SAP, Sun.

Die WfMC hat folgende Ziele definiert:

- Ausweitung des WfMS-Marktes,

- Minimierung des Risikos beim Einsatz von Workflow-Produkten,
- Erzielen der Interoperabilität der verschiedenen Systeme,
- Standards für WfMS und deren Umgebung zu definieren und
- Entwicklung eines allgemein gültigen Referenzmodells.

Um diese Ziele zu erreichen, hat die WfMC Terminologien für das Gebiet des Workflow-Managements und der Schnittstellen definiert. Diese Schnittstellen sollen Kommunikation verschiedener Workflow-Management-Systeme untereinander und mit anderen Applikationen ermöglichen. Im Folgenden werden drei grundlegende Konzepte der Workflow Management Coalition eingeführt, nämlich das Basismodell, die abstrakte Produktstruktur und das Referenzmodell des Workflow-Management-Systems in Anlehnung an [WfMC95].

2.6.1 Basismodell des WfMS

Die Workflow Management Coalition entwarf ein *Basismodell* des Workflow-Management-Systems. Dieses Basismodell bringt zum einen die wesentlichen Charakteristika eines WfMS zum Ausdruck und zum anderen die Beziehung zwischen den Funktionalitäten. Das Bündnis erarbeitete ein allgemeines Modell, sodass es Ähnlichkeiten mit Produkten, die sich auf dem Markt befinden, aufweist. Generell werden im Basismodell zwei Ebenen unterschieden: „Build Time“ und „Run Time“, wobei die letztere in zwei weitere Bereiche unterteilt ist (siehe Abbildung 2.6).

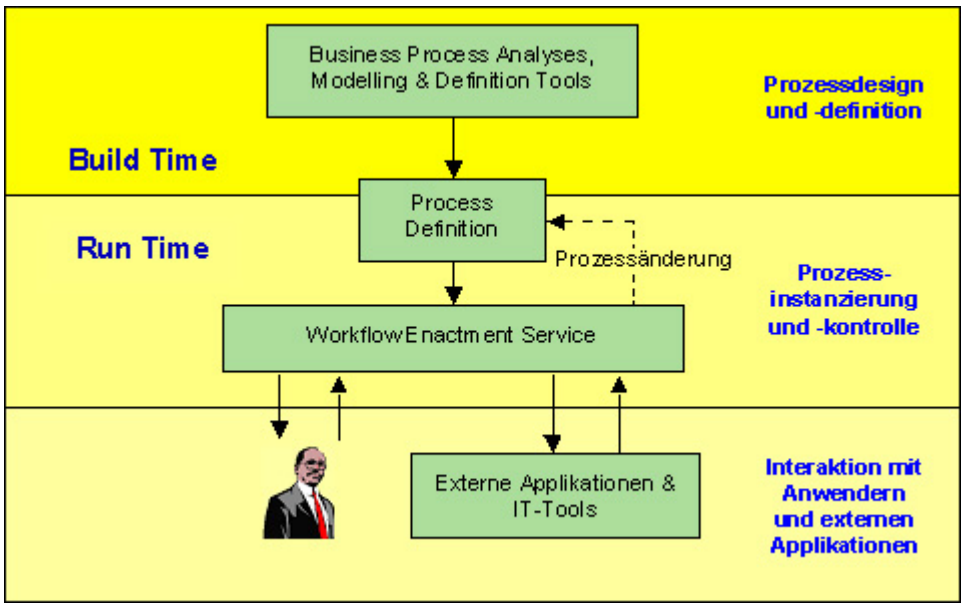

Abbildung 2.6: Das Basismodell der WfMC

In „**Build Time**“ werden Geschäftsprozesse analysiert und modelliert. Die zur Verfügung stehenden Werkzeuge setzen die Geschäftsprozesse aus der realen Welt in die

formale, für den Computer verständliche Form um. Als Ergebnis bekommt man eine Prozessdefinition bzw. ein Workflow-Schema.

In „**Run Time**“ geht es darum, wie der modellierte Prozess durch die Abteilungen des Unternehmens läuft. Im ersten Bereich der „Run Time“ finden Prozessinstanzierung und Prozesskontrolle statt, die von der Kernkomponente Workflow-Management-Engine durchgeführt werden. Im zweiten Bereich kommt es zu Interaktionen zwischen dem modellierten Prozess und den Anwendern oder den externen Applikationen. Die Anwender benutzen dafür besondere die von dem WfMS bereitgestellten Werkzeuge. Mit diesem Basismodell war die Grundlage für Standardisierungsmaßnahmen geschaffen.

2.6.2 Produktstruktur des WfMS

In einem weiteren Schritt legte die WfMC ein allgemeines Implementierungsmodell eines Workflow-Management-Systems fest. Beim Entwurf einer *abstrakten Produktstruktur* beschränkte man sich auf die wichtigsten funktionalen Komponenten innerhalb eines Workflow-Management-Systems und auf die Schnittstellen. Generell werden drei Komponententypen unterschieden:

- **Softwarekomponenten**, die eine Unterstützung für verschiedene Funktionalitäten innerhalb eines Workflow-Systems bereitstellen,
- **System- und Kontrolldaten**, auf die die Softwarekomponenten zugreifen und
- **externe Applikationen und Datenbanken**, die nicht zur Workflow-Management-Anwendung gehören, auf die aber als Teil des gesamten Workflow-Management-Systems zugegriffen werden kann.

Im Folgenden werden die Funktionen der wichtigsten Komponenten der abstrakten Struktur erläutert (vgl. Abbildung 2.7). Das *Process Definition Tool* (Prozessdefinitionswerkzeug) wird benutzt, um die Beschreibung eines Geschäftsprozesses, die oft in gewöhnlicher Normalsprache vorliegt, in eine für den Computer verständliche Form zu transformieren. Das kann mit Hilfe einer formalen Prozessdefinitionssprache, eines objektorientierten Modells oder auch einer Scriptsprache geschehen. Einige Prozessdefinitionswerkzeuge sind speziell für bestimmte Workflow-Engines konzipiert, andere (z.B. ARIS, Bonapart) konzentrieren sich vollständig auf die Prozesse als solches.

Process Definition (Prozessdefinition) setzt sich zusammen aus den notwendigen Informationen, die für die Ausführung des Prozesses von der Workflow-Management-Engine benötigt werden. Dazu zählen zum Beispiel Start- und Stopkriterien des Prozesses, eine Ansammlung von Aktionen, die Computer und Menschen durchführen müssen sowie Regeln für die Reihenfolge ihrer Ausführung. Die Struktur einer Prozessdefinition unterliegt seitens der WfMC keiner Standardisierung. Wichtig ist es, dass die Prozessdefinitionen alle notwendige Informationen enthalten, damit die Ausführungskomponente des WfMS die entsprechenden Prozesse abarbeiten kann.

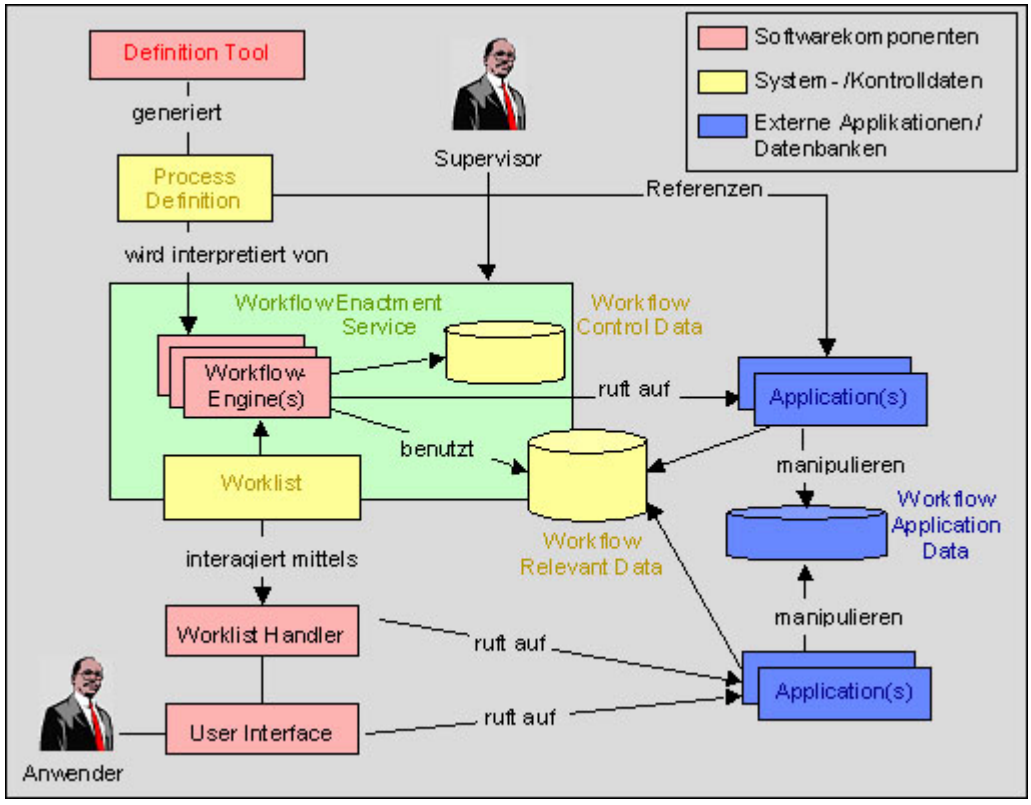

Abbildung 2.7: Abstraktes Implementierungsmodell des WfMS

Kern der abstrakten Produktstruktur ist der *Workflow Enactment Service* (Workflow-Ausführungsservice), der die Runtime-Umgebung für die Workflows darstellt. Diese Komponente interpretiert die Prozessdefinitionen, die in „Build Time“ des Basismodells modelliert und definiert wurden, kontrolliert dabei die Instanzierung der unterschiedlichen Prozesse und regelt deren Abwicklung. Zur Bewältigung dieser Aufgaben stehen eine oder mehrere miteinander kooperierenden *Workflow-Management-Engines* zur Verfügung, welche die Ausführung von individuellen Workflow-Instanzen verschiedener Prozesse verwalten. Ferner übernimmt eine Workflow-Management-Engine auch die Aufgabe, sofern es angefordert wird, externe Applikationen aufzurufen. Das kann zum Beispiel ein Email- oder ein Textverarbeitungsprogramm sein.

Die WfMC trennt in ihrer abstrakten Struktur zwischen *Workflow Control Data* (Workflow-Kontrolldaten), *Workflow Relevant Data* (Workflow-Relevanten-Daten) und *Workflow Application Data* (Workflow-Applikationsdaten). Bei Kontrolldaten handelt es sich um interne Daten, die lediglich von der Workflow-Engine genutzt werden. Sind im System mehrere Workflow-Engines vorhanden, so wird ihr Zusammenspiel über entsprechende Kontrolldaten zentralisiert oder auch verteilt geregelt. Auskunft über Zustandsübergänge eines Geschäftsprozesses, wie zum Beispiel die Reihenfolge der Aktivitäten, geben die Workflow-Relevanten-Daten. Workflow-Applikationsdaten stammen von den externen Applikationen und werden auch direkt von diesen manipuliert. Dabei kann die Workflow-Engine für die Übertragung der Applikations-

daten verwendet werden, damit sie entweder für die Benutzer verfügbar oder zwischen verschiedenen externen Applikationen austauschbar sind.

Mit Hilfe einer zur Verfügung stehenden *Worklist* (Arbeitsliste) ist die Workflow-Engine in der Lage, jedem Benutzer die zu bearbeitenden Arbeitsschritte anzubieten. Die Arbeitsliste ist also eine Liste mit Aufgaben, die in der Regel eine oder mehrere Einzelaufgaben umfasst, die der Benutzer zu erledigen hat.

Um die verschiedenen Einzelschritte der Arbeitsliste zu bearbeiten und um diese verwalten zu können, benötigt der Benutzer einen *Worklist-Handler* (Arbeitsliste-Verwalter). Dieser realisiert die Interaktion zwischen der Arbeitsliste, der Workflow-Engine und dem Bearbeiter. Typische Funktionen sind das Auswählen einer Einzelaufgabe, das Sortieren der Einzelaufgaben nach verschiedenen Merkmalen (z.B. Priorität, Status, Datum), die Wiedervorlage oder das Weiterleiten von Einzelaufgaben, das Anzeigen eines Workflows (z.B. Statusverfolgung) und das Aufrufen der Applikation mit den entsprechenden Daten als Teil der Bearbeitung einer Einzelaufgabe.

2.6.3 Das Workflow Referenzmodell der WfMC

Aus der Workflow-Produktstruktur entwickelte die Workflow Management Coalition ein *Referenzmodell*, das die allgemeine Komponente eines WfMS und deren Schnittstellen beschreibt. Ausgehend von der Workflow-Laufzeitumgebung (in Abbildung 2.8 als Workflow Enactment Service dargestellt) stehen fünf Schnittstellen, die auf unterschiedlichen Ebenen das Zusammenspiel diverser Produkte ermöglichen sollen, zur Standardisierung an:

1. zu Process Definition Tools,
2. zu Workflow Client Application,
3. zu aufgerufenen Applikationen,
4. zu anderen Workflow-Systemen,
5. zu Administrations- und Überwachungswerkzeugen

Schnittstelle 1 regelt die Bedingungen für Prozessstart und -ende. Weiterhin muss sie auch die einzelnen Aktivitäten innerhalb des Prozesses einschließlich der angebundenen Applikationen und der relevanten Daten erkennen. Die Schnittstelle hat außerdem die Aufgabe, Zugriffspfade und die entsprechenden Datentypen, Übergangsbedingungen und Flussregeln zu identifizieren. Über die Schnittstelle zu den *Workflow-Client-Applikationen* kann der Workflow Enactment Service mit den Applikationen interagieren, die die Anwender bei ihrer Arbeitsbewältigung unterstützen. Die Schnittstelle wird hauptsächlich über die Workflow Client Application (Arbeitslisten-Handler) realisiert. Sie ist die Client-basierte Integrationsplattform für die verschiedenen Programme, die mit entsprechenden Daten automatisch aktiviert werden können. Die Workflow Client Application leitet die Ergebnisse anschließend an den Workflow Enactment Service weiter.

Die dritte Schnittstelle dient zur Aktivierung bestimmter Applikationen durch den Workflow Enactment Service. Diese Applikationen sind in der Regel beim Server integriert und laufen ohne Benutzerinteraktion ab (z.B. Fax- oder E-Mail-Programme).

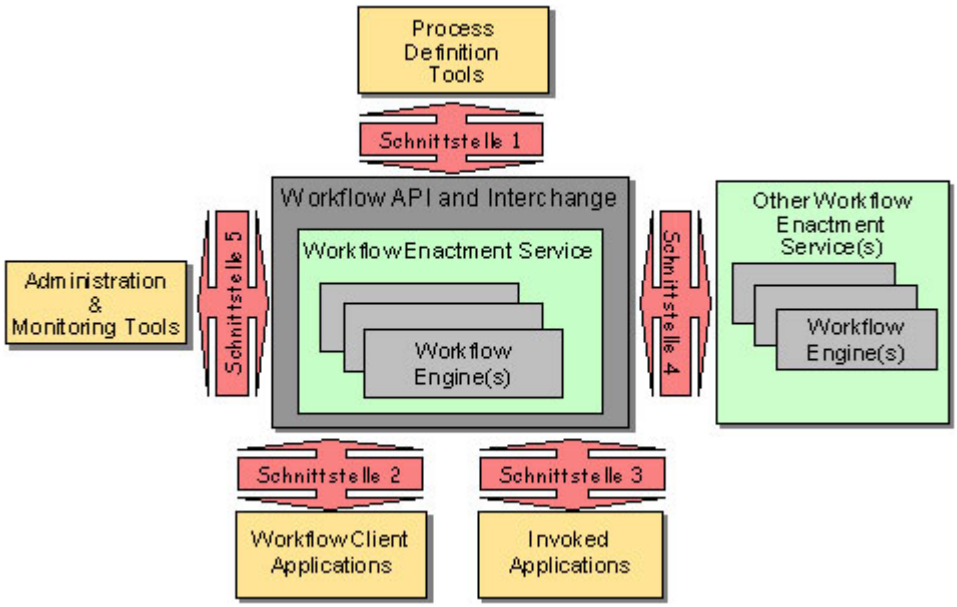

Abbildung 2.8: Workflow Referenz Modell [WfMC95]

Der Schwerpunkt bei der Ausarbeitung des Konzepts wurde auf das Erzielen der Interoperabilität der verschiedener WfMS-Produkte gelegt. Für diese Kommunikation sorgt die Schnittstelle 4. Die WfMC beschreibt sieben unterschiedliche Ebenen der Interoperabilität. Diese kann von der Koexistenz der WfMS auf derselben Hard- und Software-Plattform (Level 1) über den kompletten Austausch von Prozessdefinitionen und allen Workflow-Relevanten-Daten (Level 5) bis hin zum einheitlichen Look&Feel (Level 7) reichen. Die letzte zu standardisierende Schnittstelle betrifft die Systemadministration. Sie soll das gemeinsame Administrieren und Überwachen verschiedener Hersteller mit einem Tool eines beliebigen Anbieters ermöglichen. Typische Aufgaben eines Administrationswerkzeugs sind Benutzerverwaltung, Rollenkonzept-Management, Berechtigungsmanagement und Ressourcenkontrolle.

Kapitel 3

J2EE und Enterprise JavaBeans

3.1 J2EE Komponentenmodell

Die Firma Sun Microsystems hat im Jahre 2000 mit *J2EE* (**J**ava™ **2** Platform, **E**nterprise **E**dition) eine auf der Programmiersprache Java basierende Plattform vorgestellt, welche die Entwicklung unternehmensweiter Anwendungen (*Enterprise Applications*) erleichtern soll. J2EE ist kein Produkt im eigentlichen Sinne, sondern es wird ein allgemeiner Rahmen zur Erstellung solcher Anwendungen basierend auf einem Komponentenmodell definiert.

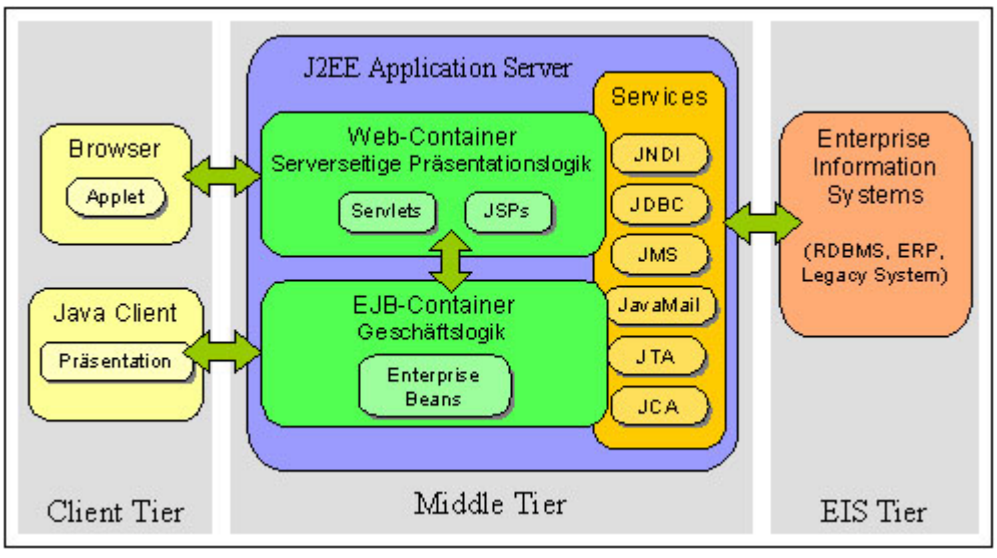

Abbildung 3.1: J2EE Architekturmodell

Die J2EE-Plattform ermöglicht die Realisierung verteilter Applikationen, die von einer klassischen Client-Server-Anwendung bis hin zu einer E-Commerce-Applikation rei-

chen können, durch die so genannte *Drei-Schichten-Architektur*. Abbildung 3.1 bietet einen graphischen Überblick über die verschiedenen Komponenten dieser Architektur. Dabei wird die eigentliche Geschäftslogik zum einen von der Benutzerschnittstelle und zum anderen von Systemdiensten abgekoppelt. Diese Trennung wird mittels einer Middleware vollzogen. Logisch gesehen ist Middleware zwischen den eigentlichen Endbenutzeranwendungen und den persistenten Datenspeichern angesiedelt.

Sun hat für J2EE eine Spezifikation entwickelt sowie eine Referenzimplementierung zur Verfügung gestellt, die zum Testen von J2EE Applikationen dienen soll. Die Spezifikation gibt die nachstehend aufgelistete Menge von Standard-Erweiterungen vor, die von einem jeden J2EE Applikationsserver zwingend unterstützt werden muss:

- JDBC Extension
- RMI-IIOP
- Enterprise Java Beans (EJB)
- Java Servlets
- JavaServer Pages (JSP)
- Java Messaging Service (JMS)
- Java Naming and Directory Interface (JNDI)
- Java Transaction API (JTA)
- JavaMail

Um nun mit diesen Diensten große und verteilte unternehmensweite Anwendungen realisieren zu können, ist zusätzlich die Verwendung von J2EE Technologien nötig, die sich in ihrer Gesamtheit in drei Gebiete aufteilen lassen:

- **Komponenten-Technologien**. Diese Technologien sind für die Bereitstellung des wichtigsten Teils einer Anwendung zuständig: die Business-Logik. Die zur Verfügung stehenden Komponenten sind JSP, Servlets und die Enterprise JavaBeans (EJB), wobei sich letztere wiederum aufteilen lassen in Session Beans, Entity Beans und Message Driven Beans.

- **Service-Technologien**. Diese Technologien stellen den Komponenten einer Anwendung zusätzliche Dienste zur Verfügung. Es handelt sich dabei z.B. um JDBC, JTA und JNDI.

- **Kommunikations-Technologien**. Diese Technologien ermöglichen es den zuvor erwähnten Komponenten und Diensten innerhalb einer J2EE Applikation miteinander zu kommunizieren. An dieser Stelle lässt sich eine Unterteilung machen in Internet Protokolle (HTTP, TCP/IP, SSL), Remote Object Protocols (RMI, RMI-IIOP, JavaIDL), JMS und JavaMail.

Im Vergleich zu anderen Middleware-Techniken wie Transaktionsmonitoren, Message-Queuing-Systemen und Object Request Brokern (ORB) erhebt J2EE den Anspruch, konsequent auf einem Komponentenmodell zu basieren und portabel zu sein. ***Komponenten*** sind Softwareeinheiten auf Applikationsebene wie Applets, Applikations-

Clients, EJB- oder Web-Komponenten. Sie erlauben die Entwicklung von wiederverwendbaren Modulen und werden von Entwicklern benutzt, um die wesentlichen Bestandteile einer Enterprise-Applikation, die Benutzerschnittstelle und die Geschäftslogik, zu erzeugen. Die Komponenten werden von Systemdiensten wie Transaktionsverwaltung, Datenbankanbindung oder Sicherheit unterstützt, die den Programmierprozess vereinfachen und den Komponenten erlauben, so installiert zu werden, dass sie die Ressourcen, die von der jeweiligen Umgebung angeboten werden, optimal nutzen können. Die Konfiguration der Komponenten stützt sich auf die Beschreibungssprache *Extensible Markup Language* (XML).

Zentraler Bestandteil des J2EE-Entwicklungsmodells sind so genannte **Container**, die J2EE-Äquivalenten zur Java VM der *J2SE* (**J**ava™ **2** Platform, **S**tandard **E**dition). Die Container sind standardisierte Laufzeitumgebungen, die den Komponenten spezifische Dienste anbieten. Hierzu zählt beispielsweise ein Namensdienst oder die Verwaltung des Lebenszyklus von Komponenten. Die Realisierung von Containern wird in der Regel auf der Basis existierender Systeme durchgeführt. So arbeiten **Web-Container** (Servlets und JSP-Seiten) mit Web-Servern zusammen. In diesem Fall nimmt der Container Anfragen der Web-Clients entgegen, leitet diese an ein Servlet weiter und schickt die generierte Antwort an den anfragenden Client zurück. **EJB-Container** verwenden oft existierende Transaktionsmonitore oder Applikationsserver und nutzen die von diesen Systemen angebotene Transaktionsverwaltung oder bieten standardisierten Zugang zu Informationssystemen.

Durch Container ist es auch möglich, das Applikationsverhalten erst zur Installationszeit zu bestimmen und nicht schon im Quelltext. Dies geschieht beim *Deployment-Prozeß*, der es ermöglicht die fertigen Applikationen an verschiedene Umgebungen anzupassen. Er beinhaltet sowohl die Installation als auch die Konfiguration der Anwendung. Die J2EE bietet hier einen Standard für so genannte Deployment-Deskriptoren. Das sind XML-basierte Textdateien, welche die Beziehungen der Komponenten untereinander sowie externe Abhängigkeiten beschreiben.

Die J2EE bietet verschiedene Vorteile, die sich aus dem Komponentenmodell ergeben:

- **Plattformunabhängigkeit**. J2EE setzt zu 100 % auf die Java Plattform und ist für alle Systeme verfügbar, auf denen Java-Technologie unterstützt wird. Dadurch wird die Portierbarkeit auf Hardware- und Betriebssysteme gewährleistet.

- **Vereinfachte Architektur und Entwicklung**, da sich das komponentenbasierte Applikationsmodell leicht und flexibel auf die gewünschte Funktionalität der Anwendung abbilden lässt. Gleichzeitig wird eine sinnvolle Arbeitsteilung für verschiedene Entwicklerteams ermöglicht.

- **Standardisierung und Anbieterunabhängigkeit**. J2EE-Applikationen sind entgegen feste Standards der J2EE-Spezifikation entwickelt. Es wird damit gewährleistet, dass Applikationen auf jeden J2EE-konformen Applikationsserver mit minimalem Customizing-Aufwand portierbar sind. Der Standard erleichtert erheblich die Entwicklung robuster, skalierbarer

und wartbarer Applikationen, die auf Wiederverwendbarkeit hin konzipiert sind.

- **Enterprise Application Integration**. J2EE ist eine geeignete Plattform zur Integration von Systemlandschaften. Ob Datenbanken, MOM-Produkte oder Legacy-Systeme – J2EE definiert einen Standard für die Integration der wichtigsten *EIS-Systeme* (Enterprise Information System). Dazu zählen beispielsweise Standard-APIs wie JDBC, Java Transaction API (JTA), Java Naming and Directory Interface (JNDI), Java Messaging Service (JMS), JavaMail, Java IDL.

- **Skalierbarkeit, Lastverteilung & Fehlertoleranz**. Durch den Einsatz Cluster-fähiger Applikationsserver ist es möglich, Applikationen auf mehreren Servern parallel zu betreiben; die Gesamtlast wird damit auf einen Verbund von Servern verteilt. Anwender können somit ihre Systeme beliebig skalieren und auf geänderte Lastanforderungen reagieren.

3.2 Enterprise JavaBeans

Enterprise **J**ava**B**eans (EJB) sind die Komponenten, welche die Geschäfts-Logik enthalten, und sind momentan durch die *EJB 2.0-Spezifikation* [EJB01] definiert. In der Drei-Schichten-Architektur der J2EE sind sie in der mittleren Schicht platziert und verbinden somit die Präsentationskomponenten der Web-Schicht mit den Daten der EIS-Schicht. Die Enterprise JavaBean Architektur erleichtert die Erstellung von Applikationen, da der Anwendungsentwickler von grundlegenden Programmieraufgaben wie Transaktions- und Zustandsverwaltung oder vom Multithreading befreit wird. Durchgeführt werden die Aufgaben vielmehr durch den Applikationsserver. Ein wesentlicher Vorteil von EJB ist darin zu sehen, dass die Beans grundsätzlich auf unterschiedlichen Plattformen ohne Veränderung des Quellcodes und ohne neue Kompilierung einsetzbar sind.

Allgemein können EJBs als **Geschäftsobjekte** bezeichnet werden, durch die das Verhalten der Geschäftslogik charakterisiert wird. Als solche müssen sie gewisse Anforderungen erfüllen. Sie müssen:

- einen Zustand erhalten,
- auf gemeinsamen Daten operieren,
- Transaktionen ausführen können,
- eine große Anzahl von Clients verwalten,
- entfernten Zugriff auf Daten haben,
- den Zugriff von außerhalb regeln und
- wiederverwendbar sein.

EJBs werden von EJB-Containern, die das Lebenszyklusmanagement übernehmen und weitere verschiedene Dienste bereitstellen, erzeugt und verwaltet. Das Verhalten einer EJB ist nicht vollständig in ihrer Implementation bestimmt. Beispielsweise werden

Sicherheitsfragen und das Behandeln von Transaktionen erst bei der Installation auf dem Applikationsserver durch Deployment-Deskriptoren geregelt.

Die EJB-Architektur unterscheidet drei Typen von Enterprise JavaBeans: Session-Beans, Entity-Beans und Message-Driven-Beans.

3.2.1 Session-Beans

Session-Beans modellieren Abläufe und Vorgänge (z.B. Anlegen eines Kunden-Kontos, Durchführung einer Buchung). Sie werden dazu benutzt, um Geschäftsobjekte zu implementieren, die benutzerspezifische Geschäftslogik ausführen. Sie sind normalerweise nicht persistente, relativ kurzlebige Komponenten. Der Lebenszyklus der Session-Beans wird vom Container bestimmt und verwaltet. Beim Herunterfahren des EJB-Servers/Containers bzw. einem möglichen Absturz geht auch die Verbindung zwischen Client und Session-Bean verloren. Eine Session-Bean implementiert das Java-Interface `javax.ejb.SessionBean`.

Hierbei kann zwischen zustandsbehafteten Session-Beans (*Stateless SessionBean*) und zustandslosen Session-Beans (*Stateful SessionBean*) unterschieden werden. Zustandsbehaftete Session-Beans speichern benutzerspezifische Daten über mehrere Methodenaufrufe hinweg. Aufrufe von Methoden können den Zustand der Bean verändern. Eine zustandsbehaftete Session-Bean ist eine private Ressource eines Clients am Server, die ihm exklusiv zur Verfügung steht. Zustandslose Session-Beans speichern von einem Methodenaufruf zum nächsten keine benutzerspezifischen Daten. Eine Methode arbeitet nur mit den Daten, die ihr als Parameter übergeben wurden.

3.2.2 Entity-Beans

Entity-Beans repräsentieren Geschäftsobjekte, deren Daten sich in einem dauerhaften Speicher befinden. Eine Entity-Bean repräsentiert z.B. einen bestimmten Kunden oder ein Konto. Mehrere Clients können ohne Probleme gleichzeitig ein und dieselbe Entity-Bean nutzen. Dabei wird unterschieden, ob die Persistenz von einem Container (CMP: *Container Managed Persistence*) oder von der EJB selbst (BMP: *Bean Managed Persistence*) verwaltet wird. Bei Entity-Beans mit container-gesteuerter Persistenz übernimmt der Container die Aufgabe, die Daten persistent zu machen. Die Bean muss lediglich deklarieren, welche Attribute persistent sind. Entity-Beans mit bean-gesteuerter Persistenz sind selbst dafür verantwortlich, dass ihre Daten persistent gemacht werden, d.h. auf eine Datenbank geschrieben werden. Hier ist der Programmierer für das Implementieren der Datenbankoperationen (z.B. mit JDBC) zuständig. Für den Client ist die gewählte Persistenzschicht vollkommen transparent. Eine Entity-Bean implementiert `javax.ejb.EntityBean`.

3.2.3 Message-Driven-Beans

Message-Driven-Beans (MDB) sind mit der Spezifikation 2.0 zu den Enterprise JavaBeans hinzugekommen. Sie können am besten als zustandslose asynchrone Konsumenten von Nachrichten beschrieben werden. Sie enthalten die Geschäftslogik für die empfangenen Nachrichten und verarbeiten diese asynchron. Diese Nachrichten stam-

men typischerweise von einem JMS-Provider (*Java Messaging Service*-Provider). Ab der EJB 2.1 Spezifikation sollen aber auch beliebige andere Nachrichtentypen unterstützt werden. Eine Message-Driven-Bean horcht an einem Topic oder einer Queue und verarbeitet die Nachrichten, die von JMS-Clients in dieses Topic/diese Queue gestellt wurden. MDB speichern von einem Methoden-Aufruf zum nächsten keine Daten. Sie sind also zustandslos, aber sie können Instanz-Variablen verwenden, deren Werte während der Lebensdauer der Instanz gespeichert werden. Mehrere Instanzen eines Message-Driven-Beans ermöglichen die parallele Verarbeitung von Nachrichten einer Queue oder eines Topics.

Die Nachrichten an Message-Driven-Beans müssen nicht von einem EJB stammen. Sie können auch von externen JMS-Clients kommen, z.B. von Java-Applikationen oder Legacy-Anwendungen (beispielsweise über MQSeries von IBM). MDB können alle Nachrichten aus einem Topic oder einer Queue eines JMS-konformen Nachrichtendienstes lesen.

3.3 Message Oriented Middleware (MOM)

Bei der Übermittlung von Geschäftsdaten muss gewährleistet sein, dass die Daten auch tatsächlich unversehrt ankommen. Geschäftssysteme werden immer mehr verteilter und heterogener. Das bedeutet für die Anwendungen, dass sie nun mit anderen, bestehenden und neuen Anwendungen interagieren und kommunizieren müssen. Hinzu kommt, dass die Kommunikationsverbindungen zwischen Anwendungen ab und zu ausfallen. Um in solchen Situationen dennoch die Übermittlung von Daten zwischen verteilten Anwendungen garantieren zu können, wird seit jeher auf *Message Oriented Middleware* zurückgegriffen. MOM erlaubt eine lose Kopplung von verteilten Anwendungen sowie eine hohe Zuverlässigkeit bei der Kommunikation zwischen Anwendungen.

Das Programmiermodell der MOM ist ereignisorientiert und beruht auf dem Austausch von selbstbeschreibenden Nachrichten, die spezifische Daten enthalten. Die zugrundeliegende Programm-zu-Programm-Kommunikation wird auch *Message-Queuing* genannt. *Message* bedeutet, dass Programme durch Senden von Daten in **Nachrichten** (Messages) kommunizieren und nicht durch wechselseitiges, direktes Aufrufen. *Queuing* heißt, dass Programme über **Warteschlangen** (Queues) miteinander kommunizieren. Dadurch ist es nicht notwendig, dass diese Programme zeitlich parallel ausgeführt werden. Sie sind also in der Lage, Informationen zu senden und zu empfangen, ohne dass eine direkte Verbindung zwischen ihnen besteht. Sie kommunizieren miteinander durch Ablegen und Herunternehmen der Nachrichten in Queues. Die wichtigsten Charakteristika von Message-Queuing sind:

- **Zeitunabhängige (asynchrone) Kommunikation**. Der Austausch von Nachrichten zwischen dem sendenden und dem empfangenden Programm ist zeitunabhängig. Das sendende Programm kann die Verarbeitung fortsetzen, ohne auf die Rückmeldung des Empfängers der Nachricht zu warten. Die MOM hält die Nachrichten solange in der Queue, bis sie verarbeitet werden.

- **Verbindungslose Kommunikation**. Sendende und empfangende Programme benutzen nur Queues für die Kommunikation. MOM ist für alle anfallende Aktivitäten, die zum Aufbau und Durchführung einer solchen Kommunikation verbunden sind, verantwortlich. Dazu zählt zum Beispiel das Erhalten der Queues und der Beziehungen zwischen Programmen und Queues, das Handling der Netzwerk-Restarts, sowie die Bewegung der Nachrichten durch das Netzwerk.

- **Parallelverarbeitung**. Das MOM-Modell erlaubt eine 1:1-Beziehung zwischen den kommunizierenden Programmen. Das bedeutet, dass genau ein Empfänger von genau einem Sender Nachrichten erhalten kann. Es kann aber auch Anwendungsstrukturen und Nachrichten-Übertragungsformen unterstützen, die viel komplexer sind: many-to-one, one-to-many.

Es existieren zwei Kommunikationsmodelle, die sich in Message Oriented Middleware etabliert haben. Diese sind Publish-and-Subscribe und Point-to-Point, die allgemein auch als *Message Domains* bezeichnet werden und im Folgenden vorgestellt werden.

3.3.1 Kommunikationsmodelle

MOM stellt den eigentlichen Dienst des Übertragens von Nachrichten zur Verfügung. Die Benutzer dieser Dienste werden als MOM-Clients bzw. in diesem Abschnitt einfach Clients bezeichnet. In einer Applikation nimmt ein Client eine von verschiedenen Rollen ein, im Allgemeinen zunächst einmal die Rolle des *Produzenten* (Producer) und/oder des *Konsumenten* (Consumer). Ein Produzent erzeugt eine Nachricht und gibt diese weiter an die MOM zum Versenden. Der Konsument empfängt die Nachricht vom Provider und verarbeitet sie weiter. Hier sieht man ganz deutlich, dass die verschiedenen Clients voneinander, sowie von dem MOM-System getrennt sind und nicht direkt miteinander kommunizieren. Dadurch können auch andere, nicht MOM-Clients eingebunden werden oder die MOM-Systeme ausgetauscht werden.

Innerhalb der Message Domains gibt es für die oben erwähnten allgemeinen Rollen spezielle Rollen. Bei einem Point-to-Point Modell wird aus dem Produzent ein *Sender* und aus dem Konsument ein *Receiver*. Analog wird bei Publish-and-Subscribe aus dem Produzent ein *Publisher* und aus dem Konsument ein *Subscriber*.

Bei dem *Point-to-Point* Modell wird eine Nachricht an eine Queue geschickt (vgl. Abbildung 3.2). Jede Nachricht hat genau einen Empfänger (Receiver). Der Empfänger bestätigt den Erhalt der Nachricht. Es gibt keine Zeit-Abhängigkeiten zwischen Sender und Empfänger. Die Queue speichert persistente Nachrichten ab, bis sie vom Receiver gelesen werden oder sie je nach Typ der Nachricht nach einem Zeit-Limit ablaufen. Auch bei mehreren Empfängern wird die Nachricht nur einmal gesendet.

Das Point-to-Point Modell wird vor allem dort eingesetzt, wo eine zeitliche Entkopplung zwischen Produzenten und Konsumenten von Nachrichten stattfinden muss, und wo jede Nachricht von nur einem Konsumenten empfangen werden darf.

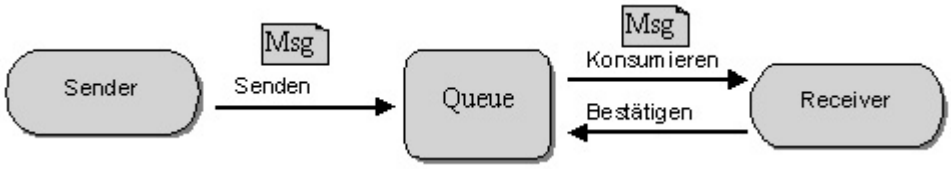

Abbildung 3.2: Point-to-Point Message-Domain

Bei dem *Publish-and-Subscribe* Modell werden Nachrichten so genannten Themen (Topics) zugeordnet. Produzenten erzeugen Nachrichten und publizieren (publish) diese anschließend unter einem oder unter mehreren Themen (vgl. Abbildung 3.3). Konsumenten können diese Themen abonnieren (subscribe). Sie erhalten alle Nachrichten, welche unter dem entsprechenden Thema publiziert werden. Sind mehrere Konsumenten für ein bestimmtes Thema angemeldet, so werden Nachrichten an alle Konsumenten übermittelt (multicast).

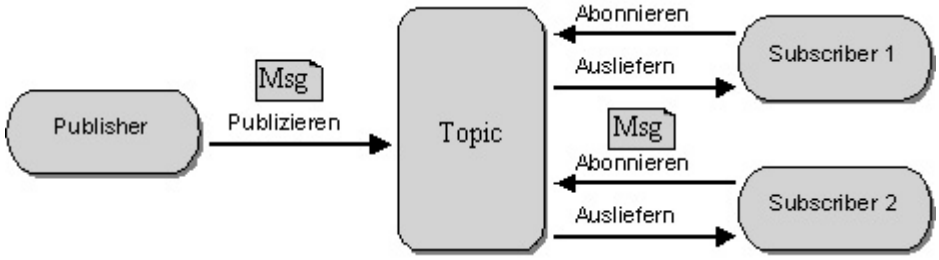

Abbildung 3.3: Publish-and-Subscribe Message-Domain

Das Publish-and-Subscribe Modell wird vor allem für Systeme eingesetzt, in denen Nachrichten in Echtzeit übermittelt werden müssen, oder in Systemen, in denen dieselbe Nachricht von mehreren Konsumenten empfangen werden muss.

Die beiden Message-Domains sind nur mit einem gewissem Aufwand austauschbar, was vornehmlich an der historischen Entwicklung von MOM-Systemen liegt. Dabei sind die jeweiligen Modelle für das gedachte Einsatzgebiet einfacher zu nutzen.

3.4 MQSeries

MQSeries ist ein Middleware-Produkt der Firma IBM für kommerzielles Message-Queuing. MQSeries wird in Client-Server- oder in verteilten Umgebungen eingesetzt. MQSeries ist auf einer Vielzahl von Plattformen lauffähig. Dadurch können Programme in unterschiedlichen Rechnerarchitekturen auf verschiedenen Plattformen ausgeführt werden. Für alle Plattformen ist derselbe Queuing-Mechanismus gültig.

Den Kern von MQSeries bildet der *Message-Queue-Manager* (MQM). Der MQM realisiert die Run-Time-Umgebung von MQSeries. Seine Aufgabe besteht darin, die Queues und die Messages zu verwalten. Er stellt das *Message Queue Interface* (MQI) für die Kommunikation mit den Anwendungen zur Verfügung. Die Applikationen rufen Funktionen des Queue-Managers durch Ausgabe von API-Calls auf. Der MQPUT-API-Call z.B. legt eine Message auf eine Queue, die von einem anderen Programm mit Hilfe eines MQGET-API-Calls gelesen werden soll.

Ein Programm kann Nachrichten an ein anderes Programm schicken. Das Ziel-Programm läuft dabei entweder auf derselben Maschine wie der Queue-Manager oder auf einem Remote-System wie z.B. ein Server oder ein Host. Das Remote-System verfügt über seinen eigenen Queue-Manager bzw. seine eigene Queue. Der MQM überträgt Nachrichten zu einem anderen MQM über *Channels*, wobei bestehende Netzwerk-Facilities wie TCP/IP, SNA oder SPX verwendet werden. Auf derselben Maschine können mehrere Queue-Manager residieren, dabei benötigen sie für die Kommunikation Channels. Für den Anwendungsprogrammierer bleibt die Kommunikation transparent, auch der Ort, zu dem er die Messages schickt. Er legt seine Nachrichten auf eine Queue und überlässt es dem Queue-Manager, die Ziel-Maschine zu suchen und die Nachricht auf diese zu übertragen.

Die oben angesprochenen Channels von MQSeries stellen Kommunikationspfade dar. Dabei wird zwischen zwei Channel-Typen unterschieden:

- Message Channel
- Message-Queue-Interface Channel (MQI-Channel)

Ein *Message-Channel* verbindet zwei Queue-Manager über *Message-Channel-Agenten* (MCAs), auch Treiber (mover) genannt. Ein Channel ist immer unidirektional und integriert zwei Message-Channel-Agenten, einen Sender, einen Empfänger und ein Kommunikations-Protokoll. Ein MCA stellt ein Programm dar, das Nachrichten von einer Transmission-Queue in die Ziel-Queue überträgt.

Ein MQI-Channel verbindet einen MQSeries-Client mit einem Queue-Manager auf der Server-Seite. Clients besitzen keinen eigenen Queue-Manager. Ein MQI-Channel ist immer bidirektional (MQI-Call und Antwort). Abbildung 3.4 zeigt beide Channel-Typen. Es sind insgesamt vier Maschinen dargestellt. Zwei Clients sind mit ihrer Server-Maschine über MQI-Channels und der Server ist mit einem anderen Server oder Host mittels zweier unidirektionaler Message-Channels verbunden.

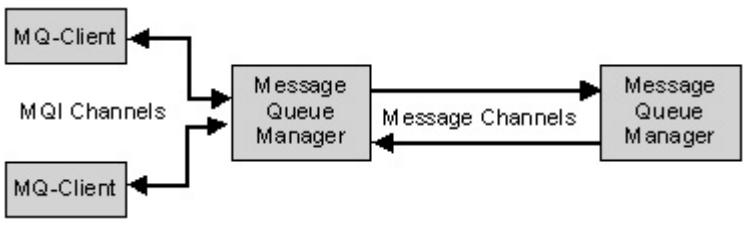

Abbildung 3.4: MQSeries Channels

MQSeries unterscheidet vier verschiedene Message-Typen, die in Tabelle 3.1 aufgelistet sind. Darüber hinaus kann zwischen **persistenten** und **nicht-persistenten** Messages gewählt werden. Das Applikations-Design bestimmt, ob eine Nachricht ihr Ziel unbedingt erreichen muss oder diese gelöscht wird, wenn das Ziel nicht innerhalb einer vorgegebenen Zeit erreicht wird. Die Übertragung von persistenten Nachrichten wird sichergestellt, d.h. sie werden protokolliert, um bei Systemausfällen erhalten zu bleiben. In einem AS/400-System heißen diese Protokolle „Journal Receivers“.

Message-Typ	Beschreibung
Datagram	Eine Message enthält Informationen, auf die keine Antwort erwartet wird
Request	Eine Message, für die eine Antwort angefordert wird
Reply	Eine Antwort-Message auf eine Request-Message
Report	Eine Message, die einen Event beschreibt wie zum Beispiel Fehlermeldung oder Bestätigung beim Eintreffen einer Nachricht

Tabelle 3.1: Message-Typen von MQSeries

3.5 JMS Standard

MOM-Produkte werden seit den 70er Jahren entwickelt und erfolgreich eingesetzt. Dabei stellte jeder Hersteller seine eigene Programmierschnittstelle (API) zur Verfügung. Mit der Einführung eines Messaging-Standards hat sich dies nun aber geändert. Führende Hersteller existierender Messaging Systeme (IBM, Oracle, Weblogic, Software AG, Sybase, Novell u.a.) haben sich unter der Schirmherrschaft von Sun Microsystems zusammengetan und einen offenen Standard für Java-MOM-Produkte geschaffen. Das Ergebnis ist der **J**ava **M**essaging **S**ervice (JMS). Ein Message-Service Produkt, welches die JMS Spezifikation und eventuell zusätzliche, spezifische Funktionen implementiert, nennt man *JMS Provider*. Typische Funktionen können sein: Sicherheitsfunktionen, Administrationswerkzeuge, Load Balancing und vieles mehr. Der iBus von SoftWired zeichnet sich z.B. dadurch aus, dass eine Vielzahl von Transport-Protokollen „eingeklinkt“ werden können, wie TCP/IP, SSL, HTTP, IP Multicast oder SMS und WAP [Ma00]. Weblogic unterstützt z.B. XML als Dokumenttyp.

3.5.1 JMS synchrone und asynchrone Kommunikation

In einer *traditionellen synchronen* Kommunikation muss der sendende und der empfangende Client die Kommunikationsaktivitäten selber koordinieren. Das bedeutet, damit eine gegebene Nachricht an einen Client geliefert werden kann, muss einer der Clients den anderen zum Senden bzw. zum Empfangen auffordern. Die Kommunikationspartner sind solange beschäftigt, bis beide die Kommunikation abgeschlossen haben. Es findet ein sogenanntes **Blocking** statt. Die Clients können nichts anderes machen als zu kommunizieren (mindestens der entsprechende Thread). Im schlimmsten

Fall muss der sendende Client beliebig lange versuchen den empfangenden Client zu erreichen.

In einer *traditionellen asynchronen* Kommunikation werden Nachrichten dann abgeliefert, wenn sie anfallen. Der empfangende Client muss also den Sender nicht zum Senden aufrufen, dies geschieht ereignisgesteuert.

In der JMS Spezifikation wird das traditionelle End-to-End Kommunikationsmodell, das eben besprochen wurde, nicht verwendet. JMS stellt modifizierte Kommunikationsmuster vor. Der sendende Client ist immer asynchron (*asynchronous send*). Mit dem Senden an einen Message-Server (z.B. Router, Broker) hat er seine Arbeit getan. Der empfangende Client kann entweder synchron oder asynchron sein. Er muss die Nachrichten beim Message Server abholen. Synchrones Empfangen (*synchronous receipt*) geschieht, indem der Client die passende Receiver Methode für eine eintreffende Nachricht wählt oder periodisch die Verbindung auf neue Nachrichten überprüft (polling). Im asynchronen Fall (*asynchronous receipt*) setzt der empfangende Client einen MessageListener auf. Eintreffende Nachrichten werden automatisch durch den Router abgeliefert, wobei der Router die `onMessage()` Methode des MessageListener aufruft.

In allen Fällen ist der Router für die korrekte Ablieferung der Nachrichten zuständig. Der sendende Client kann dabei noch bestimmte Parameter setzen, zum Beispiel Liefergarantie, Time to Live, Quality of Service oder was im Fehlerfall zu geschehen hat.

3.5.2 JMS Architektur

Eine JMS Applikation verwendet entweder das Point-to-Point oder das Publish-and-Subscribe Messaging-Modell. Diese Message-Domains können jedoch auch kombiniert werden.

Eine JMS Applikation ist aus folgenden Elementen zusammengesetzt:

- **JMS Clients**. Dies sind Java Programme, welche Nachrichten senden und empfangen.

- **Nicht-JMS Clients**. Diese Clients benutzen das API des Message Systems, also nicht JMS. Falls das System eine JMS Komponente besitzt, wird das System vermutlich JMS und fremde Clients besitzen.

- **Messages**. Jede Applikation definiert bestimmte Nachrichten, die für die Kommunikation zwischen den Clients eingesetzt werden sollen.

- **JMS Provider**. Dies ist ein Messaging System, welches JMS implementiert. Zusätzlich werden andere administrative und Kontrollfunktionen implementiert, die für den Betrieb des Messaging Systems benötigt werden.

- **Administrierte Objekte**. Das sind JMS Objekte, welche von einem Administrator für den Einsatz durch Clients eingesetzt werden. Administrierte Objekte werden von JMS Provider verwaltet.

JMS setzt voraus, dass jeder JMS Provider unter Umständen ziemlich unterschiedliche Technologien einsetzten wird. Daher werden sich auch die Managementsysteme im Wesentlichen unterscheiden. Damit JMS Clients portabel sind, muss daher von einem konkreten Managementsystem abstrahiert werden.

Dies wird erreicht, indem man JMS *administrierte Objekte* definiert, welche vom Administrator des JMS Providers kreiert und administriert werden. Zum späteren Zeitpunkt werden diese vom Client eingesetzt. Der Client verwendet diese Objekte mit Hilfe von JMS API Schnittstellen, welche portabel sind. Es gibt zwei Typen von JMS administrierten Objekten:

- **ConnectionFactory**. Das ist ein Objekt, welches der Client benutzen kann, um eine Verbindung aufzubauen.

- **Destination**. Damit kann der Client das Ziel und die Quelle der Nachrichten angeben.

Administrierte Objekte werden vom Administrator in ein JNDI (Java Naming and Directory Interface) Namespace eingetragen. Der JMS Client kennt typischerweise die JMS administrierten Objekte, die er benötigt, und weiß wie der Client zum JNDI Namen dieser Objekte gelangt.

Beim Aufbauen einer Kommunikation sind noch weitere wichtige Komponenten zu nennen. Das sind als erstes die *Ziele* (Destinations), an die sich eine bestimmte Kommunikation richtet. Je nach Modell kann dies ein Topic oder eine Queue sein. Des weiteren sind dies die *Verbindungen* (Connections), die eine logische Verbindung zwischen Client und Provider aufbauen. Vor jeder Kommunikation muss eine Connection aufgebaut werden, die nach der Kommunikation wieder geschlossen werden muss. Auf den Connections setzen dann die Sessions auf. Dabei handelt es sich um einen single-threaded Kontext, um Nachrichten zu senden oder sie zu empfangen. Man kann eine bis mehrere Sessions pro Connection erzeugen. Eine Session ist eine JMS Entität, die als atomare Transaktion aufgefasst werden kann.

JMS ist nicht grundlegend auf Multithreading ausgelegt, da dadurch ein bestimmter Overhead und eine bestimmte Komplexität hätte eingebaut werden müssen. Das aktuelle JMS Design beschränkt die Mehruserfähigkeit auf jene Teile, die natürlicherweise von mehreren Clients benutzt werden. Die betreffenden Teile werden in der Tabelle 3.2 aufgelistet.

Eine JMS Nachricht hat einen recht einfachen aber dennoch sehr flexiblen Aufbau, der auch einen Einsatz in heterogenen Nachrichten-Netzwerken erlaubt. Eine JMS Nachricht besteht aus drei Bestandteilen. Diese sind in der Tabelle 3.3 zusammengefasst.

JMS Object	Unterstützt Concurrent Use
Destination	Ja
ConnectionFactory	Ja
Connection	Ja
Session	Nein
MessageProducer	Nein
MessageConsumer	Nein

Tabelle 3.2: Multithreading der JMS Objekte

Der *Header* beinhaltet die Standardeigenschaften, die von Clients und Provider genutzt werden, um die Nachricht zu identifizieren und sie an das Ziel zu bringen. Man kann auch zusätzliche *Einstellungen* (Properties) für eine Nachricht setzen, falls die im Header vorhandenen nicht ausreichen. Diese Einstellungen sollen die Kompatibilität zu anderen Messaging Systemen ermöglichen. In dem *Body* liegen die eigentlichen Daten, die gesendet werden. Dabei gibt es unterschiedliche Typen von Nutzdaten, die in den sechs verschiedenen Interfaces von JMS vorgegeben werden und von Providern implementiert werden müssen.

Header										Properties	Body
JMS Destination	JMS Delivery Mode	JMS MessageID	JMS Timestamp	JMS Expiration	JMS Redelivered	JMS Priority	JMS ReplyTo	JMS CorrelationID	JMS Type	(Application & Provider defined)	Message, TextMessage, StreamMessage, MapMessage, ObjektMessage, ByteMessage

Tabelle 3.3: Aufbau einer JMS Nachricht

Kapitel 4

Beispiel-Implementierung einer Workflow-Engine

Im Rahmen dieser Diplomarbeit wurde eine Integration eines Workflow-Management-Systems (iHub-WfMS) in eine bestehende E-Commerce Anwendung sowie eine Portierung des iHub-WfMS auf den JBoss-Applikationsserver durchgeführt. Die genauere Vorgehensweise wird in den folgenden zwei Kapiteln näher erläutert.

4.1 Der iHub-Prototyp

Der iHub (***i**ntegration **Hub***) stellt einen Workflow/Messaging-Prototyp der Firma iT media Consult GmbH in Stuttgart dar. Er kombiniert Elemente des Workflows mit denen einer Message Oriented Middleware (MOM) und ist in der Lage, nach Vorgaben eines Workflows die Nachrichten transaktionssicher aus einer Message-Queue in eine andere zu übertragen. Das Konzept von iHub sieht vor, Integration von Anwendungen, Wiederverwendung sowie Verteilung von Prozessen und Daten zu ermöglichen. Darüber hinaus sollen Geschäftsprozesse automatisiert werden, um Informationen unter Verwendung von definierten Regeln von Anwendung zu Anwendung weiterzureichen. Der iHub besteht aus einem MessageBroker und einem iHub-Workflow-Management-System, wie es in Abbildung 4.1 schematisch dargestellt ist.

Zur Aufgabe des *MessageBrokers* gehört die Kommunikation mit angeschlossenen Systemen. Die Art dieser Systeme kann dabei unterschiedlich sein. Es können verschiedene Anwendungsprogramme sein, die für die Abarbeitung eines Geschäftsprozesses eingesetzt werden, damit ein bestimmtes Ziel erreicht wird. Der MessageBroker empfängt Nachrichten, die die Anwendungssysteme an ihn schicken und stößt die folgende Abarbeitung durch das angeschlossene Workflow-Management-System an. Nach der erfolgten Verarbeitung sendet der MessageBroker eine entsprechende Nachricht an das jeweilige System zurück. Das WfMS generiert einen Workflow und überlässt ihm

die Steuerung über den definierten Geschäftsvorfall. Eine weitere Aufgabe des MessageBrokers ist die Transaktionssicherung der Daten, die während der Übermittlung mit den Systemen übertragen werden.

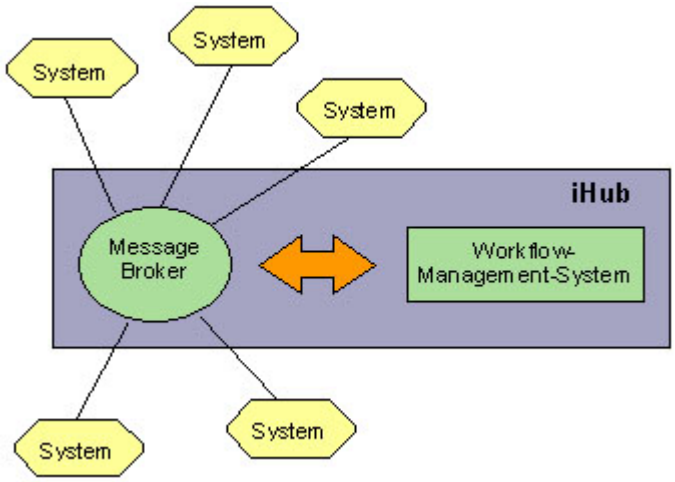

Abbildung 4.1: iHub-Aufbau

Das *iHub-Workflow-Management-System* (iHub-WfMS) bildet den Kern des iHub. Es ist vielseitig einsetzbar und kann auch unabhängig vom iHub genutzt werden. Das WfMS hat das Ziel, einen prozessorientierten Arbeitsablauf einheitlich und effektiv umzusetzen. Es steuert und kontrolliert die Abläufe von Geschäftsprozessen zwischen beteiligten Prozessen nach einer festen Definition (Prozessdefinition). In der Prozessdefinition wird festgehalten, welche Daten für die Bearbeitung des Prozesses notwendig sind, zu welchem Zeitpunkt diese benötigt werden, um neue Informationen bzw. Dokumente erzeugen zu können. Außerdem verbindet das WfMS alle Anwendungen miteinander, die für die Durchführung der Kernprozesse notwendig sind.

Da das iHub-Workflow-Management-System auch unabhängig vom iHub eingesetzt werden kann, um bestimmte Aufgaben selbständig zu erledigen, wird das System in diesem Kapitel als **Server** für eine prozessorientierte Verarbeitung betrachtet. Dem zufolge werden Anwendungen bzw. Programme, die dessen Dienst in Anspruch nehmen, als **Clients** bezeichnet. Zum Beispiel wäre der MessageBroker innerhalb des iHub-Systems ein Client in Bezug auf das WfMS. Auch der Begriff von Workflow wird in diesem Kapitel anders behandelt, als in den anderen Kapiteln. Vielmehr geht es hier um eine Komponente des iHub-WfMS, die die Eigenschaften eines Geschäftsprozesses dieser Applikation realisiert.

4.1.1 Funktionsweise des iHub-WfMS

Mit dem Einsatz des iHub-Workflow-Management-Systems wird die Verarbeitung jeglicher Workflows gewährleistet, die in das Gerüst eines XML-Repository-Trees passen. Das *XML-Repository-Tree* enthält die Definition aller Workflows (Prozessdefinition) und beinhaltet somit das „Wissen“ über die Steuerung der Abläufe innerhalb des iHub-WfMS. Der Inhalt soll von einem *Prozess Definition Tool* (Prozessdefinitionswerkzeug) modelliert werden können oder direkt von einem Benutzer in Form eines

Baumes in einer XML-Datei abgelegt werden. In Abbildung 4.2 ist ein Beispiel einer aus zwei Workflows bestehenden Prozessdefinition zu sehen.

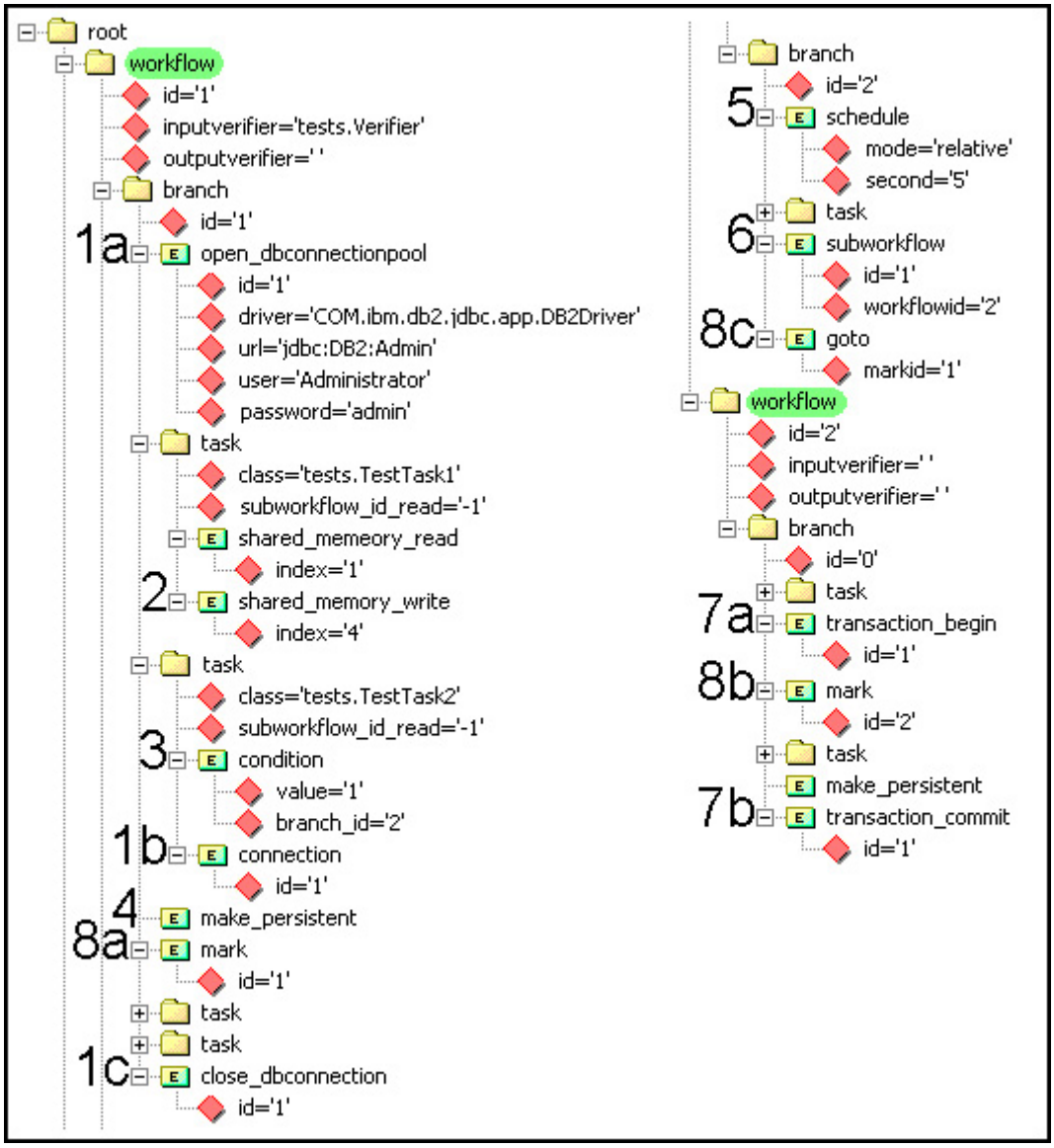

Abbildung 4.2: Beispiel einer Prozessdefinition

Der Workflow wird gestartet, indem das WfMS von seinem Client (z.B. MessageBroker) eine Nachricht erhält, die den auszuführenden Workflow anhand einer eindeutigen Workflow-Nummer (*Workflow-ID*) identifiziert. Nach dem Beenden des Workflows sendet das WfMS sein Ergebnis an den Client zurück. Anhand einer Prozess-Nummer (*Prozess-ID*), die der Client unterhält, kann er die Antwort des Workflow-Management-Systems seiner Anfrage zuweisen. Ein Workflow wird aus einzelnen Teilschritten, auch **Tasks** genannt, dargestellt, die sequenziell oder parallel ausgeführt werden können. Neben der Prozessdefinition sind die Tasks weitere Komponenten, die wäh-

rend „Build Time“[10] vom Benutzer modelliert und anschließend implementiert werden müssen. Der Benutzer ist in der Gestaltung der Tasks und deren Aufgaben völlig frei, bis auf die Einschränkung, dass sie ein bestimmtes Interface implementieren müssen. Über dieses Interface wird die Kommunikation mit dem WfMS hergestellt. Zusätzlich kann eine Definition eines Workflows mehrere *Äste* (Branches) enthalten, die zusammenhängende Bereiche eines Workflows enthalten können. Wie es weiter unten erläutert wird, kann die Reihenfolge der Verarbeitung zwischen verschiedenen Ästen durch Bedingungen festgelegt werden.

Das Konzept des iHub-WfMS ermöglicht eine parallele Abarbeitung mehrerer unabhängiger Workflows. Dabei kann jeder Workflow in Sub-Workflows verzweigen, die wiederum parallel zum Haupt-Workflow abgearbeitet werden können. Ein *Sub-Workflow* unterscheidet sich prinzipiell nicht von einem Workflow, außer dass er innerhalb des Lebens-Zyklus vom Haupt-Workflow ausgeführt wird. Ebenfalls kann jeder Sub-Workflow wieder in eigene Sub-Workflows verzweigen. Bei der Beispiel-Prozessdefinition in Abbildung 4.2 ist unter Ziffer 6 die Definition eines Sub-Workflows zu sehen. Dabei legt das Attribut „id“ die eindeutige Identifikation des Sub-Workflows fest. Das Attribute „workflowid“ repräsentiert den Workflow, der als Sub-Workflow ausgeführt werden soll.

Die Prozessdefinition eines Workflows kann bedingte Verzweigungen enthalten, durch die ein Workflow unterschiedliche Pfade durchlaufen kann. Die Pfade werden über *Bedingungen* (Conditions) modelliert. In Abbildung 4.2 wird eine Verzweigung unter Ziffer 3 modelliert. Falls das Ausführungsergebnis der Task-Klasse „`tests.TestTask2`“ ein Condition-Objekt mit dem Wert „2“ enthält, so wird der Workflow zu einem anderen Abschnitt der Prozessdefinition (Branch 2) umgeleitet und dort fortgeführt.

Darüber hinaus kann ein zeitlicher Ablauf mit einem *Scheduler* (Ablaufplaner) geregelt werden. Dabei kann ein Workflow an einer beliebigen Stelle unterbrochen und nach einer im XML-Repository-Tree angegebenen Wartezeit (absolut oder relativ zum aktuellen Datum) fortgesetzt werden. In der Beispiel-Prozessdefinition (Ziffer 5) wird ein relativer Ablaufplan benutzt, der für fünf Sekunden die Abarbeitung des Workflows unterbricht.

Um den Datenverlust im Falle eines Systemabsturzes zu verhindern, kann der Workflow *persistent* gemacht werden, damit eine richtige Weiterverarbeitung nach erneutem Starten des Systems sichergestellt werden kann. Dazu wird der aktuelle Systemzustand in eine temporäre Datei geschrieben. Der Sicherungspunkt, an dem der Workflow persistent gemacht wird, kann durch ein Flag in der Prozessdefinition (Ziffer 4 in Abbildung 4.2) festgelegt werden. Bei jedem Neustart des WfMS wird überprüft, ob abgebrochene Workflows vorhanden sind, die dann gegebenenfalls fortgeführt werden können.

[10] Die Workflow Management Coalition unterscheidet in ihrem Basismodell des WfMS zwei Ebenen: „Build Time“ und „Run Time“. In der ersten Ebene werden Geschäftsprozesse aus der realen Welt in die formale, für den Computer verständliche, Form umgesetzt (vgl. Abschnitt 2.6.1 „Basismodell des WfMS“).

Die Kommunikation zwischen Tasks wird zum einen dadurch ermöglicht, dass das Ergebnis nach jedem ausgeführten Task an den folgenden Task weitergereicht wird. Zum anderen kann ein Task sein Ergebnis in einen allen Workflows gemeinsamen Speicher schreiben, wenn es in der Prozessdefinition verlangt wird, wie es in Abbildung 4.2, Ziffer 2 zu sehen ist.

Ein Workflow oder mehrere Teile eines Workflows können innerhalb einer *Transaktion* ablaufen. Dafür werden der Anfang (Begin) und das Ende (Commit) der Transaktion in der Prozessdefinition festgelegt (Ziffer 7a und 7b). Außerdem kann ein Task ein Rollback und die erneute Ausführung der aktuellen Transaktion bewirken.

Bei der Abarbeitung eines Workflows sind auch *Schleifen* (Loops) möglich, d.h. einen mehrmaligen Durchlauf von Teilen eines Workflows. Eine Schleife kann zum einen durch einen Task ausgelöst werden, indem er das System dazu veranlasst, zu einer in der Prozessdefinition eingebauten *Marke* (Ziffer 8a und 8b in Abbildung 4.2) zurück zu springen. Zum anderen kann es in der Definition eines Workflows durch ein *Goto*-Element angegeben werden, an welche Marke gesprungen werden soll, um dort die Ausführung fortzuführen. In Abbildung 4.2 wird eine Goto-Schleife (Ziffer 8c) mit der Marke 1 (Ziffer 8a) gezeigt.

In der Beschreibung eines Workflows kann ein *JDBC-Connection-Pool* angelegt werden, indem alle für eine Datenbankverbindung notwendigen Parameter angegeben werden. Ein Task kann dann eine dieser Verbindungen für seine Zwecke nutzen (vgl. Ziffer 1a, 1b und 1c).

4.1.2 Die Komponenten des iHub-WfMS

Im Folgenden wird auf die Komponente des iHub-Workflow-Management-Systems näher eingegangen. Dabei wird die Funktionsweise sowie der strukturelle Aufbau der wichtigsten Komponenten beschrieben. Die Implementierung des iHub-Systems ist in Programmiersprache Java erfolgt. Abbildung 4.3 zeigt alle wichtigen Komponenten und ihre Beziehungen untereinander.

An dieser Stelle soll gleich vorweggenommen werden, dass der Gebrauch des Begriffs „Event“ nicht mit der gewöhnlichen Bedeutung in Zusammenhang gebracht werden darf. Es handelt sich hier nicht um Ereignisse wie Mausbewegung oder Tastaturaktion, die vom Benutzer ausgelöst und von Java-AWT abgefangen und verwaltet werden. Vielmehr geht es hier um Objekte (z.B. StartEvent, FinishedEvent, TaskEvent), die zwischen den Komponenten des iHub-Workflow-Management-Systems weitergereicht werden und bestimmte Informationen enthalten, die der Empfänger lesen aber auch verändern kann[11].

Die *Workflow-Engine* (WFEngine) bildet eine Schnittstelle des Workflow-Management-Systems zur Außenwelt und koordiniert die für das System benötigten Ressourcen. Beim Systemstart liest die WFEngine Eigenschaften aus einer Systemdatei, die für die Systemkonfiguration notwendig sind. Diese Informationen werden im Laufe der Applikation verwendet und können mit einem Text-Editor erstellt werden. Außer-

[11] Im Pattern Design werden die entsprechenden Objekte als Value Objects bezeichnet, vgl. [Al01b].

dem wird von der WFEngine ein Scheduler gestartet, der den zeitlichen Ablauf der Tasks aller Workflows innerhalb der WFEngine regelt. Als nächstes wird das Recovery-Verzeichnis überprüft. Wurde das WfMS nicht ordnungsgemäß beendet, so stellt die WFEngine den Systemzustand vor der letzten Sicherung wieder her. Die WFEngine kann StartEvents von einem Client (z.B. iHub-MessageBroker) empfangen, wobei es sich um die Art der Clients handelt, die bereits in dem Abschnitt 4.1 „Der iHub-Prototyp“ beschrieben worden ist. Danach wird ein Workflow-Manager instanziiert und das empfangene StartEvent an ihn weitergeleitet. Die WFEngine kann neben einem StartEvent auch ein FinishedEvent von einem Workflow-Manager empfangen. Dies bedeutet, dass die Abarbeitung eines Geschäftsprozesses abgeschlossen wurde. Der entsprechende Client wird darüber informiert und der Workflow-Manager, der diese Arbeit koordiniert hatte, wird samt aller gesicherten Daten gelöscht.

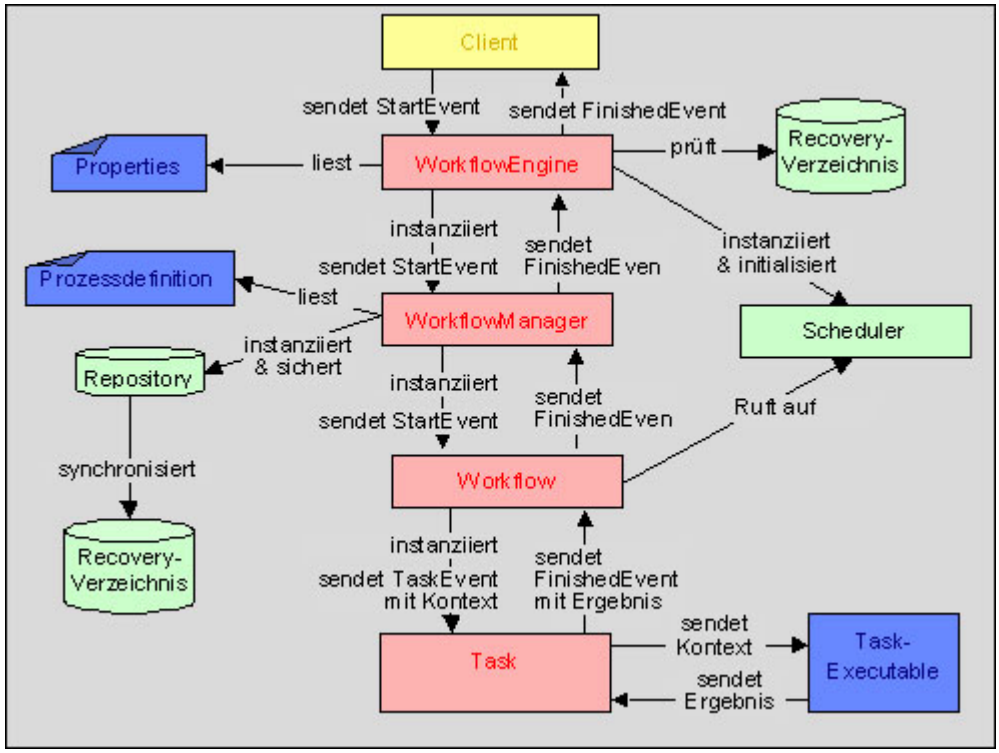

Abbildung 4.3: Komponenten des iHub-WfMS

Die Benachrichtigung des Clients erfolgt über eine Schnittstelle `MessageBroker-Listener`, die der Client implementiert. Diese enthält eine einzige öffentliche Methode, `readEvent()`, mit dem Parameter FinishedEvent.

Zu den Aufgaben eines *Workflow-Managers* (WFManager) gehört die Steuerung eines oder mehrerer Workflows sowie die Kommunikation zwischen ihnen. Er dient somit als zentrale Anlaufstelle für alle aktive Workflows, die von ihm verwaltet werden. Während der Initialisierung des WFManagers wird die Prozessdefinition eingelesen und ein Repository-Objekt instanziiert. In dem *Repository* werden alle Daten gehalten, die für die Ausführung von Workflows notwendig sind. Das sind zum Beispiel Defini-

tionen verschiedener Workflows (Prozessdefinition) oder die vom Client übergebene Prozess-ID.

Der WFManager kann drei Arten von Events empfangen, die er in einer Warteschlange des Repository sichert und dann eine nach der anderen verarbeitet. Neben dem StartEvent und dem FinishedEvent, die vom WFManager einfach weiter geleitet werden (vgl. Abbildung 4.3), erhält er einen SubEvent. Dieses Event veranlasst, dass vom WFManager ein neuer Sub-Workflow instanziiert wird. Werden alle Events aus der Warteschlage verarbeitet, so bleibt der WFManager inaktiv, bis ein neues Event eintritt.

Ein *Workflow* steuert den Ablauf der Tasks. Die Verarbeitung der eingetroffenen Events wird, ähnlich dem WFManager, über eine sich im Repository befindende Warteschlange gehandhabt. Erhält ein Workflow ein StartEvent, so beginnt er die Workflow-Definition aus der Prozessdefinition abzuarbeiten, die ihm der WFManager zugeteilt hat. Abbildung 4.2 zeigt zwei Workflow-Definitionen einer Beispiel-Prozessdefinition. Der Workflow geht bei der Verarbeitung der Definitionselemente sequenziell vor. Ein TaskEvent wird von der Komponente Task nach der Beendigung der Task-Ausführung übermittelt und enthält das Ergebnis des sendenden Tasks. Je nach Workflow-Definition speichert der Workflow über den WFManager das Ergebnis im Repository.

Die eigentliche Ausführung der beliebigen Arbeitsvorgänge eines Geschäftsprozesses werden vom Anwendungsprogrammierer, der das WfMS benutzt, programmiert. Der Benutzer schreibt die Ausführungslogik in spezielle Objekt-Klassen, *TaskExecutables* genannt. Die Kommunikation mit dem System muss allerdings über festgelegte Schnittstellen (*TaskInterface*) ablaufen, die jeder TaskExecutable implementieren soll. Durch die Schnittstelle soll verhindert werden, dass der Benutzer möglichen Einfluss auf den Geschäftsprozess bzw. auf das WfMS nehmen kann.

Soll eine schon bestehende Applikation als TaskExecutable fungieren, kann diese auch über einen Adapter angebunden werden. Ein Task übernimmt die Kommunikation mit TaskExecutables, deren Instanz er bei seiner Instanziierung erhält. Empfängt ein Task ein TaskEvent, ruft er über das TaskInterface die Methode `execute()` des TaskExecutables auf und übergibt ihm den Kontext aus dem TaskEvent. Nach Abarbeitung der Ausführung des TaskExecutables gibt die `execute()`-Methode ein neues Ergebnis-Objekt zurück und der Task sendet es zurück an den Workflow. Um Transaktionen und Persistenzierung zu gewährleisten, müssen die Objekte, die von einem TaskExecutable an das WfMS gesendet werden, die Schnittstellen `java.lang.Cloneable` und `java.io.Serializable` bzw. `java.io.Externalizable` erweitern.

4.1.3 Beispiel eines Zähler-Geschäftsprozesses

Anhand eines Beispiels soll nun die Funktionsweise des iHub-Workflow-Managements-Systems beispielhaft gezeigt werden. Es soll ein Geschäftsprozess „Counter“ modelliert werden, der einen einfachen Zähler simuliert. Zwischen jeder Inkrementierung des Zählers soll eine Zeitlang gewartet werden (z.B. fünf Sekunden). Als Abbruchbedingung wird eine bestimmte Größe (z.B. 10) gewählt, die der Zähler erreichen soll. Das Diagramm zu dem beschriebenen Vorgang ist in Abbildung 4.4 zu sehen.

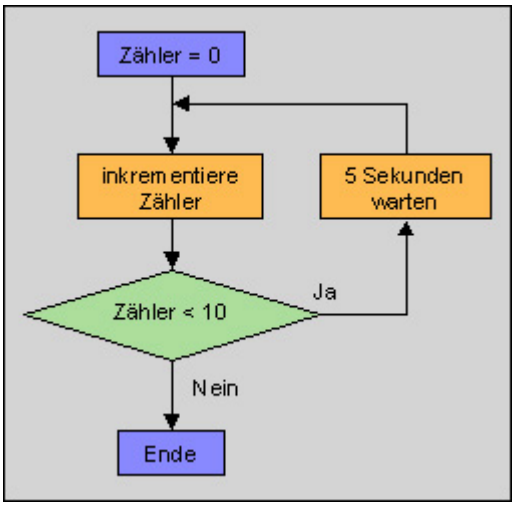

Abbildung 4.4: Diagramm des Geschäftsvorgangs „Counter“

```
1.  <?xml version="1.0" ?>
2.  - <root>
3.    - <workflow id="1" inputverifier=""
                         outputverifier="">
4.      - <branch id="0">
5.          <mark id="10" />
6.        - <task class="example.task.Count"
                         subworkflow_id_read="-1">
7.            <shared_memory_write index="1" />
8.            <shared_memory_read index="1" />
9.            <!-- count < MAX_VALUE -->
10.           <condition value="1" branch_id="2" />
11.           <!-- count >= MAX_VALUE -->
12.           <condition value="2" branch_id="1" />
13.       </task>
14.     </branch>
15.   - <branch id="1">
16.     - <task class="example.task.End"
                         subworkflow_id_read="-1" />
17.     </branch>
18.   - <branch id="2">
19.       <schedule mode="relative" second="5" />
20.       <goto markid="10" />
21.     </branch>
22.   </workflow>
23. </root>
```

Abbildung 4.5: XML-Repository-Tree des „Counters“

Abbildung 4.5 zeigt die Prozessdefinition in Form eines XML-Repository-Trees. Der Vorgang ist als Workflow mit drei Ästen (Branches) modelliert. Der erste Ast (vgl. Abbildung 4.5 Zeile 4 bis14) enthält eine Marke sowie einen Task mit dem TaskExecutable `example.task.Count`, dessen Java-Codebeispiel weiter unten aufgeführt ist. Dem Task wird eine Speicherstelle im Repository eingeräumt (Abbildung 4.5 Zeile 7 und 8). In diese Speicherstelle legt er den neuen inkrementierten Wert des Zählers ab (<shared_memory_write>-Tag) und holt den alten Wert bei der nächsten Inkrementierung wieder heraus (<shared_memory_read>-Tag). Außerdem wird mit den Bedingungs-Parameter in den Zeilen 10 und 12 eine Schleife modelliert. Solange der Zähler den maximalen Wert von 10 nicht erreicht hat, wird die erste Bedingung ausgeführt und der Workflow springt zum Ast mit der id=2 (Zeile 18). Hier wird relativ zur aktuellen Uhrzeit fünf Sekunden lang gewartet und anschließend zur Marke 10 in der Zeile 5 gesprungen. Andernfalls wird die zweite Bedingung ausgeführt und es wird zum Ast mit id=1 gewechselt (Zeile 15), wo der TaskExecutable `example.task.End` ausgeführt und damit der Geschäftsprozess beendet wird.

Quell-Code des TaskExecutable “Count.java”:

```
1.  package example.task;

2.  import java.util.*;
3.  import ihub.gwe.helper.*;
4.  import ihub.gwe.interfaces.*;

4.  // Objekt Count entnimmt aus dem gemeinsamen Speicher des
5.  // Workflow-Management-Systems den aktuellen Wert des Zählers
6.  // und erhöht ihn um eins.
7.  public class Count implements TaskInterface {
```

```
8.    static final public long MAX_COUNT_VALUE = 10;
9.    // Methode der Schnittstelle „ihub.gwe.interfaces.TaskInterface“
10.   public Object execute(Object fromPrevious, Vector fromSubWf,
                            Vector fromMemory, HashMap connections)
11.     throws Exception {
12.     Vector vec = new Vector();
13.     // hole den aktuellen Wert des Zählers aus dem Speicher des WfMS
14.     IntObject count = ((IntObject) fromMemory.get(0));
15.       if (count == null)
16.         count = new IntObject(1);
17.       else
18.         count.inc();
19.     System.out.println("Count-Task: Zähler=" + count.intValue());
20.     // Schreibe den inkrementierten Wert des Zählers
21.     // in den Speicher zurück
22.     vec.add(count);
23.     if (count.intValue() < MAX_COUNT_VALUE)
24.       // der aktuelle Wert des Zählers ist nicht groß genug
25.       vec.add(new Condition(1));
26.     else
27.       // die Abbruchbedingung ist erreicht worden
28.       vec.add(new Condition(2));
29.     return vec;
30. }
```

Der TaskExecutable `example.task.Count` implementiert die Schnittstelle `ihub.gwe.interfaces.TaskInterface` und enthält eine einzige Methode `execute()`, die diese Schnittstelle erweitert (vgl. Codebeispiel von „Counter.java“ Zeile 7 und 10). In diesem Beispiel wird nur ein Parameter der Methode benutzt. Der Vollständigkeit halber werden an dieser Stelle alle Parameter erläutert. Der erste Parameter *fromPrevious* beinhaltet das Ergebnis vom vorherigen Task oder, wenn dieser Task der erste im Workflow ist, den Startwert des Workflows. Der zweite, *fromSubWf*, enthält das Ergebnis eines Sub-Workflows, falls es im XML-Repository-Tree verlangt wird, sonst *null*. *fromMemory* enthält Werte aus dem Repository. Die Anzahl und die Speicherindizes im Repository werden im XML-Repository-Tree angegeben. In diesem Beispiel ist es eine Speicherstelle zum Schreiben und Lesen mit Index=1 (Abbildung 4.5 Zeile 7 und 8). Der Parameter *connections* enthält JDBC-Connections. Aus welchen ConnectionPools die Connections stammen, wird ebenfalls im XML-Repository-Tree angegeben. In den Zeilen 13 bis 17 des Quell-Codes wird aus dem Speicher des WfMS der alte Wert des Zählers geholt und inkrementiert. In den Zeilen 21 bis 26 wird anhand des neuen Zählerwerts die Bedingung für das weitere Verlaufen des Workflows gesetzt. Nachdem der Ergebnis-Vektor aus der Zeile 11 mit dem neuen Wert-Objekt (Zeile 20) sowie dem Bedingungs-Objekt gefüllt wurde, wird dieser an die Task-Komponente übergeben (vgl. Abbildung 4.3).

4.2 Vergleich von iHub-Workflow-Management-System mit der WfMC

In dem vorherigen Kapitel ist das Referenz-Modell der Workflow Management Coalition vorgestellt worden. In diesem Abschnitt soll ein Vergleich zwischen der standardisierten Spezifikation der WfMC und dem implementierten Prototypen der iT media GmbH iHub-Workflow-Management-System durchgeführt werden. Der Architekturvorschlag der WfMC identifiziert fünf Schnittstellen zwischen den Komponenten eines Workflow-Management-Systems, wobei die Funktionalität dieser Komponenten nicht

genau definiert ist [Ja97]. Wie es bereits im Abschnitt 2.6.3 „Das Workflow Referenzmodell der WfMC“ eingeführt wurde, ist das Referenzmodell aus folgenden Teilen zusammengesetzt:

- Workflow Enactment Service,
- Process Definition Tools,
- Administrating and Monitoring Tools,
- Workflow Client Applications,
- Invoked Applications und
- Other Workflow Enactment Services

Bei „*Workflow Client Applications*“ und „*Process Definition Tools*“ handelt es sich um Werkzeuge. Diese Werkzeuge sind im Prototyp iHub-WfMS als solche noch nicht vollständig vorhanden bzw. sind gerade im Entwicklungsstadium. So können zur Zeit die Prozessdefinitionen mit einem gewöhnlichen Text-Editor oder mit einem XML-Entwicklungswerkzeug erstellt werden. In einem laufenden Projekt wird daran gearbeitet, die Prozessdefinitionen mit einem UML-Tool (Enterprise Architect) zu erzeugen und anschließend in den iHub XML-Repository-Tree zu konvertieren.

Ein Werkzeug, welches das gemeinsame Administrieren und Überwachen des WfMS ermöglicht, das „*Administrating and Monitoring Tool*“, ist im iHub-WfMS kürzlich eingebaut worden. Damit kann der Benutzer den Ablauf des Workflows verfolgen sowie die Daten im gemeinsamen Speicher des Repository ansehen. Außerdem kann man sich den Kontext eines TaskExecutables anzeigen lassen.

„*Invoked Applications*“ stellen Applikationsprogramme dar, welche zur Ausführung elementarer Workflows aufgerufen werden. Die Aktivierung einer bestimmten Applikation durch das WfMS des Prototyps erfolgt über Adapter, wobei die Applikation als TaskExecutable an den ausführenden Workflow angebunden werden kann (vgl. dazu Abschnitt 4.1.2 „Die Komponenten des iHub-WfMS“).

Die Komponente „*Other Workflow Enactment Services*“, die mit der entsprechenden Schnittstelle dafür Sorge tragen soll, dass eine Integration von verteilten Workflow-Engines verschiedener Hersteller zustande kommen kann, ist im iHub-Prototyp nicht realisiert. Allerdings fehlt im Vorschlag der WfMC die Definition einer verteilten Ausführung eines Workflows [Ja97].

Der „*Workflow Enactment Service*“ stellt die Runtime-Umgebung für die Workflows dar. Die Aufgaben dieser Komponente stimmen mit denen des Prototyps überein. Dazu zählt die Interpretation der Prozessdefinitionen, die in „Build Time“ modelliert bzw. definiert wurden, die Kontrolle der Instanzierung der unterschiedlichen Workflows und die Regelung ihrer Abwicklung. Die WfMC beschreibt in ihrer Spezifikation zwar die Aufgaben, die eine Workflow-Engine erfüllen soll, definiert jedoch nicht exakt die Dienste der Komponenten, die eine Workflow-Engine umfasst. Auch der Vorschlag der WfMC, mehrere miteinander kooperierende Workflow-Engines zur Verfügung zu stellen, wird von der iHub-WfMS-Anwendung unterstützt. Seit kurzer Zeit wurde der Prototyp erweitert, so dass mehrere Workflow-Engines miteinander rollenbasiert kommunizieren können. In der Prozessdefinition kann durch einen XML-Tag festgelegt

werden, welche Rolle bzw. welcher User mit dem Workflow fortfahren soll. Diese Bestimmung gilt auch für andere Engines. Darüber hinaus basiert das Konzept des Prototyps auf mehreren parallel laufenden Workflow-Manager, die ihrerseits mehrere kooperierende Workflows steuern, wobei die WfMC nicht auf die Einzelheiten der Implementierung der Workflow-Management-Anwendungen eingeht.

Das vorgeschlagene abstrakte Implementierungsmodell trennt zwischen Workflow-Kontrolldaten, Workflow-Relevanten-Daten und Workflow-Applikationsdaten. Die Kontrolldaten sollen lediglich von der Workflow-Engine benutzt werden und werden laut WfMC vom Workflow Enactment Service verwaltet. Die Workflow-Relevanten-Daten werden von externen Applikationen und von der Workflow-Engine benutzt. Auf die Workflow-Applikationsdaten hat das WfMS keinen Einfluss, da sie direkt von den Applikationen manipuliert werden. Die WfMC schlägt vor, dass die Workflow-Engine für die Übertragung dieser Daten verwendet werden kann. In der iHub-WfMS-Anwendung werden die Daten nicht nach diesen Kriterien unterschieden, zumindest nicht explizit. Hier werden die Kontroll- und Applikationsdaten gemeinsam im Repository der einzelnen Workflow-Manager gehalten.

4.3 Integration des iHub-WfMS in eine bestehende Anwendung

Als nächstes soll in diesem Abschnitt untersucht werden, wie das iHub-Workflow-Management-System mit anderen Applikationen zusammengesetzt werden kann. Wie bereits in Abschnitt 4.1 „Der iHub-Prototyp“ erwähnt, kann das iHub-WfMS unabhängig vom iHub eingesetzt werden. Der Zugang zum System für dessen Clients und externe Applikationen wird über wohldefinierte Schnittstellen verfügbar gemacht. Einerseits ist es die *MessageBrokerListener*-Schnittstelle, die auf eine Benachrichtigung der Workflow-Engine über erfolgte Arbeit wartet und andererseits die *TaskInterface*-Schnittstelle, die für die Ausführung der TaskExecutables benutzt wird. Man kann das iHub-WfMS als eine eigenständige Komponente betrachten, die zur Ausführung bestimmter Dienste verwendet werden kann. Eine Applikation, die diese Dienste in Anspruch nehmen soll, heißt eCCo.

4.3.1 Der eCCo-Prototyp und seine Funktionsweise

Das eCCo (***e****lectronic* ***C****ommerce* ***Co****ncept*) ist ein Prototyp einer E-Commerce-Applikation der Firma iT media Consult GmbH. Dieses Produkt dient zum Kauf von Fahrzeugen und einiger dazugehörigen Cross Selling[12]-Produkten sowie dem Bezahlvorgang im Web.

Die Architektur des eCCo besteht aus fünf Schichten. Abbildung 4.6 zeigt eine Architekturübersicht mit den wichtigen Komponenten der Applikation. Wie es sich leicht erkennen lässt, ist das Konzept des eCCo eng mit dem J2EE-Architekturmodell für

[12] Beim „**Cross Selling**“ (dt. Querverkauf) wird dem Käufer eines Artikels ein Verweis auf weitere Artikel gemacht. Das können alternative Artikel oder passender Zubehör sein. Zum Beispiel kann beim Kauf eines Staubsaugers ein Querverweis auf einen Staubsaugerbeutel und einen Staubsaugerfilter gemacht werden.

Java-Enterprise-Anwendungen verbunden. Merkmale dieses Modells sind zum einen die Aufteilung der Integration-Tier und der Resource-Tier, die in früheren Modellen in einem gemeinsamen Layer angesiedelt waren. Um eine bessere Kapselung und eine sich daraus ergebende Wart- und Erweiterbarkeit zu erreichen, wurden die Zugriffe auf die Backend-Systeme von diesen und von der Business-Tier abgetrennt.

Zum anderen wurden in das Modell neu entworfene *Design Patterns* [Al01a, Al01b] für die Presentation-, die Business- und die Resource-Tier aufgenommen. Diese Entwurfsmuster spiegeln die langen Erfahrungen des *Sun Java Centers*, der *Enterprise Consulting Division* von Sun Microsystems wieder. Die J2EE Design Patterns geben den Entwicklern von Enterprise-Anwendungen mit der Java 2 Plattform neben den Architekturvorgaben noch Best Practices und Implementierungsstrategien an die Hand.

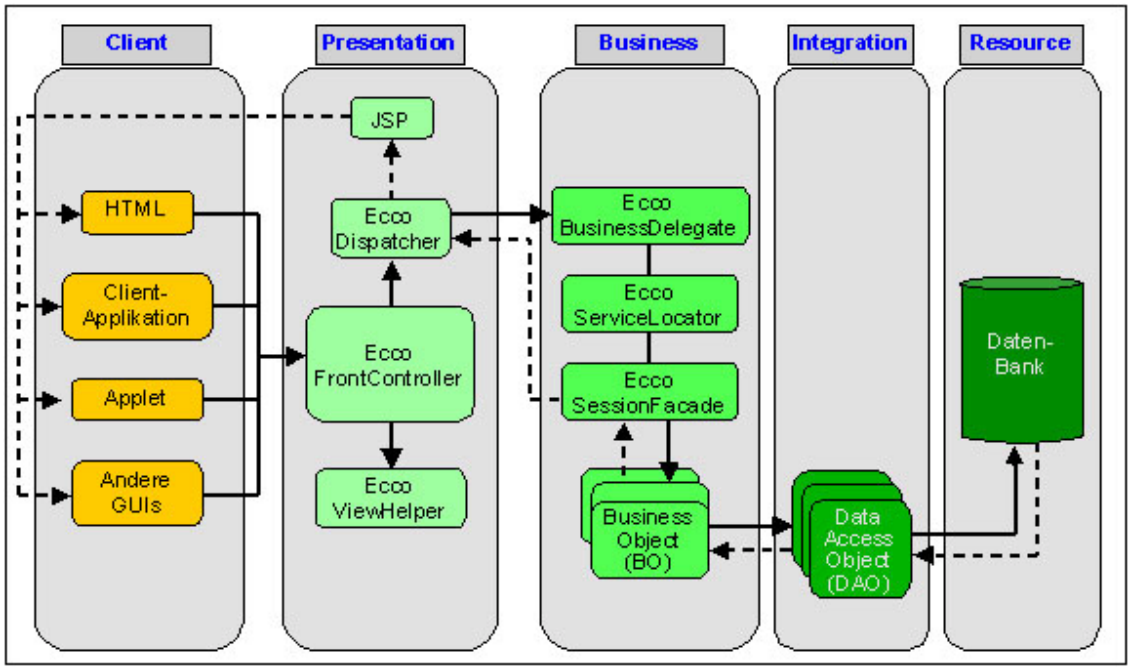

Abbildung 4.6: Schichten-Architektur des eCCo

Tabelle 4.1 listet Komponenten der jeweiligen Schichten aus der Abbildung 4.6 mit ihren Funktionen auf. Einem Benutzer der eCCo-Applikation stehen viele Aktionen zur Verfügung. Er kann zum Beispiel sich beim eCCo-System anmelden, über eine Suchmaske nach vorhandenen Fahrzeugen in der Datenbank suchen, diese in einem Warenkorb ablegen oder die Fahrzeuge kaufen. Diese Aktionen werden als Anfragen (Request) an das eCCo geschickt und von einem Servlet, dem *EccoFrontController*, empfangen. Die Anfrage wird weiter an die Business-Schicht geleitet, wo die Logik der Applikation angesiedelt ist. Handelt es sich um eine Aktion, bei der Informationen aus der Datenbank benötigt werden, so werden diese über *Data Access Objects* (DAO) der Integrations-Schicht geholt. Nach der Bearbeitung der Anfrage wird an den Client eine entsprechende JSP (JavaServer Pages) gesendet.

Presentation Tier	
eCCo Front Controller	Zentraler Zugriffspunkt für Verwaltung der ankommenden Anfragen. Bewerkstelligt Authentifizierung/Autorisierung des Benutzers, hält den Benutzer-Kontext. Zeigt Exceptions zur Client-Tier an.
eCCo View Helper	Repräsentiert das Datenmodell der Views, bezieht die Daten vom System. Die Views (JSPs) übernehmen lediglich die Ausgabeformatierung.
eCCo Dispatcher View	Kombiniert eine Dispatcher-Komponente in Abstimmung mit dem Front Controller und dem View Helper.
Business Tier	
eCCo Business Delegate	Dient als Client-Side-Presentation der Business-Objekte. Entkoppelt Presentation- und Business-Tier. Überträgt Exceptions aus der Business-Tier in die der Presentation-Tier. Spielt die Rolle eines Proxys, da der Einsatz von EJBs noch nicht erfolgt ist. Business Delegate ist konzeptionell und zur Erweiterung des Systems unabdingbar.
eCCo Session Facade	Verbirgt die Komplexität der Business-Objekte. Bildet den Prozessfluss über die bei einem Request benutzten Business-Objekte ab.
eCCo Service Locator	Zuständig für den Service-Lookup. An dieser Stelle kann z.B. eine Middleware angebunden werden oder direkt auf die Server-Klassen zugegriffen werden.
Integration Tier	
eCCo Data Access Object (DAO)	Modelliert die Persistenz der Business-Objekte, also den eigentlichen Zugriff auf die Backend-Systeme.

Tabelle 4.1: Beschreibung der einzelnen Komponenten des eCCo [eCCo01]

4.3.2 Implementierung der Integration

Nachdem nun die betreffenden Anwendungen iHub-WfMS und eCCo vorgestellt wurden, folgt die Beschreibung der während der Diplomarbeit durchgeführten Integration der beiden Applikationen.

Durch die Integration des iHub-WfMS erhält die eCCo-Anwendung eine zentrale Steuerungs-Komponente, die den Ablauf der gesamten Anwendung kontrolliert. Als ein weiterer Vorteil gilt die Trennung der Applikations-Logik und die dadurch erhöhte Modularität der Anwendung. Die Logik wird in den Task-Objekten realisiert, dessen

Zusammenhang in einer Prozessdefinition (XML-Datei) festgelegt wird. Deshalb kann die Logik jederzeit verändert werden, ohne dass andere Komponenten modifiziert werden müssen.

Die Aufgabe der Integration des iHub-WfMS kann man in folgende Unterpunkte aufteilen:

1) Implementierung eines WfMS-Clients für das iHub-WfMS

2) Integration des WfMS-Clients in die eCCo-Applikation

3) Erstellung einer Prozessdefinition und Realisierung der TaskExecutables

An dieser Stellt ist es notwendig, eine gewisse Klarheit im Gebrauch des Begriffs *Client* einzubringen. Solange man ein einfaches Client/Server-System betrachtet, ist es nicht schwer diese beiden Bereiche zu trennen. Sobald es sich aber um ein komplexeres System mit mehreren Komponenten handelt, die sowohl eine Client- als auch eine Server-Rolle tragen, ist Vorsicht geboten. Daher wird in diesem Abschnitt zwischen den Clients für das eCCo-System (im Folgenden *Web-Clients* genannt) und dem Client für das iHub-Workflow-Management-System (im Folgenden *WfMS-Client* genannt) unterschieden.

Der WfMS-Client, genannt *EchubBroker*, stellt einen Singleton[13] dar, da die Workflow-Engine des iHub-Prototyps nur einen einzigen Client bedienen kann. So wurde es mit dem Singleton-Entwurfsmuster möglich, dass eine einzige EchubBroker-Instanz gleichzeitig mehrere Anfragen an das iHub-WfMS zur Bearbeitung der Geschäftsprozesse stellen kann. Zu den Aufgaben des EchubBroker zählt unter anderem die Vergabe von eindeutigen Prozess-IDs, die Verwaltung der Workflow-IDs sowie die Erzeugung eines StartEvents und dessen Übermittlung an die Workflow-Engine. Der WfMS-Client muss für die Kommunikation mit dem iHub-WfMS die *MessageBroker-Listener*-Schnittstelle erweitern.

Die Verbindung zwischen der Presentation- und der Business-Schicht erfolgt durch die Applikationskomponente *EccoDispatcher*. Sie erhält die Anfrage des Web-Clients und reicht diese in die Geschäfts-Schicht zur weiteren Bearbeitung weiter. In der Geschäfts-Schicht ist die Anwendungslogik lokalisiert. Jedes Objekt der EccoDispatcher-Klasse startet über den WfMS-Client einen Arbeitsvorgang. Dieser Arbeitsvorgang wird durch einen oder mehrere Workflows beschrieben und hängt von der vom Benutzer gewählten Aktion ab. Darüber hinaus muss der EchubBroker sicher stellen, dass während der Workflow-Bearbeitung das EccoDispatcher-Objekt in Warte-Zustand versetzt wird. Nach der erfolgten Verarbeitung des Arbeitsvorgangs wird der WfMS-Client über die Callback-Methode der *MessageBrokerListener*-Schnittstelle `readEvent(FinishedEvent event)` informiert. Der Parameter vom Datentyp `FinishedEvent` enthält das Ergebnis der Workflow-Bearbeitung. Im Falle der eCCo-Applikation handelt es sich dabei stets um eine Adresse der JSP, die an den Benutzer als Antwort seiner

[13] Ein **Singleton** ist ein Erzeugungsmuster, das genau eine Instanz von einer Klasse erzeugt und somit einen globalen Zugriffspunkt auf das Objekt bereit stellt [Ga96].

Anfrage geschickt wird. Zum Schluss wird das entsprechende EccoDispatcher-Objekt aus dem Warte-Zustand zurück geholt und aktiviert.

Die Integration des WfMS-Clients in die eCCo-Applikation ist in der Presentation-Tier, in der Klasse *EccoDispatcher*, untergebracht. Diese Schicht ist für den Empfang von Anfragen eines Web-Clients sowie deren Interpretation und Weiterleitung an den Business-Tier zur Verarbeitung zuständig. Die Verarbeitung übernimmt dann das Workflow-Management-System. Für jede Anfrage des Web-Clients, die von der Aktion des Benutzers abhängt, wählt der EccoDispatcher den passenden Workflow aus und veranlasst durch den WfMS-Client seine Bearbeitung.

Als letztes bleibt noch die Erstellung einer Prozessdefinition und Realisierung der TaskExecutables. Eine einfache Darstellung der Prozessdefinition kann wie folgt beschrieben werden. Die Bearbeitung jeder möglichen Aktion des Benutzers, wie zum Beispiel die Anmelde-Prozedur oder Auswahl eines Fahrzeugs, wird durch einen Workflow beschrieben. Ein Workflow enthält in seiner Definition ein oder mehrere TaskExecutables (Java-Klassen), welche die zugrundeliegende Business-Logik abbilden. Die Business-Logik beschreibt die Geschäftsprozesse.

Kapitel 5

Implementierung einer EJB-Applikation

Der Schwerpunkt dieser Diplomarbeit liegt in der Portierung des iHub-WfMS auf einen Applikationsserver. Die dadurch entstandene EJB-Applikation heißt *iHubEJB*. In diesem Kapitel werden sämtliche Anwendungs-Komponenten des iHub-WfMS bezüglich ihrer Funktionalität analysiert, um aus diesen Erkenntnissen eine Architektur für iHub-EJB-Applikation auf der J2EE Plattform aufzuzeigen.

5.1 Applikationsserver und Entwicklungsumgebung

Sun Microsystems hat eine Referenzimplementierung von EJB als Bestandteil der Java 2 Enterprise Edition (J2EE) zur Verfügung gestellt. Im Anfangsstadium dieser Arbeit kam diese Referenzimplementierung zum Einsatz. Bei der Implementierung der Message-Driven-Beans (MDB) hat es sich aber herausgestellt, dass die Sun Reference Implementation von J2EE fehlerhaft ist. Die parallele Ausführung der MDB-Instanzen wurde von dem EJB-Container der Referenzimplementierung nicht unterstützt. Daraufhin wurde *JBoss* Version 3.0.6 eingesetzt. Mit dieser J2EE-Implementierung war das Problem bezüglich der Parallelität der Message-Driven-Beans gelöst.

JBoss ist ein vollständig in Java implementierter, standardkonformer Open Source J2EE Application Server. Zu seiner Ausstattung zählt nicht nur die obligatorische EJB Unterstützung oder die Integration von verbreiteten Webservern wie *Tomcat* oder *Jetty*, sondern auch die Integration von Fremdsystemen mit der Java Connector Architecture (JCA). Das modulare Design des JBoss Applikationsservers, das im Kern aus einem Java Management Extension Server (JMX) besteht, ermöglicht eine Erweiterung über komponentenbasierte Plug-Ins. JBoss bietet unter anderem JBossMQ für Java Messaging Service, JBossTX für JTA/JTS, JBossCMP (Container Managed Persistence) zur Unterstützung der Persistenz.

Als Entwicklungsumgebung, kurz IDE (Integrated Development Environment), wurde ***Eclipse*** Version 2.1 gewählt. Dabei handelt es sich um ein Open Source Project, dass von IBM ins Leben gerufen wurde. Der Entwickler erhält eine umfangreiche Entwicklungsumgebung, die über sogenannte Plug-Ins beliebig erweiterbar ist. Plug-Ins bilden die kleinsten Funktionseinheiten innerhalb der Plattform. Weitere Open Source Werkzeuge wurden in Eclipse eingebunden, um die Entwicklung der Applikation effektiver zu gestalten.

Zum Kompilieren wurde ***Ant*** (Another Neat Tool) benutzt, das im Rahmen des Jakarta-Projekts entwickelt wurde. Ant ist ein plattformübergreifendes Build-Werkzeug, das mit Java arbeitet und als Format für Buildfiles XML verwendet. Das Tool ist speziell auf die Erfordernisse des Build-Prozesses in der Java-Umgebung zugeschnitten und ist besonders gut zur Automatisierung komplexer Build-Aufgaben geeignet.

Zur Generierung der Deployment-Deskriptoren und EJB-Stubs wie EJBObject- und EJBHome-Interfaces wurde ***XDoclet*** eingesetzt, die eine erweiterte Javadoc-Engine darstellt. Dazu werden in dem Bean-Code Javadoc-Tags eingebettet. In dem Build-Skript sorgt der Ant-Task *ejbgen* für die Erzeugung des Quellcodes der entsprechenden EJB-Stubs und der Deployment-Deskriptoren, die auf den Applikationsserver abgestimmt sind. Die folgenden zwei Codebeispiele zeigen die Generierung einer einfachen Session Bean.

Quellcode-Auszug aus der Datei „WorkflowBean.java:

```
1.  /**
2.   * @ejb:bean name="Workflow"
3.   *           display-name="Workflow Bean"
4.   *           type="Stateful"
5.   *           transaction-type="Container"
6.   *           jndi-name="ejb/Workflow"
7.   *
8.   * @ejb:ejb-ref ejb-name="Task"
9.   *              ref-name="ihub/Task"
10.  *
11.  * @author Eugen Resch
12.  */
13. public class WorkflowBean implements SessionBean {...}
```

Im Bean-Header wird mit der Syntax @ejb:bean gefolgt von Attributen die Eigenschaften der Bean näher ausgezeichnet. Das kann z.B. die Art der Bean sein (zustandslos oder zustandsbehaftet bei einer Session-Bean), das Transaktions-Konzept oder der JNDI-Pfad.

Quellcode-Auszug aus der Datei „build.xml“

```
1. <?xml version="1.0" encoding="UTF-8" ?>
2.   <project name="iHub EJB Build Script" default=" xdoclet-generate "
3.                                                          basedir=".">
4.   ... <!-- Property definitions /-->
5.   <target name="xdoclet-generate" depends="init">

6.     <taskdef name="ejbdoclet" classname="xdoclet.ejb.EjbDocletTask">
7.       <classpath refid="xdoclet.path"/>
8.     </taskdef>
```

```
9.      <ejbdoclet sourcepath="${source.ejb.dir}"
10.       destdir="${source.ejb.generate.dir}"
11.       classpathref="base.path"
12.       excludedtags="@version,@author"
13.       ejbspec="${ejb.version}"
14.       force="${xdoclet.force}">

15.         <fileset dir="${source.ejb.dir}">
16.           <include name="**/*Bean.java"/>
17.         </fileset>
18.         <packageSubstitution packages="repository,task,workflow"
19.                                       substituteWith="interfaces"/>
20.         <remoteinterface/>
21.         <localinterface/>
22.         <homeinterface/>
23.         <localhomeinterface/>
24.         <deploymentdescriptor destdir="${source.resources.dir}"
25.                                           validatexml="false" />
26.     </ejbdoclet>
27.   </target>
```

In dem Ant-Buildfile wird ein durch die XDoclet-Bibliothek definierter Task *ejbdoclet* aufgerufen. Die Art der Bean oder des Source-Codes bestimmt hier, was der Entwickler generieren kann wie zum Beispiel EJBs, Web-Dateien oder Deskriptoren.

Darüber hinaus wurde die Entwicklungsumgebung um ein weiteres Plug-In, das **JBOSS-IDE,** erweitert. Dieses Plug-In ermöglicht es einen Debugger zu verwenden, der zur Laufzeit Schritt für Schritt durch den Quellcode der auf dem JBoss-Applikationsserver laufenden Klassen gehen kann.

5.2 Design von iHubEJB

Das Design von iHubEJB besteht, strukturell gesehen, aus zwei Teilen. Die Abbildung 5.1 verdeutlicht es graphisch. Der erste Teil umfasst die Klassen *Engine, EngineDaemon* und *EngineEventListener* und ist für den Empfang der Anfragen eines Clients sowie für das Weiterreichen dieser zur weiteren Bearbeitung zuständig. Hinzu kommt auch die Vermittlung der bidirektionalen Kommunikation zwischen dem Client und den Arbeitsvorgängen des Geschäftsprozesses.

Das *Engine*-Objekt ist als ein Singleton implementiert und kann somit Anfragen mehrerer Clients verarbeiten. Singleton (vgl. [Ga96]) ist ein objektbasiertes Erzeugungsmuster, dessen Zweck es ist genau ein Objekt von einer Klasse zu erzeugen und einen globalen Zugriffspunkt auf das Objekt zu geben. Neben dem Speichern der Referenz des Clients, initialisiert die *Engine*-Klasse ein *EngineEventListener*-Objekt. Die Aufgabe der Kommunikation mit dem funktionalen zweiten Teil wird unter diesen zwei Klassen aufgeteilt. Die *Engine* dient hauptsächlich zum Senden der Nachrichten an eine EJB Komponente des zweiten Teils – *ManagerMDBean*. Den Empfang der Nachrichten übernimmt dann die *EngineEventListener*-Klasse, die die Schnittstelle *javax.jms.Message-Listener* erweitert.

Der zweite Teil ist für die Verarbeitung der einzelnen Arbeitsvorgänge zur Umsetzung eines Geschäftsprozesses verantwortlich. Die bereits angesprochene *ManagerMDBean*-Klasse erweitert die *MessageDrivenBean*-Schnittstelle. Wie es oben (Abschnitt 3.2.3 „Message-Driven-Beans“) bereits erläutert wurde, sind Message-Driven-Beans in EJB 2.0 Spezifikation eingeführt worden um einen Asynchronbetrieb von ankommenden JMS Nachrichten zu gewährleisten [EJB01 §15.2].

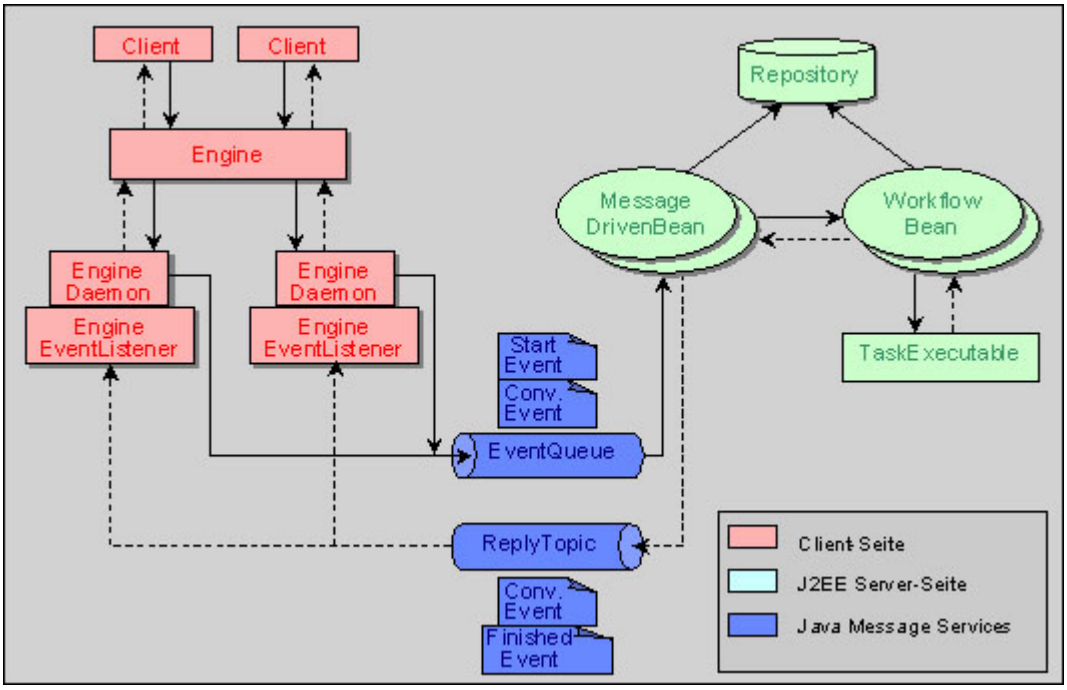

Abbildung 5.1: Struktureller Aufbau von iHubEJB

Zur Hauptaufgabe der *ManagerMDBean* gehört das Empfangen und Senden der Nachrichten. In diese JMS Nachrichten werden Value Objects verpackt. *Value Objects* werden für den Transfer von Daten zwischen den zwei Teilen der iHubEJB-Anwendung sowie den einzelnen EJB Komponenten eingesetzt. Value Objects bündeln Daten und verbessern durch Minimierung der Netzwerklast die Performanz der Anwendung [Al01b]. Bei jedem Aufruf einer Enterprise Bean, sei es eine Session Bean oder eine Entity Bean, erfolgt über ein Netzwerk ein entfernter Methodenaufruf (Remote Method Invocation). Es gibt eine ganze Reihe an Value Objects im iHubEJB System. Die wichtigsten davon sind jedoch *StartEventVO*, *TaskEventVO* und *FinishedEventVO*.

Die *ManagerMDBean*-Klasse initialisiert außerdem die Enterprise Beans *RepositoryBean* und die *WorkflowBean*. In der *RepositoryBean* werden alle Daten abgelegt, die für einen Geschäftsprozess notwendig sind. Dazu zählt zum Beispiel die Prozess-Definitionsbeschreibung oder die aktuelle Position eines Geschäftsprozesses. Die *WorkflowBean* ist eine Session-Bean und ist für die Abarbeitung der einzelnen Schritte aus der Prozess-Definitionsbeschreibung verantwortlich. Die Bean greift auf die Daten der *RepositoryBean* zurück und verändert sie gegebenenfalls. Falls bei der Abarbeitung der Prozess-Definitionsbeschreibung ein *Task*-Objekt vorkommt, so initialisiert die Bean dieses Objekt und ruft dessen Callback-Funktion `execute()` auf. Die *Task*-Objekte

implementieren die Logik der Arbeitsvorgänge und stellen ganz gewöhnliche Java-Klassen dar.

Die strukturelle Aufteilung des gesamten Systems erkennt man sofort, wenn der Ort betrachtet wird, an dem jeder der beiden Teile abläuft. Der erste Teil, die so genannte Client-Seite [Mo01a], befindet sich lokal in der Java VM, wobei der zweite Teil auf dem entfernten J2EE-Server läuft.

Die Kommunikation zwischen den beiden Teilen erfolgt nicht direkt, sondern baut auf der Nachrichtenübertragung des Java Messaging Services auf. Alle Nachrichten, die zum *ManagerMDBean* geleitet werden, werden an eine Queue (`EventQueue`) gesendet. Die Nachrichten, die für die andere Richtung bestimmt sind, also von *ManagerMDBean* zu *EngineEventListener*, werden an ein Topic (`ReplyTopic`) geleitet. Das Point-to-Point Modell mit der `EventQueue` wurde aus dem Grund gewählt, weil jede von der *Engine* gesendete Nachricht genau von einem *ManagerMDBean*-Objekt empfangen wird. Dies entspricht der Eigenschaft einer JMS Queue. Im Gegensatz dazu werden Nachrichten, die von `ReplyTopic` empfangen werden, an alle vorhandenen *EnginEventListener* verschickt (Publish-and-Subscribe Modell). So bleibt die Option zur weiteren Erweiterung des iHubEJB Systems offen, wenn die Zahl der *Engine-EventListener's* variabel sein sollte.

Das ursprüngliche Design von iHub-WfMS ist um eine weitere Eigenschaft erweitert worden. Das *Task*-Objekt ist nun in der Lage eine Art **Interaktion** mit dem Client zu führen. Die Notwendigkeit für eine Interaktion ergibt sich aus der Tatsache, dass der lokale Client von dem entfernten *Task*-Objekt, das sich auf dem J2EE Server befindet, getrennt ist. Dabei kann der Server sogar auf einem anderen Rechner laufen. Nun ist es ja nicht auszuschließen, dass in einem Arbeitsvorgang, der durch einen Task beschrieben wird, während eines Geschäftsprozesses eine Bestätigung seitens des Clients erfolgen muss. Das kann unter anderem eine Auswahl von vorgegebenen Alternativen sein, die der Benutzer von der Kommandozeile oder von einem GUI aus machen soll. Der Dialog läuft über die oben beschriebene nachrichtenorientierte Kommunikation mit Unterstützung des JMS ab. Auf weitere Details der iHubEJB-Implementierung wird im nächsten Abschnitt eingegangen.

5.2 Einschränkungen der EJB 2.0 und Lösungen

Das Design des iHubEJB bringt zwingende Modifikationen der ursprünglichen Struktur der Workflow-Management-Engine mit sich. Durch die Restriktionen der EJB-Spezifikation, auf die weiter unten näher eingegangen wird, kommt es schon in vornherein zu einer wesentlichen Veränderung des Gesamtsystems.

Bei der Implementierung der EJB Komponenten gibt es eine Reihe von Einschränkungen, die eingehalten werden müssen. Sie ergeben sich aus der Tatsache, dass der EJB Container für Sicherheit, Ressourcenverwaltung, Nebenläufigkeit und andere Aufgaben zur Unterhaltung der Enterprise JavaBeans verantwortlich ist und diese von der Container Software auf mehrere Java VMs verteilt werden können.

5.2.1 Thread-Verwaltung

Bei der Programmierung von EJB ist es nicht möglich eigene *Thread*-Klassen zu schreiben. Laut der EJB 2.0 Spezifikation ist die Verwaltung von Threads, wie zum Beispiel Starten, Stoppen, Suspendieren oder Wiederaufnehmen eines Threads, zu vermeiden [EJB01 §24.1.2]. Diese Funktionen sind für den EJB Container vorgesehen. Die Fähigkeit des Containers, den Lebenszyklus einer EJB zu bestimmen, würde durch Multithreading im Anwendungscode der EJB zu Einschränkungen führen. Anwendungen mit der Multithread-Eigenschaft sind nach wie vor möglich, jedoch liegt die Überwachung von Multithreading im Aufgabebereich des Containers und nicht in dem der Enterprise Bean.

5.2.2 Nebenläufigkeit

Das Problem der Nebenläufigkeit wird in EJB mit einer einfachen Lösung umgangen. EJB verbietet den nebenläufigen Zugriff auf Bean-Instanzen [Mo01a]. Es können zwar mehrere Clients mit einem EJB-Objekt verbunden sein, aber nur ein Client kann auf einmal auf die Bean-Instanz zugreifen. Abbildung 5.2 zeigt zwei Clients, die das gleiche EJB-Objekt verwenden. Der zweite Client kann erst dann auf die Bean-Instanz zugreifen, wenn der erste Client seinen Methodenaufruf beendet hat.

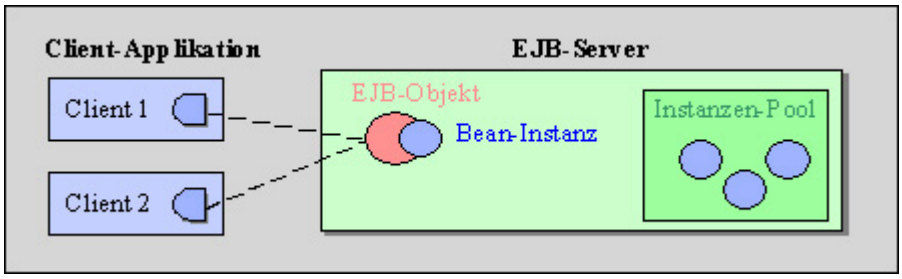

Abbildung 5.2: Gemeinsamer Zugriff auf ein EJB-Objekt [Mo01a]

5.2.3 Reentranz

Ein wichtiger Aspekt, den man an dieser Stelle nicht außer Acht lassen darf, ist das Konzept der *Reentranz* [Mo01a]. Reentranz liegt dann vor, wenn ein Kontroll-Thread in eine Bean-Instanz erneut einzutreten versucht. Die Voreinstellung von Bean-Instanzen in EJB schreibt Nicht-Eintrittsfähigkeit vor [EJB01 §§ 7.11.8, 10.5.11], was bedeutet, dass solche Rückkopplungsschleifen nicht erlaubt sind. Man spricht von Rückkopplung, wenn Bean A eine Methode von Bean B aufruft, die dann wiederum Bean A aufruft.

Abbildung 5.3 zeigt dieses Szenario. Hier ruft der Client eine Methode von Bean A auf. Als nächstes ruft Bean A eine Methode von Bean B auf. Bis jetzt verläuft die Interaktion korrekt, da Bean A der Client von Bean B ist. Wenn nun aber Bean B versuchen würde eine Methode an Bean A aufzurufen, dann würde dieser Aufruf blockiert wer-

den, weil sich dieser Thread bereits in Bean A befindet. EJB erlaubt nicht, dass ein Kontroll-Thread eine Bean-Instanz erneut betritt.

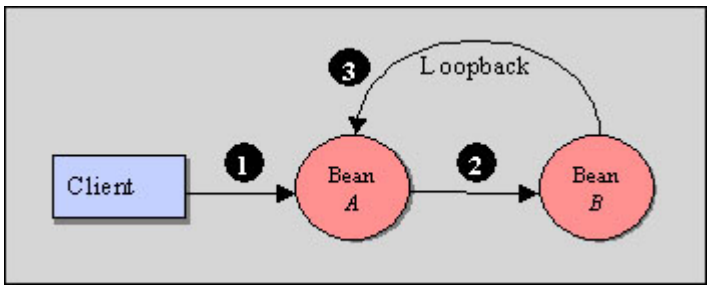

Abbildung 5.3: Ein Rückkopplungsszenario [Mo01a S.65]

Die Vorgehensweise hinsichtlich der Reentranz unterscheidet sich je nachdem, ob es sich um Session-Beans oder Entity-Beans handelt. Session-Beans können nie wiedereintrittsfähig sein und lösen eine *RemoteException* aus. Dasselbe geschieht mit Entity-Beans, sofern sie im Deployment-Deskriptor nicht ausdrücklich als wiedereintrittsfähig konfiguriert werden, wovon die Spezifikation jedoch abrät. Wie es oben bereits besprochen wurde, werden die Client-Zugriffe auf eine Bean synchronisiert, d.h. dass nur ein Client auf einmal auf eine bestimmte Bean zugreifen kann. Das Problem mit einem wiedereintrittsfähigem Code besteht darin, dass das EJB-Objekt, das Methodenaufrufe an der Bean-Instanz abfängt und delegiert, nicht zwischen einem wiedereintrittsfähigem Code und einem Multithread-Zugriff des gleichen Clients im gleichen Transaktionskontext unterscheiden kann.

Um diesen Sachverhalt im Zusammenhang mit Interaktionen zwischen Beans besser zu verstehen, gilt in EJB ein grundlegendes Konzept: Enterprise Beans greifen aufeinander mit Hilfe von Remote-Referenzen zu.

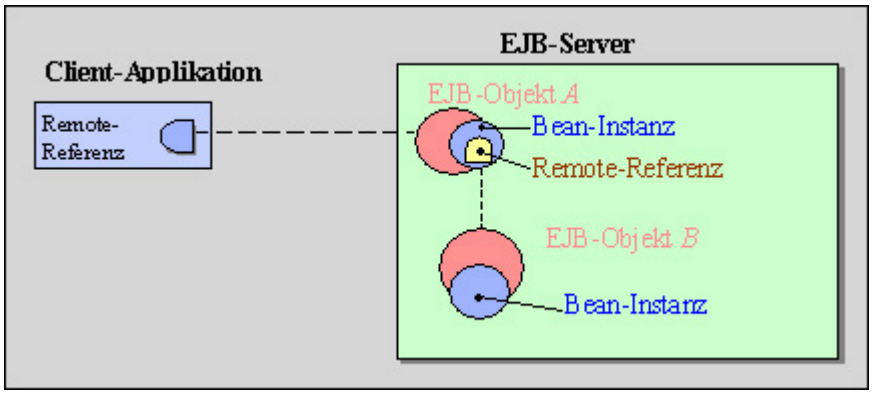

Abbildung 5.4: Interaktion unter Enterprise Beans [Mo01a]

Dieses Konzept beschreibt die Art und Weise, wie Beans miteinander interagieren. Wenn zwei Beans miteinander arbeiten, dann geschieht das mit anderen Worten genauso, wie ein Applikations-Client das tun würde, also unter Verwendung des Remote-Interfaces, das von einem EJB-Objekt implementiert wird (vgl. Abbildung 5.4).

Diese Regel ist für eine vollständige Ortstransparenz notwendig, die als eine der Forderungen an eine Komponentenarchitektur gestellt wird. Da Interaktionen mit Beans immer über Remote-Referenzen laufen, können Beans ihren Aufenthaltsort verändern, ohne dass dies den Rest der Anwendung beeinträchtigt.

5.2.4 Lösung zum Thread-Problem und Reentranz

Wichtige Klassen des iHub-Systems sind von der *Thread*-Klasse abgeleitet worden. Diese Klassen, wie zum Beispiel *Workflow* und *WFManager*, laufen nach der Initialisierung eigenständig und verarbeiten unabhängig von dem aktuellen Status die ankommenden Events, sei es *WFStartEvent*, der von *WFEngine* an *WFManager* geschickt wird oder *WFTaskEvent*, der während der Verarbeitung eines Tasks an *Workflow* versandt wird. Die Verwendung von *Thread*-Objekten entkoppelt das ganze System. Das bedeutet letztendlich, dass der Kontrollfluss nach dem Aufruf der `readEvent()` Methode der jeweiligen Klasse zu dem Aufrufenden zurückkehrt. Abbildung 5.5 zeigt zur Verdeutlichung einen Teilausschnitt aus dem Sequenzdiagramm für diese Klassen. Das Diagramm ist einfacher gehalten um eine bessere Übersicht zu ermöglichen.

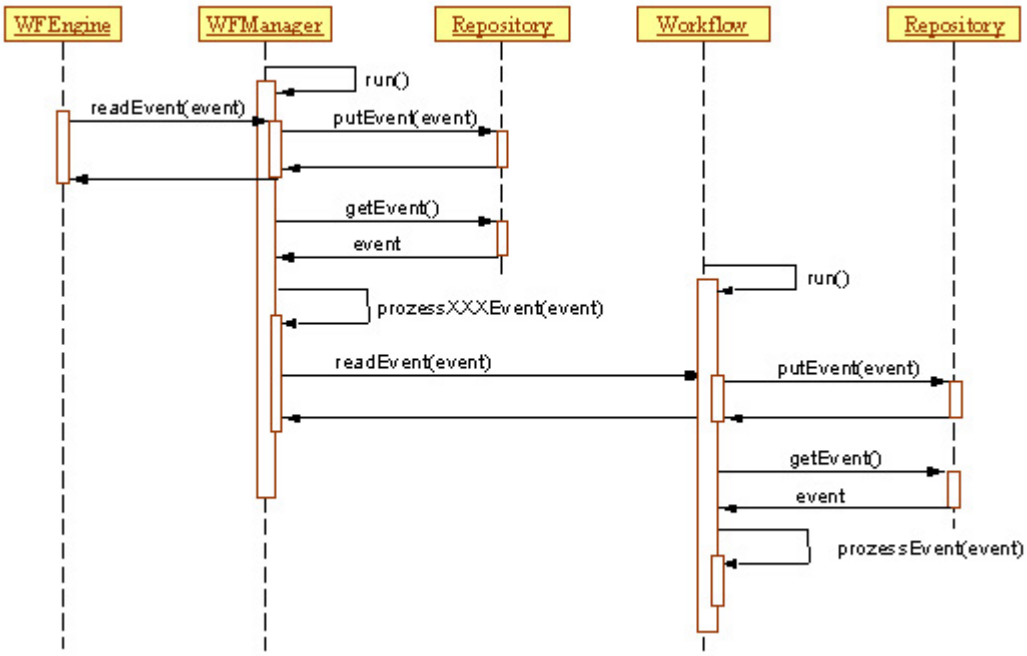

Abbildung 5.5: Sequenzdiagramm von iHub

Die bereits oben besprochene Einschränkung der Thread-Verwaltung durch EJB-Container (Konzept der Reentranz) macht es unmöglich eigene *Thread*-Klassen in iHubEJB zu verwenden. Dadurch geht die Eigenschaft einer entkoppelten Verbindung unter den Komponenten verloren. Auf der anderen Seite wird von der Anwendung iHubEJB eine asynchrone Kommunikation mit ihren Clients erwartet. Als Lösung bietet sich der Einsatz von Message-Driven-Beans (MDB) an. MDB sind zustandslose, transaktionssichere, auf dem J2EE Server laufende Komponente [Mo01b], die, im Gegenteil zu Ses-

sion Beans und Entity Beans, zur asynchronen Weiterverarbeitung von JMS Nachrichten dienen. MDB werden von EJB Container in gleicher Art und Weise verwaltet wie zustandslose Session Beans. Die Message-Driven-Beans erweitern die *MessageListener*-Schnittstelle und enthalten deshalb eine `onMessage()` – Methode. Diese Methode wird nach Erhalt einer Nachricht automatisch aufgerufen. Folglich bewirkt dieser Aufruf wie jeder andere Aufruf einer Geschäftsmethode, dass eine Bean-Instanz der Klasse *ManagerMDBean* aus dem Instanzen-Pool geholt und dem EJB-Objekt zugewiesen wird. Eine MDB-Instanz bleibt solange blockiert, bis der Methodenaufruf von `onMessage()` zurückkehrt. Würden in der Zeit der Blockierung weitere Nachrichten an die *ManagerMDBean* gesendet, so würden diese an andere Instanzen aus dem gleichen Instanzen-Pool weitergeleitet.

Das Design von iHubEJB wurde so konzipiert, dass es während der Laufzeit zu keinen Problemen mit Reentranz kommen kann. Abbildung 5.6 zeigt die zeitliche Abfolge der Zugriffe unter den Komponenten. Die *ManagerMDBean* erzeugt zuerst eine *Repository*-Bean und eine *Workflow*-Bean, welche die Referenz auf eine gerade erzeugte *Repository*-Bean erhält. Danach wird *Repository* mit notwendigen Daten für eine weitere Bearbeitung des Workflows gefüllt, unter anderem auch die Referenz auf die *Workflow*-Bean. Als nächstes wird die `readEvent()` – Methode von *Workflow* aufgerufen. Während der Abarbeitung von Arbeitsvorgänge greift die *Workflow*-Bean auf Geschäftsmethoden der *Repository*-Bean zu. Ist der Vorgang des Workflows beendet, kommt es zu einer Interaktion mit dem Client des WfMS oder im Falle eines Abbruchs kehrt der Methodenaufruf `readEvent()` zur Message-Driven-Bean zurück. Das Ergebnis des Aufrufs wird in ein Nachrichten-Objekt verpackt und über JMS an den *EngineEventListener* geschickt, der es an den Client des WfMS weiterreicht. Handelt es sich dabei um eine Konversations-Anfrage, so wird die Antwort des Clients zurück an die *ManagerMDBean* geschickt, die dann wieder die `readEvent()` – Methode des Workflows aufruft, der für diese Interaktion zuständig ist.

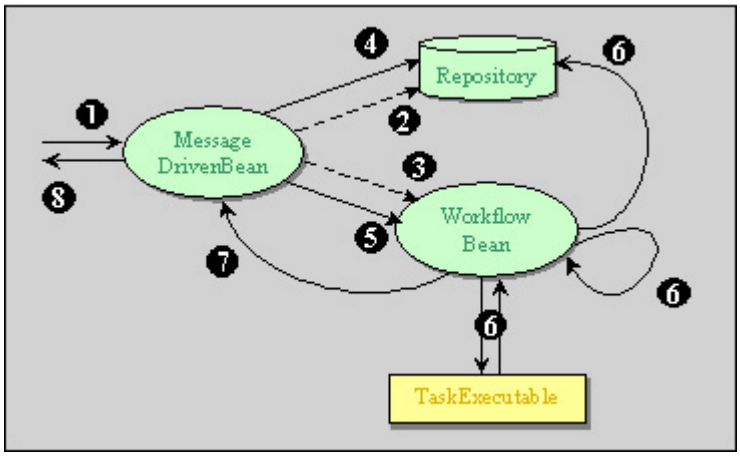

Abbildung 5.6: Zugriffe der iHubEJB-Komponenten untereinander

Die Verwendung der nachrichten-orientierten Kommunikation lässt den Client nach der Nachrichtenübertragung von *Engine* mit seiner Berechnung fortfahren. Nach der Bear-

beitung der Anfrage wird der Client durch das Aufrufen seiner Callback-Funktion wieder benachrichtigt.

5.2.5 Übergabe von Referenzen mit JMS

Laut der EJB 2.0 Spezifikation [EJB01 §24.1.2] darf die Referenz auf die eigene Instanz (*this*) als Argument bei dem Aufruf einer Komponenten-Methode nicht mitgegeben werden. Ebenfalls darf *this* nicht als Rückgabewert einer Methode verwendet werden. Stattdessen soll die Methode `getEJBObject()` des *SessionContext*- oder *EntityContext*-Objekts benutzt werden. Das folgende Beispiel verdeutlicht diesen Sachverhalt. Bean `A` soll ihre eigene Referenz an Bean `B` übergeben. Nach dieser Übergabe soll dann Bean `B` Methoden von Bean `A` aufrufen. Die *this*-Referenz zeigt auf die Bean-Instanz, die sich im EJB-Container befindet. Der Container ist der einzige, der Methoden einer Bean-Instanz direkt aufrufen kann und deshalb kann *this* nicht übergeben werden. Bean-Clients greifen auf die Bean-Instanz indirekt zu, indem sie einen Methodenaufruf an ein Objekt vom Typ Remote-Interface tätigen. Es ist die Referenz an dieses Objekt, also die Remote-Referenz der Bean, die Bean `A` an Bean `B` übergibt.

Bei der Verwendung der Methode `getEJBObject()` des *SessionContext*- bzw. *EntityContext*-Objekts stellt EJB Container das Remote-Interface der Bean bereit. Das EJB-Objekt, das von der Methode `getEJBObject()` zurückgegeben wird, erweitert nur die `java.rmi.Remote`-Schnittstelle. Für eine Übertragung dieses Objekts über ein Netzwerk mit Hilfe von JMS ist das nicht ausreichend. Das JMS API definiert für diesen Zweck einen Nachrichtentyp *ObjectMessage* und schreibt vor, dass dieser Nachrichtentyp zum Senden eines serialisierbaren Objekts benutzt wird [J2EE03]. Mit anderen Worten ausgedrückt, soll dieses Objekt die `java.io.Serializable` implementieren.

Das Schlüsselwort für die Lösung des Problems ist ein *Handle*-Objekt. Die `getHandle()`- Methode des EJB-Objekts liefert ein `javax.ejb.Handle`-Objekt zurück. Dieses ist eine serialisierbare Referenz auf das EJB-Objekt. Das bedeutet, dass das Handle-Objekt mittels Objekt-Serialisierung zum Speichern dessen Inhalts in einer Datei oder zum Versenden über das Netzwerk verwendet werden kann [Ne97]. Mit dem *Handle* kann man also eine Remote-Referenz auf ein EJB-Objekt wiedererzeugen, die auf den gleichen Typ von Session Bean oder dieselbe eindeutige Entity Bean verweist, von der das *Handle* kommt [Mo01a]. Das folgende Codebeispiel zeigt, wie *Handle*-Objekte und Remote-Referenzen in der Methode `onMessage()` der Klasse *ManagerMDBean* verwendet werden:

Quellcode-Auszug aus der Klasse „ManagerMDBean.java“:

```
1.  public void onMessage(Message inMessage) {

3.    EventVO evo = (EventVO) ((ObjectMessage) inMessage).getObject();
4.    switch (evo.getType()) {
5.      case EventVO.START_EVENT :
6.        // Create a Repository
7.        Repository repository = createRepository("iHubEJBRepos");
8.        // Create a Workflow with the Reference to the Repository
9.        WorkflowHome home = lookupWorkflow();
```

```
10.       Workflow workflow =
11.       home.create(repository,
                      getProcDef((StartEventVO) evo.getXmlWorkflowId()),
                      ((StartEventVO) evo).getXmlWorkflowId());

15.       // Write the new Workflow-Reference to the Repository
16.       repository.setWorkflow(workflow.getHandle(), null);
17.       ...
18.       break;
19.     case EventVO.CONVERSATION_EVENT :
20.       // Get the Handle-Object of Repository from the Value-Object
21.       Handle handle =
                    ((ConversationEventVO)evo).getRepositoryHandle();
22.       // Get the Repository-EJBObject
23.       Repository repository = (Repository) handle.getEJBObject();
24.       // Get the Handle-Object of Workflow from Repository
25.       handle =
           repository.getWorkflow(((ConversationEventVO)evo).getWfId());
26.       // Get the Workflow-EJBObject
27.       Workflow workflow = (Workflow) hadle.getEJBObject();
28.       ...
29.       break;
30.   }
31.   ...
32. }
```

5.3 Leistungsvergleich

Mit Hilfe einer Testapplikation *Counter*, die bereits im Abschnitt 4.1.2 kurz eingeführt wurde, wurden Zeitmessungen an iHub-WfMS und iHubEJB vorgenommen. Allerdings wurde auf das Warten nach jedem Inkrementieren des Zählers verzichtet und die Abbruchbedingung des Zählers auf 10.000 festgelegt. Diese elementaren Messungen sollen als Anhaltspunkt für einen grundsätzlichen Leistungsvergleich zwischen dem vorgestellten iHub-Workflow-Management-System und dessen portierter Version dienen.

Alle Messungen wurden lokal ausgeführt. Als Referenzmaschine diente ein Pentium III Prozessor 700 MHz mit 256 MB RAM. Als Laufzeitumgebung für Java wurde Java 2 SDK, Standard Edition Version 1.4.1 sowie Java 2 SDK, Enterprise Edition Version 1.3.1 verwendet. Als Applikationsserver wurde JBoss 3.0.6 eingesetzt.

Es wurden 20 Testläufe durchgeführt. Wie zu erwarten war, lag die Geschwindigkeit des iHub-WfMS viel höher als die des iHubEJB. Die Ausführungszeit der Testapplikation *Counter* mit iHubEJB hat fast um das vierfache länger gedauert als mit iHub-WfMS (vgl. Tabelle 4.1).

	iHub-WfMS	*iHubEJB*
durchschnittliche Zeit (msec)	7.955	30.607

Tabelle 5.1: Performance-Vergleich

Ursache für diese Differenz ist unter anderem in der Architektur des EJB-Komponentenmodells zu suchen. Die EJB-Architektur nimmt dem Anwendungsentwickler Aufgaben wie z.B. eine low-Level Transaktions- und Zustandsverwaltung, Zugriffskontrolle, Nebenläufigkeit und verteilte Ressourcenverwaltung für die Programmierung ab. Dies schlägt sich in der Ausführungsgeschwindigkeit nieder. Darüber hinaus spielt noch die Tatsache eine bedeutende Rolle, dass der Client und der J2EE-Server auf verschiedenen virtuellen Maschinen laufen. Jeder Methodenaufruf an einer Remote-Referenz verbirgt einen Datentransport vom Stub zum Server und zurück zum Stub, was ebenfalls einen beträchtlichen Zeitaufwand nach sich zieht.

Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Implementierung eines Workflow-Management-Systems vorgestellt. Nach einer genauen Darstellung des Aufbaus und der Funktionsweise des betrachteten WfMS wurde ein Vergleich mit dem standardisierten Referenzmodell der Workflow Management Coalition gemacht.

Es wurde ferner eine Integration des WfMS in einer bestehenden E-Commerce-Applikation durchgeführt. Des weiteren wurde ein Konzept zur Portierung des WfMS auf die J2EE-Plattform entwickelt. Mit der Zusammenführung des WfMS mit J2EE als Basis kann sich der Wunsch nach einer schnellen Integration von Altsystemen in die neue verteilte Internet/Intranet-Welt erfüllen. Diese Arbeit macht deutlich, dass die Portierung bestimmter Applikationen auf die J2EE-Plattform gegebenenfalls Konzeptveränderungen nach sich zieht. Bei der Programmierung von EJB Komponenten gelten eine Reihe von Einschränkungen, die eingehalten werden müssen. Das hängt damit zusammen, dass der EJB Container für die Ausführung einiger Dienste wie zum Beispiel Threading, Security oder Ressourcenverwaltung verantwortlich ist und Enterprise JavaBeans von der Container Software auf mehrere Java VMs verteilt werden können.

Für die Kommunikation der verteilten Module wurde das Konzept der Message Oriented Middleware verwendet. Durch die Kombination des JMS-Standards mit den Message-Driven-Beans Komponenten wurde die Voraussetzung dafür geschaffen, dass die Kommunikation asynchron verläuft und die Anfragen an das WfMS parallel verarbeitet werden.

Mit der Einführung des neuen Proposed Final Draft der EJB 2.1-Spezifikation eröffnen sich neue Möglichkeiten, die zur Erweiterung dieser Arbeit führen können. Die neue Spezifikation schreibt vor, dass EJB-Container einen Timer-Service anbieten müssen. Dieser kann zur Realisierung des Schedulers in dem Workflow-Management-System verwendet werden, um langläufige Geschäftsprozesse und periodische Aktivitäten anzustoßen.

Im Zusammenhang mit der Workflow-Engine wäre eine Entwicklung weiterer leichtgewichtiger Enterprise Applications wie Persistenz-, Kontext- und Transaktions-Manager sinnvoll. Die Verwendung dieser Komponenten mit dem WfMS kann eine breite Unterstützung bei Realisierung komplexer EAI-Anwendungen wie z.B. Mitarbeiter-Portale finden.

Literaturverzeichnis

[Al01a] D. Alur, J. Crupi, D. Malks: *Core J2EE Patterns - Best Practices and Design Strategies*. Prentice Hall PTR, 1st ed., 2001.

[Al01b] D. Alur, J. Crupi, D. Malks: *Sun Java Center J2EE Patterns*. First Public Release, Version 1.0 Beta, 2001.

[Be96] P. A. Bernstein: *Middleware: A model for distributed system Services*. In: Communications of the ACM, Vol. 39, No. 2, February 1996, S.86-98.

[Bö97] M. Böhm: *Modellierung von Workflows*. In: S. Jablonski, M. Böhm, W. Schulze (Hrsg): Worklfow-Management - Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie, dpunkt, 1997.

[Bu97] C. Bußler: *Organisationsverwaltung in Workflow-Management-Systemen*. Habilitationsschrift, Erlangen, 1997.

[eCCo01] A. P. Kalantidis: *Redesign eCCo*. iT media Consult GmbH, internes Whitepaper, 2001.

[EJB01] Sun Microsystems: *Enterprise JavaBeans™ Specification*. Version 2.0, Final Release, 14. Aug. 2001.

[Er93] G. Erdl, G. Schönecker: *Vorgangssteuerungssysteme im Überblick*. In: Office Management, 41 (1993) 3, S. 14-21.

[Ga96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Westley-Longman, 1. Auflage, 1996.

[Ha93] M. Hammer, J. Champy: *Reengineering the Cooperation - A Manifesto of Business Revolution*. Harper Business, New York, 1993.

[HaLa91] K. Hales, M. Lavery: *Workflow Management Software - the Business Opportunity*, London, 1991.

[Hase87] U. Hasenkamp: *Konzipierung eines Bürovorgangssystems*. Habilitationsschrift, Köln, 1987.

[Heil94] H. Heilmann: *Workflowmanagement - Integration von Organisation und Informationsverarbeitung. Theorie und Praxis der Wirtschaftsinformatik*. In: HMD (31), Heft 176, März 1994, S. 8-21.

[Ja97] S. Jablonski: *Architektur von Workflow-Management-Systemen*. In: Informatik Forschung und Entwicklung, 12 (1997), S.72-81, Springer, 1997.

[Ko76] E. Kosiol: *Organisation der Unternehmung*. 2. Aufl., Gabler, Wiesbaden, 1976.

[Le97] F. Lehmann: *Gebrauchssprachliche Methoden*. In: S. Jablonski, M. Böhm, W. Schulze (Hrsg): Workflow-Management - Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie, dpunkt, Heidelberg, 1997.

[Le98] F. Lehmann, E. Ortner: *Workflowsysteme: ein interdisziplinares Forschungs- und Anwendungsgebiet*. In: Informatik/Informatique, 2/1998, S. 3-12.

[Ma00] S. Maffeis, T. Haas, B. Kelly: *Gut verteilt ist halb gewonnen! Robuste Anwendungen mit JMS*. In: Java™ Spektrum 3/2000, SIGS Conferences, 2000, S.34-46.

[Mo01a] R. Monson-Haefel: *Enterprise Java Beans*. O'Reilly & Associates, 2. Aufl., 2001.

[Mo01b] R. Monson-Haefel: *Enterprise Java Beans*. O'Reilly & Associates, 3rd ed., 2001.

[Ne97] A. Newman: *Java - Referenz & Anwendungen*. Que, Special Edition, 1997.

[Or97] E. Ortner: *Glossar*. In: S. Jablonski, M. Böhm, W. Schulze (Hrsg): Workflow-Management: Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie, dpunkt, Heidelberg, 1997, S.485-493.

[Pa72] D. Parnas: *On the Criteria To Be Used in Decomposing Systems into Modules*. Communications of the ACM, Vol. 15, No. 12, 1972, S. 1053-1058.

[Ph97] M. Philipp: *Anforderungen der Internen Revision an Workflow-Managementsysteme*. In: Zeitschrift für Interne Revision, 1/1997, S. 24-39.

[Pi95] A. Ricot, P. Rohrbach: *Organisatorische Aspekte von Workflow-Management-Systemen*. In: IM Information Management, 1/1995, S. 28-35.

[Re94] H. Reinermann: *Vorgangssteuerung in Behörden. Theorie und Praxis der Wirtschaftsinformatik*. In: HMD (31), Heft 176, März 1994, S. 22-34.

[Te95] S. Teufel, C. Sauter, T. Mühlherr, K. Bauknecht: *Computerunterstützung für die Gruppenarbeit*. Bonn: Addison-Wesley, 1995.

[Ve95] G. Versteegen: *Alles im Fluß - Die Ansätze der Workflow Management Coalition*. In: iX Multiuser Multitasking Magazin, 3/1995, Heinz Heise, Hannover, S.152-160.

[We96] D. Weiß, H. Krcmar: *Workflow Management - Herkunft und Klassifikation*. Arbeitspapiere Lehrstuhl für Wirtschaftsinformatik Nr. 95, Universität Hohenheim, 1996.

[WfMC95] Workflow Management Coalition: *The Workflow Reference Model*. Document Status-Issue 1.1, Document Number TC00-1003, Januar 1995.

[WfMC99] Workflow Management Coalition: *Workflow Management Coalition - Terminology & Glossary*. Document Status-Issue 3.0, Document Number WFMC-TC-1011, Januar 1999.

Internet-Seiten

[J2EE] Sun Microsystems: *Java 2 Platform, Enterprise Edition - API Specification*. Version 1.3, http://java.sun.com/j2ee/sdk_1.3/techdocs/api, April 2003.

[Ra] J. Rabitsch: *Einführung in das Qualitätsmanagement*. http://members.aon.at/joana.rabitsch/einfach/qual_einl.htm#27, April 2003.

[Sy] Online-News Flexible Unternehmen. *Lean Management effektiver als Mitarbeiterbeteiligung*. http://www.flexible-unternehmen.de/news/02-11-21-04.htm, April 2003.