

Diplomarbeit Nr. 1834

# Integration der OMG Workflow Management Facility in das CORBA-basierte Objektmodell von ASCEND

Andreas Kuhn

Universität Stuttgart  
Institut für Parallele und  
Verteilte Höchstleistungsrechner  
Abteilung Anwendersoftware  
Breitwiesenstrasse 20-22  
70565 Stuttgart

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Aufgabenstellung . . . . .	2
1.3	Aufbau dieser Arbeit . . . . .	2
<b>2</b>	<b>Computer Supported Cooperative Work</b>	<b>3</b>
2.1	Groupware . . . . .	3
2.2	Workflow-Management . . . . .	5
2.3	Vergleich von Workflow-Management- und Groupware-Systemen	6
2.4	Designflow-Management . . . . .	7
<b>3</b>	<b>ASCEND und CASSY</b>	<b>9</b>
3.1	ASCEND . . . . .	9
3.1.1	Das Aktivitätenmodell von ASCEND . . . . .	11
3.1.2	Beziehungstypen der Designflow-Activity . . . . .	12
3.1.3	Designflow-Beispiel . . . . .	13
3.2	CASSY . . . . .	14
3.2.1	Beschreibung der Komponenten und des aktuellen Entwicklungsstandes . . . . .	15
<b>4</b>	<b>Die Workflow Management Facility</b>	<b>19</b>
4.1	Das Workflow-Referenzmodell der WfMC . . . . .	19
4.2	Request for Proposals für eine Workflow Management Facility . .	22
4.2.1	jFlow - Joint RfP Submission . . . . .	23
4.3	Das jFlow-Objektmodell . . . . .	24
4.3.1	ExecutionElement . . . . .	25
4.3.2	WorkflowManager . . . . .	25
4.3.3	ProcessDefinition . . . . .	26
4.3.4	ProcessInstance . . . . .	26
4.3.5	ActivityInstance . . . . .	26
4.3.6	ProcessData . . . . .	27

4.3.7	WorkItem . . . . .	27
4.3.8	Observer . . . . .	28
4.3.9	Participant . . . . .	28
4.3.10	HistoryIterator . . . . .	28
4.3.11	ElementIterator . . . . .	29
4.3.12	DefinitionIterator . . . . .	29
4.4	Implementierungsaspekte der Workflow Management Facility . .	29
<b>5</b>	<b>CORBA</b>	<b>33</b>
5.1	Die Object Management Group . . . . .	35
5.1.1	Das Objektmodell der OMG . . . . .	35
5.2	Die Object Management Architecture . . . . .	36
5.3	CORBA-Spezifikation . . . . .	39
5.3.1	Die Interface Definition Language . . . . .	40
5.3.2	Der Object Request Broker . . . . .	41
5.3.3	Interoperabilität . . . . .	45
5.4	Erstellung einer CORBA-Applikation mit ORBacus . . . . .	47
<b>6</b>	<b>Realisierung des Wrappers</b>	<b>51</b>
6.1	Integration in das Aktivitätenmodell . . . . .	52
6.1.1	Beschreibung der WorkflowActivity-Klasse . . . . .	53
6.1.2	Die Workflow Activity Object Factory . . . . .	55
6.2	Implementierung . . . . .	56
6.3	Probleme bei der Implementierung . . . . .	58
6.3.1	Vorschläge zur Verbesserung der Softwarequalität . . . . .	58
<b>7</b>	<b>Zusammenfassung, Bewertung und Ausblick</b>	<b>61</b>
7.1	Zusammenfassung . . . . .	61
7.2	Bewertung . . . . .	62
7.3	Ausblick . . . . .	62
<b>A</b>	<b>ActivityInterface.idl</b>	<b>65</b>
	<b>Literaturverzeichnis</b>	<b>73</b>

# Abbildungsverzeichnis

2.1	Klassifikation von Groupware nach [Bur97] . . . . .	4
3.1	Beispiel eines Designflows . . . . .	14
3.2	Die Implementierungsstruktur des CASSY-Systems . . . . .	15
4.1	Das Workflow-Referenzmodell der WfMC . . . . .	21
4.2	Das jFlow-Objektmodell . . . . .	24
4.3	Zustandsübergangendiagramm von ExecutionElement . . . . .	25
4.4	Zustandsübergangendiagramm von ProcessInstance . . . . .	26
4.5	Zustandsübergangendiagramm von ActivityInstance . . . . .	27
4.6	Zustandsübergangendiagramm von WorkItem . . . . .	28
5.1	Object Management Architecture . . . . .	36
5.2	Client und Server kommunizieren über den ORB . . . . .	37
5.3	Basis- und Applikationsdienste der OMA . . . . .	39
5.4	Die Struktur des Object Request Brokers . . . . .	41
5.5	Zugriff auf die Objektimplementierung über DII und IDL-Stub . . . . .	42
5.6	Aufruf der Objektimplementierung über DSI und IDL-Skeleton . . . . .	43
5.7	Internet Inter-ORB Protocol . . . . .	46
5.8	Erstellen einer CORBA-Applikation mit ORBacus . . . . .	48
6.1	Integration des Wrappers in die Systemstruktur von CASSY . . . . .	52
6.2	Klassendiagramm der ActivityManagement-Komponente . . . . .	53
6.3	Instanziierung einer Workflow-Aktivität . . . . .	55
6.4	Dateistruktur des Aktivitätenmanagements . . . . .	57
6.5	Dateien der Workflow Management Facility . . . . .	59

# Tabellenverzeichnis

2.1	Klassifikation von Groupware-Systemen nach Raum und Zeit . . .	4
2.2	Vergleich von Workflow-Management- und Groupware-Systemen	7
3.1	Beschreibung des Aktivitätenmodells . . . . .	11

# Abkürzungsverzeichnis

API	Application Programming Interface
ASCEND	Activity Support in Cooperative Environments for Design Issues
CAD	Computer Aided Design
CASE	Computer Aided Software Engineering
CASSY	Cooperative Activity Support System
CAX	Computer Aided X (X steht hier als Variable für ein Arbeitsgebiet)
CORBA	Common Object Request Broker Architecture
COM	Component Object Model
COS	Common Object Services
CSCW	Computer Supported Cooperative Work
DB	Database
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
EBNF	Extended Bacchus-Naur-Form
ESIOP	Environment Specific Inter-ORB Protocol
GIOP	General Inter-ORB Protocol
IBM	International Business Machines

IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
IT	Information Technology
NIIP	National Industrial Information Infrastructure Protocols
OA	Object Adapter
OMA	Object Management Architecture
OMG	Object Management Group
OO	Object Oriented
OOC	Object Oriented Concepts
OODBA	Object Oriented Database Adapter
ORB	Object Request Broker
RfP	Request for Proposals
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
WAPI	Workflow Application Programming Interfaces
WfMC	Workflow Management Coalition
WMF	Workflow Management Facility
WMS	Workflow Management System



# Kapitel 1

## Einleitung

Diese Diplomarbeit entstand im Rahmen des Forschungsprojektes *ASCEND* an der Universität Stuttgart. Sie beschäftigt sich mit der Entwicklung und Implementierung eines *Wrappers* für eine *Workflow Management Facility*. Die Funktionalität dieser Facility soll durch den Wrapper in geeigneter Weise in das bestehende Aktivitätenmodell der ASCEND-Implementierungsstudie *CASSY* eingebunden werden. Da diese Diplomarbeit auf zahlreichen vorausgegangenen Arbeiten aufbaut, müssen insbesondere auch bereits bestehende Implementierungen und deren Programm-Schnittstellen beachtet werden.

### 1.1 Motivation

Im dem Projekt ASCEND (**A**ctivity **S**upport in **C**ooperative **E**Nvironments for **D**esign **I**ssues) werden die notwendigen Konzepte für kooperative Entwurfsumgebungen (CAD, CASE, ...) untersucht. Weil diese teilweise automatisiert werden können, bietet es sich an, Techniken aus dem Bereich Workflow-Management zu untersuchen. Da Entwurfsprozesse im Gegensatz zu herkömmlichen Geschäftsprozessen eine größere Flexibilität erlauben müssen, scheinen Techniken aus dem CSCW-Bereich Lösungen zu bieten. Im Rahmen der Arbeiten des Projektes wurden eine Workflow Management Facility, eine CSCW Facility, sowie ein Kollaborations-Server entwickelt. Diese Komponenten bieten ihre Dienste über CORBA an. Diese Komponenten werden mittels eines gemeinsamen Aktivitätenmodells zusammengeführt. Durch die Verwendung von CORBA werden die Integrationsmöglichkeiten erweitert, d.h. die Implementierung kann unter Windows NT oder Unix, in C++ oder Java erfolgen. Letztendlich soll *CASSY* (**C**ooperative **A**ctivity **S**upport **S**ystem) eine effiziente Bedienoberfläche zur Verfügung stellen, die das Aktivitätenmodell einem Benutzer leicht zugänglich macht.

## 1.2 Aufgabenstellung

Die bereits existierende Workflow Management Facility ist in das Objektmodell von ASCEND unter Berücksichtigung der existierenden CSCW-Facility-Integration einzubringen. Somit soll die Grundlage für ein umfassendes Objektmodell für das Aktivitätenmanagement in ASCEND entwickelt werden.

## 1.3 Aufbau dieser Arbeit

*Kapitel 2 bis Kapitel 5* dienen der Einführung in das thematische Umfeld dieser Diplomarbeit. Sie leisten die Vorarbeit, um dem Leser die Zusammenhänge in dem schon relativ weit fortgeschrittenen Projekt ASCEND zu verdeutlichen und um das Grundwissen zum Verständnis der Ergebnisse dieser Arbeit zu vermitteln. *Kapitel 6, Kapitel 7* und der *Anhang* zeigen die gefundenen Lösungen und deren Integration in das bestehende System.

**Kapitel 2** bietet eine Einführung in die Begriffswelt und vergleicht die Technologien Workflow-Management und Groupware.

**Kapitel 3** zeigt die Vorteile von ASCEND und beschreibt das Aktivitätenmodell. Anschließend wird CASSY vorgestellt, ein System in dem die Konzepte des ASCEND-Modells implementiert werden.

**Kapitel 4** beginnt auf der Metaebene mit dem Workflow-Referenzmodell der WfMC und beschreibt die Schnittstellen der joint RfP Submission (jFlow). Anschließend wird die konkrete Implementierung einer Workflow Management Facility, welche im Rahmen der Diplomarbeit von Günter Heiß realisiert wurde, betrachtet.

**Kapitel 5** gibt einen kurzen Überblick über die Begriffswelt der OMG, beschreibt die Common Object Request Broker Architecture (CORBA) und den vorteilhaften Einsatz dieser Middleware für CASSY.

In **Kapitel 6** wird die Integration der Workflow Management Facility in das CASSY Framework dargestellt. Vorgehensweise, Lösungswege und Lösungen aber auch aufgetretene Probleme und Einschränkungen des Systems werden detailliert beschrieben.

**Kapitel 7** gibt einen Überblick über die erbrachten Leistungen dieser Diplomarbeit. Die gefundenen Lösungen werden zusammenfassend beschrieben und bewertet. Der aktuelle Entwicklungsstand von CASSY wird dargestellt, sowie ein Ausblick auf noch anstehende Aufgabenbereiche gegeben.

# Kapitel 2

## Computer Supported Cooperative Work

Im Forschungsbereich CSCW (**C**omputer **S**upported **C**ooperative **W**ork) wird untersucht, wie sich die Kooperation zwischen Menschen mit Hilfe von Computern effizienzsteigernd unterstützen lässt. Eine zentrale Rolle kommt dabei dem Begriff der *Gruppe* zu. Eine Gruppe bzw. ein Team ist im CSCW-Kontext eine Menge von Menschen, deren Mitglieder über die Zusammensetzung der Gruppe Bescheid wissen und die Lösung einer gemeinsamen Aufgabe zum Ziel haben. Um dieses Ziel zu erreichen ist Kooperation notwendig. Die technischen Hilfsmittel, die zur Kooperationsunterstützung eingesetzt werden bezeichnet man als *Groupware*. Groupware umfasst sowohl Hardware- als auch Softwarekomponenten. Im Rahmen dieser Arbeit wird jedoch nur auf die Softwarekomponenten eingegangen.

### 2.1 Groupware

Groupware-Systeme bieten Funktionen zum Zweck der Kooperationsunterstützung über zeitliche und räumliche Distanzen hinweg, wie sie bei der Zusammenarbeit von Mitgliedern eines Teams notwendig sind. Kooperation schließt Konkurrenz nicht aus, deshalb sind Koordinationsmechanismen notwendig. Koordination setzt Kommunikationsfunktionen voraus; die Kommunikationsvorgänge selbst müssen allerdings auch koordiniert werden. Groupware-Systeme bieten deshalb auch Unterstützung für Kommunikations- und Koordinationsvorgänge an.

Groupware-Systeme werden meistens für spezifische Einsatzgebiete oder für einen speziellen Benutzerkreis geschaffen. Beispiele für Groupware-Systeme sind Verhandlungssysteme, Konferenzsysteme, Systeme zur Unterstützung von Gruppenentscheidungen, Terminvereinbarungssysteme, Mehrbenutzereditoren,

E-Mail, etc.

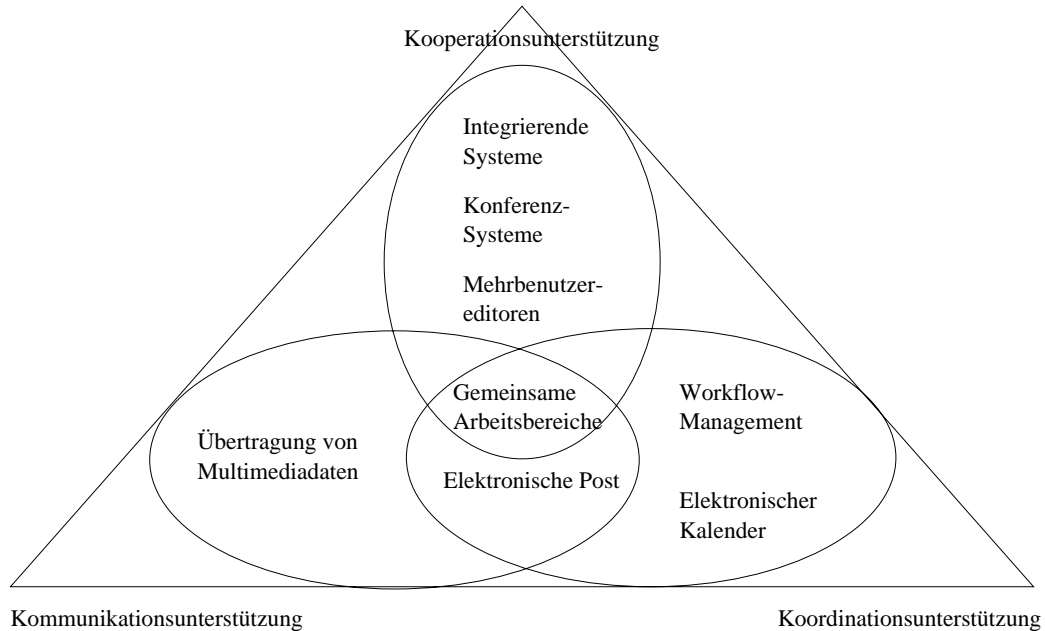


Abbildung 2.1: Klassifikation von Groupware nach [Bur97]

In [Bur97] werden Groupware-Systeme bezüglich ihrer Unterstützung für Kommunikation, Koordination und Kooperation klassifiziert. Abbildung 2.1 zeigt diese Aufteilung mit einigen Beispielen. E-Mail und Videokonferenzsysteme konzentrieren sich zum Beispiel mehr auf die Kommunikationsunterstützung, während bei Terminvereinbarungssystemen der Fokus auf der Koordinationsunterstützung liegt. Aufgrund ihrer Funktionen zur Ablaufsteuerung kann man Workflow-Management-Systeme in die Kategorie der Koordinationsunterstützenden Systeme einordnen.

Anwesenheit der Teilnehmer	zur gleichen Zeit (synchron)	zu unterschiedlichen Zeiten (asynchron)
am gleichen Ort (zentral)	direktes Gespräch	Schwarzes Brett
an unterschiedlichen Orten (verteilt)	Video-Konferenz	E-Mail

Tabelle 2.1: Klassifikation von Groupware-Systemen nach Raum und Zeit

Tabelle 2.1 zeigt eine weitere Klassifikation, die sehr häufig verwendet wird mit jeweils einem Beispiel. Diese Klassifikation unterscheidet Groupware-

Systeme nach ihrer Fähigkeit, Kooperation über räumliche und zeitliche Distanzen hinweg zu unterstützen. Auch bei dieser Taxonomie lassen sich Workflow-Management-Systeme wieder als Spezialisierung der Groupware-Systeme betrachten. In diesem Fall passen sie in die Kategorie der asynchronen, verteilten Systeme. D. h. sie bieten Kooperationsunterstützung für Benutzer an unterschiedlichen Orten zu unterschiedlichen Zeiten. Oft bieten Groupware-Systeme eine ganze Vielzahl von Funktionen, so dass sie sich nicht eindeutig nur einer Kategorie zuordnen lassen.

## 2.2 Workflow-Management

Menschliche und damit auch kooperative Arbeit läuft häufig nach bestimmten Regeln ab, deren Abläufe sich oft wiederholen. Je stärker diese Abläufe strukturiert sind, desto besser lassen sich die Abfolgen schon im Voraus bestimmen und desto besser lassen sie sich deshalb auch automatisieren. Workflow-Management-Systeme (WMS) nutzen das Wissen über die Struktur der Arbeitsabläufe aus. Sind alle, für den Arbeitsablauf notwendigen Informationen im Voraus bekannt: die Bearbeitungsschritte, deren Abhängigkeiten untereinander und etwaige Alternativen im Ablauf, zu benutzende Werkzeuge und die aufzurufenden Applikationen, kann das WMS dieses Wissen in die Koordination von Arbeitsschritten umsetzen.

Die Automatisierung eines Geschäftsvorganges wird als *Workflow* bezeichnet. Der Begriff Workflow ist in [JBS99] folgendermaßen definiert:

Ein **Workflow** ist eine zum Teil automatisch (algorithmisch) - von einem Workflow-Management-System gesteuert - ablaufende Gesamtheit von Aktivitäten, die sich auf Teile eines Geschäftsprozesses oder andere organisationelle Vorgänge beziehen. Ein Workflow besteht aus Abschnitten (Subworkflows), die weiter zerlegt werden können. Er hat einen definierten Anfang und ein definiertes Ende. Ein Workflow-Management-System steuert die Ausführung eines Workflows. Workflows sind überwiegend als ergonomische (mit Menschen als Aufgabenträgern) und nicht als technische (z. B. Einsatz von Maschinen) Prozesse zu sehen.

Das Workflow-Management-System ist für die Definition, Verwaltung und Ausführung von Workflows zuständig. Für die Beschreibung von Workflows bietet es eine Sprache an (engl. Workflow Definition Language), die entweder grafisch z. B. in Form von erweiterten Petri-Netzen oder textuell sein kann. Für Verwaltung, Wartung und Kontrolle der Workflows stellt es spezielle administrative Werkzeuge (engl. *Tools*) zur Verfügung. Das WMS bietet eine aktive Verwaltung der Workflows, sorgt dabei für die Koordination aller beteiligten Personen und Ressourcen gemäß den Zielvorgaben und stößt durchführbare Routineaufgaben automatisch an. Abschnitt 4.1 zeigt weitere Eigenschaften von WMS.

Beispiele kommerziell verfügbarer Workflow-Management-Systeme sind: Lotus Notes (IBM), Flowmark und dessen Nachfolger MQSeries (IBM) und Work-Party (SNI AG).

## 2.3 Vergleich von Workflow-Management- und Groupware-Systemen

Obwohl Workflow-Management genauso wie Groupware dem Forschungsbereich CSCW zuzuordnen ist und beide Teilbereiche bzw. die implementierten Systeme entsprechende Gemeinsamkeiten aufzuweisen haben: beide dienen der Kooperationsunterstützung, Workflow-Management-Systeme wie auch Groupware-Systeme sind verteilte Systeme, weil sie häufig räumliche Distanzen überbrücken müssen, etc., so unterscheiden sie sich doch wesentlich in ihren Anwendungsgebieten und hinsichtlich der Art und dem Zweck der Koordinationsunterstützung. [JBS99] (S. 376 ff.) vergleicht beide Systemarten nach folgenden Kriterien:

### 1. Unterscheidung zwischen Koordination auf Objektebene und Koordination auf Aktivitätsebene

Von Koordination auf Objektebene spricht man, wenn für jeden Benutzer für jedes Objekt genau festgelegt wird, welche Operationen er darauf ausführen darf und welche nicht. Objektkoordination ist vorteilhaft, wenn der strukturelle Aufbau der Objekte gut beschreibbar ist und die Tätigkeiten, mit denen die Objekte bearbeitet werden, schlecht vorstrukturierbar sind.

Für die Ausführung von Koordinationsvorgängen auf Aktivitätsebene sind genaue Kriterien definiert. Sind die Aktivitäten (Bearbeitungsschritte oder Subworkflows) in ihrem Ablauf diskret und auch zeitlich vorhersehbar, lassen sich die Abfolgen dieser Aktivitäten in Schemata angeben.

### 2. Existenz eines Schemakonzepts

Groupware-Systeme sind daraufhin optimiert, informelle, oft nicht vernünftig strukturierbare Kommunikationsprozesse möglichst effizient zu unterstützen. Die Prozesssicht der Koordinationsunterstützung wird deshalb häufig nicht modelliert. Viele Groupware-Systeme verfügen aus diesem Grund über kein Schemakonzept und falls doch, ist dieses oft stark verschieden von den Schema-Konzepten der WMS. Es entfällt die Trennung zwischen Entwicklungs-, Spezifikations- und Ablaufumgebung. Damit entfällt aber auch die Möglichkeit, Koordinationsaufgaben vorzuplanen und diese vom System automatisch ausführen zu lassen. Benutzer von Groupware koordinieren ihre Aktivitäten selbst. WMS werden primär zur Automation gut vorstrukturierbarer und damit formalisierbarer Routineprozesse eingesetzt.

### 3. Synchroner oder asynchroner Arbeitsweise der Benutzer

Die Koordinationsunterstützung durch WMS konzentriert sich auf asynchrone Kooperationsituationen, während die Koordinationsunterstützung durch Groupware eher für synchrone Kooperationsituationen ausgelegt ist. Viele reale Kooperationen lassen sich allerdings nicht auf synchrone oder asynchrone Kommunikation reduzieren, weil reale Kooperationen meistens beide Systemtypen benötigen.

Die Unterschiede zwischen Workflow-Management und Groupware liegen vor allem in der Art und Weise der Unterstützung von kooperativer Zusammenarbeit begründet. Dies kommt insbesondere in der Verteilung der aktiven und passiven Rollen zwischen System und Benutzer zum Vorschein [KKS<sup>+</sup>98]. Tabelle 2.2 fasst die Ergebnisse nochmals zusammen:

Workflow-Management	Groupware
Koordination auf Aktivitätsebene	Koordination auf Objektebene
Steuerung und Kontrolle obliegt größtenteils dem System	System stellt nur passive Infrastruktur zur Verfügung
Benutzer hat eher passive Rolle bezüglich der Steuerung	Die Kontrolle obliegt dem Benutzer
System hat eine aktive Rolle: steuert Abläufe, startet Aktionen, reicht Dokumente weiter	Interaktion des einzelnen mit der Gruppe steht im Vordergrund
Dient der Unterstützung aufgabenbezogener Koordination und Kommunikation	Gut geeignet zur Unterstützung gering strukturierter Vorgänge
Einsatz bei organisatorisch geregelten, gut strukturierten Abläufen, z. B. Büroprozesse	Einsatz eher bei informellen Aufgaben und kleinen Gruppen mit Selbstorganisation
Benutzer arbeiten vorwiegend asynchron	Benutzer arbeiten vorwiegend synchron

Tabelle 2.2: Vergleich von Workflow-Management- und Groupware-Systemen

## 2.4 Designflow-Management

Sowohl Groupware- als auch Workflow-Management-Systeme können nur Teile eines Entwurfsablaufs effizient unterstützen weil die Konzepte grundlegende Unterschiede aufweisen und nicht für die Unterstützung vollständiger Entwurfsabläufe entwickelt wurden. ASCEND will genau dieses Dilemma lösen, indem es beide Technologien unter dem Dach eines allgemeinen Aktivitätenmodells vereinigt. Das Konzept dafür nennt sich Designflow-Management. Es soll die Verwaltung von Entwurfsabläufen (engl. Designflows) in ihrer Gesamt-

heit ermöglichen, indem es die flexible Integration automatisierbarer Workflow-Aktivitäten und kooperativer Groupware-Aktivitäten unterstützt (vgl. [FM99]).

# Kapitel 3

## ASCEND und CASSY

Ein typischer Entwurfsablauf lässt sich in miteinander interagierende Teilabläufe aufspalten. Kooperation kann bei integrierten, computerunterstützten Entwicklungsumgebungen dazu beitragen, sowohl die Effizienz, als auch die Qualität des Designs zu verbessern. Dies kann durch bessere Koordination der Entwurfsschritte im Entwicklungsprozess und durch frühzeitigen Austausch der Arbeitsleistungen zwischen den Teilnehmern erreicht werden. Zur Unterstützung solcher Aktivitäten bietet es sich an, Workflow-Management- und Groupware-Technologien einzusetzen.

### 3.1 ASCEND

ASCEND (Activity Support in Cooperative ENvironments for Design Issues) ist ein Projekt, das an der Technischen Universität München entstand und jetzt an der Universität Stuttgart weiterentwickelt wird. ASCEND zielt darauf ab, Workflow-Management- und Groupware-Aspekte zu integrieren, um flexible Koordination und Kooperation in Entwurfsumgebungen zu schaffen. Integrität kann nicht automatisch kontrolliert werden, wenn die Benutzer des Systems nach wie vor auf systemexterne Kommunikationsmethoden wie Telefon oder E-Mail zurückgreifen, die sich den Kontrollmechanismen entziehen. Die manuelle Entwurfsüberwachung ist fehleranfällig. Deshalb ist es wichtig, den gesamten Entwurfsablauf integrieren zu können. Der Ansatz von ASCEND beruht dabei in besonderem Maße auf Aktivitäten.

Das Ziel von ASCEND ist es, geeignete generische Modellierungs- und Realisierungskonzepte für einen Aktivitäten-Management-Service zu erarbeiten und zu erproben. Der Aktivitäten-Management-Service vereinigt dabei die Menge aller Basisprimitive des Aktivitäten-Managements in einer Weise, vergleichbar der Integration von Basisdiensten zur Entwurfsdatenverwaltung eines CAD Frame-

work.

Die Integrationsarchitektur von ASCEND erfüllt dabei drei wesentliche Aufgaben [Fra99a]. Sie unterstützt die *Kooperation* der an einer gemeinsamen Aufgabe bzw. Projekt beteiligten Personen und Gruppen, die *Koordination* von (Entwurfs-)Abläufen und die *Interaktion* bzw. die *Kommunikation* der eingesetzten Werkzeuge mit der Entwurfsdatenverwaltung.

Charakteristische Aufgaben einer Entwurfsumgebung lassen sich unterschiedlichen Aktivitätsebenen für Kooperation, Planung und Werkzeuge zuordnen. Durch die Unterscheidung dieser Ebenen wird eine Abbildung auf die typischen Strukturen einer Entwurfsumgebung, wie sie zum Beispiel im CAX-Bereich oder der Softwareentwicklung üblich ist, ermöglicht. Der oft äußerst komplexe Gesamtprozess wird dabei in eine Menge miteinander interagierender Teilabläufe zerlegt.

Das Grundprinzip, das hinter dieser Vorgehensweise steckt, ist aus vielfältigen Anwendungen nicht nur in der Informatik bekannt: *Divide & Conquer* (dt. Teile und Herrsche). Objekte, in diesem Fall Geschäftsabläufe, die wegen ihrer Größe oder Komplexität nicht oder nur schwer beherrschbar sind, werden in kleinere Teile zerlegt, die für sich genommen überschaubarer, verständlicher und damit auch leichter zu kontrollieren sind. Für zusätzliche Quellen, sowie eine kurze Beschreibung des Projekts vgl. [Fra99b].

Wie aus [FM99] hervorgeht, gibt es bisher kaum Möglichkeiten, Arbeitsabläufe in ihrem organisatorischen Umfeld entsprechend darzustellen. Es gibt zwar Bestrebungen, in Workflow-Management-Systemen flexible und dynamische Aktivitäten zu unterstützen, genauso wie es bei Groupware-Systemen Versuche gibt, exakt vordefinierte und wiederholbare Aktivitäten zu integrieren, jedoch sind diese Technologien immer noch zu sehr den Restriktionen ihres jeweiligen Anwendungsbereichs unterworfen und eignen sich entweder mehr für das eine oder das andere Anwendungsgebiet. Abschnitt 2.3 hat allerdings auch gezeigt, dass sich Workflow-Management- und Groupware-Technologien gegenseitig sehr gut ergänzen können.

#### **Daraus ergeben sich die Vorteile des ASCEND-Modells:**

- Getrennt gekapselte Funktionalität wird zusammengefügt und durch ein einzelnes, integriertes Aktivitätenmodell verwaltet.
- Als Konsequenz daraus können existierende Dienste und Facilities wie z. B. Datenbankdienste, Notifikationsmechanismen, Benutzerverwaltung, Administrationsdienste, etc. wiederverwendet und gemeinsam genutzt werden.
- Bereits existierende Workflow-Management- und Groupware-Anwendungen können in das erweiterbare, offene Framework eingebunden werden.

- Eine Integration beider Systeme auf Basis gemeinsam genutzter Komponenten erhöht die Administrierbarkeit des Gesamtsystems im Vergleich zu den einzelnen, voneinander isolierten Systemen.
- Durch eine nahtlose Integration beider Technologien erhöht sich auch die Akzeptanz seitens der Benutzer, da hier eine einheitliche, ergonomische Benutzerschnittstelle gewählt werden kann.

### 3.1.1 Das Aktivitätenmodell von ASCEND

Durch einen Designflow können komplexe Entwurfsaktivitäten dargestellt werden. Dieser kann CSCW-Aktivitäten enthalten, die von Menschen dynamisch koordiniert werden müssen und Workflow-Aktivitäten, die automatisiert werden können. Designflows lassen sich in einzelne, elementare Beschreibungselemente, sogenannte *Design Items* zerlegen. Die Zuordnung von Design Item zu einem *Actor* (dt. Ausführer) muss eindeutig sein. Actor kann eine einzelne Person, ein Team oder auch ein Computerprogramm sein. Tabelle 3.1 zeigt die Basis-komponenten des ASCEND-Aktivitätenmodells in einer EBNF-ähnlichen Notation [FM99].

primitive-activity	::=	design-step   groupware-primitive
basic-activity	::=	workflow-activity   groupware-activity   primitive-activity
workflow-activity	::=	workflow-definition ( {basic-activity } )
groupware-activity	::=	groupware-primitive
workflow-definition	::=	workflow-well-formed-formula ( {workflow-primitive } )
workflow-primitive	::=	sequention-execution   and   or   ...
groupware-primitive	::=	tele-conference   shared editing   BBS   interview   scheduler   ...
designflow-activity	::=	designflow-definition ( {basic-activity } )
designflow-definition	::=	negotiation   usage   delegation   constraints

Tabelle 3.1: Beschreibung des Aktivitätenmodells

**Primitive-Activity:** Ein einzelner Entwurfsschritt oder ein *Groupware-Primitive*. Beides kann nicht weiter untergliedert werden. Da die Primitive-Activity in allen anderen Aktivitäten enthalten sein kann, können diese auch als Spezialisierungen der Primitive-Activity betrachtet werden. Jeder Primitive-Activity ist ein Actor zugeordnet, der für die Activity verantwortlich ist.

**Basic-Activity:** Bietet für alle Aktivitäten die gleiche minimale Menge an Funktionalität. Da alle Arten von Aktivitäten gleichzeitig auch Basic-

Activities sind, wird auf diese Weise der rekursive Aufbau von Designflows ermöglicht.

**Workflow-Activity:** Dient der Kapselung von Worklow-Abläufen. Ein Worklow-Ablauf besteht aus mindestens zwei, durch Workflow-Primitive miteinander verknüpften Aktivitäten. Die Aktivitäten sorgen für die korrekte Ausführung des Workflows.

**Groupware-Activity:** Kann nicht rekursiv definiert werden und ist im allgemeinen weniger strukturiert bzw. weniger strukturierbar als Workflow-Activities. Sie steht in direktem Zusammenhang mit einem einzelnen, kooperativen Vorgang, z. B. Telekonferenz, Interview, Verteilungsdienst, etc.

**Designflow-Activity:** Eine Designflow-Activity repräsentiert oftmals eine sehr komplexe Entwurfsaufgabe. Sie besteht für gewöhnlich aus Aktivitäten mit unterschiedlichen Typen und kann auch noch weitere, gekapselte Designflows enthalten. Zur Modellierung der Funktionalität stehen verschiedene *Beziehungstypen* zur Verfügung. Die Designflow-Activity kann als eine Spezialisierung kooperativer Aktivitäten betrachtet werden, welche Entwurfsspezifische Funktionalität bietet.

### 3.1.2 Beziehungstypen der Designflow-Activity

Designflow-Activities erlauben die Beschreibung verschiedener Funktionalitäten.

**Delegation:** Durch die *Delegation*-Beziehung wird eine Auftraggeber-/Auftragnehmer-Beziehung dargestellt. Dies gestattet die Dekomposition des Entwurfsablaufs. Die *delegierende* Aktivität ist für die Integration der Ergebnisse der *delegierten* Aktivität verantwortlich.

**Usage:** Aktivitäten, welche über eine *Usage*-Beziehung verbunden sind, können vorzeitig freigegebene Versionen von Design-Items untereinander austauschen. Dazu kann eine Aktivität ein Design-Item bei einer anderen Aktivität registrieren lassen, um dann im Falle einer Änderung des gewünschten Datums benachrichtigt zu werden.

**Negotiation:** Die *Negotiation*-Beziehung hat sehr allgemeingültige Primitive, welche die Modellierung nahezu jeder Interaktion zur Abstimmung von Entwurfsentscheidungen ermöglichen. Folglich können daraus fast alle Kommunikationsprimitive aufgebaut werden. Typische Beispiele für Primitive des Abstimmungsprotokolls sind `Propose`, `Accept` und `Reject` zum Vorschlagen, Annehmen und Abbrechen einer Auftragspezifikation. Diese Primitive können zwischen verschiedenen Aktivitäten eingesetzt

werden, um (Entwurfs-)Entscheidungen zu fällen, beispielsweise zur Konfliktlösung bei konkurrierendem Ressourcenzugriff.

**Design Constraints:** Durch *Design Constraints* können Restriktionen bezüglich der Ablaufreihenfolge von Aktivitäten ausgedrückt werden. Sie bieten Koordinationsmöglichkeiten, die Workflow-Konstrukten ähnlich sind, jedoch nicht genauso strikt sind wie diese. Typische Restriktionen sind: *after*, *directly after*, *implies*, *enables*, *limits*, *must appear*, *not*.

### 3.1.3 Designflow-Beispiel

Mit Hilfe des Aktivitätenmodells von ASCEND soll der gesamte Entwicklungsprozess unterstützt werden können. Klassisches Beispiel für einen solchen Entwicklungsprozess ist z. B. der Entwicklungsprozess eines Softwaresystems. Dieser besteht aus mehreren Phasen, welche nach einem wohldefinierten Schema angeordnet sind. Das *Wasserfall-Modell* [Som98] ist eine Möglichkeit, den Prozessverlauf darzustellen. Es unterteilt den Lebenslauf eines Softwaresystems in die Phasen „requirements definition“, „system and software design“, „implementation and unit testing“, „integration and system testing“ und „operation and maintenance“, wobei die letzte Phase nicht mehr dem Entwicklungsprozess zuzuordnen ist. Jede der Phasen wird mit einer Verifikation des erstellten Produkts abgeschlossen. Abhängig von der Phase, in welchem sich der Entwicklungsprozess befindet, kann dies formal (z. B. über Beweistechniken) oder informell (z. B. in Diskussionen) geschehen. Während sich für die Steuerung des gesamten Entwicklungsprozesses ein Workflow-Management-System anbietet, können Diskussionen mit Unterstützung von Groupware-Systemen durchgeführt werden. Abhängig vom Diskussionsergebnis wird dann im WMS die nächste Phase begonnen, oder in der aktuellen Phase verblieben. Die Vorgänge erfordern hierbei den Zugriff auf dieselbe Datenbasis und werden potentiell von derselben Person durchgeführt.

Wie dieser Entwicklungsprozess durch das Designflow-Modell dargestellt werden kann, wurde in der Diplomarbeit von Marcello Mariucci bereits ausführlich untersucht [Mar98]. Deshalb sei hier nur ein schemenhaftes Beispiel vorgestellt, um zu zeigen, wie ein Entwicklungsprozess prinzipiell durch ein *Gantt-Diagramm* (siehe Abbildung 3.1) dargestellt werden kann. Wichtig ist die Feststellung, dass das Beispiel sowohl unstrukturierte Groupware-Aktivitäten enthält, als auch strukturierte Workflow-Aktivitäten und Designflow-Aktivitäten, die noch weiter unterteilt werden können.

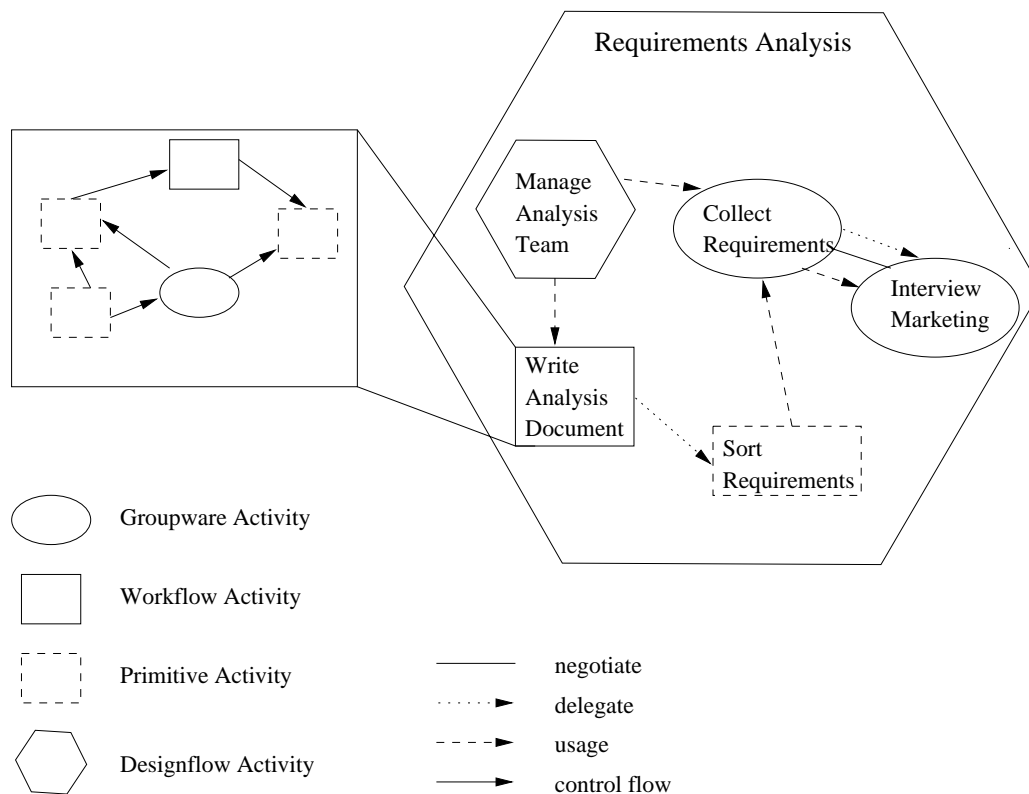


Abbildung 3.1: Beispiel eines Designflows

### 3.2 CASSY

CASSY (Cooperate Activity Support SYstem) implementiert die Konzepte des ASCEND-Modells zur Unterstützung verteilter Entwurfsumgebungen und soll unter Beweis stellen, dass die Konzepte, die im Projekt ASCEND erarbeitet wurden, effizient realisiert werden können. CASSY wurde so konzipiert, dass durch seinen CORBA-basierten und dadurch auch komponentenbasierten Ansatz ein Höchstmaß an Erweiterbarkeit, Plattformunabhängigkeit und Wiederverwendbarkeit erreicht wird. Das Ziel ist die Bereitstellung einer allgemeinen Entwurfsumgebung, zur Unterstützung sowohl formalisierbarer, als auch nicht formalisierbarer Abläufe.

Der Entwurf stützt sich in besonderem Maße auf die Dienste des CORBA-Systems, respektive auf *CORBA Services* [OMG98], um die Wiederverwendbarkeit existierender Groupware- und Workflow-Tools zu ermöglichen und um zahlreiche Software-, Hardware-, und Internetumgebungen unterstützen zu können. CORBA ist eine von der OMG standardisierte *Middleware*, die eine platt-

formübergreifende Kommunikation räumlich verteilter Objekte ermöglicht (siehe Kapitel 5). Definition von Middleware nach [JBS99]:

**Middleware** ist Anwendungsneutrale Software, die zwischen Anwendungsprogrammen und Betriebssystem, Datenbanksystemen und anderen Ressourcenmanagern läuft. Sie stellt Infrastrukturdienste zur Verfügung, auf deren Basis Applikationen ortstransparent entwickelt und betrieben werden können. Solche Dienste sind unter anderem Kommunikations-, Datenmanagement-, Prozessmanagement-, Transaktionsmanagement- sowie Netzwerk- und Systemmanagement-Dienste.

CORBA bildet als Middleware eine Kommunikationsplattform für die CASSY-Komponenten. Aufgrund der Vorteile des programmiersprachen-, betriebssystem- und hardwareunabhängigen CORBA-Systems kann CASSY den Anforderungen an Erweiterbarkeit, Plattformunabhängigkeit und Wiederverwendbarkeit gerecht werden.

### 3.2.1 Beschreibung der Komponenten und des aktuellen Entwicklungsstandes

Zum gegenwärtigen Zeitpunkt befindet sich CASSY noch in der Entwicklung. Abbildung 3.2 zeigt den modularen Aufbau von CASSY, bestehend aus fünf Komponenten und der Middleware CORBA. Die Systemarchitektur von CASSY kombiniert Workflow-Management-eigene, Groupware-eigene, als auch gemeinsam genutzte Komponenten.

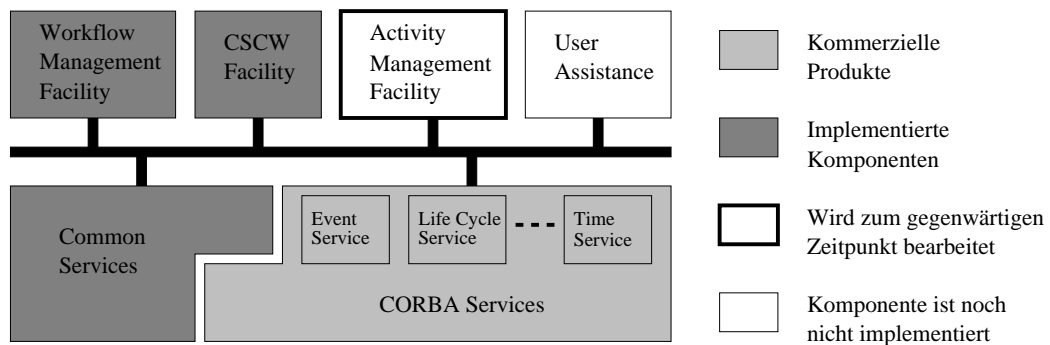


Abbildung 3.2: Die Implementierungsstruktur des CASSY-Systems

**CORBA Services:** CORBA Services stehen generell zur Verfügung. Sie werden benötigt, damit CASSY auch auf verteilten, heterogenen Computerplattformen

men eingesetzt werden kann. Hierbei kommt das für den nicht kommerziellen Gebrauch frei verfügbare CORBA-System ORBacus [OOC00] in der Version 3.3.1 zur Einsatz.

**Common Services:** Die Common Services stellen Basisdienste für alle Activity-Komponenten zur Verfügung, beispielsweise die Management- und Nachrichtendienste.

**CSCW Facility:** Da es keinen Standard für eine CSCW-ähnliche Facility gab, musste für CASSY eigens eine solche Facility konstruiert werden. Dies wurde im Rahmen der Diplomarbeit von Michael Muchitsch verwirklicht [Muc97].

Die CSCW Facility basiert auf einem erweiterten Modell der *Joint Common Business Objects*-Veröffentlichung der *Object Management Group* (OMG). Es handelt sich dabei um ein Objektmodell des NIIP-Konsortiums für *Common Business Objects*, das im Rahmen der OMG-Arbeitsgruppe *OMG Business Objects Task Force* eingebracht wurde. Dieses Objektmodell des NIIP-Konsortiums stellt die Basis für das CSCW-Facility-Objektmodell dar und wurde, um den speziellen Anforderungen von CASSY gerecht zu werden, um zusätzliche Funktionen erweitert. Die Erweiterungen des CSCW-Facility-Objektmodells gegenüber des NIIP-Objektmodells liegen vorwiegend in der umfassenderen Betrachtung der Möglichkeiten für die Unterstützung kollaborativer Aktivitäten begründet.

**Workflow Management Facility:** Basierend auf dem Workflow-Management-System Flowmark der Version 2.3 (IBM) wurde der jFlow-Vorschlag der *Object Management Group* (siehe Kapitel 4) für eine CORBA-basierte Workflow Management Facility im Rahmen der Diplomarbeit von Günter Heiß [Hei98] umgesetzt.

**Activity Management Facility:** Die Activity Management Facility implementiert das in Abschnitt 3.1.1 vorgestellte Aktivitätenmodell von ASCEND. Sie bietet Schnittstellen zu den verschiedenen Aktivitätstypen des ASCEND-Modells an, mit denen sowohl Workflow-Management- als auch Groupware-Aktivitäten unterstützt werden können. Der Aufbau sollte modular und leicht erweiterbar sein. Außerdem soll die Activity Management Facility in möglichst vielen verschiedenen Software- und Hardware-Umgebungen, als auch im Internet-Umfeld anwendbar sein. Sie soll nicht von speziellen Workflow-Management- und Groupware-Systemen abhängig sein, damit sie flexibel an unterschiedliche Systeme angepaßt werden kann. Die wesentlichen Teile der Activity Management Facility wurden von Markus Rösner durch seine Studienarbeit [Rös99] und anschlie-

ßende HiWi-Tätigkeit erarbeitet. Das Ziel dieser Diplomarbeit ist, einen Wrapper zu entwickeln, der die Workflow Management Facility in das Aktivitätenmodell von ASCEND einbindet. Somit trägt diese Arbeit auch zur Entwicklung der Activity Management Facility bei.

**User Assistance:** Die User-Assistance-Komponente soll die Funktionalität des Aktivitätenmanagements für den Benutzer verfügbar machen. Diese Komponente muss noch implementiert werden.

Die User-Assistance-Komponente, auch Benutzerassistent genannt, soll eine möglichst gute Abbildung zwischen Bedienung und Sprache der Activity Management Facility darstellen. Da die Benutzerschnittstelle für die Akzeptanz des Benutzers besonders wichtig ist, muss besonders darauf geachtet werden, die Bedienung möglichst einfach und intuitiv zu gestalten. Für die Realisierung des Benutzerassistenten sollen auch Agentenkonzepte mit einbezogen werden. Dadurch ergeben sich erweiterte Kooperationsmöglichkeiten für die Benutzerassistenten unterschiedlicher Benutzer.

Zusätzlich zur Implementierung von CASSY wurde die Anwendbarkeit des Designflow-Modells zur Beschreibung realistischer Anwendungsszenarien in der Diplomarbeit von Marcello Mariucci untersucht und anhand von Beispielen verifiziert [Mar98].



# Kapitel 4

## Die Workflow Management Facility

Die Workflow Management Facility als Teil von CASSY, wurde nach einem Vorschlag der *joint Request for Proposals Submission* für eine Workflow Management Facility (jFlow) entwickelt. jFlow wurde als gemeinsamer Vorschlag verschiedener Anbieter aufgrund eines Aufrufs der *Object Management Group* (OMG - siehe Abschnitt 5.1) eingereicht, mit dem Ziel einen Standard für CORBA-basierte Workflow-Management-Applikationen zu schaffen.

jFlow orientiert sich sehr stark an dem Workflow-Referenzmodell der *Workflow Management Coalition* (WfMC). Deshalb werden nachfolgend zuerst die WfMC und ihre Ziele kurz vorgestellt und das Workflow-Referenzmodell beschrieben. Danach werden die wichtigsten Anforderungen des *Request for Proposals* gezeigt und der jFlow-Vorschlag sowie das Objektmodell dieses Vorschlags erklärt. Am Ende dieses Kapitels werden die Implementierung und die Anwendbarkeit der Workflow Management Facility beurteilt.

### 4.1 Das Workflow-Referenzmodell der WfMC

Die Workflow Management Coalition wurde 1993 gegründet, um Standards zur Beschreibung von Workflow-Management-Systemen und -Umgebungen zu erstellen. Die über 100 Mitglieder der WfMC kommen aus Hersteller-, Anwender- und Forscherkreisen. Die Ziele der WfMC sind:

- Förderung des allgemeinen Verständnisses für Workflow-Management,
- Entwicklung einer Terminologie zur Beschreibung von Workflow-Management-Systemen und ihres Umfelds,
- Standardisierung und Bereitstellung eines Workflow-Referenzmodells und

- Zusammenarbeit mit anderen Industriekonsortien zur Definition weiterer gemeinsamer Standards.

Im Mittelpunkt der Standardisierungsbemühungen der WfMC steht das Referenzmodell für Workflow-Management-Systeme. Das Workflow-Referenzmodell ist ein *Metamodell*, also ein Modell das beschreibt, wie der Modellierungsvorgang - in diesem Fall das Erstellen einer Workflow Management Facility - beschaffen sein muss. Für ein Metamodell können durchaus unterschiedliche Implementierungen existieren. Der prinzipielle Aufbau des Produkts ist jedoch durch das Metamodell vorgegeben. In [JBS99] findet sich folgende Erläuterung zum Begriff Metamodell:

Das Nachdenken über das Modellieren, die Modellierungskonzepte und Modellierungsbegriffe führt zu einer Vorstellung vom Modellieren, einem Modell des Modellierens, einem **Metamodell**.

Das Workflow-Referenzmodell der WfMC wurde in besonderem Maße darauf ausgerichtet, Interoperabilität zwischen unterschiedlichen Workflow-Management-Systemen zu unterstützen, denn zum gegenwärtigen Zeitpunkt ist in aller Regel keine Zusammenarbeit zwischen den Workflow-Management-Systemen verschiedener Hersteller möglich. Das Workflow-Referenzmodell zeigt die funktionalen Bereiche auf, die von der Workflow Management Facility angesprochen werden und zeigt typische Anwendungsgebiete eines WMS.

Abbildung 4.1 illustriert die wesentlichen Komponenten und Schnittstellen des Workflow-Referenzmodells [WfM95]. Im Mittelpunkt der Architektur steht der *Workflow Enactment Service*. Diese Komponente ist für die Abwicklung und Koordination laufender Workflows zuständig. Dieser Dienst kann durch eine oder mehrere *Workflow Engines* (dt. Laufumgebungen für Workflows) realisiert sein. Damit Komponenten unterschiedlicher WMS-Hersteller mit dem System verbunden werden können, wurden im Referenzmodell der WfMC fünf separate Schnittstellen definiert:

1. **Process definition tools** (dt. Workflow-Modellierungswerkzeuge) dienen dem Austausch von Workflow-Schemata. Sie ermöglichen die Beschreibung von Workflows zum Zeitpunkt der Erstellung (*Buildtime*) und die Beschreibung des Datenaustausches von Workflows mit der Workflow-Ausführungsumgebung zur Laufzeit (*Runtime*). Die formale Beschreibung von Workflows wird häufig durch eine grafische Oberfläche unterstützt. Es ist jedoch auch möglich, Workflows anhand einer formalen Sprache textuell zu beschreiben. Die Definition eines Workflows umfasst dabei Aspekte wie die Bestimmung der logischen Abfolge und eventueller Parallelisierbarkeit von Aktivitäten, Festlegung der Start- und Endbedingungen von Akti-

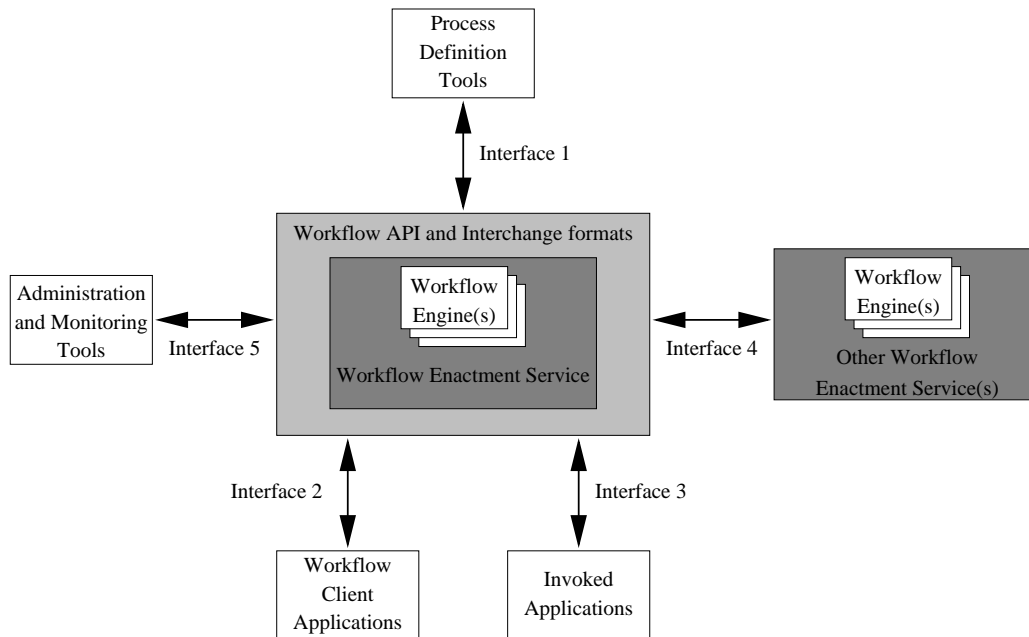


Abbildung 4.1: Das Workflow-Referenzmodell der WfMC

vitäten, der Zuordnung zu Bearbeitern und der Verknüpfung von Aktivitäten mit Applikationen.

2. **Workflow client application interfaces**, also Schnittstellen für Anwendungsprogramme, dienen der Unterstützung verschiedener Benutzerschnittstellen. Client-Applikationen stellen für den Benutzer die Schnittstelle zur Workflow-Engine dar. Sie ermöglichen das Instanzieren und die Kontrolle von Prozessinstanzen (z. B. *start*, *resume*, *suspend*, *terminate*, *abort*, ...), die Abfrage des Prozesszustandes und die Manipulation von Prozessdaten und Worklists zur Laufzeit.
3. **Invoked application interfaces** sind Schnittstellen für die Unterstützung verschiedener IT-Anwendungen. Da ein WMS nur den Rahmen zur Ausführung von Workflows zur Verfügung stellt, besteht die Notwendigkeit externe Applikationen einbinden zu können, mit denen der Benutzer seine Aufgaben erfüllen kann. Beispiele sind Dokumentenmanagementsysteme, Textverarbeitungsprogramme, Tabellenkalkulationprogramme und andere Anwendungen. Durch die Standardisierung der Schnittstelle 3 soll die Integration externer Anwendungen wesentlich erleichtert werden. Anwendungen, die direkt mit der Workflow Engine interagieren können, bezeichnet man als „workflow enabled“. Alle anderen Anwendungen benötigen dafür

ein Zwischenprogramm (*Application Agent*).

4. **Workflow interoperability interfaces** sind Schnittstellen für die Interaktion mit anderen Workflow-Systemen. D. h. Schnittstelle 4 dient dem Austausch Workflow-relevanter Daten zwischen verschiedenen Workflow Engines. Ziel ist es, einen Subworkflow eines WMS einem anderen WMS zur Ausführung übergeben zu können.
5. **Administration and monitoring interfaces** sollen die Anwendung von Administrations- und Monitoringdiensten eines Anbieters auf dem WMS eines anderen Anbieters ermöglichen. Administrations- und Monitoringdienste dienen der Systemüberwachung und Leistungsanalyse. Sie erleichtern damit das Management von Workflow-Anwendungen und -Umgebungen.

Die *Workflow API* (WAPI) ist eine Menge gemeinsamer *Application Programming Interfaces* (API) und den dazugehörigen Datenaustauschformaten zur Unterstützung der fünf Schnittstellen des Workflow-Referenzmodells. Durch die WAPI wird die Funktionalität der Schnittstellen zusammengefasst und die Wiederverwendung von Programmcode bei überschneidenden Schnittstellenfunktionen ermöglicht.

## 4.2 Request for Proposals für eine Workflow Management Facility

Im Mai 1997 veröffentlichte die Object Management Group einen *Request for Proposals* (RfP) zur Spezifikation einer Workflow Management Facility [OMG97b]. Ein RfP ist eine Aufforderung an die Mitglieder der OMG, Vorschläge zu dem spezifizierten Thema einzureichen. Dieser RfP hatte das Ziel, einen objektorientierten Framework zu schaffen, um unterschiedlichen Workflow-Management-Systemen eine Zusammenarbeit zu ermöglichen. Der RfP enthielt sowohl obligatorische als auch optionale Anforderungen an die Spezifikation. Die obligatorischen Anforderungen betreffen folgende Aspekte:

- Beschreibung der kompletten semantischen Definition des Workflow-Metamodells.
- Spezifikation von Schnittstellen bezüglich der Ausführung von Workflows.
- Zugriff auf Zustandsinformationen der Workflows.
- Zugriff auf die Ausführungsgeschichte der Workflows.

- Möglichkeiten zur Verschachtelung von Workflows.

Anforderungen bezüglich der Metadaten von Workflowinstanzen zählen zu den optionalen Anforderungen des RfP und wurden in jFlow noch nicht oder nur teilweise spezifiziert:

- Definition von Workflow-Schemata.
- Verschachtelung von Workflow-Schemata.
- Unterstützung von Ad-hoc-Workflows.
- Level-of-Service-Parameter.

### 4.2.1 jFlow - Joint RfP Submission

jFlow ist einer der Vorschläge, die aufgrund des RfP der OMG eingereicht wurden [OMG97c]. Eine kritische Betrachtung von jFlow und RfP, sowie weitere Quellen liefert [Hei98]. Der jFlow-Vorschlag ist eine Spezifikation für die Implementierung einer CORBA-basierten Workflow Management Facility. In jFlow wurden ein Framework für Workflows und Schnittstellen spezifiziert, welche von einer großen Anzahl von Workflow-Anbietern und -Benutzern unter der Schirmherrschaft der Workflow Management Coalition entwickelt wurden. jFlow wurde von 19 Firmen zusammen erarbeitet und wird von 18 weiteren Firmen unterstützt. Die spezifizierten Schnittstellen für die Interaktion mit einer Workflow-Prozess-Ausführungsumgebung sollen ein möglichst breites Spektrum an Implementierungen erlauben, insbesondere auch solche Implementierungen, die auf bereits existierenden Workflow-Systemen basieren. Dies kann als wesentliche Stärke des jFlow-Vorschlags angesehen werden, da er von zahlreichen Workflow-Anbietern unterstützt wird. Auf diese Weise soll das Ziel des RfP, die Zusammenarbeit unterschiedlicher WMS zu ermöglichen, erreicht werden.

Alle obligatorischen Anforderungen des RfP wurden in jFlow erfüllt. jFlow basiert in seiner Konzeption auf dem Workflow-Referenzmodell der WfMC. In jFlow wurden *workflow client application interfaces* (Schnittstelle 2), *interoperability interfaces* (Schnittstelle 4) und *process monitoring interfaces* (Teil der Schnittstelle 5) des Workflow-Referenzmodells spezifiziert. *Process definition interfaces* (Schnittstelle 1) und *administration interfaces* (Teil der Schnittstelle 5) sollen erst in späteren RfPs spezifiziert werden. *Invoked application interfaces* (Schnittstelle 3) wurden nicht spezifiziert. D. h. jFlow spezifiziert zwar die wichtigsten, allerdings nicht alle Elemente des Workflow-Referenzmodells.

Inzwischen gibt es schon eine überarbeitete, zweite Version von jFlow. Da die Workflow Management Facility jedoch auf der ersten jFlow-Version basiert, wurde in diesem Kapitel noch diese ältere jFlow-Version beschrieben.

### 4.3 Das jFlow-Objektmodell

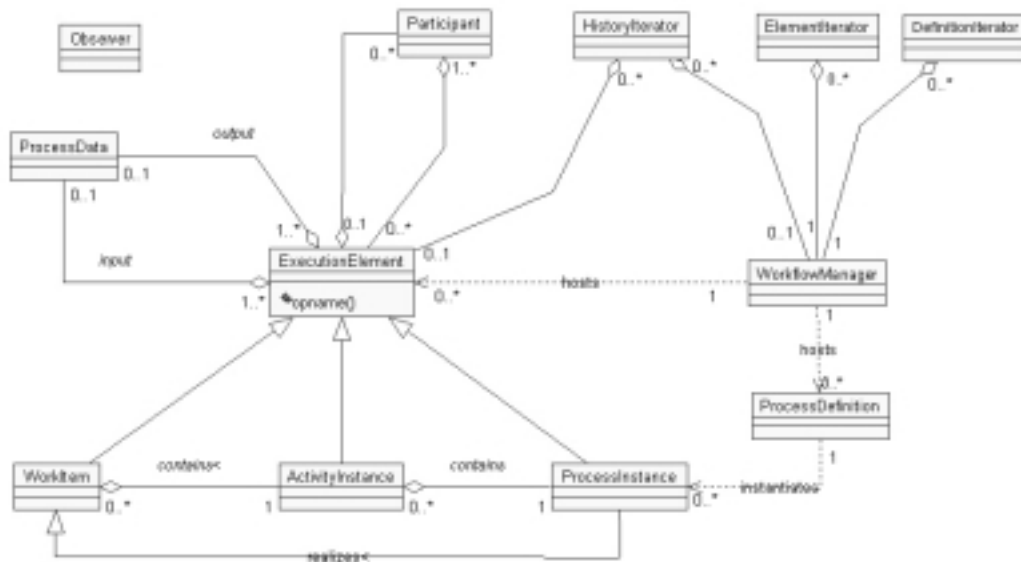


Abbildung 4.2: Das jFlow-Objektmodell

Abbildung 4.2 [OMG97c] zeigt das jFlow zugrunde liegende Objektmodell. Zentrales Interface ist `ExecutionElement`, welches den Kern einer Workflow-Ausführung implementiert. `ExecutionElement`-Objekte werden vom Workflow-Manager verwaltet. Dieser verwaltet außerdem auch den Ablauf (engl. *History*) und die Erzeuger (engl. *Factories*) der `ExecutionElement`-Objekte. Für das `ExecutionElement`-Interface gibt es drei Spezialisierungen:

- `ProcessInstance` repräsentiert die Instanz eines spezifischen Workflow-Prozess-Modells.
- `ActivityInstance` repräsentiert die einzelnen Prozessschritte im Workflow-Prozess-Modell von `ProcessInstance`.
- `WorkItem` ordnet die Prozessschritte den einzelnen Workflow-Teilnehmern zu. Da `WorkItem`-Objekte auch von `ProcessInstance`-Objekten realisiert werden können, die von einem anderen `WorkflowManager`-Objekt verwaltet werden, ist es auch möglich, komplexe Workflowprozesse verteilt auszuführen.

Mit Hilfe dieser Elemente lassen sich aus `ExecutionElement`-Objekten komplexe Workflow-Prozess-Instanzen aufzubauen. Das *BoWorkflow*-Modul von

jFlow definiert die Schnittstellen der Workflow Management Facility. Diese werden im Folgenden beschrieben.

### 4.3.1 ExecutionElement

Das ExecutionElement-Interface definiert generische Operationen und Attribute, welche alle ausführbaren Workflow-Objekte, also ProcessInstance-, ActivityInstance- und WorkItem-Objekte gemeinsam haben. Ein ExecutionElement-Objekt steht stellvertretend für eine Komponente einer Workflow-Instanz. Der aktuelle Zustand von ExecutionElement kann jederzeit ermittelt und Zustandsübergänge ausgeführt werden. Abbildung 4.3 zeigt das Zustandsübergangsdiagramm [OMG97c].

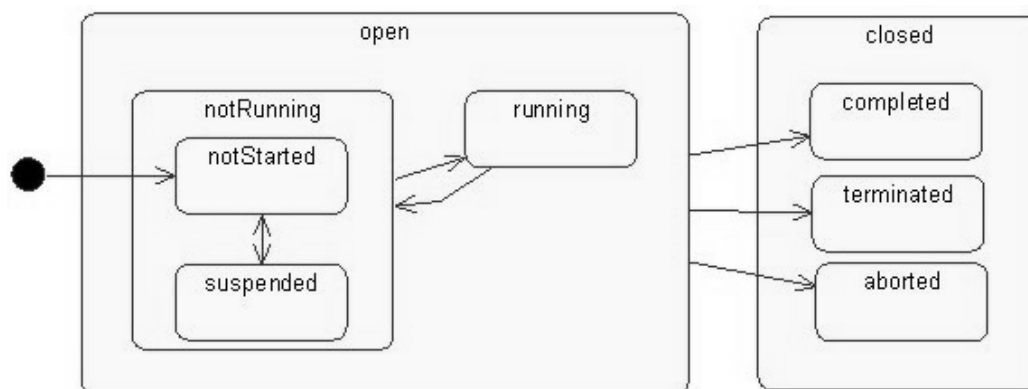


Abbildung 4.3: Zustandsübergangsdiagramm von ExecutionElement

### 4.3.2 WorkflowManager

Das WorkflowManager-Interface nimmt im jFlow-Objektmodell eine wichtige Rolle ein. Workflow-Manager verwalten die *Factories* für ausführbare Elemente und ermöglicht den Zugriff auf gefilterte Mengen von Workflow-Entitäten und deren Ablauf. Der Workflow-Manager ermöglicht den Zugriff auf Prozeßdefinitionen, indem er Referenzen auf ProcessDefinition-Objekte liefert und er ermöglicht den Zugriff auf Prozessinstanzen, Aktivitäten und Workitems, indem er Referenzen auf ExecutionElement-Objekte liefert. Diese Referenzen können entweder anhand eines eindeutigen Schlüssels direkt vom WorkflowManager-Objekt bezogen werden oder durch Angabe eines Filters über das DefinitionIterator-Interface. Leider sind in jFlow keine Factories für Workflow-Manager spezifiziert. Es wird angenommen, daß diese bereits existieren und in einem CORBA Name Service gefunden werden können.

### 4.3.3 ProcessDefinition

Das ProcessDefinition-Interface repräsentiert ein ausführbares Workflow-Modell. ProcessDefinition ist eine Factory zur Erzeugung von ProcessInstance-Objekten, also für die Instanzen eines Workflow-Modells. Da das Erzeugen von Prozessdefinitionen nicht Gegenstand von jFlow ist, wird davon ausgegangen, dass ProcessDefinition-Objekte genauso wie WorkflowManager-Objekte von außen zur Verfügung gestellt werden.

### 4.3.4 ProcessInstance

Ein ProcessInstance-Objekt ist die Instanz einer bestimmten Workflow-Prozess-Definition. Sie repräsentiert ein aktives Workflow-Prozess-Modell und liefert Informationen, um die Ausführung dieses Modells kontrollieren zu können. Abbildung 4.4 zeigt das Zustandsübergangsdiagramm [OMG97c].

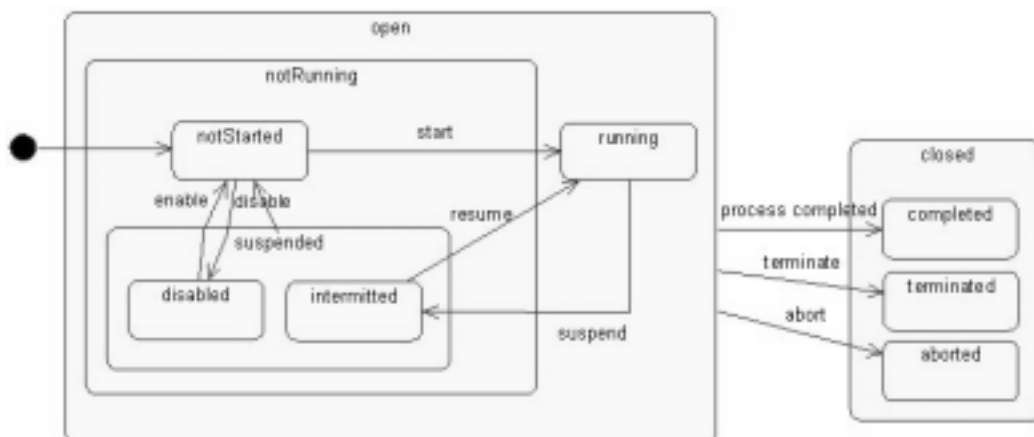


Abbildung 4.4: Zustandsübergangsdiagramm von ProcessInstance

### 4.3.5 ActivityInstance

Eine ActivityInstance ist ein Prozessschritt innerhalb eines ProcessInstance-Objekts. Durch ActivityInstance wird eine Aktion innerhalb eines Workflows beschrieben. Ihr ist ein verantwortlicher Benutzer oder eine Gruppe und ein oder mehrere Bearbeiter zugeordnet. Eine ActivityInstance stellt sozusagen einen „aktiven Knoten“ dar und wird im wesentlichen zur Aufzeichnung des Fortschritts einer Prozess-Instanz verwendet. Die Zustandsübergänge sind in Abbildung 4.5 zu sehen [OMG97c].

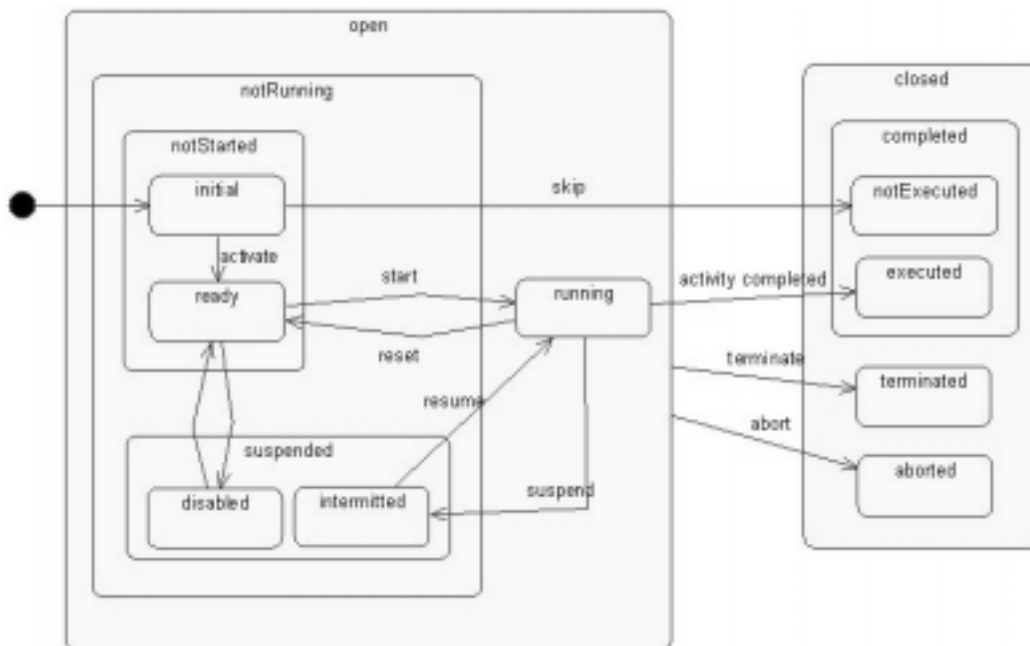


Abbildung 4.5: Zustandsübergangsdiagramm von ActivityInstance

### 4.3.6 ProcessData

Das ProcessData-Interface erlaubt den Zugriff auf die Prozessrelevanten Daten, welche an der Ausführung eines ProcessInstance-Objekts beteiligt sind. Im ProcessData-Objekt wird die Menge der relevanten Prozessdaten beschrieben, die mit einem bestimmten ProcessInstance-Objekt in Zusammenhang stehen. Prozessdaten sind prozessbezogene Eigenschaften eines ExecutionElement-Objekts und können zwischen verschiedenen ExecutionElement-Objekten, die in einer Prozess-Instanz enthalten sind, ausgetauscht werden.

### 4.3.7 WorkItem

Ein WorkItem ist ein Teil einer Arbeit, der genau einem Bearbeiter (repräsentiert durch ein Participant-Objekt) zugeordnet ist. Es entspricht also der Zuordnung eines bestimmten Workflow-Teilnehmers zu einem ActivityInstance-Objekt (welches mehrere WorkItem-Objekte umfassen und somit auch mehrere Bearbeiter haben kann) und erlaubt die Überwachung der Arbeitsausführung. D. h. durch das Interface wird die Aufgabe des Teilnehmers bei der Ausführung eines Prozessschrittes repräsentiert, der dem ActivityInstance-Objekt zugeord-

net ist. Abbildung 4.6 zeigt das Zustandsübergangsdiagramm eines `WorkItem`-Objekts [OMG97c].

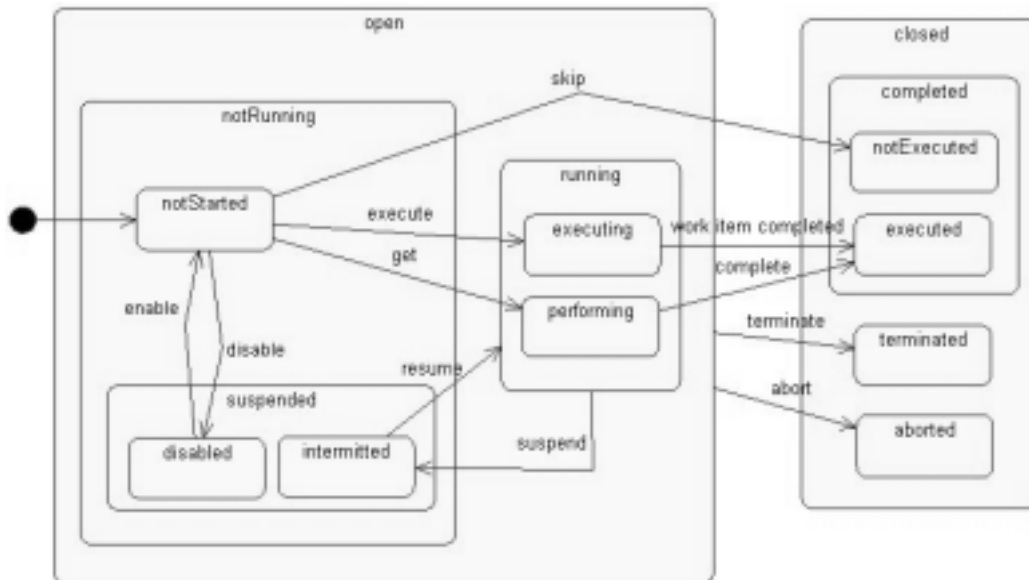


Abbildung 4.6: Zustandsübergangsdiagramm von `WorkItem`

### 4.3.8 Observer

`Observer` wird über Zustandsänderungen der `ExecutionElement`-Objekte benachrichtigt. Dieses Interface wird benutzt, um Zustandsänderungen von `ExecutionElement`-Objekten aufzuzeichnen. Es empfängt Mitteilungen, welche den Zustand von `ExecutionElement`-Objekten betreffen.

### 4.3.9 Participant

Ein `Participant`-Objekt dient als Platzhalter für Bezüge in ein Organisationsmodell. Das Interface ist in `jFlow 1.0` noch nicht vollständig spezifiziert. Es repräsentiert die an der Ausführung eines `ProcessInstance`-Objekts beteiligten Ressourcen oder Personen.

### 4.3.10 HistoryIterator

`HistoryIterator` definiert einen Iterator für die Ablaufdaten. Das Interface ermöglicht den Zugriff auf gefilterte Mengen von aufgezeichneten Prozessdaten, die mit `ExecutionElement`-Objekten eines bestimmten Typs verbunden sind.

### 4.3.11 ElementIterator

Das `ElementIterator`-Interface definiert einen Iterator für `WorkflowElement`-Objekte. `ElementIterator` ermöglicht den Zugriff auf gefilterte Mengen von `WorkflowElement`-Objekten eines spezifischen Typs (`ProcessInstance`, `ActivityInstance` oder `WorkItem`), die im gleichen Workflow-Kontext vorkommen.

### 4.3.12 DefinitionIterator

`DefinitionIterator` ermöglicht den Zugriff auf gefilterte Mengen von Prozessdaten, die von einem `WorkflowManager`-Objekt verwaltet werden. Durch das Interface wird ein Iterator für die Menge der Prozessdefinitionen definiert und damit, ähnlich wie durch `ElementIterator` oder `HistoryIterator`, ein einfacher Zugriff auf die Objekte ermöglicht.

## 4.4 Implementierungsaspekte der Workflow Management Facility

Bei der Integration eines Programms bestehen generell zwei verschiedene Möglichkeiten, um auf die Funktionen des Programms zuzugreifen. Entweder bietet das zu integrierende Programm bereits von sich aus über ein API Schnittstellen an, auf die man von außen zugreifen kann oder es muss durch einen speziell zum Zweck der Integration zu implementierenden *Tool Agent* nutzbar gemacht werden. Es liegt auf der Hand, dass eine Applikation die mit einem API versehen ist, wesentlich einfacher und damit zeitsparender und nahtloser integriert werden kann. Nicht zuletzt weil das Workflow-Management-System Flowmark über ein *C++ Client API* verfügt (siehe [Flo96] Kapitel 8 und 9), dürfe die Wahl auf dieses System gefallen sein.

Im Rahmen des Projekts ASCEND wurde durch die Diplomarbeit von Günter Heiß bereits eine Workflow Management Facility anhand der jFlow-Spezifikation implementiert. Die Workflow Management Facility integriert das Workflow-Management-System Flowmark (IBM) der Version 2.3. Für Details bezüglich der Integration von Flowmark sei auf [Hei98] und die technische Dokumentation von Flowmark verwiesen. Die Spezifikation der Workflow Management Facility ist durch jFlow schon vorgegeben und wurde in den vorausgegangenen Abschnitten beschrieben. Deshalb sollen hier nur die Unterschiede zwischen der Spezifikation aus jFlow und der konkreten Implementierung der Workflow Management Facility von Günter Heiß, sowie die Ursachen dieser Unterschiede beschrieben werden.

**1. Operationen die in jFlow spezifiziert sind, aber in Flowmark keine Entsprechung finden:**

- (a) Das setzen der Namen von Workflow-Elementen (`ProcessDefinition`, `ProcessInstance`, `ActivityInstance` und `WorkItem`) ist in der C++ Client API nicht vorgesehen.
- (b) Flowmark besitzt kein `ActivityInstance`-Interface und keine `History`-Verwaltung.

**2. Funktionalität die in Flowmark vorhanden ist, in jFlow allerdings nicht oder nur ungenau spezifiziert ist:**

- (a) Das `Participant`-Interface ist in jFlow nicht genau spezifiziert und bedarf deshalb einer eigenen Lösung.

**3. Unvollständige bzw. noch nicht implementierte Funktionen der WMF:**

- (a) `Observer`, `HistoryIterator` und `ActivityInstance` sind in der WMF nicht implementiert.
- (b) Einzelne Klassenmethoden wie `ProcessDefinition::list_process_instances` wurden wegen 1. (a) nicht implementiert. D. h. die Funktionalität von `ElementIterator` steht praktisch nicht zur Verfügung.

**4. Fehler bei der Implementierung der WMF:**

- (a) Bei der Compilierung des Quellcodes aufgetretene Syntaxfehler erschwerten die Portierung auf ein Windows NT-System.
- (b) Das `BoWorkflow-Server`-Programm stürzt aus ungeklärten Gründen häufig ab und erfordert oft mehrere Start-Versuche.
- (c) Die korrekte Funktionsweise der Zustandsübergangsfunktionen konnte aufgrund von Fehlern im `BoWorkflow-Client`-Programm nicht verifiziert werden.

Die aufgelisteten Unzulänglichkeiten zeigen, dass das Vorhandensein einer API noch keinen Garant für eine problemlose Integration darstellt, denn oft resultieren die Schwierigkeiten aus Unterschieden in den verwendeten Modellen, wie die unterschiedliche semantische Interpretation von `WorkItem` zwischen der C++ Client API und jFlow zeigt. Während sich einfache Funktionen evtl. noch auf Umwegen realisieren lassen, muss bei grundlegenden Modell-Unterschieden entweder auf Teile der Funktionalität verzichtet werden oder über andere Realisierungsmöglichkeiten nachgedacht werden.

*MQSeries Workflow* ist die nächste Generation des Workflow-Management-Systems Flowmark der IBM. MQSeries liefert eine offene, skalierbare Informations-Infrastruktur, die den Vorgaben des Workflow-Referenzmodells der WfMC entspricht und für den Einsatz auf unterschiedlichen Plattformen konzipiert wurde. Trotz der erweiterten Funktionalität ist MQSeries weitgehend Flowmark-kompatibel. Eine zukünftige Arbeit soll die Workflow Management Facility auf den neuesten Stand bringen und auf MQSeries basieren. Idealerweise soll dabei auch schon die Anpassung an jFlow Version 2.0 erfolgen. Die neue WMF wird nach Möglichkeit dieselben Schnittstellen verwenden, wie die bisherige. Aus diesem Grund ist es besonders wichtig, dass die Schnittstellenspezifikation genau beachtet und die Verwendung der Schnittstellen gut dokumentiert wird, damit die spätere Zusammenarbeit der Module mit möglichst geringen Problemen vonstatten gehen kann.



# Kapitel 5

## CORBA

Abschnitt 3.2 hat gezeigt, dass CORBA (**C**ommon **O**bject **R**quest **B**roker **A**rchitecture) einen wesentlichen Bestandteil des Systems CASSY ausmacht. Als Middleware-Komponente abstrahiert CORBA von der darunterliegenden Plattform und bildet somit eine Basis, auf der die anderen Komponenten von CASSY aufbauen können. Damit schafft CORBA die notwendigen Voraussetzungen für ein verteilungstransparentes System, durch das viele Eigenschaften von CASSY überhaupt erst ermöglicht werden:

- CORBA basiert auf einem *objektorientierten Modell*. Dies ermöglicht eine höhere Wiederverwendung von Programmcode und eine höhere Softwarequalität.
- Zugriffe von CORBA-Programmen auf entfernte Objekte sind *verteilungs-transparent*. D. h. es wird auf entfernte Objekte mit denselben Methoden zugegriffen, wie auf lokale Objekte. Eine Client-Applikation hat in der Regel keine Kenntnis vom genauen Aufenthaltsort der Objekte.

Zur Überbrückung zeitlicher und räumlicher Distanzen setzen CSCW-Systeme auf verteilten Systemen auf und bilden somit verteilte Anwendungen. Auch Workflow-Management-Anwendungen müssen, um gesamtbetriebswirtschaftliche Abläufe unterstützen zu können, wegen des Umfangs solcher Abläufe und aus Gründen räumlicher Distanz, als verteiltes System realisiert werden. Ein einzelner, zentraler Workflow-Management-Server ist oft nicht ausreichend.

- CORBA ist *hardware-, betriebssystem- und programmiersprachenunabhängig*. Dies ermöglicht den Einsatz von CASSY in einem heterogenen Systemumfeld. Beispielsweise kann eine Server-Applikation der Workflow Management Facility in C++ realisiert sein und auf einer UNIX-Maschine

ablaufen, während die Client-Applikation in Java realisiert ist und auf einem Windows-System bedient wird.

- Der Einsatz von CORBA erleichtert die Entwicklung eines *offenen Systems*. Über den ORB können Programme verschiedener Hersteller zusammenarbeiten. Dies ist für CASSY ein nicht unwesentlicher Aspekt.

Mit diesen Eigenschaften lassen sich die Ziele von CASSY, existierende Workflow-Management- und Groupware-Systeme und -Werkzeuge wiederzuverwenden und dabei möglichst viele Software-, Hardware- und Internetumgebungen zu unterstützen, verwirklichen. Zur Realisierung von CASSY wird momentan das CORBA-System *ORBacus* des Anbieters OOC (Object Oriented Concepts) in der Version 3.3.1 verwendet [OOC00]. Diese Version ist zum gegenwärtigen Zeitpunkt die aktuellste Version, von der alle Sourcen verfügbar sind und die außerdem zu den vorausgegangenen Implementierungen der anderen Diplomarbeiten noch voll kompatibel ist. Die Verwendung von ORBacus 4.0 würde, aufgrund von Neuerungen und Anpassungen an die neueste CORBA-Spezifikation, Änderungen im Quellcode der anderen Arbeiten erfordern und wurde deshalb nicht eingesetzt. ORBacus ist frei verfügbar und darf für nicht kommerzielle Zwecke kostenlos verwendet werden. Nähere Informationen zu ORBacus, Dokumentation, die jeweils aktuellste Version des ORBacus und Zusätze für den Einsatz mit Java kann man über die Internetseite von OOC (<http://www.ooc.com/>) beziehen. Es gibt noch eine Anzahl weiterer Middleware-Produkte verschiedener Anbieter, von denen jedoch keines dieselbe Flexibilität wie CORBA bietet. Beispielsweise ist Microsofts *Component Object Model* (COM) nicht plattformunabhängig und die *JavaBeans*-Technologie ist nicht programmiersprachenunabhängig. CORBA-Systeme dagegen sind bei zahlreichen Anbietern verfügbar, z. B. Orbix (IONA), DSOM (IBM), NEO (Sun), VisiBroker (Inprise/Borland) und werden oft sogar als Freeware- oder Shareware-Produkt angeboten, z. B. ORBacus (OOC), JacORB (Freie Universität Berlin) und OmniORB (AT&T).

Für eine Einführung in CORBA sei auf [Red97] verwiesen. [HV99] bietet fundierte Details und ist daher gut als Nachschlagewerk geeignet. Die *Object Management Group* (OMG) beschreibt sich selbst, ihre Ziele und ihre aktuellen Tätigkeiten auf der Internetseite <http://www.omg.org>, über die auch die CORBA-Spezifikation und alle weiteren Spezifikationen der OMG der Öffentlichkeit zugänglich sind. Die Internetseite bietet eine Suchfunktion für OMG-Spezifikationen und liefert zahlreiche Verweise auf weitere Informationsquellen.

## 5.1 Die Object Management Group

Die *Object Management Group* (OMG), 1989 gegründet, ist heute ein internationaler Zusammenschluss aus über 800 Hardwareherstellern, Softwareentwicklern, Netzwerkbetreibern und kommerziellen Nutzern von Computersoftware. Die OMG zählt Akteure aus allen nennenswerten Gebieten der Softwaretechnologie zu ihren Mitgliedern, die alle das gemeinsame Ziel haben, objektorientierte Technologien in der Softwareentwicklung zu fördern.

Das Hauptziel der OMG ist die Definition einer universellen Kommunikationsplattform, welche eine Zusammenarbeit zwischen heterogenen, objektorientierten Systemen erlaubt. Die Definition dieser Kommunikationsplattform ist Gegenstand der *Object Management Architecture* (OMA). Weitere Ziele der OMG sind die Vereinheitlichung der verwendeten Begriffe zur Beschreibung verteilter Systeme, sowie die Definition allgemeiner Komponenten, welche in einer Vielzahl von Anwendungsgebieten zum Einsatz kommen.

Interessant ist das pragmatische Vorgehen der OMG, die nicht einfach irgend einen weiteren Standard definiert, sondern auf der kommerziell verfügbaren Software ihrer Mitglieder aufbaut und daraus die Standards entwickelt. Es werden Schnittstellenspezifikationen und Protokolle entwickelt, wozu die Mitglieder Vorschläge einreichen können. Diese Vorschläge werden dann von der OMG veröffentlicht, so dass unter allen interessierten Mitgliedern darüber diskutiert und abgestimmt werden kann.

Im Zuge der Standardisierung wurden von der OMG ein abstraktes Objektmodell und eine objektorientierte Referenzarchitektur definiert: die *Object Management Architecture*. Ein wesentlicher Bestandteil dieser Architektur ist der *Object Request Broker* (ORB), woraus sich für die Architektur die bekannte *Common Object Request Broker Architecture* (CORBA) ableitet. Der ORB ermöglicht die Kommunikation zwischen beliebigen CORBA-Objekten, wobei das CORBA-Objektmodell eine Konkretisierung des abstrakten OMG-Objektmodells darstellt. Die aktuelle CORBA-Spezifikation befindet sich in der Version 2.3.1 [OMG99b]. Version 3.0 der CORBA-Spezifikation steht kurz vor der Veröffentlichung.

### 5.1.1 Das Objektmodell der OMG

Die OMA basiert auf einem Kern-Objektmodell (engl. Core Object Model), das alle wesentlichen Konzepte der Objektorientierung - Kapselung, Vererbung, Polymorphie, etc. - unterstützt. Objekte werden in diesem Objektmodell als grundlegende, eindeutig identifizierbare Bausteine verteilter Anwendungen definiert. Sie verfügen über nach außen hin sichtbare Eigenschaften (Attribute) und Operationen (Methoden), über die das Verhalten der Objekte gesteuert werden kann. Die Implementierung eines Objekts ist vollständig von der Beschreibung der Eigen-

schaften und Operationen des Objekts getrennt. Dies wird durch die Verwendung einer von der OMG standardisierten, abstrakten Beschreibungssprache, der *Interface Definition Language* (IDL) erreicht. Über die Schnittstelle (engl. Interface) lassen sich alle von außen sichtbaren Eigenschaften eines Objekttyps festlegen. Objekte sind im Sinne dieses Objektmodells eigenständige Bestandteile einer verteilten Architektur, die miteinander durch gegenseitige Nutzung der exportierten Schnittstellen kommunizieren.

## 5.2 Die Object Management Architecture

Die *Object Management Architecture* (OMA) ist ein Referenzmodell für eine Systemarchitektur zur Entwicklung objektorientierter, verteilter Anwendungen, welche unabhängig von heterogenen Systemumgebungen ablauffähig sind [OMG97a]. Grundlage der OMA ist das OMG-Objektmodell.

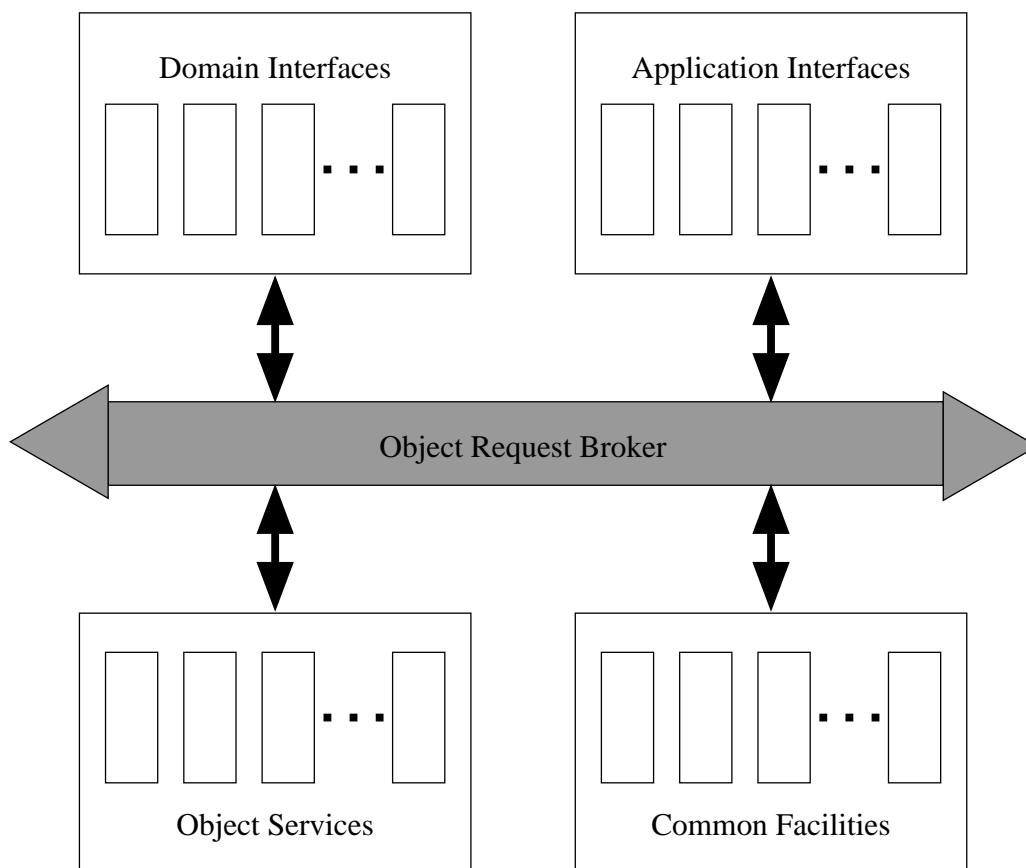


Abbildung 5.1: Object Management Architecture

Über die Schnittstelle eines Objekts angebotene Operationen werden in der OMG-Terminologie als *Services* bezeichnet. Ruft ein Objekt einen Service auf, wird es zum *Client* bezüglich der Dienstanforderung. Entsprechend wird das den Service implementierende Objekt zum *Server* bezüglich der Dienstanforderung. Zur Realisierung des Services kann sich der Server weiterer Objekte bedienen und tritt diesen gegenüber folglich selbst als Client auf. Gruppen von Objekten, die gemeinschaftlich eine Aufgabe wahrnehmen, werden zu Komponenten zusammengefasst und als solche in die Referenzarchitektur der OMA (siehe Abbildung 5.1 [OMG97a]) eingeordnet. Die OMA ist ein abstraktes Referenzmodell, für das es keine reale Implementierung gibt. Das Modell (genauer: Artefakt) wird konkretisiert durch weitere Spezifikationen der OMG, die in den Implementierungen verschiedener Anbieter realisiert sind.

**Der Object Request Broker** ist das Kernstück der Systemarchitektur. Beim ORB handelt es sich um eine logische Komponente, welche das Kommunikationsmedium zwischen den Objekten repräsentiert. Der ORB ist nicht als eigenständige, eindeutig identifizierbare Komponente realisiert. Er stellt vielmehr eine Kombination aus den in jedem Objekt enthaltenen Kommunikationsmechanismen und weiteren Komponenten, wie implementierungsspezifische Bibliotheken oder Dämon-Prozessen dar.

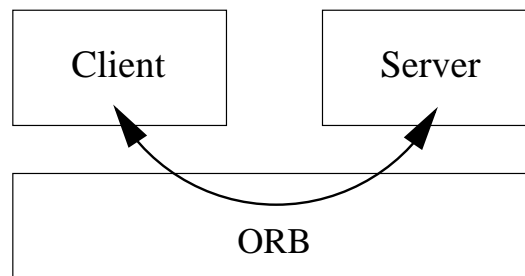


Abbildung 5.2: Client und Server kommunizieren über den ORB

Zu den Aufgaben des ORB zählt das Ermitteln des physikalischen Aufenthaltsortes eines Objekts anhand der jedem Objekt eindeutig zugeordneten *Objektreferenz*, das Versenden und Empfangen von Nachrichten in Byteströme (*marshalling* und *unmarshalling*), sowie die Konvertierung der Byteströme in plattformunabhängige Darstellungsformate. Der ORB reduziert den Informationsbedarf des aufrufenden Objekts (Client) somit auf die Kenntnis der vom Zielobjekt (Server) nach außen hin bereitgestellten Schnittstelle und entbindet den Client von Aspekten wie Aufenthaltsort, Realisierungsplattform und Implementierungssprache des Zielobjekts.

Die OMG stellt keine Realisierung eines solchen ORB zur Verfügung, sondern definiert einen Standard (CORBA) nach dem sich konkrete Implementierungen richten müssen [OMG99b]. Neben der Spezifikation des ORB definiert die OMG eine Reihe weiterer Komponenten, ebenfalls ohne dafür konkrete Implementierungen anzubieten. Diese Komponenten werden in Abhängigkeit ihrer Anwendungsbereiche einer der folgenden vier Kategorien zugeordnet:

**Object Services** bzw. *Common Object Services* (COS) sind horizontale, also anwendungsübergreifende Dienste, die grundlegende Aufgaben in verteilten, objektorientierten Anwendungen wahrnehmen können. Diese Low-Level-Dienste stellen die grundlegende Funktionalität für höhere Anwendungen zur Verfügung oder dienen der Entwicklung verteilter Applikationen.

Die Object Services der OMG werden allgemein unter dem Begriff *CORBAservices* zusammengefasst und sind im OMG-Dokument [OMG98] beschrieben. Zu den CORBAservices gehören unter anderem der *Naming Service* als zentrales Verzeichnis für Objektreferenzen und der Zuordnung von Namen zu Objektreferenzen, der *Life Cycle Service* für das Erzeugen, Kopieren, Migrieren und Löschen von Objekten, der *Event Service* für asynchrone Kommunikation bei Ereignissen, der *Externalisation Service*, der *Transaction Service*, etc.

**Common Facilities** stellen Dienste bereit, die nicht den allgemeinen Charakter der Object Services besitzen, jedoch trotzdem in einer Vielzahl von Anwendungsbereichen Verwendung finden. Dienste dieser Kategorie stehen auf einer höheren Ebene (vgl. Abbildung 5.3) und können sich somit auch der Object Services bedienen.

Common Facilities werden von der OMG als *CORBAfacilities* zusammengefasst und teilen sich gemäß [OMG95] in folgende vier Bereiche auf:

- *User Interface*. Dienste zur Bereitstellung von Benutzerschnittstellen.
- *Information Management*. Dienste für Modellierung, Speicherung, Verarbeitung und Austausch von Daten.
- *Systems Management*. Administrative Dienste zur Verwaltung verteilter Objekte.
- *Task Management*. Infrastruktur-Dienste die den Benutzer bei der Bearbeitung seiner Aufgaben unterstützen sollen. Hierzu gehört unter anderem auch eine Workflow Management Facility. Da diese Facility noch nicht spezifiziert ist, wurde für CASSY eine eigene Facility entwickelt.

**Domain Interfaces** wurden früher als vertikale Dienste bezeichnet, weil sie auf ein bestimmtes Anwendungsgebiet spezialisiert sind. Sie sind bezüglich ihres Aufbaus standardisiert, jedoch nicht hinsichtlich der von ihnen bereitgestellten Dienste. Domain Interfaces wurden für eine große Anzahl verschiedener Problembereiche von der OMG spezifiziert. Dazu gehören Dienste für das Finanzwesen (CORBAfinance), das Gesundheitswesen (CORBAmed), den Telekommunikationsbereich (CORBAtel), das Produktionswesen (CORBAmanufacturing), den elektronischen Handel (Electronic Commerce) und für Geschäftsabläufe (Business Object).

**Application Interfaces** sind nicht standardisierte, anwendungsspezifische Schnittstellen. Sie beschreiben konkrete Produkte eines Herstellers, für die keine Spezifikation der OMG vorliegt. Diese Dienste stellen die oberste Schicht der OMA dar, wie in Abbildung 5.3 zu sehen ist.

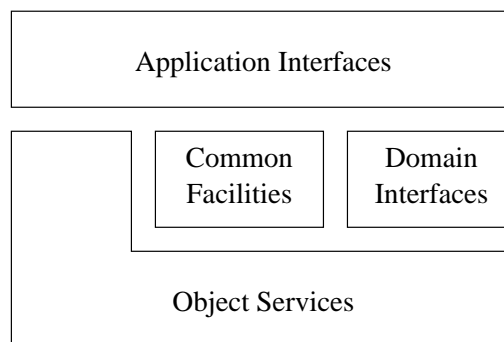


Abbildung 5.3: Basis- und Applikationsdienste der OMA

### 5.3 CORBA-Spezifikation

Die CORBA-Spezifikation beinhaltet das OMG-Objektmodell und dessen grundlegende Komponenten, das sogenannte *CORBA Core*. Des Weiteren werden in dem Dokument die IDL, der ORB und seine Bestandteile und Eigenschaften sowie Interoperabilitätsaspekte und der Austausch mit anderen Middleware-Technologien (Interworking) beschrieben. Der CORBA-Standard ist im OMG-Dokument [OMG99b] spezifiziert. Die Spezifikation gibt keine Bauanleitung zur Erstellung eines ORB, sondern definiert dessen Eigenschaften und gibt Richtlinien für die Implementierung vor. Die minimalen Anforderungen, denen ein

CORBA-System genügen muss, um von der OMG das Prädikat „CORBA-Compliant“ erhalten zu können, beinhalten die Implementierung der Kern-Komponenten gemäß der CORBA-Spezifikation, sowie mindestens einer Abbildung von IDL auf eine Programmiersprache gemäß der *Language Mapping Specification* für diese Programmiersprache. Die wichtigsten Teile des CORBA-Standards werden in den folgenden Unterabschnitten kurz zusammengefasst.

### 5.3.1 Die Interface Definition Language

Die OMG Interface Definition Language ist eine Deklarationsprache zur implementierungsunabhängigen Beschreibung von Objektschnittstellen [OMG99a]. Die syntaktischen Regeln der IDL sind dieselben wie im ANSI C++ Standard spezifiziert. Allerdings wurden neue Schlüsselwörter hinzugenommen, um die Beschreibung verteilter Konzepte zu ermöglichen. Die Grammatik der OMG IDL ist eine Teilmenge des ANSI C++ Standards, mit zusätzlichen Konstrukten für Operationsaufrufe versehen. D. h. IDL unterstützt die C++ Syntax für Konstanten, Typen und Operationsdeklarationen, beinhaltet aber keine algorithmischen Strukturen oder Variablen. Weil sie über keine Kontrollstrukturen verfügt, ist die IDL also keine berechnungsvollständige Sprache. In Bezug auf die Ausdrucksfähigkeit zur Beschreibung von Schnittstellendefinitionen stellt dies jedoch keinerlei Einschränkung dar.

Da es sich bei der IDL um eine rein deskriptive Sprache handelt, ist für die Implementierung der definierten Objekte eine Abbildung auf eine konkrete Programmiersprache notwendig. Diese Abbildung geschieht durch einen *IDL-Compiler*, basierend auf dem für die jeweilige Zielsprache spezifischen CORBA Language Mapping. Bisher wurden von der OMG CORBA Language Mappings für die Sprachen Ada, C, C++, COBOL, Java und Smalltalk spezifiziert, weitere sind geplant. Die Zielsprache muss nicht notwendigerweise eine objektorientierte Sprache sein. Durch das Language Mapping ist genau festgelegt, wie die Übersetzung von IDL in Sprachkonstrukte der jeweiligen Programmiersprache abzubilden ist. Diese Vorgehensweise schafft nicht nur die Unabhängigkeit der CORBA-Schnittstelle von der Programmiersprache an sich, sondern ermöglicht auch die Kommunikation von Client- und Server-Anwendungen, welche mit völlig unterschiedlichen Programmiersprachen realisiert wurden.

Diese Vorgehensweise, zuerst die Schnittstellendefinition und danach erst die Implementierung der Funktionen vorzunehmen, bezeichnet man als *Top-Down*. Der Umstand, dass für die Sprache Java nicht nur ein „IDL to Java“ sondern auch ein „Java to IDL“ Language Mapping existiert, erlaubt auch die umgekehrte Herangehensweise (*Bottom-Up*).

### 5.3.2 Der Object Request Broker

Der ORB stellt das Medium für die Kommunikation zwischen den einzelnen Objektimplementierungen dar. Als solches ist er für die Lokalisierung der Objektimplementierungen über die global eindeutigen Objektreferenzen (*Interoperable Object Reference* - IOR), die Weiterleitung der Methodenaufrufe (Requests) an die einzelnen Objektimplementierungen, sowie die Übertragung der Ergebnisse (Responses) verantwortlich. Abbildung 5.4 zeigt den strukturellen Aufbau des ORB [OMG99b]. Dabei ist zu beachten, dass es sich beim ORB um eine logische Komponente handelt, die nicht zwangsläufig durch ein einzelnes Objekt realisiert sein muss. Die Funktionalität des ORB kann sowohl in den Client- und Serverimplementierungen integriert sein, als auch durch spezielle Betriebssystem- oder Bibliotheksfunktionen erbracht werden.

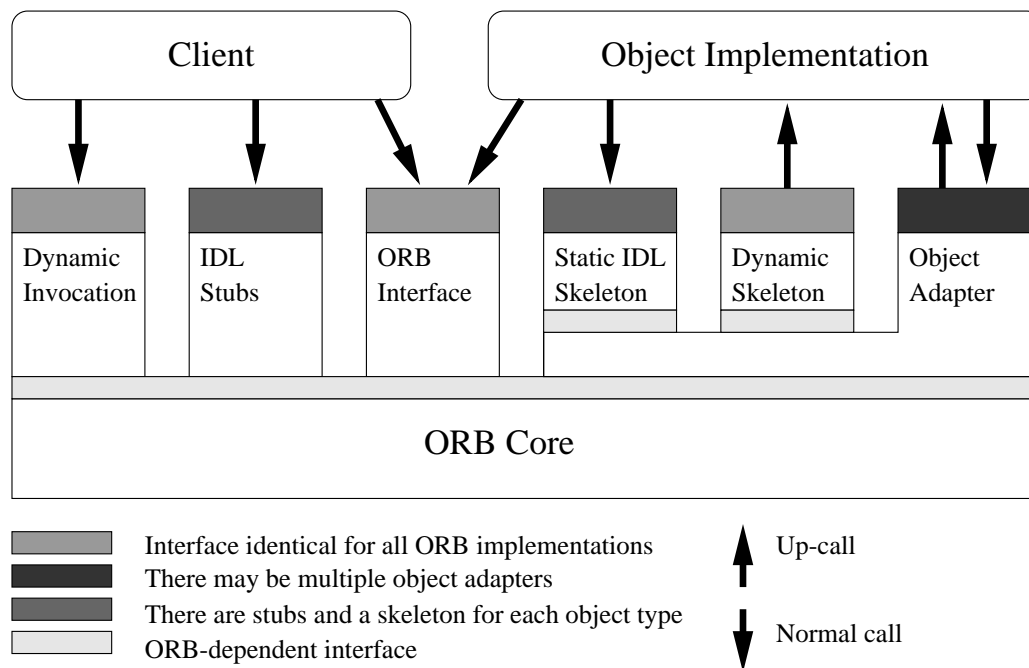


Abbildung 5.4: Die Struktur des Object Request Brokers

Zur Erfüllung seiner Aufgaben verfügt der ORB über eine Menge unterschiedlicher Schnittstellen. Der IDL-Compiler generiert aus der IDL-Schnittstellendefinition der einzelnen Objekte IDL-Stubs für die Client-Komponenten und IDL-Skeletons für die Server-Komponenten. Der erzeugte IDL-Skeleton dient als Basis für die Objektimplementierung, der IDL-Stub dient dem Client als Kommunikationsschnittstelle, über die er Requests an ein Objekt verschicken kann. Für den Aufruf eines Objekts bieten sich dem Client zwei

Möglichkeiten: Die eine ist der asynchrone Aufruf, bei dem der Client so lange warten muss, bis eine Antwort zurück kommt. Hierfür wird die genau einmalige Ausführung garantiert (*Exactly-Once-Semantik*). Die andere Möglichkeit ist der asynchrone Aufruf, bei dem der Client seine Aufgabe weiterführen kann, direkt nachdem er der Aufruf abgesetzt hat. In diesem Fall darf der Client aber keine Antwort erwarten. Es gilt die *Best-Effort-Semantik*; dabei können Nachrichten verloren gehen, ohne dass ihr Verlust bemerkt wird.

Mit der Verwendung Compiler-generierter IDL-Stubs ist der Nachteil verbunden, dass die Objektschnittstelle zum Erstellungszeitpunkt des Client vollständig bekannt sein muss und somit statisch festgelegt ist. Dies erweist sich für eine Gruppe von Anwendungen, z. B. Trader als zu inflexibel. Ähnliche Probleme ergeben sich bei einer nachträglichen Änderung der Schnittstellendefinition. Die erforderliche erneute Compilierung führt zu geänderten IDL-Stubs und IDL-Skeletons, was wiederum eine Anpassung der betroffenen Objektimplementierungen und aller Clients notwendig macht. Um solche Probleme vermeiden zu können, bietet der ORB eine Möglichkeit um Methodenaufrufe dynamisch, d. h. zur Laufzeit des Programms zu generieren. Dies wird mittels des *Dynamic Invocation Interface* (DII) realisiert.

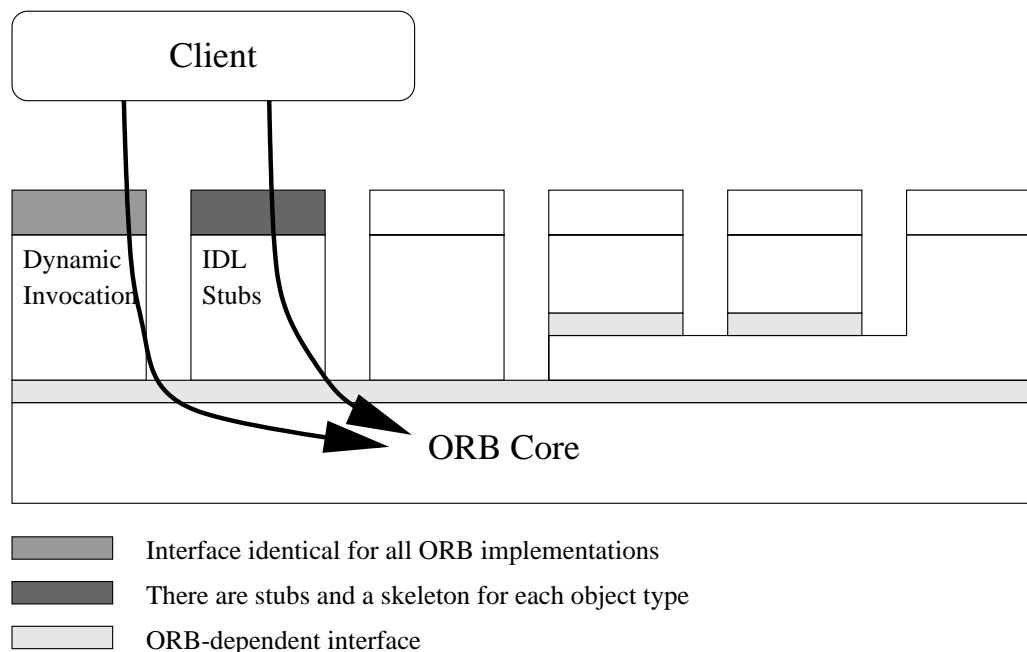


Abbildung 5.5: Zugriff auf die Objektimplementierung über DII und IDL-Stub

Das DII erlaubt dem Client, durch Zugriff auf das *Interface Repository*, Methoden und Parameter eines Objekts zur Laufzeit zu ermitteln. Das Interface Re-

pository ist sozusagen eine Art Datenbank für IDL-Schnittstellenbeschreibungen. Diese Vorgehensweise bietet den Vorteil, dass der Client zum Zeitpunkt seiner Compilierung über keine Kenntnisse bezüglich der Schnittstellen der Objektimplementierung verfügen muss. Änderungen der Schnittstellenbeschreibung erfordern somit nicht die Anpassung des Clients. Beim Aufruf eines Dienstes ist für die angesprochene Objektimplementierung nicht erkennbar, ob der eingegangene Request über DII oder IDL-Stub initiiert wurde. In Abbildung 5.5 ist ein Objektaufruf durch den Client dargestellt [OMG99b].

Eine ähnliche Flexibilität wird für die Server-Komponente durch das *Dynamic Skeleton Interface* (DSI) erreicht. Das DSI erlaubt die Implementierung eines Servers, dessen Methoden zum Zeitpunkt der Compilierung nicht bekannt sind. Dies ist z. B. dann der Fall, wenn der Server einen Interpreter darstellt, der erst zur Laufzeit den Programmcode einliest und auf diese Weise erst über die bereitzustellenden Objektschnittstellen und den Implementierungscode erfährt. Die Methodenaufrufe eines Objekts sind in Abbildung 5.6 dargestellt [OMG99b]. Auch hier ist der Zugriff transparent; das Objekt hat keine Kenntnis über die Art des Aufrufs.

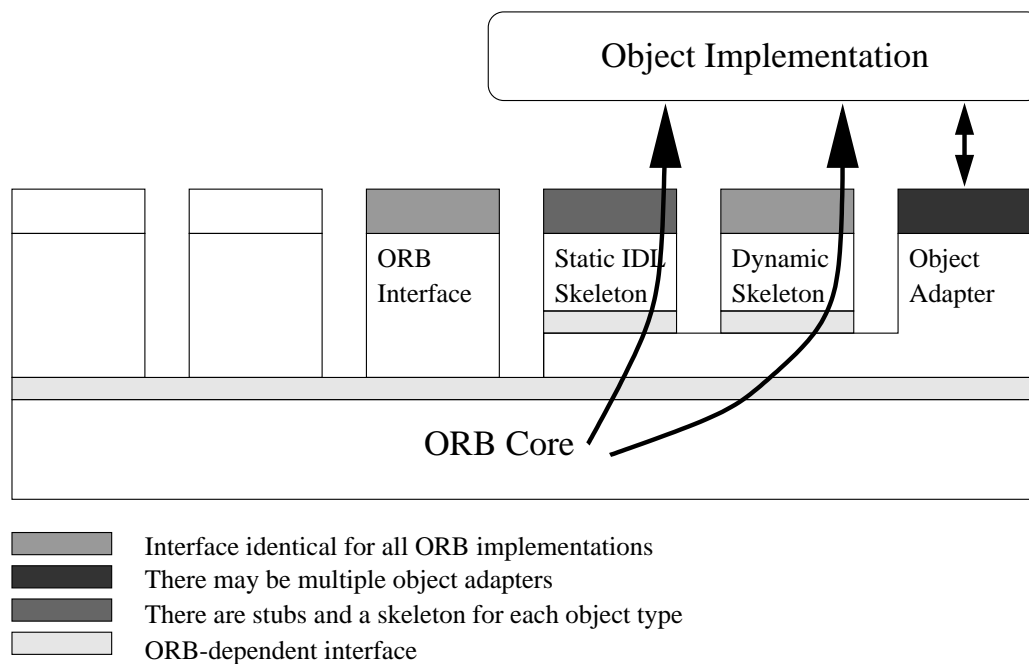


Abbildung 5.6: Aufruf der Objektimplementierung über DSI und IDL-Skeleton

Wie in Abbildung 5.6 zu sehen ist, stellt der ORB neben den bereits dargestellten noch zwei weitere Schnittstellen zur Verfügung: die Schnittstelle zum ORB

selbst (*ORB Interface*) und den *Object Adapter* (OA). Es handelt sich bei diesen Schnittstellen um sogenannte Pseudo-Schnittstellen, weil sie zwar in der IDL spezifiziert sind, aber keine CORBA-Objekte im Sinne des OMG-Objektmodells beschreiben.

Das ORB Interface stellt allgemeine Dienste zur Verfügung, die sowohl für den Client als auch die Objektimplementierung von Nutzen sind. Diese Dienste beinhalten unter anderem Operationen für die Konvertierung einer Objektreferenz in einen String, die entsprechende Rückkonvertierung vom String in eine Objektreferenz und Operationen zur Ermittlung von Basisdiensten, wie beispielsweise dem Naming Service. Die Dienste des ORB Interface sind weder von der Schnittstelle des Objekts noch vom verwendeten Object Adapter abhängig. Da der Großteil der Funktionalität des ORB durch die anderen Schnittstellen (DII, DSI, OA, Stubs, Skeletons) realisiert wird, gibt es nur wenige Dienste, die durch das ORB Interface angeboten werden.

Der Object Adapter ist das Bindeglied zwischen ORB und Objektimplementierung. Er steht somit nur den Objektimplementierungen zur Verfügung. Vom Client eingegangene Requests an den ORB werden durch den OA auf die gewünschten Methoden des Zielobjekts abgebildet. Dies erlaubt gleichzeitig auch die Überprüfung der Zugriffsrechte des Client an das aufzurufende Objekt. Da die Anforderungen an den ORB je nach Einsatzgebiet stark unterschiedlich sein können, wurden im CORBA-Standard mehrere, auf jeweils spezielle Nutzungsszenarien abgestimmte Arten von OAs vorgesehen. Dennoch sollte ein OA stets ein möglichst großes Anwendungsgebiet unterstützen, damit die Anzahl verschiedener OAs möglichst klein bleibt. Denn der Wechsel zu einem anderen OA erfordert stets eine erneute Compilierung der Objektimplementation. Innerhalb des OMG-Standards werden zwei Object Adapter spezifiziert: Der *Basic Object Adapter* (BOA) wurde für Anwendungen konzipiert, in denen eine relativ kleine Anzahl von Objekten, die über mehrere Rechner verteilt sind, miteinander kommunizieren. Dem gegenüber steht der *Object-Oriented Database Adapter* (OODBA). Er wurde für die Zusammenarbeit mit objektorientierten Datenbanken (OODB) vorgesehen. Der OODBA muss folglich mit sehr umfangreichen Objektmengen umgehen können. Zu seinen Aufgaben gehören das Zurückschreiben des Objektzustands nach Zustandsänderungen und die automatische Wiederherstellung des Objektzustands aus der Datenbank nach einem Neustart.

In der Version 2.0 des CORBA-Standards wurde der Aufbau der Interoperable Object References spezifiziert, um die Kommunikation von Object Request Brokern verschiedener Anbieter zu ermöglichen. Damit wurde der Basic Object Adapter durch den *Portable Object Adapter* (POA) abgelöst, welcher *Interoperabilität* auch über ORB-Grenzen hinaus ermöglicht.

### 5.3.3 Interoperabilität

Interoperabilität bezeichnet man im Allgemeinen als die Fähigkeit verschiedener Hard- und Softwarekomponenten, miteinander zu interagieren. Im Kontext von CORBA bezeichnet Interoperabilität die Fähigkeit eines Clients auf ORB A in OMG IDL spezifizierte Operationen eines Objekts auf ORB B aufzurufen, wobei ORB A und ORB B völlig unabhängig voneinander entwickelt worden sein können. D. h. Aufrufe zwischen Client und Objektimplementierung müssen davon unabhängig sein, ob sie auf demselben oder auf völlig verschiedenen ORBs ausgeführt werden, ohne dabei grundlegende Änderungen existierender Anwendungen zu erfordern. Interoperabilität kann als Erweiterung des Transparenzbegriffs angesehen werden, derart dass unterschiedliche ORB-Implementierungen mit eingeschlossen werden.

Wegen der Vielzahl unterschiedlicher ORB-Implementierungen sind mehrere unterschiedliche Ansätze notwendig, um Interoperabilität zu erreichen. Das CORBA-Interoperabilitätsmodell, das von der OMG erstmals im CORBA 2.0 Standard spezifiziert wurde, umfaßt deshalb folgende Elemente:

- **ORB Interoperability Architecture.**  
Festlegungen über die Interoperabilitätsarchitektur des ORB. Der einheitliche Aufbau der Objektreferenzen, der IORs ist eine wichtige Voraussetzung für die Kommunikation über Rechner- und Plattformgrenzen hinweg.
- **Inter-ORB Bridge Support.**  
Unterstützung für Inter-ORB Brücken. Brücken (engl. bridges) und Halb-Brücken sind Dienste, welche die transparente Kommunikation über ein Transportprotokoll hinweg ermöglichen. Eine Halb-Brücke realisiert nur die Umwandlung von oder zu einem Protokoll, während eine Voll-Brücke beide Umwandlungen beinhaltet, also auch durch zwei passende Halb-Brücken realisiert sein kann. Der Inter-ORB Bridge Support kann verwendet werden, um Interoperabilität mit nicht-CORBA-Systemen z. B. Microsofts COM zu ermöglichen.
- **General Inter-ORB Protocol (GIOP) & Internet Inter-ORB Protocol (IIOP).**  
Allgemeine- und Internet-Inter-ORB-Protokolle. Sie dienen der Beschreibung von Protokollen zur Kommunikation zwischen verschiedenen ORBs. GIOP und IIOP werden von jeder standardkonformen CORBA-Plattform bereitgestellt.

Das General Inter-ORB Protocol wurde speziell für die „ORB zu ORB“-Interaktion entworfen. Es beschreibt die allgemeine Kommunikation zwischen

Client und Server anhand der Spezifikation von Transfer-Syntax und Nachrichtenformaten für die Kommunikation zwischen den ORBs. Da es direkt auf allen verbindungsorientierten Protokollen aufbaut, ist es nicht auf höhere RPC-Mechanismen angewiesen. GIOP spezifiziert die gemeinsamen Protokollelemente unterschiedlicher Transportprotokolle und kann somit auf eine Menge unterschiedlicher Transportprotokolle abgebildet werden. Ähnlich wie eine IDL-Beschreibung auf eine Programmiersprache, muss GIOP zuerst auf ein Protokoll abgebildet werden, bevor es anwendbar ist.

Das Internet Inter-ORB Protocol stellt eine Spezialisierung des abstrakten GIOP dar, indem es spezifiziert, wie GIOP-Nachrichten über TCP/IP-Verbindungen ausgetauscht werden können. Durch IIOP wird ein standardisiertes Interoperabilitätsprotokoll für das Internet spezifiziert. Es kann benutzt werden, um mehrere ORBs miteinander zu verbinden, indem Halb-Brücken geschaffen werden, die auf IIOP basierend miteinander kommunizieren (siehe Abbildung 5.7 [OMG99b]). IIOP kann aber auch genauso gut für den ORB-internen Nachrichtenaustausch oder einfach als Kommunikationsprotokoll zwischen zwei Halb-Brücken verwendet werden.

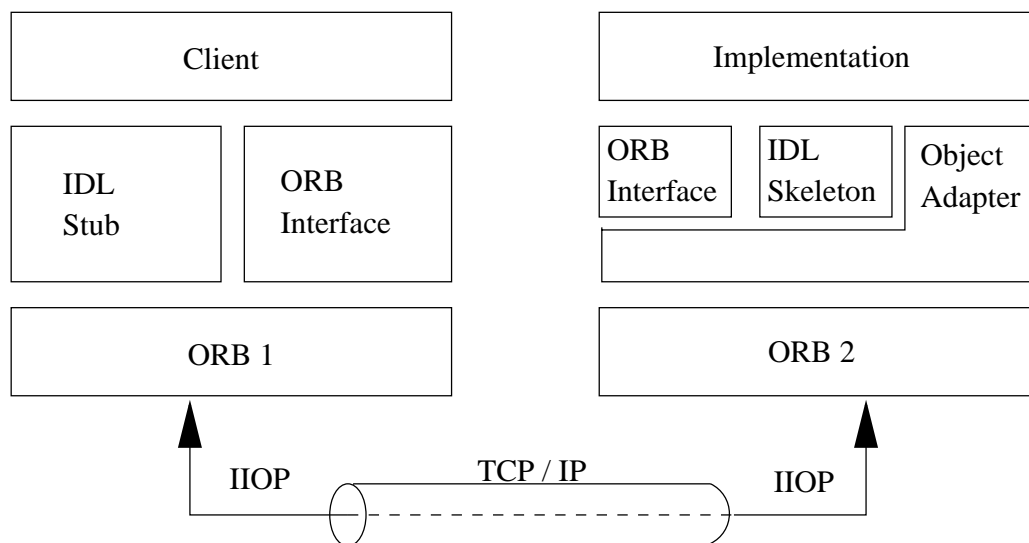


Abbildung 5.7: Internet Inter-ORB Protocol

Erwähnenswert wäre noch, dass es für die einfache Interoperation mit Benutzerseiten, welche bereits eine bestimmte Netzwerk- oder Verteilungsinfrastruktur etabliert haben, das *Environment Specific Inter-ORB Protocol* (ESIOP) gibt. ESIOPs können für spezielle Umgebungen angepasst werden. Es muss dabei aber darauf geachtet werden, dass die Konformität zu GIOP und dem CORBA-

Interoperabilitätsmodell bewahrt bleibt, damit auf einfache Weise Brücken implementiert werden können.

## 5.4 Erstellung einer CORBA-Applikation mit ORBacus

In IDL definierte Schnittstellen müssen zuerst durch einen IDL-Compiler auf eine reale Programmiersprache abgebildet werden, bevor sie in einem Programm verwendet werden können. Der IDL-Compiler erzeugt ausgehend von der gegebenen IDL-Definition ein Programmgerüst (*Skeleton*), welches als Grundlage für die eigentliche Abbildung der Attribute und Methoden auf die entsprechenden C++-Konstrukte dient. Die von CORBA bereitgestellten, implementierungsunabhängigen Mechanismen ermöglichen die Kommunikation des Objekts mit seiner Umwelt. Die Implementierung des Objekts beschränkt sich dadurch auf die Bereitstellung der spezifizierten Funktionalität. Zusätzlich zum Skeleton erzeugt der IDL-Compiler einen objektspezifischen Funktionsrumpf (*Stub*). Der Stub ermöglicht es anderen Objekten, die an der Objektschnittstelle angebotenen Methoden aufzurufen, ohne dass dabei für den Client ersichtlich wird, dass es sich um Methodenaufrufe an entfernten Objekten handelt. Der Unterschied zu lokalen Methodenaufrufen wird für den Client erst in Bezug auf Antwortzeiten und bei erforderlicher Fehlerbehandlung sichtbar.

Genaue Implementierungsdetails von ORB und IDL-Compiler sind durch die CORBA-Spezifikation nicht vorgeschrieben. Deshalb gibt es Unterschiede bezüglich der Realisierung eines CORBA-Programms, je nach Anbieter des verwendeten CORBA-Systems. An einem kurzen, zum besseren Verständnis sehr allgemein gehaltenen Beispiel soll die prinzipielle Vorgehensweise der Erstellung einer Client-/Server-Applikation mit dem CORBA-System ORBacus 3.3.1 und der Programmiersprache C++, welche im Projekt CASSY vorwiegend zum Einsatz kommen, vorgestellt werden.

Abbildung 5.8 zeigt, welche Dateien durch den IDL-Compiler aus der Schnittstellendefinition generiert werden und wie Dateien, die diese Schnittstelle benutzen, mit diesen im Zusammenhang stehen. Freilich können in einer Schnittstellenbeschreibungsdatei oder einer Quellcode-Datei mehrere Definitionen bzw. Klassen enthalten sein. Es liegt im Ermessen des Programmierers, diese in geeigneter Weise zusammenzufassen. Allerdings ist es für die Übersichtlichkeit, das Verständnis und folglich für die Fehlervermeidung von Vorteil, wenn in jeder Datei nur eine Klasse implementiert wird, Dateinamen den Klassennamen entsprechen und die Klassenstruktur sich im Wesentlichen auf die Dateistruktur übertragen lässt.

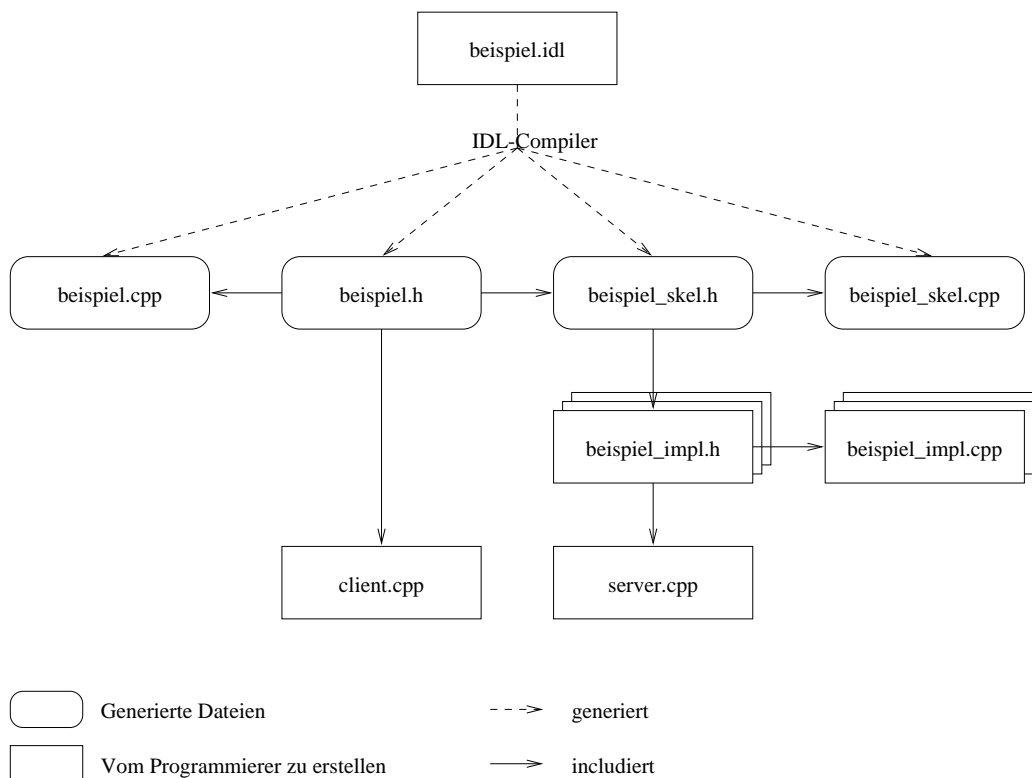


Abbildung 5.8: Erstellen einer CORBA-Applikation mit ORBacus

Das hier vorgestellte Beispiel geht von einer IDL-Definition mit dem Namen „beispiel“ aus. Wie zu sehen ist, werden die Dateien durch ihren Suffix unterschieden. In C++ werden Klassendeklarationen und allgemeine Definitionen in Header-Dateien („.h“) beschrieben; Quellcode-Dateien („.cpp“) beinhalten den Programmcode der Klasse.

`beispiel.h` ist die Definitionsdatei für den Client-Stub und `beispiel.cpp` enthält den Programmcode dafür. Entsprechend steht für die Server-Seite ein Skeleton mit der Definitionsdatei `beispiel_skel.h` und der Datei `beispiel_skel.cpp` mit dem Programmcode für das Skeleton zur Verfügung. Wie man in der Abbildung sieht, verwendet das Skeleton zusätzlich auch noch die Stub-Dateien. Die Objektimplementierung wird in `beispiel_impl.h` definiert und `beispiel_impl.cpp` realisiert. Im Gegensatz zu den generischen Stub- und Skeleton-Dateien müssen diese Dateien vom Programmierer erzeugt werden. Wie die Abbildung schon andeutet, kann die Objektimplementierung über mehrere Dateien verteilt sein. Die Dateinamen können beliebig gewählt werden, es ist jedoch üblich den Suffix „\_impl.h“ bzw. „\_impl.cpp“ zu verwenden, damit Dateien der Objektimplementierung einfacher von Stub- oder Skeleton-Dateien

zu unterscheiden sind.

Die Client-Applikation (`client.cpp`) stützt sich ausschließlich auf die in IDL spezifizierte Schnittstelle. Sie ist dadurch relativ einfach zu implementieren und lässt sich sehr flexibel an unterschiedliche Plattformen anpassen, indem das Programm für die entsprechende Plattform neu compiliert wird. Im Programmcode sind Änderungen nur dann erforderlich, wenn die IDL-Schnittstellendefinition geändert wird. Nachdem die Client-Applikation den ORB initialisiert hat, kann sie sofort auf die in der IDL-Schnittstelle spezifizierte Funktionalität zugreifen, ohne sich darum kümmern zu müssen, ob die Zugriffe lokal oder entfernt ausgeführt werden.

Der Aufbau der Server-Komponente (`server.cpp`) erfordert für gewöhnlich etwas mehr Aufwand als bei der Client-Komponente. Während die Funktionalität der Objektimplementierung bereits in den „`impl.cpp`“-Dateien vorhanden ist, bietet der *Server* diese Dienste durch einen laufenden Prozess (oder durch *Threads*) nach außen an. Der prinzipielle Aufbau eines CORBA-Servers sieht wie folgt aus:

1. Initialisierung von ORB und BOA
2. Erzeugen der Objektinstanz
3. Registrieren der Instanz beim CORBA Naming Service
4. Objektreferenz beim ORB verfügbar machen
5. Bereitmelden des Servers (Event Loop)

Das Erstellen einer CORBA-Applikation ist nicht immer so einfach, wie man anhand dieses kleinen Beispiels vermuten könnte. IDL-Schnittstellen können unter Umständen sehr umfangreich werden. Wenn ein Programm evtl. auch noch mit mehreren IDL-Schnittstellen interagieren muss und weiteren Objektimplementierungen gegenüber gleichzeitig als Client auftritt, können zwischen Dateien und Beziehungen bei Operationsaufrufen äußerst komplexe Abhängigkeiten entstehen. Kapitel 6 zeigt Lösungswege, wie diese Komplexität bewältigt werden kann.



# Kapitel 6

## Realisierung des Wrappers

Nicht alle Applikationen können direkt über CORBA-Aufrufe gesteuert werden. Der *Wrapping*-Mechanismus liefert einen Lösungsansatz zur Applikationsintegration (engl. *to wrap* = ein-/verpacken). Programme, die a priori nicht via CORBA aufgerufen werden können, müssen zum Zweck des Aufrufs mit Hilfe eines Wrapping-Mechanismus mit einer IDL-Schnittstelle versehen werden.

Ein *Wrapper* für CORBA dient der Einbindung eines bereits bestehenden Produkts ohne IDL-Schnittstelle in die CORBA-Umgebung. Der Wrapper übernimmt dazu die anfallenden Übersetzungsfunktionen, um das Produkt über seine API an das CORBA-System zu binden. Genau dieses Verfahren wurde bei der Implementierung der Workflow Management Facility auch angewendet, um die Funktionalität von Flowmark für CASSY zur Verfügung zu stellen. Dabei wurde eine standardisierte Schnittstelle gewählt (jFlow), um die Möglichkeit zu schaffen, auch andere Workflow-Management-Systeme auf die gleiche Weise integrieren zu können. Der hier vorgestellte Wrapper setzt auf dieser Schnittstelle der WMF auf und integriert sie durch geeignete Übersetzungsfunktionen in das bestehende Aktivitätenmodell von CASSY. In Abbildung 6.1 ist das Prinzip der Integration schematisch dargestellt. Folglich handelt es sich hier also um zwei aufeinander aufbauende Wrapper. Im Folgenden sei zum Zwecke der Unterscheidung mit „Wrapper“ stets die Aufgabe dieser Diplomarbeit gemeint und die Workflow Management Facility sei als solche bezeichnet.

Für die Realisierung ihrer Funktionen verwendet die WMF sogenannte *Proxy*-Mechanismen. Um ein Programm vor einem Objekt zu verbergen, wird es zur Unterstützung von entfernten Aufrufen in ein Proxy- und ein Implementierungsobjekt spezialisiert. Das Implementierungsobjekt befindet sich auf der Maschine, auf der das zugehörige Programm ausführbar ist, das Proxy-Objekt ist Teil der Arbeitsliste eines Benutzers. Somit ist eine Verteilung bezüglich der Programmausführung möglich. Diese Proxy-Objekte sind es, auf die der Wrapper zugreift und über das `WorkflowActivity`-Interface in CASSY verfügbar macht.

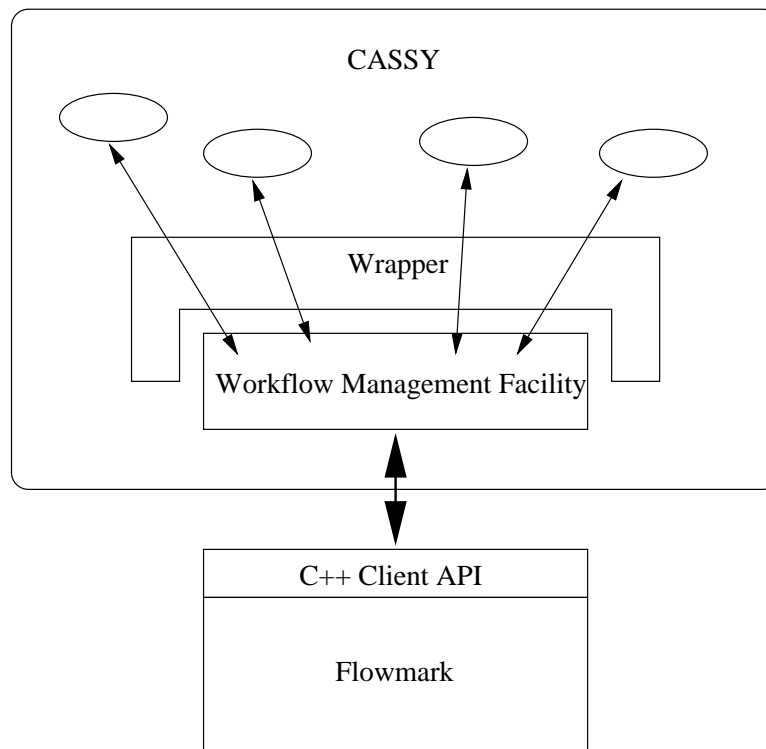


Abbildung 6.1: Integration des Wrappers in die Systemstruktur von CASSY

Realisierungsgrundlage dieser Arbeit ist die OO-Middleware CORBA unter Verwendung von ORBacus Version 3.3.1 [OOC00]. Die Implementierung wurde in C++ programmiert, wobei die Entwicklungsumgebung Microsoft Visual C++ 6.0 verwendet wurde. Als Systemplattform kam Microsoft Windows NT 4.0 mit dem Service Pack 6 zum Einsatz und für die Modellierung der Klassendiagramme wurde Rational Rose verwendet.

## 6.1 Integration in das Aktivitätenmodell

Wie in Abbildung 6.2 zu sehen ist, wurde die `WorkflowActivity`-Klasse ebenso wie die `DesignflowActivity`-, die `GroupwareActivity`- und die `ToplevelActivity`-Klasse als Spezialisierung von der `BasicActivity`-Klasse abgeleitet. Dies geht aus den Eigenschaften der `BasicActivity` hervor, welche allgemeine Grundfunktionalitäten für alle Arten von Aktivitäten zur Verfügung stellt. `PrimitiveActivity` stellt hierbei eine Ausnahme dar, da diese eine nicht weiter spezifizierbare Aktivität repräsentiert, die auch nicht weiter untergliedert werden kann und deshalb direkt

von der `Activity`-Klasse - welche die minimale Funktionalität für Aktivitäten definiert - abgeleitet wurde. Die Funktionalität der `ToplevelActivity` ist im Wesentlichen identisch mit der (Grund-)Funktionalität der `BasicActivity`, allerdings mit dem Unterschied, dass sie als die oberste Aktivität keine Eltern-Aktivität haben kann und über zusätzliche Funktionen für die allgemeine Verwaltung aller Aktivitäten verfügt.

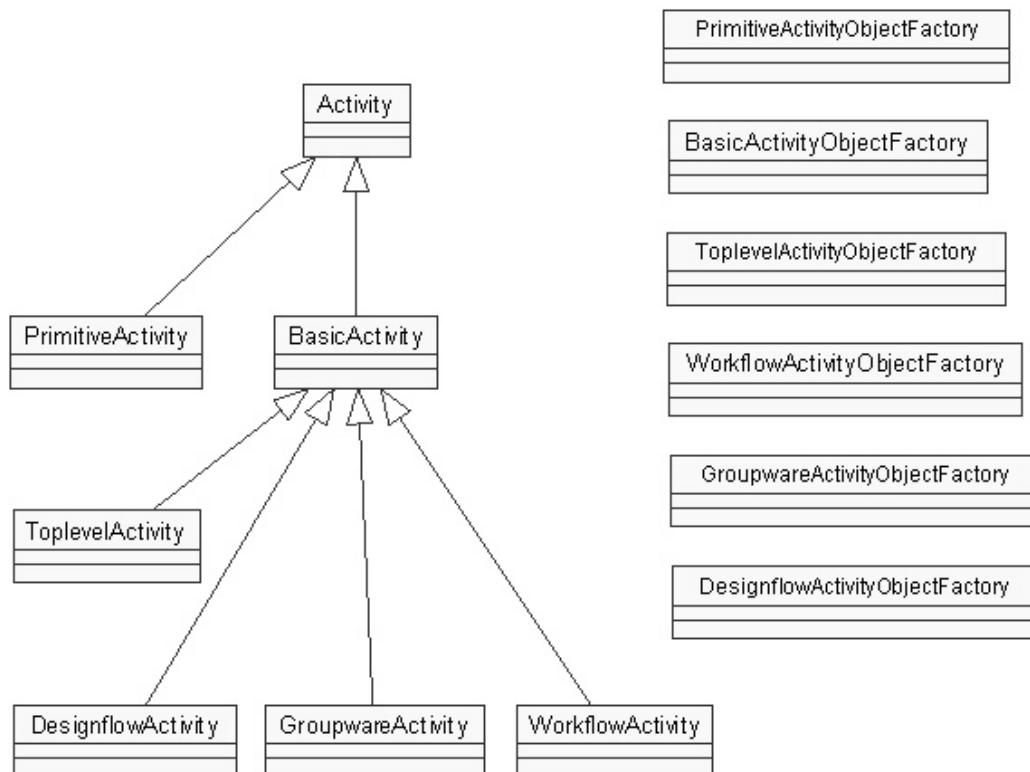


Abbildung 6.2: Klassendiagramm der ActivityManagement-Komponente

Für jede der Aktivitäts-Klassen - mit Ausnahme der abstrakten `Activity`-Klasse, welche durch `PrimitiveActivity` konkretisiert wird - existiert eine *Factory*, um die Erzeugung neuer CORBA-Objekte durch den Client zu ermöglichen.

### 6.1.1 Beschreibung der `WorkflowActivity`-Klasse

Die wesentliche Aufgabe der `WorkflowActivity`-Klasse besteht darin, die Funktionen der Superklassen `Activity` und `BasicActivity` in geeigneter Weise an die `Workflow Management Facility` weiterzuleiten. Dies sind insbesondere **Methoden der Activity-Klasse:**

Zustandssteuerung:

getState(), init(), start(), stop(), reset(), resume(), cancel() und terminate()

Benutzerverwaltung:

getActor(), setActor(), getName() und setName(). Diese Funktionalität wird vorerst noch nicht implementiert, da im jFlow-Modell keine explizite Unterstützung einer Benutzerverwaltung vorgesehen ist.

Identität:

getID() und setID.

Ressourcenverwaltung:

getResources(), getResource(), addResource(), removeResource() und testResource().

Persistenz:

saveActivity() und loadActivity().

Hierarchieverwaltung für Eltern-Aktivitäten:

getParentActivity() und setParentActivity().

### **Methoden der BasicActivity-Klasse:**

Hierarchieverwaltung für Kind-Aktivitäten:

getChildActivities(), getChildActivity(), testChildActivity(), removeChildActivity() und addChildActivity().

Ergänzt werden diese Methoden durch zusätzliche Workflow-spezifische Methoden aus der **WorkflowActivity-Klasse**:

Nachrichtenaustausch:

sendMsg() und receiveMsg().

Kontrollfluss:

getDefinition() und setDefinition().

Auf die genauen Details soll hier nicht weiter eingegangen werden, da diese auch aus der IDL-Spezifikation ersichtlich sind. Wichtig sind die verschiedenen Funktionsgruppen, welche durch die Aktivitäten beherrscht werden. Wie man auch sieht, wird der größte Teil der Funktionalität bereits in der abstrakten Activity-Klasse definiert. Dies ist besonders interessant, da diese Methoden für alle Aktivitäten (sowohl Groupware- als auch Workflow-Management- oder Designflow-Aktivitäten) zur Verfügung stehen.

### 6.1.2 Die Workflow Activity Object Factory

Ein *Factory*-Objekt ermöglicht den Zugriff auf mehrere zusätzliche Objektinstanzen einer Objektimplementierung. Die Objektreferenz des Factory-Objekts wird an einer allgemein bekannten Stelle hinterlegt. Client-Programme wissen, dass sie nur diese Objektreferenz erhalten müssen, um Zugriff auf die Objektreferenzen weiterer Objekte im System zu erhalten. Dies reduziert die Anzahl der zu veröffentlichen Objektreferenzen und ermöglicht eine verteilte Ausführung. Ohne Verwendung von Factories müßte die zum Objekt zugehörige Implementierung in dem Server sein, in dem die Erzeugung des Objekts ausgelöst wurde. Die Erzeugung eines neuen Objekts durch ein Client-Programm wäre unmöglich, genauso wie die über mehrere Rechner verteilte Ausführung.

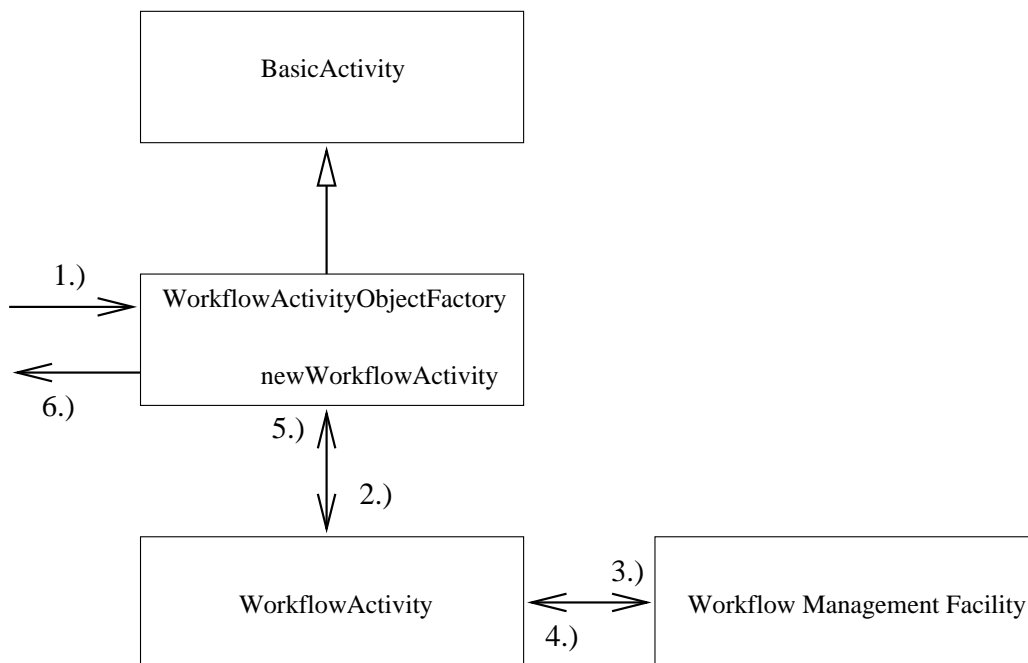


Abbildung 6.3: Instanziierung einer Workflow-Aktivität

Die Instanziierung eines `WorkflowActivity`-Objekts geschieht mittels der `WorkflowActivityObjectFactory` auf folgende Weise (vgl. Abbildung 6.3):

1. Die Factory wird benachrichtigt, ein neues `WorkflowActivity`-Objekt zu erzeugen.
2. Die Factory erzeugt ein neues `WorkflowActivity`-Objekt, welches wiederum

3. die Workflow Management Facility aufruft, um ein Session-Objekt zu instanziiieren. Details zum Flowmark-Server-Login und der Session-Verwaltung sind [Hei98] Seite 71 ff. zu entnehmen.
4. Nach dem erfolgreichen Login am Flowmark-Server
5. wird die Nachricht an die Factory weitergeleitet.
6. Daraufhin gibt die Factory die Objektreferenz an den Aufrufer zurück und das Objekt kann verwendet werden.

## 6.2 Implementierung

Die Implementierungsstruktur des Wrappers auf Dateiebene ist in Abbildung 6.4 dargestellt. Wie man sieht, handelt es sich um eine Spezialisierung der allgemein gehaltenen Abbildung 5.8, wobei Dateibezeichnungen und die Dateien der Objektimplementierung entsprechend angepasst wurden. Die Aufteilung in drei logische Gruppen *Schnittstelle*, *Implementierung* und *Ausführung* ist wie folgt begründet:

1. Zur Schnittstelle gehören die IDL-Schnittstellendefinition des Aktivitätenmanagements und die daraus generierten Programmdateien. In der Schnittstellendefinition (`ActivityInterface.idl`) sind die Beschreibungen aller Aktivitäten zusammengefasst.
2. Die Implementierung umfasst die vom Programmierer entwickelten Objektimplementierungen. Wie man sieht, wurden die Include-Beziehungen der Dateien an die Klassenhierarchie angepasst.
3. Ausgeführt werden die Funktionen der Objektimplementierung durch ein Client- oder Server-Programm. Diese beinhalten eine „`main()`“-Funktion, welche die Objektimplementierung direkt oder indirekt aufruft. Aus diesen Dateien wird nach der Compilierung und dem Binden mit den inkludierten Dateien ein lauffähiges Programm erzeugt. In diesem Fall der Wrapper und ein Testprogramm, das die Funktionen aufrufen kann.

Ursprünglich war die Objektimplementierung des gesamten Activity-Managements in einer einzigen Datei untergebracht (`ActivityInterface_impl.cpp`). Dem Umfang der Aktivitätsklassen entsprechend war die Datei sehr groß und zudem leider auch nicht dokumentiert. Durch die Aufteilung der monolithischen `ActivityInterface`-Implementierung in mehrere separate Bestandteile entsprechend jeder Klasse wurde die Übersichtlichkeit der Programmstruktur erhöht und zudem die Möglichkeit geschaffen, weitere Komponenten in das Aktivitätenmanagement zu integrieren, ohne die Anpassung fremder

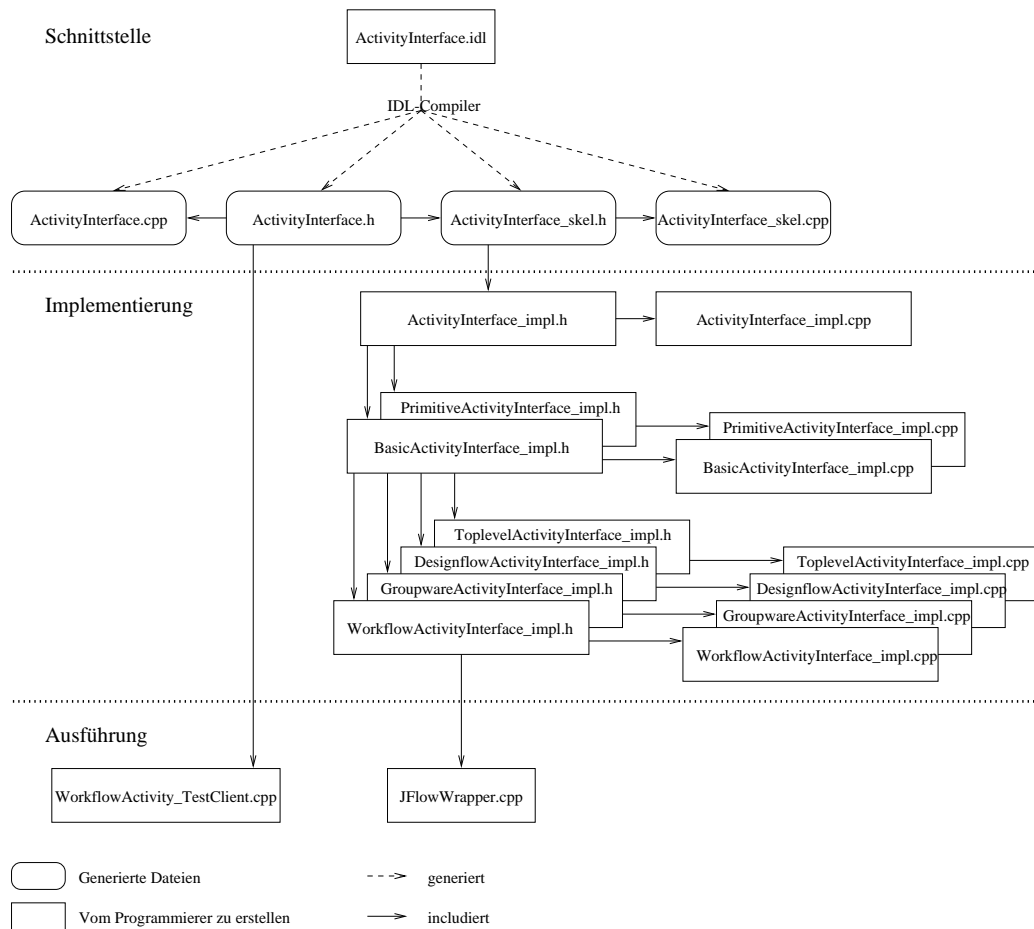


Abbildung 6.4: Dateistruktur des Aktivitätenmanagements

Komponenten zu erfordern. Zur Erweiterung der Schnittstelle muss die IDL-Datei selbstverständlich neu kompiliert werden. Jedoch können neue Spezialisierungen der `BasicActivity` einfach realisiert werden, indem die Klasse in einer neuen Datei realisiert wird, welche die Datei `BasicActivity_impl.h` inkludiert. D. h. keine der anderen Implementierungsdateien muss dafür geändert werden.

Da die Bezeichnung „Server“ je nach Kontext verschiedenen Bedeutungen zugeordnet ist, z.B. als dedizierter Rechner, auf dem *Server*-Applikationen laufen oder als eine Software, welche verschiedene Funktionen für andere Programme (Clients) anbietet, sei hier darauf hingewiesen, dass im Rahmen dieser Diplomarbeit und insbesondere im Zusammenhang mit CORBA mit *Server* stets eine Software gemeint ist. Ein Server kann gegenüber anderen Programmen allerdings auch als Client auftreten, wenn er deren Dienste in Anspruch nimmt. Die jeweilige Bedeutung geht allerdings meistens aus dem Kontext hervor, ansonsten wird ex-

plizit darauf hingewiesen. Beispielsweise tritt der Wrapper gegenüber der Workflow Management Facility als Client auf, weil er deren Dienste in Anspruch nimmt und verhält sich gegenüber Anforderungen seitens des Activity-Managements als Server.

## 6.3 Probleme bei der Implementierung

Unkalkulierbare Faktoren wie Probleme beim Installieren von lauffähigen CASSY-Komponenten, teilweise nicht verfügbarer Funktionalität in den Komponenten und zu Beginn der Arbeit mangels genauer Kenntnisse über die Implementierung der vorausgegangenen Arbeiten nicht abschätzbarer Zeitaufwand für die Analyse sind Faktoren, die bei einer Implementierungsaufgabe zu schlecht vorausplanbaren Zeitplänen führen können. Nachfolgend sind deshalb Vorschläge aufgelistet, welche dazu führen sollen, unnötigen Arbeitsaufwand zu vermeiden.

### 6.3.1 Vorschläge zur Verbesserung der Softwarequalität

Abbildung 6.5 entstand als Ergebnis der Untersuchung der Workflow Management Facility um deren Zusammenhänge zu verstehen. Zunächst sieht die Grafik sehr unübersichtlich und chaotisch aus. Sie zeigt die Include-Abhängigkeiten der Dateien, aus denen die WMF besteht. Wegen anfänglicher Probleme den WMF-Quellcode zu compilieren, schien eine genauere Betrachtung der Abhängigkeiten sinnvoll. Diese Abhängigkeiten lassen sich durch das Beachten weniger, einfacher Richtlinien sehr stark reduzieren, wodurch die Übersichtlichkeit und das Verständnis der Zusammenhänge verbessert werden.

- In C++ existiert für eine Datei mit Programmcode („`.cpp`“-Datei) stets auch eine zugehörige *Header*-Datei („`.h`“-Datei), welche allgemeine Deklarationen und die Prozedurköpfe enthält. Dadurch ist die Verwendung der Funktionen schon beschrieben, ohne dass der gesamte Programmcode gelesen werden muss. Es ist Sinn und Zweck der Header-Datei, die Funktion des Programmcodes zu beschreiben. Deshalb sollten sie auch alle benötigten „`#include`“-Direktiven beinhalten. Es ist nicht sinnvoll, dass die „`.cpp`“-Datei außer für die zugehörige Header-Datei weitere „`#include`“-Direktiven beinhaltet.
- Wenn in einer Datei stets nur eine Klasse implementiert wird, die Zuordnung von Dateinamen zu Klassennamen einfach und eindeutig ist und die Include-Beziehungen der Dateien aus der Klassenstruktur abgeleitet wurden, ergibt sich zwangsläufig eine intuitive und dadurch weniger fehleranfällige Beziehungsstruktur.

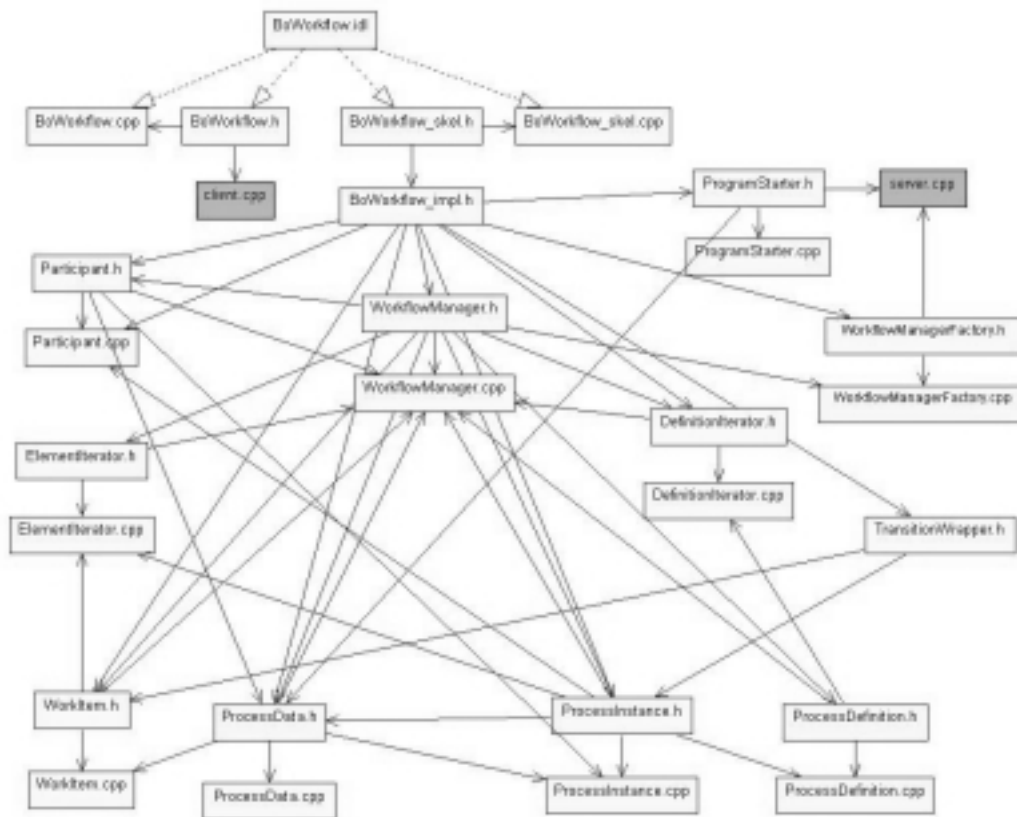


Abbildung 6.5: Dateien der Workflow Management Facility

- Mehrfaches includieren derselben Datei lässt sich auf diese Weise auch leichter vermeiden und erhöht die Übersichtlichkeit zusätzlich.

Trotz der verwirrenden Include-Abhängigkeiten hat Abbildung 6.5 ihren Zweck erfüllt. Sie gibt Aufschluss über die implementierten Programmkomponenten der WMF und zeigt außerdem die nicht dokumentierte Komponente Programm-Starter.

Nachfolgend aufgelistet sind weitere Möglichkeiten, um die Konsistenzhaltung der Schnittstellendefinitionen zu erleichtern oder zumindest eine einfachere Einarbeitung bei der Entwicklung weiterer Komponenten für CASSY zu ermöglichen.

- Die Gliederung der Verzeichnisse orientiert sich an den Komponenten des Systems. Jede Komponente befindet sich in einem eigenen Verzeichnis, um Verwechslungen zu vermeiden. Die CSCW-Facility befindet sich im Verzeichnis „CSCW-Facility“, die Workflow-Management-Facility im

Verzeichnis „BoWorkflow“, das Aktivitäten-Management im Verzeichnis „Activity-Management“, der jFlow-Wrapper im Verzeichnis „JFlowWrapper“ und der ORBacus im Verzeichnis „OB-3.3.1“.

- Zusätzlich gibt es genau ein Verzeichnis für alle IDL-Spezifikationen von CASSY. Dies erleichtert die Suche nach den IDL-Dateien, vermeidet Konsistenzprobleme und ermöglicht die rasche Anpassung an eine andere ORBacus Version indem einfach alle IDL-Definitionen des Verzeichnisses neu kompiliert werden.
- Da die IDL-Dateien alle im IDL-Verzeichnis liegen, sollten im Verzeichnis jedes Teilprojekts (z. B. JFlowWrapper) entsprechende Links auf die IDL-Dateien vorhanden sein und die Namen in einer Projektbeschreibungdatei stehen, damit eine eindeutige Zuordnung möglich ist.
- Jedes Verzeichnis sollte eine eigene *ReadMe*-Datei haben, welche die Funktionalität der enthaltenen Komponenten und deren Gebrauch erklärt. Denn oft sind bestimmte Voraussetzungen notwendig, damit ein spezieller Dienst gestartet werden kann, z. B. Name-Service muss vorher gestartet werden.
- Momentan wird das Entwicklungssystem Microsoft Visual C++ 6.0 verwendet. Das oberste CASSY-Verzeichnis sollte eine VS6-Projektdatei enthalten, die alle Teilprojekte von CASSY enthält. Jedes Teilprojekt sollte in seinem Verzeichnis eine Projektdatei enthalten, welche die Erstellung der Programme dieses Teilprojekts erlaubt.
- Systemvariablen bieten eine elegante Lösung, um Pfadangaben flexibel an verschiedene Systeme anpassen zu können und ermöglichen damit ein compilierbares Visual C++ Projekt auf einfache Weise auf einen anderen Computer übertragen zu können. Die benötigten Systemvariablen sollten ebenfalls in der Projektbeschreibungdatei des jeweiligen Teilprojekts beschrieben sein. Beispiel:  
Umgebungsvariable: %CASSY%  
Pfad: C:/Projekte/Cassy  
Beschreibung: Das Hauptverzeichnis von CASSY  
Durch Verwendung von Systemvariablen können Pfade relativ zu dem Pfad der Systemvariablen gesetzt werden. Somit muss bei Änderungen des absoluten Pfades lediglich der Pfad für die Variable angepasst werden und nicht alle Pfade.
- Gleiches gilt für spezielle Compileroptionen, Linkeroptionen, Präprozessoroptionen etc.

# Kapitel 7

## Zusammenfassung, Bewertung und Ausblick

### 7.1 Zusammenfassung

Das Ziel dieser Arbeit war die Integration der Workflow Management Facility in das CASSY-Aktivitätenmodell. Zu diesem Zweck mussten ein Wrapper entwickelt und geeignete Möglichkeiten zur Integration der WMF-Funktionalität in das Aktivitätenmodell untersucht werden. Dabei musste besonderes Augenmerk auf die gegebenen Randbedingungen, bedingt durch die vorausgegangenen Arbeiten, gelegt werden.

Ausgangspunkt war die Integrationsarchitektur CASSY, in der eine Workflow Management Facility und die Klassen `Activity`, `ToplevelActivity`, `BasicActivity` und `PrimitiveActivity` der Activity Management Facility realisiert waren. Der Quellcode dieser Implementierungen stand in C++ zur Verfügung, wenngleich die Compilierung dieser Komponenten in der Entwicklungsumgebung Microsoft Visual C++ 6.0 erst nach kleinen Änderungen im Quellcode und dem Setzen passender Umgebungsvariablen des Systems und der Entwicklungsumgebung möglich war. Als Middleware wurde das CORBA-System ORBacus 3.3.1 verwendet, dessen Quellcode von Object Oriented Concepts für den nicht kommerziellen Einsatz kostenlos zur Verfügung gestellt wird, und für das verwendete System (Windows NT 4.0) compiliert.

Nun ist das Projekt CASSY seiner Verwirklichung einen Schritt näher gekommen. Durch den Wrapper wird ein Test der ActivityManagement-Komponente für Workflow-Aktivitäten ermöglicht. Diese Arbeit hat allerdings auch gezeigt, dass die Integration einer externen Applikation nicht völlig unproblematisch ist. Insbesondere durch die zwei aufeinander aufbauenden Wrappings geht durch Unterschiede in den Modellen beim Übergang von Flowmark zu dem jFlow-Modell

und vom jFlow-Modell zur ActivityManagement-Komponente stets ein Teil der Funktionalität verloren, weil sich manche Eigenschaften und Funktionen nicht in das jeweils andere Modell übertragen lassen (Abschnitt 4.4). Die Verluste sind um so geringer, je besser die Modelle zueinander passen. Aus diesem Grunde werden auch noch weitere Arbeiten auf diesem Gebiet für CASSY notwendig sein (siehe Abschnitt 7.3). Das perfekte Wrapping ist jedoch nur bei vollständig kompatiblen Modellen möglich.

## 7.2 Bewertung

Die Ziele der Diplomarbeit wurden erreicht und die Workflow Management Facility durch einen Wrapper unter Berücksichtigung bestehender Implementierungen in das Aktivitätenmodell integriert. Darüberhinaus wurden Problembereiche bestehender Implementierungen von CASSY aufgezeigt und Vorgehensweisen und Lösungsvorschläge erarbeitet, welche bei der Integration weiterer Komponenten von Nutzen sein werden.

## 7.3 Ausblick

Nachdem die Workflow Management Facility nun in das Aktivitätenmanagement integriert ist, müssen noch geeignete Wege gefunden werden, um auch die Benutzerverwaltung zu integrieren. Denn in der jFlow-Spezifikation ist dieser Dienst nur rudimentär beschrieben, er muss also durch eine eigene Lösung eingebracht werden. Wie Abschnitt 3.2 gezeigt hat, müssen zur Komplettierung von CASSY noch die Activity Management Facility verfullständigt und die User Assistance Komponente implementiert werden. Dann kann das System erprobt werden und es können weitergehende Untersuchungen auf dieser neuen Technologie aufbauen. Folgende Liste gibt einen Überblick über die zukünftigen Arbeiten des Projekts CASSY:

- Integration der Benutzerverwaltung.
- Neuimplementierung der Workflow Management Facility für den Einsatz mit MQSeries und evtl. Anpassung an den jFlow 2.0 Standard.
- Implementierung einer geeigneten Benutzerschnittstelle für CASSY.
- Koordinationsmechanismen für die verschiedenen Aktivitäten.
- Ergänzung von CASSY um ein Konfliktmodell

- Test und Verifikation von CASSY durch die Realisierung von Test-Anwendungen.
- Implementierung realer Designflow-Szenarien, welche die Möglichkeiten des ASCEND-Modells ausschöpfen, um die Anwendbarkeit des Modells zu belegen und um Erfahrungen mit dem Modell zu sammeln.
- Ergänzung und Vervollständigung von CASSY um fehlende Funktionen, um ein ganzheitliches, zusammenhängendes und vollständiges System zu erhalten.

Ich bin überzeugt, dass zukünftige CSCW-Systeme in zunehmendem Maße auf eine Vereinheitlichung der Benutzerführung abzielen werden. Gleichzeitig ist eine stetig zunehmende Automatisierung in betrieblichen Abläufen zu beobachten: Von Einbenutzeranwendungen über Mehrbenutzeranwendungen hin zu Groupware- und Workflow-Management-Systemen erfolgt eine zunehmende Erweiterung des Anwendungsgebiets der Software. Abschnitt 2.3 hat gezeigt, dass Groupware und Workflow-Management ähnliche Zielsetzungen (z. B. Kommunikationsunterstützung) haben, sich in ihren Einsatzgebieten jedoch unterscheiden. Somit kann die Erweiterung des Anwendungsgebietes und damit die Erweiterung der Computerunterstützung durch die Kombination beider Technologien erreicht werden.

Dass diese Verschmelzung der Technologien möglich ist, wird am Beispiel von CASSY deutlich. Notwendig ist dafür ein integriertes und offenes System, das dem Benutzer genügend Freiraum lässt, es seinen speziellen Bedürfnissen anzupassen und das flexibel auf sich verändernde Anforderungen reagieren kann.



# Anhang A

## ActivityInterface.idl

```
#include "ActorInterface.idl"

module ActivityInterface
{
    interface Activity;
    interface PrimitiveActivity;
    interface BasicActivity;
    interface ToplevelActivity;
    interface GroupwareActivity;
    interface WorkflowActivity;
    interface DesignflowActivity;

    // Type definitions

    // ActivitySequence

    typedef sequence<Activity> ActivityList;
    typedef sequence<ActorInterface::BasicResourceAdapter> ResourceList;

    // Activity states

    enum stateType { initialized,
                    started,
                    stopped,
                    canceled,
                    not_initialized,
                    finished };

    enum ActivityType { ACTIVITY,
                      PRIMITIVEACTIVITY,
                      BASICACTIVITY,
                      TOPLEVELACTIVITY };

    enum BasicActivityType { WORKFLOWACTIVITY,
                            GROUPWAREACTIVITY,
                            DESIGNFLOWACTIVITY };
}
```

```

/*****
/*****
/**
/**          Activities          **
/**          **
/*****
/*****

/*****
//          Activity
/*****

interface Activity {

/***** Attributes *****/

    attribute unsigned long resources_length;

/***** Methods *****/

    ActivityType is_a();

//state management
    boolean getstate(out stateType state);
    boolean init();
    boolean start();
    boolean stop();
    boolean terminate();
    boolean reset();
    boolean cancel();
    boolean resume();

//actor management
    boolean getActor(out ActorInterface::Actor actor);
    boolean setActor(in ActorInterface::Actor actor);
    boolean getName(out string name);
    boolean setName(in string name);

//ID
    boolean getID(out string id);
    boolean setID(in string name);

//hierarchy (parents only)
    boolean getParentActivity(out BasicActivity activity );
    boolean setParentActivity(in BasicActivity activity);

//resource management
    boolean getResources( out ResourceList resources);
    boolean setResourceList(in ResourceList resources, in long len);
    boolean getResource(in string id,
                        out ActorInterface::BasicResourceAdapter bra);
    boolean addResource(in ActorInterface::BasicResourceAdapter bra);
    boolean removeResource(in string id);
    boolean testResource(in string id);

//persistency
    boolean saveActivity();

```

```

    boolean loadActivity(in ToplevelActivity top);

};

/*****
//
//                               PrimitiveActivity
//
interface PrimitiveActivity:Activity { };

/*****
//
//                               BasicActivity
//
interface BasicActivity:Activity {

/***** Attributes *****/

    attribute unsigned long childActivities_length;
    attribute unsigned long childActivities_free;

/***** Methods *****/

    // hierarchy (children)
    // parents are already implmented via Activity interface
    boolean getChildActivities(out ActivityList activities);
    boolean getChildActivity(out Activity activity, in string id);
    boolean addChildActivity(in Activity childActivity);
    boolean removeChildActivity(in string id);
    boolean testChildActivity(in string id);

};

/*****
//
//                               ToplevelAktivity
//
interface ToplevelActivity : BasicActivity {

/***** Attributes *****/

    attribute unsigned long activities_length;
    attribute unsigned long activities_free;
    attribute unsigned long actual_activities_length;

/***** Methods *****/

    boolean showAllActivities (out ActivityList activities);
    boolean getActivity(out Activity activity, in string id);
    boolean removeActivity(in string id);
    boolean addActivity(in Activity activity);
    boolean testActivity(in string id);

};

```

```

//*****
//
//                               Workflow Activity
//*****

interface WorkflowActivity : BasicActivity {

//***** Attributes *****

    attribute unsigned long identifier;
    attribute unsigned long links;
    attribute unsigned long composition;
    attribute unsigned long controlflow;

//***** Methods *****

    boolean sendMessage(out string message);
    boolean receiveMessage(in string message);
    boolean getDefinition(in string definition);
    boolean setDefinition(out string definition);

};

//*****
//
//                               Groupware Activity
//*****

interface GroupwareActivity : BasicActivity {

//***** Attributes *****

//***** Methods *****

};

//*****
//
//                               Designflow Activity
//*****

interface DesignflowActivity : BasicActivity {

//***** Attributes *****

//***** Methods *****

};

//*****

```



```
        in unsigned long activities_len);

    ToplevelActivity create_empty_ToplevelActivity_object(in string id);
};

//*****
//                               WorkflowActivityObjectFactory
//*****

interface WorkflowActivityObjectFactory {

    WorkflowActivity create_WorkflowActivity_object(in string name,
        in string id,
        in ActorInterface::Actor actor,
        in WorkflowActivity parent,
        in ResourceList resources,
        in unsigned long resources_len,
        in ActivityList childs,
        in unsigned long childs_len);

    WorkflowActivity create_empty_WorkflowActivity_object(in string id);
};

};
```

# Literaturverzeichnis

- [Bur97] Cora Burger. *Groupware, Kooperationsunterstützung für verteilte Anwendungen*. dpunkt Verlag, 1997.
- [Flo96] IBM Flowmark. IBM Flowmark Programming Guide. Programmdokumentation SH19-8240-02, IBM, 1996.
- [FM99] Aiko Frank and Bernhard Mitschang. Towards an Activity Model for Design Applications. Paper, Universität Stuttgart, IPVR, 1999.
- [Fra99a] Aiko Frank. Projekt: ASCEND, Unterstützung für kooperative Entwurfsabläufe. Online-Dokument, Universität Stuttgart, IPVR, <http://www.informatik.uni-stuttgart.de/ipvr/as/projekte/JB9899Projekte/ascend.html>, 08. Juli 1999.
- [Fra99b] Aiko Frank. The ASCEND project (Activity Support in Cooperative ENvironments for Design Issues). Online-Dokument, Universität Stuttgart, IPVR, [http://www.informatik.uni-stuttgart.de/ipvr/as/projekte/ascend/ascend\\_home.html](http://www.informatik.uni-stuttgart.de/ipvr/as/projekte/ascend/ascend_home.html), 08. Feb. 1999.
- [Hei98] Günter Heiß. *Evaluierung der Vorschläge für eine CORBA Workflow Facility und die Implementierung eines solchen Dienstes*. Technische Universität München, Institut für Informatik, 1998. Diplomarbeit.
- [HV99] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley, 1999.
- [JBS99] Stefan Jablonski, Markus Böhm und Wolfgang Schulze. *Workflow-Management, Entwicklung von Anwendungen und Systemen. Facetten einer neuen Technologie*. dpunkt Verlag, 1999.
- [KKS<sup>+</sup>98] Thomas Kindler, Ottokar Kulendik, Kerstin Schneider, Reiner Siebert, Tobias Soyez und Kurt Rothermel. Verbundprojekt Poliflow. Abschlussbericht, Universität Stuttgart, IPVR, 1998.

- [Mar98] Marcello Mariucci. *Untersuchung kooperativer Abläufe anhand von Beispielen und Entwicklung eines kooperativen Entwurfsvorgangmodells für die Integration von Workflow- und CSCW-Aspekten*. Technische Universität München, Institut für Informatik, 1998. Diplomarbeit.
- [Muc97] Michael Muchitsch. *Entwurf und prototypische Implementierung einer CORBA CSCW-Facility*. Technische Universität München, Institut für Informatik, 1997. Diplomarbeit.
- [OMG95] OMG. Common Facilities Architecture. OMG-Dokument formal/98-07-10, Object Management Group, November 1995. Version 4.0.
- [OMG97a] OMG. A Discussion of the Object Management Architecture. OMG-Dokument formal/00-06-41, Object Management Group, Januar 1997.
- [OMG97b] OMG. Workflow Management Facility, Request for Proposals. OMG-Dokument cf/97-05-06, Object Management Group, 1997.
- [OMG97c] OMG Business Domain Task Force BODTF-RFP 2 Submission. Workflow Management Facility, jFlow - Joint RFP Submission. OMG-Dokument bom/97-08-05, Object Management Group, 1997.
- [OMG98] OMG. CORBA Services: Common Object Services Specification. OMG-Dokument formal/98-12-09, Object Management Group, Dezember 1998.
- [OMG99a] OMG. C++ Language Mapping Specification. OMG-Dokument formal/99-07-41, Object Management Group, Juni 1999.
- [OMG99b] OMG. The Common Object Request Broker: Architecture and Specification. OMG-Dokument formal/99-10-07, Object Management Group, Oktober 1999. Revision 2.3.1.
- [OOC00] OOC. ORBacus for C++ and Java. Online-Dokument, Object Oriented Concepts, <http://www.ooc.com/ob/>, 2000. Version 3.3.1.
- [Red97] Jens-Peter Redlich. *Corba 2.0 Praktische Einführung für C++ und Java*. Addison-Wesley, 1997.
- [Rös99] Markus Rösner. *Entwurf und Implementierung gemeinsamer CSCW- und Workflow-Aktivitäten für CASSY*. Universität Stuttgart, Institut für Informatik, 1999. Studienarbeit.

- [Som98] Ian Sommerville. *Software Engineering*. Addison-Wesley, 5th edition, 1998.
- [WfM95] WfMC. Workflow Management Coalition, The Workflow Reference Model. Online-Dokument TC00-1003, Workflow Management Coalition, <http://www.aiim.org/wfmc/>, 19. Jan. 1995. Version 1.1.



Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, den 31. Juli 2000

---

(Andreas Kuhn)