Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

CORBA-Integration des Workflow-Management-Systems IBM FlowMark

Diplomarbeit

Aufgabenstellung und Betreuung:

Prof. Dr. habil. E. Rahm,

Dipl.-Math. R. Müller

vorgelegt von

Uwe Neubert

Leipzig, Februar 1999

Vorwort

Das Thema der vorliegenden Arbeit entstand im Rahmen des Projektes „HematoWork“. Im Laufe der Projektes wurde immer deutlicher, daß es beim Einsatz von Workflow-Management-Systemen in der Praxis Ansatzpunkte für eine Verbesserung der Workflow-Modellierungsmöglichkeiten gibt. Ursprünglich vom medizinischen Standpunkt ausgehend, können die in dieser Arbeit dargestellten Erkenntnisse jedoch auch auf andere Anwendungsgebiete von Workflow-Management-Systemen übertragen werden.

Bei folgenden Personen möchte ich mich bedanken:

- Herrn Prof. Dr. E. Rahm für die Aufgabenstellung und die freundliche Unterstützung der Arbeit,
- Herrn Prof. Dr. A. Winter für hilfreiche Informationen über das Krankenhausinformationssystem (KIS),
- Herrn R. Müller für die intensive Betreuung der Arbeit,
- meinem Kommilitonen Daniel Käps für die Unterstützung in Orbix-Problemen.

Inhaltsverzeichnis

1 Einleitung

1.1 Übersicht

Die vorliegende Arbeit entstand im Rahmen des Projektes „HematoWork“. Ziel dieses Projektes, das als Kooperation zwischen dem Institut für Informatik und dem Institut für Medizinische Informatik, Statistik und Epidemiologie der Universität Leipzig bearbeitet wird, ist die rechnergestützte Durchführung von Therapiestudien in der Hämato-Onkologie. Eine Unterstützung der Durchführung von Studien durch die Informationsverarbeitung verspricht wesentliche Verbesserungen in der Kommunikation und Dokumentation der beteiligten Institutionen. Durch die damit verbundene Erhöhung der Zuverlässigkeit der Ergebnisse kann die Qualität der statistischen Auswertung verbessert werden, die für eine Aussage über den Erfolg einer Studie von entscheidender Bedeutung ist.

Im Projekt sollen neben verschiedenen Applikationen, beispielsweise für die Berechnung von Chemotherapien, auch *Workflow-Management-Systeme* (*WfMS*) zum Einsatz kommen. Während die einzelnen Applikationen nur für Teilaufgaben zuständig sind, liegt es in der Verantwortung des WfMS, ganze Behandlungs- und Dokumentationsabläufe korrekt zu steuern.

Für die Kommunikation zwischen WfMS und Applikationen sowie zwischen den Applikationen untereinander soll die *Middleware CORBA* verwendet werden. CORBA erlaubt eine Einführung verteilter Applikationen, wobei von konkreten technischen Bedingungen weitgehend abstrahiert werden kann. Zusätzlich kann durch CORBA-Module (z.B. durch den *Object Transaction Manager; OTM*) eine hohe Zuverlässigkeit des Gesamtsystems gewährleistet werden.

Die vorliegende Arbeit hat das Ziel, CORBA und das Workflow-Management-System IBM FlowMark zu integrieren. Als CORBA-Implementierung wurde *Iona Orbix, Version 2.3c*, als WfMS *IBM FlowMark, Version 2.3* verwendet. Bei der Integration werden sowohl Modellierungs- als auch Ausführungsaspekte von Workflows in IBM FlowMark berücksichtigt.

Die Integration von CORBA und IBM FlowMark besitzt zwei Schwerpunkte.

Einerseits muß die *Middleware* in das WfMS eingebunden werden. Ziel dieser Art der Integration ist es, Arbeitsschritte, die durch CORBA-Objekt-Methoden implementiert werden, auszuführen.

Andererseits ist eine Integration des WfMS in CORBA nötig. Dadurch können Applikationen mit dem WfMS kommunizieren, beispielsweise ist es möglich, bei der Patientenaufnahme patientenspezifische Workflows zu starten. Die Integration von IBM FlowMark in CORBA kann bereits vollständig durch eine Ergänzung der IBM FlowMark-API’s um CORBA-

Kommunikationsaspekte erfolgen[1]. Deshalb wird der Schwerpunkt der Arbeit auf der Integration von CORBA in IBM FlowMark liegen.

Zunächst wird medizinische Hintergrund des Projektes „HematoWork“ skizziert. Daran anschließend werden die Anforderungen der onkologischen Station an eine Integration beschrieben (Kapitel 2). Es folgt eine Darstellung des WfMS IBM FlowMark in relevanten Eigenschaften. Der anschließende Abschnitt beschreibt Elemente des CORBA-Standards.

Bei der Beschreibung von IBM FlowMark und CORBA wird besonders auf die Kommunikationsmöglichkeiten eingegangen. Der Systemanalyse von IBM FlowMark und CORBA schließt sich eine Darstellung der technischen Anforderungen an eine CORBA-Integration an.

Die Ergebnisse der Analyse finden Eingang in einen konzeptionellen Entwurf (Kapitel 4), dessen praktische Umsetzung im Kapitel 4 betrachtet wird. Es werden die Workflow-Modellierungssprache *CORBA Workflow Definition Language* (*CWDL*) für die Buildtime, ein Übersetzer von CWDL in die Workflow-Definitionssprache von IBM FlowMark, *FlowMark Definition Language* (*FDL*), und verschiedene Module für eine CORBA-Runtime-Integration beschrieben. Es schließt sich eine kritische Betrachtung der Lösung an, die jedoch gleichzeitig einen Ausblick auf umfangreiche weitere Aufgaben sowie Forschungsansätze und Standardisierungsbemühungen auf dem bearbeiteten Gebiet bieten soll.

1.2 Medizinische Grundlagen

1.2.1 Medizinische Therapiestudien

Die Onkologie beschäftigt sich mit der Erforschung und Behandlung von Krebs. Als eine häufige Todesursache steht diese Krankheit weltweit im Mittelpunkt vieler Forschungsarbeiten. Führen Ergebnisse dieser Arbeiten zum Vorschlag neuer Behandlungsmethoden, so müssen diese nach vielen Vortests zuletzt klinisch getestet werden. Die Therapiestudien, die im Projekt „HematoWork“ durch den Einsatz von automatischer Informationsverarbeitung unterstützt werden sollen, sind die *kontrollierten klinischen Therapiestudien* (Abbildung 1-1).

[1] Die Ergänzung der API von IBM FlowMark um CORBA-Kommunikationsaspekte kann im wesentlichen durch eine Übersetzung der in Form von C++ -Klassen angebotenen Signatur des WfMS in die Schnittstellenbeschreibungssprache IDL erfolgen.

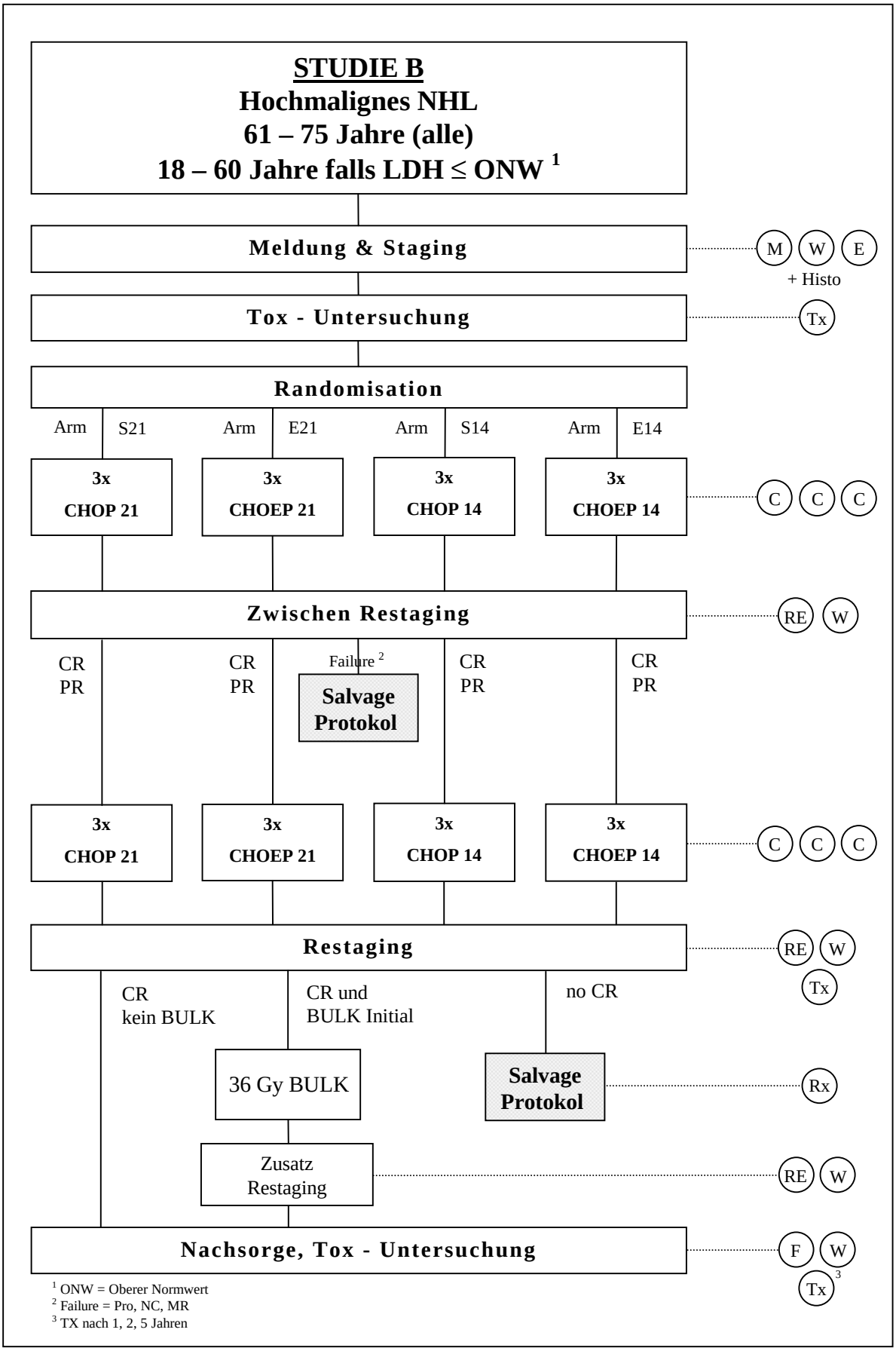


Abbildung 1-1: Hochmalignes NHL: Der Behandlungsablauf in der Studie B. Nach [NHL94].

In einer solchen Studie werden bereits praktizierte Behandlungsabläufe mit den neuen Therapiemethoden verglichen.

Nicht zuletzt aufgrund ethischer Forderungen dürfen nur solche Behandlungsmethoden klinisch getestet werden, deren Erfolgsaussicht nicht schlechter als bei bisherigen Therapien ist. Gleichzeitig dürfen in einer Studie jedoch auch keine Therapien miteinander verglichen werden, deren Unterschied bereits durch Vortests ausreichend gezeigt wurde, da allen an einer Studie teilnehmenden Patienten eine gleiche Heilmöglichkeit geboten werden soll.

[Vi81a] schreibt:

> „Sie [die randomisierten, kontrollierten klinischen Studien] sind zulässig, solange keine gesicherten Ergebnisse zur Überlegenheit einer der untersuchten Therapien vorliegen bzw. solange diese kontrovers beurteilt werden."

Die Voraussetzungen für klinisch kontrollierte Studien sowie nur marginale Erfolgsunterschiede trotz Einsatz neuer Wirkstoffe und Methoden führen zum Vergleich von Behandlungsmethoden, deren Unterschied nur durch eine statistische Auswertung ermittelt werden kann.

[Sch95] schreibt dazu:

> „Zielsetzung kontrollierter klinischer Studien ist es, vergleichende Aussagen ... zwischen Therapien zu treffen und zwar so, daß diese Vergleiche so wenig wie möglich durch Störfaktoren verfälscht und verzerrt sind. Der Einfluß von Störfaktoren kann zu diesem Zweck zwar nicht ausgeschaltet, wohl aber kontrolliert werden, indem man dafür sorgt, daß die Störfaktoren in beiden Gruppen [in den zu vergleichenden Patientenkollektiven (Anm. des Verfassers)] gleich sind."

Deshalb muß eine Studie in ihrer Durchführung und Dokumentation strengsten Richtlinien genügen. Diese Richtlinien werden in einem *Studienprotokoll* festgehalten.
Das Studienprotokoll enthält neben den die Zielgruppe definierenden *Ein-* und *Ausschlußkriterien* für Patienten (z.B. dem Alter), alle nötigen Informationen über den *Behandlungsablauf* (z.B. über die Voruntersuchungen wie die Toxizitätsuntersuchung, die Dosierung der einzelnen Medikamente und die Aufeinanderfolge von Chemotherapien; Abbildung 1-1), die beteiligten *Institutionen* und die *Dokumentationsrichtlinien* (Dokumentationsformulare und –termine).
Die zu vergleichenden Behandlungsmethoden sind in Form von Therapiearmen in den Behandlungsablauf des Patienten integriert. Ein Patient wird immer genau einem der alternativen Therapiearme zugewiesen. Die Zuweisung erfolgt im Rahmen der *Randomisation*, einem Verfahren, das die zufällige Verteilung der Patienten auf die Therapiearme und damit die bereits zitierte gleiche Verteilung der Störfaktoren ([Sch95]) gewährleistet.

Zur Größe einer Stichprobe, d.h. der Anzahl der an der Studie teilnehmenden Patienten, schreibt [Üb81]:

> „Die Fallzahl muß hinreichend groß, aber nicht zu groß sein. Die Festlegung der Fallzahl ist nicht ein statistisches Problem, sondern ein Bewertungsproblem."

Ähnliche Überlegungen enthält auch [Sch95]).

Durchführende einer Studie sind die behandelnden *Kliniken*, die *Studienleitung* (Koordination, Randomisation sowie statistische Auswertung) und *weitere Einrichtungen*, wie z.B. die *Histologie*. Die Dokumentation des Behandlungsablaufes für einen an der Studie teilnehmenden Patienten erfolgt in Form von *Studienbögen*. Studienbögen können *Meldebögen, Chemotherapiebögen, Kontrollbögen* usw. sein. Im Verlauf einer Behandlung müssen sie in unterschiedlicher Art, teilweise mehrfach, erstellt und der Studienleitung termingerecht zugesandt werden. Damit kann von der Studienleitung einerseits der erfolgreiche Verlauf der Studie überprüft werden, andererseits kann die Studienleitung bereits während der Studie statistische Auswertungen vornehmen und den weiteren Verlauf der Studie planen (etwa den Abbruch bei einer hohen Rückfall- und Todesrate bei Anwendung der neuen Behandlungsmethode oder eine Stichprobenvergrößerung).

Aufgrund praktischer Probleme werden die Richtlinien vielfach nicht eingehalten. Besonders im organisatorischen Bereich treten Probleme auf. U.a. [Sch95] schreibt:

> „Klinische Studien scheitern am häufigsten an mangelhafter organisatorischer Planung."

In der Studie festgelegte Studienbögen werden nur mit großer Verzögerung oder gar nicht erstellt. Zusätzlich kommt es vor, daß ein ausgefüllter Studienbogen nur unvollständig oder nicht korrekt ist. Besonders Abweichungen vom Studienprotokoll werden nicht ausreichend für die Studie dokumentiert. Deshalb wird im Projekt „HematoWork" versucht, diese Probleme durch Einsatz von Informationsverarbeitung, besonders aber durch die Nutzung von WfMS, zu lösen.
Hervorzuheben sind damit zwei Punkte, die der Einsatz von WfMS adressieren soll[2]:

- Die statistische Auswertung der Studie muß durch eine protokollgerechte Durchführung gewährleistet sein. Grobe Abweichungen vom Studienprotokoll, die nicht durch Ereignisse (z.B. Komplikationen) im Behandlungsverlauf des Patienten verursacht werden, sollen vermieden werden.

- Es muß eine zentrale Koordination der Durchführung einer Studie stattfinden können. Das ist speziell für die Durchführung von multizentrischen Studien unerläßlich.

Die im Projekt „HematoWork“ betrachteten Studien fokussieren alternative Behandlungsmethoden der *Hämato-Onkologie* ([MH98]).

1.2.2 Die Hämato-Onkologie

Die *Hämato-Onkologie* beschäftigt sich mit der Diagnose und Behandlung von *nicht-soliden Tumoren* wie z.B. *Lymphomen* und *Leukämien*. Die Behandlung eines nicht-soliden Tumors kann durch *Poly-Chemotherapie* oder *Radiotherapie*, aber auch *chirurgisch* erfolgen. Meist werden die einzelnen Formen der Behandlung miteinander kombiniert, um ein optimales Ergebnis zu erzielen. Der Schwerpunkt des Projektes „HematoWork“ liegt hauptsächlich auf der Durchführung von Therapiestudien mit chemotherapeutischen Behandlungsmethoden. Der Unterschied zwischen zwei Behandlungsmethoden der Chemotherapie spiegelt sich in der unterschiedlichen Ausprägung einzelner Parameter wider. Die Wirkung der unterschiedlichen Parameter-Werte ist damit Gegenstand einer Studie. Untersuchte Parameter können beispielsweise die *Dosis* eines Medikamentes, die verwendeten *Zytostatika*[3], die *Dauer* der Chemotherapie oder auch die *zeitliche Aufeinanderfolge* von Chemotherapien sein. Einen Ausschnitt aus dem Studienprotokoll für hochmaligne *Non-Hodgkin-Lymphome* (*NHL*) zeigte bereits Abbildung 1-1. Das Standard-Protokoll für die Behandlung von Patienten mit hochmalignen Non-Hodgkin-Lymphomen ist derzeit das *CHOP-Protokoll*[4]. Seit seiner Einführung vor 20 Jahren war es nicht mehr möglich, den Behandlungserfolg durch Dosissteigerung einzelner Medikamente oder Hinzunahme neuer Zytostatika zu verbessern ([NHL94]). Allerdings wurden nicht alle Variationen von Dosissteigerung und Verwendung weiterer Zytostatika in einer *randomisierten* Studie überprüft ([NHL94]). Insbesondere wurde das Zytostatikum Etoposid noch nicht in einer solchen Studie für den Einsatz bei der Chemotherapie von hochmalignen NHL bewertet. Deshalb ist eine solche Evaluierung als Therapie *CHOEP*[5] Gegenstand der Studie B für hochmalignes NHL. Diese Studie enthält zusätzlich Varianten von CHOP und CHOEP, die als Intensivierung der jeweiligen Schemata durch Intervallverkürzungen realisiert werden.

---

[2]WfMS sollen die in [MH98] beschriebenen 4 Funktionsarten (*Treatment Functionality*, *Intra-hospital Communication Functionality*, *Inter-hospital Communication Functionality* und *Tracking Functionality*) steuern.
[3] Zellgift.
[4] Die verwendeten Zytostatika sind gemäß [NHL94] Cyclophosphamid, Doxorubicin, Vincristin und Prednisolon.
[5] In Ergänzung zum CHOP-Protokoll wird zusätzlich das Zytostatikum Etoposid verwendet.

Organisatorische Probleme im Bereich der studienbasierten Hämato-Onkologie sind neben den bereits genannten allgemeinen Problemen auch Ereignisse im Behandlungsablauf des Patienten (z.B. Komplikationen).

> Komplikationen können entstehen, wenn einzelne Medikamente auf den Patienten toxisch wirken. Sie können nur kurzzeitig andauern und damit den Behandlungsablauf eines Patienten nicht wesentlich beeinflussen.
>
> Es ist aber auch möglich, daß sich selbst durch die Applikation zusätzlicher Medikamente die Reaktion des Patienten nicht ändert. Das kann zum Ausschluß des Patienten aus der Studie führen.

Die Komplikationen werden oft nicht ausreichend für die Studie dokumentiert.

1.3 WfMS - Werkzeuge zur Unterstützung von Arbeitsvorgängen

Um die genannten Anforderungen an eine Studie zu erfüllen und die organisatorischen Probleme zu lösen, wird eine Automatisierung wichtiger Vorgänge angestrebt.
Die Automatisierung kann durch eine Einführung von geeigneten Anwendersystemen, beispielsweise der Patientendatenbank ([Br97]), eines Moduls zur Berechnung der patientenspezifischen Medikation (z.B. in Abhängigkeit von der Körperoberfläche des Patienten; [Be97]) und des datenbankgestützten Dokumentenstruktur-Editors und -Generators ([Jö97]), erfolgen.
Damit wird jedoch nur die studiengerechte Durchführung *einzelner* Arbeitsschritte unterstützt. Darüber hinausgehend ist es auch möglich, *alle* für eine Studie *relevanten Prozesse* zu automatisieren. Das kann unter Verwendung von *Workflow-Management-Systemen* (*WfMS*), aber auch mit *Groupware* erfolgen.

> Bereits in [Du81] wird eine Prozeß-Steuerung in Form von Entscheidungstabellen diskutiert:
>
> „Ein DV-System zur Unterstützung klinischer Studien muß folgende Funktionen bereitstellen:
> - Formulargenerator
> - Input-Output-Generator
> - Protokoll-Generator
> - Entscheidungstabellen-Generator
> - Übertragungsprozeduren.“
>
> [Hö81] verwendet bereits den Begriff „Real-Time-Prozeßsteuerung“.

Zunächst werden grundlegende Definitionen und Aufgaben von WfMS genannt ([WfMC96]).

Business Process	„A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships.“
Workflow	„The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“
Workflow Management System (WfMS)	„A system that defines, creates and manages the execution of workflows through the use of software, running on one ore more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.“

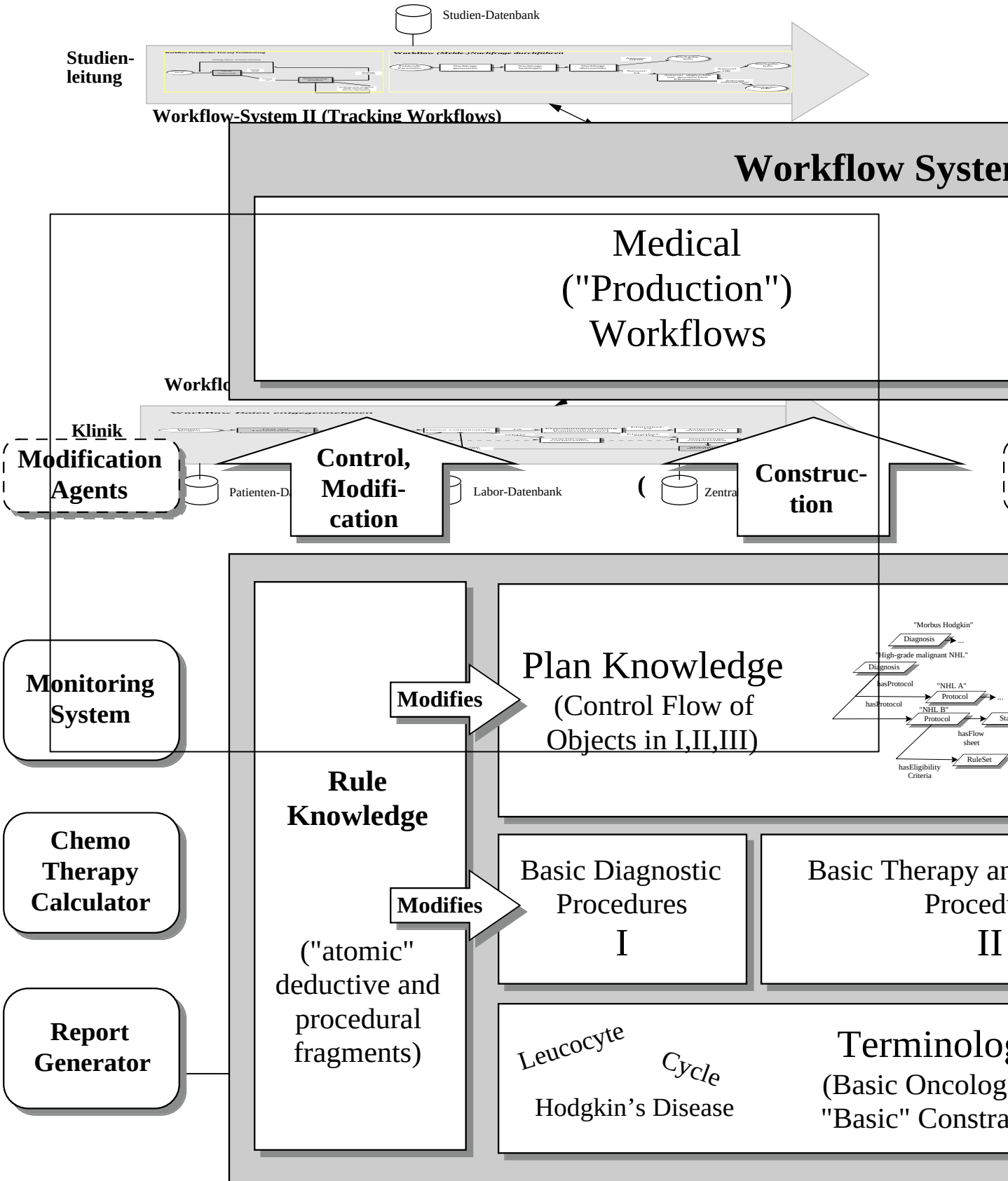
Es werden eine oder mehrere *zusammengehörige Aktivitäten* im Rahmen einer *Organisationsstruktur* unter Verwendung von *Rollen* und *funktionalen Beziehungen* durch eine Menge von *prozeduralen Regeln automatisiert*.

Dafür ist ein WfMS nötig, das die *Definition*, die *Instantierung* und *Ausführung* von Workflows unter Verwendung von (verteilten) *Workflow Engines*, die die Prozeßdefinition *interpretieren*, ermöglicht, mit den Workflow-Teilnehmern *kommuniziert* und Applikationen *ausführt*.

Damit läßt sich ein WfMS unter 2 Aspekten betrachten:

- Elemente von Business-Prozessen (Modellierungsmächtigkeit des WfMS)
- Modellierungs- und Laufzeitkomponente eines WfMS (Implementierung der Elemente von Business-Prozessen).

Im folgenden soll unter einem WfMS ein System verstanden werden, das es erlaubt, Arbeitsprozesse zu modellieren und ihre Ausführung zu unterstützen. Im WfMS modellierte und ausgeführte Prozesse sind stark strukturiert und sie werden häufig wiederholt. Die Koordination der einzelnen Arbeitschritte erfolgt gemäß dem modellierten Workflow. Der Benutzer erfährt während der Ausführung von Workflows keine Unterstützung bei der Änderung der Ablaufstruktur. Typische Vertreter so definierter WfMS sind IBM FlowMark und COSA Workflow.
Im Gegensatz dazu ermöglicht es Groupware (z.B. Lotus Notes), sich bereits im Ablauf befindliche Prozesse noch während ihrer Ausführung zu definieren. Es ist möglich, den nächsten für die Ausführung eines Vorganges erforderlichen Schritt durch den Endanwender definieren zu lassen. Hierbei ist es jedoch erforderlich, daß der Benutzer über Prozeß-, Organisations- und Modellierungswissen verfügt. Damit ist Groupware ein geeignetes Mittel, schwach strukturierte, sich meist nicht wiederholende, nur wenige Schritte umfassende Arbeitsvorgänge zu automatisieren.


Studien-Datenbank
Studien-leitung
Workflow-System II (Tracking Workflows)
Workflow System
Medical ("Production") Workflows
Klinik
Modification Agents
Control, Modifi-cation
Construc-tion
Labor-Datenbank
Monitoring System
Chemo Therapy Calculator
Report Generator
Modifies
Plan Knowledge
(Control Flow of Objects in I,II,III)
Rule Knowledge
("atomic" deductive and procedural fragments)
Basic Diagnostic Procedures
I
Basic Therapy ar
Proced
II
Leucocyte
Cycle
Hodgkin's Disease
Terminolog
(Basic Oncolog
"Basic" Constra
"Morbus Hodgkin"
Diagnosis
"High-grade malignant NHL"
hasProtocol
"NHL A"
Protocol
"NHL B"
hasFlow sheet
RuleSet
hasEligibility Criteria

- *Terminologische Basis* mit onkologischen Begriffen und ihrer Beziehung untereinander (Schicht 3; *Terminological Base*).

Die aufeinander aufbauenden Schichten werden um Wissen in Form von Regeln ergänzt.

Gegenwärtig werden die datenbanktechnischen Voraussetzungen für die Wissensbasis erarbeitet. Das erfolgt unter Verwendung des objekt-orientierten Datenbanksystems O_2 auf dem Betriebssystem *SunSolaris*.

Das *Workflow-System* (WfS; WfMS mit allen Workflows) steuert die einzelnen Applikationen in *Treatment-Workflows* und stellt Elemente für eine *intra-* und *inter-hospital communication* bereit.

1.4 Aufgabenstellung der Diplomarbeit

Die Steuerung eines Behandlungsablaufs soll durch den Einsatz eines WfMS erfolgen, das auf der onkologischen Station für die Abarbeitung der *Treatment Workflows* sowie für die Koordination der *Kommunikation* zuständig ist. Während des Ablaufs eines Workflows sind daher verschiedenste Tätigkeiten notwendig, z.B. *Dokumentation der Patientendaten* in der Patienten-Datenbank, die *Chemotherapie-Berechnung*, und die *Erstellung von Studienbögen*. Ebenso ist die *Kommunikation* mit dem klinikinternen Labor und anderen Einrichtungen sicherzustellen. Im Rahmen dieser Zielstellung muß das WfMS in der Lage sein, die einzelnen *Applikationen auszuführen* und auf ihre Ergebnisse als relevante *Daten für die Ablaufsteuerung* zuzugreifen. Zusätzlich muß es möglich sein, durch eine Applikation auch Workflows, z.B. einen Workflow zur Kommunikation der onkologischen Station mit dem Labor, im WfMS zu starten. Damit ist das WfMS einerseits steuerndes Element des Gesamtsystems, andererseits kann es aus Sicht der anderen Applikationen ebenfalls nur ein Modul im verteilten System der onkologischen Station sein. Deshalb ist eine Integration des WfMS in CORBA notwendig. Es sind zwei Schwerpunkte zu betrachten:

- die Integration von CORBA in das WfMS IBM FlowMark und
- die Integration des WfMS IBM FlowMark in CORBA.

Der zweite Schwerpunkt kann ohne Probleme unter Verwendung der *FlowMark C++* und *C-API's* gelöst werden und ist nicht Gegenstand dieser Arbeit. Wie bereits in Abschnitt 1.1 erwähnt, kann das WfMS unter Verwendung dieser API's als CORBA-Objekt gekapselt werden.

Im Gegensatz zum zweiten Schwerpunkt erweist sich der erste als wesentlich komplexer. Die Zielstellung der CORBA-Integration führt zu einer Untergliederung der Problemstellung in eine *Buildtime*- und eine *Runtime-Integration.*
Eine CORBA-Integration zur *Buildtime* stellt dem Prozeß-Modellierer ein Werkzeug zur problemnahen Prozeß-Modellierung zur Verfügung (einfache Spezifikation von *Aktionen* und *Bedingungen* durch *CORBA-Methodenaufrufe*).
Die *Runtime-Integration* beinhaltet alle Kommunikationsaspekte zwischen IBM FlowMark und CORBA zur *Ausführung* von Workflows.

> Dies betrifft sowohl die Ausführung von Methoden von CORBA-Objekten als auch den Datenaustausch mit diesen.
> Der Datenaustausch dient einerseits der Weiterleitung der Daten an folgende *Workflow-Aktivitäten*, die wiederum Methodenaufrufe auf CORBA-Objekten sein können. Andererseits muß das WfMS über den weiteren Ablauf eines Workflows entscheiden. Deshalb muß für das WfMS eine Möglichkeit gefunden werden, die eine Auswertung von Ergebnissen von Methodenaufrufen in Form von Vergleichen einfacher Terme (Summen, Produkte) erlaubt.

Die Arbeit beschäftigt sich mit der Integration von CORBA in das WfMS, wobei die beteiligten Applikationen und Daten durch CORBA-Objekte repräsentiert werden.

Eine CORBA-Integration von *Aufbauorganisationen* (z.B. Nutzer, Rollen, Organisationen) wird im Projekt „HematoWork“ nicht angestrebt.

> Eine CORBA-Integration der Aufbauorganisation kann mit IBM FlowMark, aber auch mit *COSA* nur als einfache Datenübertragung von Daten über die Organisationsstruktur zwischen verschiedenen Datenbanken erfolgen. Darüber hinausgehende Ansätze, etwa WfMS-Benutzer mit ihren Berechtigungen selbst als CORBA-Objekte darzustellen, können in diesen WfMS nicht realisiert werden.

Zunächst jedoch werden im folgenden Kapitel in einer Analyse die Anforderungen der onkologischen Station an ein WfMS diskutiert. Daran anschließend werden IBM FlowMark und CORBA vorgestellt und bezüglich ihrer Kommunikationsmöglichkeiten betrachtet. Aus der Systemanalyse werden Schlußfolgerungen in der Form von technischen Anforderungen an eine CORBA-Integration in das WfMS IBM FlowMark gezogen. Ausgehend von diesen Forderungen wird ein Lösungskonzept, die *CORBA Workflow Definition Language* (*CWDL*), präsentiert. Die

praktische Umsetzung des Konzeptes beeinflußt *Buildtime* und *Runtime* von Workflows und wird im letzten Teil der Arbeit betrachtet.

2 Analyse

Die vorliegende Arbeit betrachtet ausschließlich die Integration von CORBA in die Ablauforganisation von IBM FlowMark. Damit werden CORBA-Methoden als Applikationen und Daten betrachtet. Es werden zunächst die Anforderungen der onkologischen Station als Forderungen des Anwendungsgebietes diskutiert. Daran anschließend werden die Systeme IBM FlowMark und CORBA in einer Systemanalyse dargestellt. Darauf basierend werden im Rahmen der technischen Anforderungen der Methodenaufruf und der Datenaustausch zwischen CORBA-Objekten und dem WfMS beschrieben.

2.1 Anforderungen der onkologischen Station

Die Anforderungen der onkologischen Station sind Forderungen des Anwenders, die bereits teilweise von den Systemen CORBA, IBM FlowMark sowie von den bereits erstellten Applikationen ([Be97], [Br97], [Jö97]) erfüllt werden. Trotzdem werden sie an dieser Stelle mit aufgeführt. Neben technischen Forderungen sind vor allem Probleme des Anwendungsgebietes Bestandteil der folgenden Betrachtungen.

(A) Technische Forderungen
- (1) Die Integration von CORBA in IBM FlowMark ist möglich unter
- (2) der Verwendung der bisherigen Betriebssysteme.
- (3) Dabei sollen bereits bestehende Anwendungen ohne großen Aufwand in das neue System integriert werden können.
- (4) Die Verwaltung des Systems soll in einfacher Weise erfolgen.
- (5) Das System ist in Hard- und Software erweiterbar.
- (6) Das System ist stabil gegenüber externen Einflüssen.
- (7) Das System besitzt eine sichere Benutzerverwaltung.
- (8) Die Kommunikation zwischen Applikationen führt zu keiner Verzögerung im Behandlungsverlauf eines Patienten.
- (9) Es dürfen keine Fehler im System auftreten.
- (10) Das System ist benutzerfreundlich, insbesondere
- (11) sind die Modellierung von Workflows unter Verwendung von CORBA-Methoden als Applikationen und Daten und die
- (12) Ausführung der Prozeß-Instanzen einfach,
- (13) sehr gut wartbar und
- (14) gut dokumentiert.

Die Forderungen des Anwendungsgebietes adressieren vor allem Aspekte der Funktionalität, können sich aber auch auf juristische oder benutzerergonomische Probleme beziehen.

(B) Forderungen des Anwendungsgebietes

(1) Im Projekt „HematoWork“ werden Abläufe betrachtet, die als komplexe *semantische Netze* in der Wissensbasis verwaltet werden.

Die Abläufe sind Bestandteil der Wissensbasis, da sie eine Einheit mit deren anderen Elementen bilden (Schichtenmodell der Wissensbasis; [MH98]). Gleichzeitig sind die semantischen Netze in nicht-operationaler Form formuliert. Beispielsweise werden Bedingungen wie

$$\text{Leukozytenwert} \geq 3000/\text{mm}^3$$

angegeben, d.h. es erfolgt keine Angabe der Datenquelle, die diese Information enthält. Die Wissensbasis als wichtiges Modul soll ferner beliebigen Anwendungen des Projektes „HematoWork“ den Zugriff auf die Ablauf-Definitionen ermöglichen.

Jeder Patient durchläuft nur einen geringen Teil der durch die semantischen Netze wiedergegebenen Behandlungsschemata. Deshalb ist es sinnvoll, eine redundante Repräsentation von Behandlungsabläufen zu vermeiden bzw. so weit wie möglich zu begrenzen. Es bietet sich an, Teile von Behandlungsabläufen fest im WfMS als Workflows zu definieren. Das betrifft vor allem Strukturen, die sich bei jedem Patienten wiederholen. Dies können gemäß [NHL94] *Meldung & Staging*, die *Toxizitäts-Untersuchung*, die *Randomisation*, die *Chemotherapie-Blöcke*, das *Zwischen-Restaging* sowie die *Nachsorge* mit Toxizitäts-Untersuchung sein. Um auch patientenspezifische Teilstrukturen, wie z.B. den genauen Ablauf einer Chemotherapie oder die Behandlung von Nebenwirkungen, Bestandteil eines ablauffähigen Workflows werden zu lassen, ist es entweder nötig, jede patientenspezifische Möglichkeit der Behandlung als Workflow, und damit in höchstem Maße redundant zur Wissensbasis, zu definieren, oder im Rahmen einer *Skelett-Planung* Teile von Workflows erst dann zu definieren, wenn sie tatsächlich benötigt werden. Dann können sie mit patientenspezifischen Werten in Struktur und Daten parametrisiert werden. Die zweite Möglichkeit soll im Projekt näher

untersucht werden und ist aufgrund der von ihr benötigten Funktionalität[6] unbedingt in dieser Arbeit zu betrachten.

(2) Das Gesamtsystem soll die Behandlung eines Patienten durch die *Treatment Functionality* unterstützen.

In Chemotherapie-Schemata wird oft eine Aufeinanderfolge von einzelnen Chemotherapien in Form von relativen Zeitangaben definiert (z.B. darf in der NHL-Studie B im Arm S21 die Therapie CHOP erst am 22. Tag nach Ende der vorhergehenden beginnen; [NHL94]). Das WfMS soll eine termingesteuerte Ausführung von Aktivitäten gestatten.

(3) Das Gesamtsystem unterstützt die *Intra-hospital Communication Functionality* ([MH98]).

(4) Das Gesamtsystem unterstützt die *Inter-hospital Communication Functionality* ([MH98]).

(5) Um die Kommunikationsarten (*Intra-hospital Communication* und *Inter-hospital Communication*) zu ermöglichen, soll das WfMS eine Folge von Aktivitäten zulassen, die durch externe Ereignisse (z.B. Eingang eines Laborberichtes) *beeinflußt* werden kann.

(6) Erhält die Studienleitung die Studienbögen eines Patienten nicht fristgerecht, so muß sie in Form von Erinnerungen, Nachfragen oder Mahnungen in der Lage sein, mit einem Workflow, mindestens aber mit dem WfMS der onkologischen Station zu kommunizieren. Ein WfMS der onkologischen Station muß daher auf externe Ereignisse *reagieren* können, etwa durch Starten eines neuen Prozesses.

(7) Bei Verwendung eines verteilten Systems müssen sicherheitsrelevante Aspekte beachtet werden. Es ist nötig, Patientendaten in geeigneter Weise verschlüsselt zwischen zwei Modulen der verteilten Anwendung und den beteiligten Institutionen zu übertragen.

Gegenwärtig wird von der OMG an der Standardisierung einer *CORBAfacility CORBAmed* für eine domainspezifische Anpassung von CORBA an das Gesundheitswesen gearbeitet. Dieser Standard soll unter anderem eine Anpassung des Datenübertragungsformats *HL7* an das CORBA-Format enthalten.

[6] Parametrisierung von Workflows nicht nur hinsichtlich der Daten, sondern auch bezüglich ihrer Struktur durch zur Buildtime nicht bekannte Prozeß-Fragmente

2.2 Systemanalyse

2.2.1 IBM FlowMark

Im Rahmen dieser Arbeit wurde mit dem WfMS IBM FlowMark 2.3 gearbeitet. Die konzeptionellen Aspekte der Arbeit sind unabhängig vom verwendeten WfMS. Trotzdem müssen mit Blick auf die Implementierung die Möglichkeiten des verwendeten Systems erläutert werden.

2.2.1.1 Die Elemente eines Business-Prozesses in IBM FlowMark

Die Elemente eines Business-Prozesses lassen sich allgemein in die *Aufbau-* und die *Ablauforganisation* sowie eine *Produkt-* oder *Datenkomponente* gliedern. Während in der Aufbauorganisation der *hierarchische Aufbau* eines Unternehmens wiedergegeben wird, werden in der Ablauforganisation die *Prozesse* des Unternehmens dargestellt. Daten sind Objekte, die auf den Verlauf eines Prozesses Einfluß haben oder durch ihn selbst erzeugt werden (*Produkte*).
Mit IBM FlowMark lassen sich die in Tabelle 2-1 genannten Elemente darstellen. Diese dienen der Modellierung von Workflows.

Aufbauorganisation (Staff)	Mitarbeiter (People) Rolle (Role) Level Organisation
Ablauforganisation	
Aktivitäten und Prozesse	Prozeß (Process) Prozeß-Aktivität (Process Activity) Programm-Aktivität (Program Activity) Block Programme (Program)
Elemente des Kontroll- und Datenflusses	Kontroll-Konnektor (Control Connector) Default-Konnektor (Default Connector) Daten-Konnektor (Data Connector)
Produkte / Daten	Datenstrukturen (Data Structure), gebildet als - Menge (Array) und - Aggregation (Struct) unter Verwendung - der Typen long, float, string sowie - bereits derart definierter Typen.

Tabelle 2-1: Elemente von Business-Prozessen in IBM FlowMark.

In der Aufbauorganisation werden alle *Mitarbeiter* definiert, die an der Ausführung von Workflows beteiligt sind. Mitarbeiter besitzen *Rollen*, gehören einem (Hierarchie-)*Level* und maximal einer *Organisation* an. Eine *Rolle* spezifiziert die Kompetenzen und Berechtigungen von Mitarbeitern zum Starten von *Prozessen* und Ausführen von *Aktivitäten*.

Ein Mitarbeiter kann mit mehreren Rollen und eine Rolle mit mehreren Mitarbeitern assoziiert sein. Jede Rolle besitzt einen *Koordinator*.
Eine *Organisation* stellt eine Einheit in einem Unternehmen dar (z.B. eine Arbeitsgruppe oder ein Profit-Center). Eine Organisation kann in einer anderen Organisation enthalten sein. Jede Organisation besitzt einen *Manager*.

> Durch die Festlegung von *Rollen* können Mitarbeiter in Gruppen eingeteilt werden, die von der Organisationsstruktur unabhängig sind. Dies ermöglicht die Modellierung von *Matrix-Strukturen* in Unternehmen, die eine Unternehmensgliederung in *administrativer* und *fachlicher Dimension* darstellen.

Weitere Details der Aufbauorganisation beschreibt [IBM96a].

Die wichtigsten Elemente der Ablauforganisation sind der *Prozeß*, die *Aktivität* sowie die *Kontroll-* und die *Daten-Konnektoren*.

Ein *Prozeß* enthält zusammengehörige Aufgaben (*Aktivitäten*, *Blöcke*), *Kontroll-* und *Daten-Konnektoren*.

> Eine Aktivität wird gemäß [WfMC96] wie folgt definiert:
>
> „A Description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources to support process execution; where human resource is required an activity is allocated to a workflow participant."

Eine Aktivität bildet damit *innerhalb eines Prozesses* die kleinste logische Einheit. Sie kann jedoch weiter untergliedert werden. Diese weitere Untergliederung kann durch die Möglichkeit der *Modellierung von Subprozessen* unterstützt werden.

Eine Aktivität kann eine *Programm-Aktivität* oder eine *Prozeß-Aktivität* sein. Aktivitäten können mit Mitarbeitern (im folgenden auch synonym als *Personen* bezeichnet), *Rollen*, *Levels* und *Organisationen* assoziiert sein. Damit werden Ausführungsberechtigungen spezifiziert. Zusätzlich kann für eine Aktivität der spätestmögliche Zeitpunkt für ihre Ausführung festgelegt werden, d.h. nachdem eine Aktivität durch die Abarbeitung eines Prozesses ausführbar wurde (siehe *Runtime)*, muß sie innerhalb dieser Zeit gestartet und beendet werden (*Duration of activity*). Erfolgt das nicht, so erhält ein durch direkte Angabe der Person oder über Rollenauflösung spezifizierter Mitarbeiter eine Nachricht. Für eine Reaktion dieses Mitarbeiters kann ebenfalls eine Zeit

angegeben werden (*Duration for making decision*). Nach deren Nichteinhaltung wird der Systemadministrator benachrichtigt.
Das Konstrukt der *Prozeß-Aktivität* ermöglicht eine hierarchische Darstellung der Struktur eines Prozesses. In ihr wird ein *Prozeß* spezifiziert, der bei Ausführung der *Prozeß-Aktivität* abläuft. Die Ausführung des Prozesses erfolgt *synchron*, d.h. die *Prozeß-Aktivität* wird erst nach erfolgter Beendigung des aufgerufenen (Sub-)Prozesses beendet. Der spezifizierte *Subprozeß* kann ein beliebiger, jedoch bereits vor der aktuellen Prozeß-Modellierung im WfMS definierter Prozeß sein. Die Möglichkeit, Subprozesse auszuführen, bietet neben dem Vorteil der hierarchischen und damit für den Prozeß-Modellierer *übersichtlichen Modellierung* gleichzeitig die Möglichkeit, daß einmal definierte Prozesse mehrfach *wiederverwendet* werden können.

Eine Variante der *Prozeß-Aktivität* ist der *Block*, der ebenfalls eine hierarchische Modellierung erlaubt. Im Gegensatz zur *Prozeß-Aktivität* ist der *Block* jedoch kein eigenständiger Prozeß, d.h. er kann nicht in beliebigen Prozessen wiederverwendet werden. In ihm definierte Prozeß-Strukturen sind Bestandteil des Prozesses, in dem der *Block* definiert wurde.
Neben der *hierarchischen* Modellierung eines Prozesses dient der *Block* weiterhin zur Modellierung eines *Bundles*.

Ein *Bundle* ist ein Konstrukt, das die Ausführung von bestimmten Aktivitäten (den im Block enthaltenen) durch *mehrere Nutzer* bewirkt. Im Gegensatz dazu können alle anderen Aktivitäten nur von *einem Nutzer* ausgeführt werden, selbst wenn sich mehrere Nutzer gemäß ihrer Berechtigungen qualifizieren. Ein *Bundle* kommt zum Einsatz, wenn alle Nutzer, die für die Ausführung einer Aktivität berechtigt sind, eine Aktivität ausführen *müssen* (z.B. Wahlvorgang: alle berechtigten Personen nehmen teil).

Die *Programm-Aktivität* ist die einzige Aktivität, deren Ausführung *produktiv* ist, d.h. es werden Eingangsdaten (oder -produkte) verarbeitet und Ausgangsdaten erzeugt. Dies geschieht durch die Ausführung eines Programms.

> Ein solches Programm ermöglicht dem Mitarbeiter, die ihm zugeordnete Aufgabe zu erfüllen (z.B. einfaches Textprogramm wie MS Word: Schreiben eines Briefes). Es können aber auch automatisierbare Aufgaben durch eine Aktivität bearbeitet werden (z.B. Datenbankanfragen oder komplizierte Berechnungen wie Chemotherapie-Planungen, siehe auch [Be97]).

Eine Programm-Aktivität ruft dabei immer ein *Programm* auf, das symbolisch im WfMS definiert wurde.

Die Definition eines *Programms*, unabhängig von der Aktivität, kapselt die physischen Randbedingungen eines Programmaufrufs wie Betriebssystem (*MS Windows 3.11, MS WindowsNT, HP-UX, AIX, OS/2*) und die Art des auszuführenden Programms (ausführbare Dateien des jeweiligen Betriebssystems, *DLL*'s unter Windows und OS/2). Für die Ausführung physischer Programme kann deren Ausführungsumgebung in der Programm-Definition festgelegt werden.
Programme verarbeiten Daten, die in externen Datenquellen abgelegt sind oder ihnen vom WfMS übergeben werden. Die Datenübergabe durch das WfMS erfolgt über Kommandozeilen-Parameter und (über ein *Application Programming Interface - API*) durch *Daten-Container*. Jedes Programm besitzt hierzu einen *Input-Container*, der die zu verarbeitenden Daten enthält, und einen *Output-Container*, in den die Ausgabedaten des Programmes oder der Aktivität eingetragen werden.
Analog zur Programm-Definition besitzt jeder *Prozeß*, jede *Aktivität* und jeder *Block* einen *Input-* und einen *Output-Container*.

Das Ende einer Aktivität wird durch eine *Abschluß-Bedingung* (Exit Condition) festgestellt. Die Syntax dieser Bedingung entspricht der Syntax der Übergangsbedingungen von *Kontroll-Konnektoren* und wird zusammen mit diesen beschrieben. Wird eine Abschluß-Bedingung positiv evaluiert, so wurde die Aktivität erfolgreich beendet. Im Falle einer negativen Auswertung der Bedingung wird die Aktivität erneut zur Ausführung gebracht. Damit ist es möglich, *Schleifenkonstrukte* in einem Prozeß darzustellen.

> Die einfache Form entspricht einer Schleife mit abschließender Bedingung. Durch einen *Kontroll-Konnektor* mit einer Übergangsbedingung, dessen Ziel die Aktivität ist, und durch die Rückführung eines *Daten-Konnektors* auf die gleiche Aktivität erhält man eine Schleife mit Eintrittsbedingung.

Für die Definition der Reihenfolge der Abarbeitung von Aktivitäten und Blöcken innerhalb eines Prozesses dient der *Kontroll-Konnektor*. Ein *Kontroll-Konnektor* verbindet immer zwei Aktivitäten, wobei die eine als Quelle und die andere als Ziel fungiert. Damit wird festgelegt, daß die Ausführung der Ziel-Aktivität niemals vor der Beendigung der Quell-Aktivität erfolgen darf. Ob eine Aktivität ausgeführt wird, hängt zudem aber noch von weiteren Faktoren ab. Ein *Kontroll-Konnektor* ist mit einer *Übergangsbedingung* (Transition Condition) behaftet. Eine Übergangsbedingung ist eine aussagenlogische Formel. Diese Formel kann aus den in Tabelle 2-2 aufgeführten Operatoren, Funktionen und Operanden bestehen.
Die Anwendbarkeit der einzelnen Operatoren und Funktionen für die unterschiedlichen, in IBM FlowMark konstruierbaren Datentypen wird in [IBM96a] detailliert in Form einer kontextfreien Grammatik definiert. Die in Übergangsbedingungen verknüpften bzw. verglichenen Werte können

Konstanten oder Elemente von Output-Containern beliebiger Aktivitäten des aktuellen Prozesses sein.

Operatoren		
	Logisch	AND OR NOT, ! (Negation)
	Vergleich	= <> > < >= <=
	Arithmetisch	+ - (unär) - (binär) *, MUL (Multiplikation) /, DIV (Division) MOD
Funktionen (String-Operationen)		VALUE (Umwandeln eines Strings in eine Zahl) SUBSTR LOWER UPPER
Hilfszeichen		( )
Operanden		Konstanten von FlowMark-Basistypen Variable eines Output-Containers, referenziert durch %<Variablenname>%, wobei <Variablenname> ein Element einer beliebigen Datenstruktur einer beliebigen Aktivität des Prozesses ist

Tabelle 2-2: IBM FlowMark 2.3 : Operatoren, Funktionen und Operanden in Bedingungen. Nach [IBM96a].

Die mit *Übergangsbedingungen* behafteten *Kontroll-Konnektoren* ermöglichen neben der einfachen Festlegung der Reihenfolge der auszuführenden Aktivitäten zusätzlich die Darstellung von *Alternativen* und *Parallelisierungen* im Arbeitsfluß. In Abhängigkeit von der Formulierung der Übergangsbedingungen können mit den *Kontroll-Konnektoren* verschiedene Arten von *Verzweigungen* und *Parallelisierungen* im Prozeß modelliert werden:

- Oder-Verzweigung: Laufen aus einer Aktivität mehrere *Kontroll-Konnektoren* heraus, so *können mehrere Bedingungen erfüllt* sein.

- Und-Verzweigung (Parallelisierung): Laufen aus einer Aktivität mehrere *Kontroll-Konnektoren* heraus, so *müssen alle Bedingungen erfüllt* sein. Typischerweise ist die Bedingung aller Konnektoren gleich, oder es wurde keine Bedingung angegeben.

- Exklusiv-Oder-Verzweigung: Laufen aus einer Aktivität mehrere *Kontroll-Konnektoren* heraus, so darf nur genau eine Bedingung erfüllt sein. Typischerweise kann diese Form der Verzweigung erreicht werden, indem sich die *Übergangsbedingungen* der *Kontroll-Konnektoren* gegenseitig ausschließen. Zusätzlich gibt es das Konstrukt des *Default-Konnektors*. Läuft ein solcher Konnektor aus einer Aktivität heraus, so wird er genau dann positiv evaluiert, wenn die Übergangsbedingungen aller anderen aus der Aktivität auslaufenden *Kontroll-Konnektoren* bei ihrer Evaluierung zum negativen Resultat führten.

Die so im Prozeß entstandenen parallelen Pfade können wieder miteinander verknüpft werden. Haben mehrere *Kontroll-Konnektoren* die gleiche Aktivität als Ziel, so wird die Aktivität (bei der Prozeß-Modellierung definiert) genau dann als ausführbar gekennzeichnet[7], wenn die *Übergangsbedingung* mindestens eines (*Oder*- oder *Exklusiv-Oder-Verknüpfung*; in vielen WfMS oder Prozeß-Modellierungswerkzeugen auch als *Or-Join* bezeichnet) oder aller einlaufenden Kontroll-Konnektoren (*Und-Verknüpfung*; *And-Join*) positiv evaluiert wurde.

Eine Konstruktion von *Schleifen* durch die Bildung eines Kreises aus *Kontroll-Konnektoren* ist in IBM FlowMark nicht möglich. Schleifen werden ausschließlich durch die Anwendung von *Abschluß-Bedingungen* von *Aktivitäten* und *Blöcken* realisiert.

Daten-Konnektoren kennzeichnen den *Daten*- oder *Produktfluß* in einem Arbeitsprozeß. Ein *Daten-Konnektor* spezifiziert die Weiterleitung der Daten von der *Erzeuger-Aktivität* der Daten zur *Verbraucher-Aktivität*.
Der Datenfluß ist nicht notwendigerweise mit dem Kontrollfluß identisch.

> Trotzdem beeinflußt der Datenfluß die Kontrollstruktur: Erst wenn alle Daten für eine Aktivität auf die vom *Daten-Konnektor* beschriebene Weise vorliegen, kann sie bei positiver Evaluierung der entsprechenden *Kontroll-Konnektoren* ausgeführt werden.

Neben der Spezifizierung von *Erzeuger*- und *Verbraucher-Aktivitäten* kann als Bestandteil eines *Daten-Konnektors* ein *Mapping* vom *Output-Container* der *Erzeuger-Aktivität* in den *Input-*

[7] Falls die benötigten Daten vorliegen.

Container der *Verbraucher-Aktivität* definiert werden. Dies bedeutet, daß eine oder mehrere Abbildungsvorschriften zwischen Elementen dieser beiden Container festgelegt werden.

Neben den ersten beiden Bestandteilen des Business-Prozesses (Aufbau- und Ablauforganisation) gibt es als dritte Komponente die *Produkte* oder *Daten* (im folgenden als *Daten* bezeichnet). Die Konstruktion von *Datentypen* und *Datenstrukturen* wurde bereits in Tabelle 2-1 skizziert. Die konstruierten Datentypen finden in IBM FlowMark Eingang in die *Daten-Container*. Jeder *Daten-Container* ist eine Instanz einer im WfMS konstruierten Datenstruktur.
Eine Übersicht über die Basis-Datentypen und ihre Wertebereiche gibt Tabelle 2-3.

Long	$[-2^{31}; 2^{31}-1]$
Float	$[-2^{31}; 2^{31}-1]$
String	Keine Angabe zur maximalen Länge in der Dokumentation von IBM FlowMark Länge bis zu 3000 Zeichen, gemäß IBM Vertreter Länge bis zu 99999 Zeichen, eigener Test unter OS/2 mit FlowMark-Rexx-API Länge bis zu 512 Zeichen, eigener Test mit Buildtime-Tool (String als Default-Wert)

Tabelle 2-3: IBM FlowMark 2.3: Basis-Datentypen und ihr Wertebereich.[8] [9]

Für die Konstruktion von *Aggregationen* (*Structs*) und *Mengentypen* (*Arrays*) gilt folgende Einschränkung (data structure = structure ; entspricht einem beliebigen Datentyp; [IBM96a], S.7):

> „A data structure is an ordered list of variables, called members, that have a name and a data type. The maximum size of a data structure is 256 member items. Data types are string, long, floating point, and structure. The FlowMark workflow manager supports one-dimensional arrays of any of these data types, with a maximum size of 256 elements."

Die gleiche Quelle beschreibt einen ähnlichen Sachverhalt im Zusammenhang mit der Modellierung unter Verwendung des Buildtime-Tools auf S. 78 allerdings auch noch etwas anders:

> „... you define the member as an array. The maximum size that you can specify is 512 elements...
> Note: You can have up to 512 member items in a data structure, including member items of any nested data structures and elements of arrays."

Insbesondere werden also über die maximale Größe eines Arrays widersprüchliche Angaben gemacht.

[8] Die Wertebereiche für long- und float-Datentypen wurden in der Dokumentation von IBM FlowMark gleich angegeben ([IBM96a], S.193/194).
[9] Da in der Dokumentation von IBM FlowMark keine Angaben über die maximale Länge von Strings enthalten waren, wurde ein Vertreter von IBM befragt. Dessen Angabe stimmte jedoch nicht mit eigenen Tests überein. Ein Test erfolgte über die von IBM FlowMark angebotene Rexx-API. Dabei wurde festgestellt, daß die Länge der Strings, die über die API mit IBM FlowMark ausgetauscht werden können, kleiner als die maximale String-Länge in der Skript-Sprache Rexx ist. Ein zweiter Test wurde mit dem Buildtime-Tool von IBM FlowMark durchgeführt. Dabei wurde für ein String-Element des Input-Containers einer Aktivität ein Default-Wert festgelegt. Dieser konnte im entsprechenden Eingabefeld nur mit einer maximalen Länge von 256 Zeichen angeben werden.

Der Datenaustausch zwischen WfMS und Applikation erfolgt über die *Daten-Container* (unter Verwendung der FlowMark-API) und durch *Kommandozeilen-Parameter*, die in den symbolischen Programm-Definitionen vereinbart werden.

2.2.1.2 Buildtime

Für die Ausführung von Arbeitsabläufen als Prozesse in einem WfMS ist es zunächst nötig, alle dafür relevanten Elemente des Business-Prozesses zu modellieren. Die Modellierung wird oft auch als *Buildtime* bezeichnet. Es werden alle Elemente der *Aufbau-, Ablauf- und Datenorganisation* sowie technische Details wie der Speicherort von Programmen, ausführende Systeme (Server-Definitionen; dienen in der Prozeß-Aktivität zur Spezifizierung anderer FlowMark-Server, auf denen ein Subprozeß ausgeführt werden kann), Rechner und Datenbanken festgelegt. In einer Datenbank werden die Prozeß-Definitionen abgelegt. Gleichzeitig ist die Datenbank für eine persistente Speicherung der Zustände aller Workflows zuständig. Der Zustand eines Workflows ist die Information über den Zustand des Prozesses[10] selbst, den Zustand aller Aktivitäten[11] sowie die Menge der Instanzen aller Datenstrukturen (*Input-* und *Output-Container*). Die WfMS-interne Datenbank ist bei IBM FlowMark 2.3 *ObjectStore, Version 4.0* oder *5.0*[12] [13].
IBM FlowMark bietet für die Modellierung zwei Möglichkeiten.
Einerseits kann ein graphisches Tool verwendet werden, der *Buildtime-Client*.

> Mit ihm können alle Elemente modelliert und teilweise bereits durch einen eingebauten Test (z.B. Default-Werte in Datenstrukturen) auf ihre korrekte Definition überprüft werden. Nach Abschluß einer Prozeß-Modellierung erfolgt ein Übersetzungsvorgang, in dem der modellierte Prozeß in eine vom WfMS interpretierbare Form umgewandelt (kompiliert) wird. Dabei erfolgen weitere Tests auf eine korrekte Workflow-Definition.

Alternativ zur ersten Variante der Modellierung können Prozesse auch als Text-Beschreibung durch ein weiteres Tool (im weiteren als *Import-Tool* bezeichnet) aus einer Datei in das WfMS importiert und gegebenenfalls kompiliert werden.

---

[10] Der Zustand von Prozessen kann *ready* (der Prozeß kann gestartet werden), *running* (der Prozeß wird ausgeführt), *suspended* (der Prozeß wurde unterbrochen), *terminated* (der Prozeß wurde abgebrochen) oder *finished* (der Prozeß wurde erfolgreich beendet) sein ([IBM96b]).
[11] Der Zustand von Aktivitäten kann *idle* (die Startbedingung der Aktivität wurde noch nicht evaluiert), *ready* (die Aktivität kann gestartet werden), *running* (die Aktivität wird gerade ausgeführt), *suspended* (die Prozess-Instanz besitzt den Status *suspended*), *force-finished* (die Prozeß-Instanz wurde von einem Nutzer beendet), *finished* (die Aktivität wurde erfolgreich ausgeführt), *skipped* (die Startbedingung der Aktivität wurde mit negativem Ergebnis evaluiert) oder *disabled* (die Aktivität kann vom aktuellen Nutzer nicht bearbeitet werden) sein ([IBM96b]).
[12] Im Verlauf der Arbeit erfolgte ein Update auf ObjectStore 5.0.
[13] Im FlowMark-Nachfolge-Produkt *MQSeries Workflow* wird das IBM Produkt *DB2* verwendet.

Die Text-Beschreibung muß dabei einer *kontextfreien Grammatik* genügen. Die durch diese Grammatik definierte Sprache ist die *FlowMark Definition Language* (*FDL*). Diese Sprache ist in [IBM96a] beschrieben. Mit Kenntnis der Grammatik von FDL lassen sich die Beschreibungen der Elemente von Workflows erzeugen. Das kann durch einen einfachen Text-Editor, aber auch durch ein eigenständiges *Buildtime-Tool* erfolgen.

Als Nachteil von FDL ist hervorzuheben, daß die angegebene Syntax nicht vollständig ist.
Es werden *Strings* und *Identifier* (Bezeichner für Elemente der Input- und Output-Container) verwendet, deren korrekte Syntax und Semantik jedoch kontextabhängig sind. Beispielsweise kann der Name einer Datenbank maximal 8 Zeichen lang sein, während der Name eines Prozesses 32 Zeichen umfassen kann. Ferner können Namen als String oder Identifier angegeben werden.
Bedingungen sind ebenfalls nur in Form von Strings, nicht aber als kontextfreie Regeln Bestandteil von FDL.
Ferner gibt es Regeln, die gemäß der Grammatik alternativ oder in beliebiger Anzahl und Reihenfolge angewandt werden können. Trotzdem schließen sie sich teilweise sogar gegenseitig aus, ihre syntaktisch vorgegebene Beliebigkeit ist semantisch falsch und muß daher in einem Semantik-Test von IBM FlowMark überprüft werden. Dieser Semantik-Test läßt sich allerdings auch durch Einführung weiterer kontextfreier Regeln als Syntax-Test durchführen.

> Der gegenseitige Ausschluß von Regeln wird kurz skizziert: Ausführungsberechtigungen von Aktivitäten können entweder dynamisch zur Laufzeit (*Runtime*) aus Werten des Input-Containers oder statisch zur *Buildtime* in Form einer Assoziierung mit Rollen oder durch Spezifizierung einer Person festgelegt werden. Die entsprechenden Regeln der Grammatik erscheinen alternativ und beliebig kombinierbar. Trotzdem schließen sie sich aus, der Ausschluß wird im graphischen *Buildtime-Client* sogar zwingend vorgeschrieben.

Ein weiterer Unterschied zwischen der angegebenen Grammatik und dem System zeigt sich in der Definition von *Duration of activity* und *Duration of making decision*. Während sowohl die Grammatik als auch die Beschreibung in [IBM96a] lediglich Regeln für eine Zeitspezifikation in Tagen und Stunden enthalten, können im *Buildtime-Client* zusätzlich Minuten angegeben werden. Der Export eines derart definierten Prozesses zeigt, daß in der FDL-Ausgabe-Datei eine Minutenangabe enthalten ist.

Das Ergebnis der Prozeß-Modellierung mit Übersetzung sind *Prozeß-Templates, Programm- und Server-Definitionen* sowie *Organisations* –und *Datenstrukturen*.

2.2.1.3 Runtime

Die Ausführung der zur *Buildtime* modellierten Prozesse erfolgt zur *Runtime.*
Dies geschieht durch die Generierung einer *Prozeß-Instanz* aus einem *Prozeß-Template*, das mit instanzspezifischen Daten parametrisiert wird. Instanzspezifische Daten sind die *Instanz-ID* und dynamisch aufgelöste *Ausführungsberechtigungen* wie Rollen, Levels und Organisationen.

> Für die Interaktion mit dem WfMS steht einem *Nutzer* (einem Benutzer, der im WfMS als Person repräsentiert wird) der *Runtime-Client* zur Verfügung. Dieser enthält neben einer *Arbeitsliste* (auch als *Worklist* bezeichnet) mit allen für den Nutzer gemäß seiner Berechtigungen ausführbaren Aktivitäten eine Liste aller Prozesse, für deren Start er als Person qualifiziert ist.

Ferner können von einem Nutzer Initialwerte für die Datenstruktur des Input-Containers des instanzierten Prozesses angegeben werden, sofern diese Möglichkeit während der Buildtime spezifiziert wurde.
Der Nutzer wählt für den Start eines Prozesses das entsprechende Prozeß-Template aus der Liste der verfügbaren Prozesse aus.
Nach dem Start des Prozesses werden bei entsprechender Evaluierung der *Kontroll-Konnektoren* Aktivitäten ausführbar. Das führt dazu, daß eine entsprechende Nachricht in der *Worklist* aller für die Ausführung dieser Aktivität berechtigten Nutzer erscheint. Sobald ein Nutzer diese Aktivität startet, ist sie für alle anderen Berechtigten nicht mehr ausführbar. Eine Ausnahme ist ein *Bundle*. In diesem Fall müssen alle Nutzer die Aktivität ausführen.
Wurden alle *Kontroll-Konnektoren* im Workflow evaluiert, so kann der Prozeß beendet werden.

> Es werden alle *Kontroll-Konnektoren* eines Pfades negativ evaluiert, wenn feststeht, daß der Pfad im Workflow niemals durchlaufen wird. Das bedeutet, daß ein im Pfad enthaltener und positiv evaluierbarer *Kontroll-Konnektor* ebenfalls negativ ausgewertet wird, wenn er nur über einen bereits negativ evaluierten *Kontroll-Konnektor* erreicht werden kann.
>
> Während die negative Evaluierung für ganze Pfade erfolgt, wird bei der positiven Evaluierung nur der aktuelle *Kontroll-Konnektor* betrachtet.

Treten während der Ausführung eines Workflows externe Ereignisse, wie ein Systemfehler, auf einem Server-Rechner (FlowMark- oder Datenbank-Server) oder auf einem Client-Rechner auf, so wird dies unterschiedlich behandelt.
Wie im Buildtime-Abschnitt bereits ausgeführt, wird der aktuelle Systemzustand immer in der Datenbank abgelegt. Dadurch ist es möglich, im Falle eines Fehlers des FlowMark-Servers durch dessen Neustart mit der Abarbeitung eines Workflows nach Behebung der Probleme fortzufahren.

> Einschränkungen gibt es, wenn zum Zeitpunkt des Systemfehlers Applikationen liefen, die potentiell datenmanipulierend (d.h. die Applikation hat in Abhängigkeit von ihrer eigenen Ablaufstruktur Daten zu lesen oder zu schreiben) auf eine Datenquelle zugreifen. Durch das Fehlen eines verteilten *Commit-Protokolls* (z.B. *2-Phasen-Commit*) zwischen WfMS und Applikation können deshalb Daten-Inkonsistenzen in einer externen Datenquelle entstehen. Kommuniziert eine Applikation während ihrer Ausführung datenmanipulierend mit einem DBMS, so kann später, nach ihrem Neustart, vom WfMS nicht mehr eindeutig festgestellt werden, ob die entsprechenden Daten bereits verändert wurden. Im Falle eines ausschließlich lesenden Zugriffs der Applikation auf externe Daten entstehen keine Inkonsistenzen.

Falls der Fehler jedoch im (Betriebs-)System des WfMS-Datenbank-Servers auftritt, so ist es schwieriger, einen konsistenten Zustand herzustellen. Obgleich ObjectStore als Datenbank-Server weitreichende *Persistenz* gewährleisten sollte, ist diese praktisch nicht gegeben.

> Insbesondere erfolgt ein *Logging* der ausgeführten Datenbankoperationen ungenügend. Die angebotene Funktionalität ist zweistufig. Es ist vorgesehen, daß vom Administrator ein manuelles *Backup* generiert wird. Um auch die Veränderungen des Datenbank-Zustandes im Zeitraum zwischen den Backups zu protokollieren, gibt es ein Tool, das in Form von *Schnappschüssen* der Datenbank die entsprechende Information entnimmt. Dieses Tool ist als einfaches Programm unabhängig von der Datenbank zu starten. Das Zeitintervall für die Schnappschüsse wird vom Administrator festgelegt. Offensichtlich kann dadurch jedoch nicht die gesamte Information nach einem Datenbank-Fehler rekonstruiert werden. Es erfolgt lediglich eine Wiederholung für Transaktionen (*Redo*), die zur Zeit des letzten Schnappschusses erfolgreich abgeschlossen waren ([IBM96d]). Nur für diese Transaktionen wird die Persistenz gewährleistet.

Treten Systemfehler auf einem Workflow-Client-Rechner auf, so können diese ebenfalls zu größeren Inkonsistenzen des Gesamtsystems führen. Ist vom Systemfehler lediglich die Arbeitliste

eines Nutzers betroffen, so kann deren Inhalt nach Problembehebung wieder vom FlowMark-Server abgefragt werden. Somit führt dies zu keiner Inkonsistenz.
Schwieriger wird es, wenn auf dem Workflow-Client-Rechner zum Zeitpunkt des Systemfehlers eine Programm-Aktivität in Form einer automatischen oder manuellen Applikation ausgeführt und deren Ausführung vom Systemfehler beeinträchtigt wurde.

> Aufgrund der Protokollierung des Workflow-Zustandes in der WfMS-Datenbank und einer (einfachen) Überwachung der Ausführung der Applikation ist es für das WfMS möglich, festzustellen, ob eine Applikation erfolgreich ausgeführt oder abgebrochen wurde. Im letzeren Fall wird die Applikation in den Arbeitslisten der Nutzer erneut als ausführbar gekennzeichnet. Damit tritt das gleiche Problem wie oben auf. Greift eine Applikation potentiell datenmanipulierend auf externe Daten zu, so kann aufgrund des Fehlens eines verteilten *Commit-Protokolls* vom WfMS nicht festgestellt werden, ob beim Applikationsabbruch bereits Daten verändert wurden.

Im nächsten Abschnitt wird der CORBA-Standard betrachtet und dessen relevanten Eigenschaften für eine Integration herausgearbeitet.

2.2.2 CORBA

2.2.2.1 Einführung

Die *Common Object Request Broker Architecture* (*CORBA*) wurde Anfang der 90er Jahre als Standard der *Object Management Group* (*OMG*) eingeführt. Ziel dieses Standards ist es, dem Anwendungsentwickler ein Werkzeug bereitzustellen, welches die Erstellung von Applikationen in verteilten Systemen wesentlich erleichtert. Folgende Ziele werden angestrebt ([Ba97]):

(1) Vereinfachung der Implementierung von Applikationen, die Komponenten auf verschiedenen Hosts im Netzwerk besitzen oder mit verschiedenen Programmiersprachen implementiert werden müssen.
(2) Ermöglichung der Erstellung von offenen Anwendungen, die später auch als Komponenten von größeren Systemen genutzt werden können.

Um diese Ziele zu erreichen, ist eine Anzahl von Teilzielen zu erfüllen:

- Client/Server-Applikationen statt monolithischer Mehrbenutzer-Anwendungen,

- aber trotzdem eine Weiterverwendung von Legacy-Systemen[14] oder Daten solcher Systeme,
- verteilte Anwendungen, die über ein Netzwerk miteinander kommunizieren,
- Netzwerk- bzw. –Protokoll-Unabhängigkeit der Anwendungen,
- maximale Kapselung der einzelnen verteilten Komponenten bzgl. der Rechner-Plattform, der Implementierungssprache und des implementierten Algorithmus,
- einfacher Code, der im wesentlichen die Implementierung der Funktion der Komponente enthält und stark von der Lokation von anderen Komponenten oder Daten abstrahiert, somit wartungsfreundlich und unabhängig von Systemveränderungen ist, sowie
- die Wieder- und Mehrfachverwendung von Software-Modulen.

Die Komponenten des CORBA-Standards sind *Objekte*. Im Gegensatz dazu werden z.B. bei der Verwendung des *Remote Procedure Call* (*RPC*) *Funktionen* und *Prozeduren* verteilt implementiert.
Beide Systeme schaffen die Möglichkeit, über Grenzen, wie Netzwerke, verschiedene Betriebssysteme und unterschiedliche Programmiersprachen, hinweg zu kommunizieren. Ein großer Vorteil von CORBA gegenüber RPC ist die Schaffung eines kommerziellen Standards ([Ba97]).

Aufgrund seiner beiden Ziele unterstützt CORBA eine *Client/Server-Architektur*. Ein *Client/Server-System* besteht aus einer Menge von *Client-Programmen*, die mit anderen, im System verteilten Komponenten, den *Servern*, kommunizieren. Dabei enthält ein Client meist nur wenig Applikationslogik, sondern hauptsächlich Funktionen, die für eine Kommunikation mit dem Server nötig sind. Soll eine Interaktion zwischen Client und Nutzer stattfinden, so sind diese Funktionen ebenfalls Bestandteil des Clients. Ein Server kann seinerseits wiederum ein Client eines anderen Servers sein. Das ermöglicht eine vollständige Dekomposition eines Systems in einzelne Komponenten. Dadurch können die CORBA-Ziele bereits für kleinste Anwendungskomponenten realisiert werden.

Eine Implementierung des CORBA-Standards wird *Object Request Broker* (*ORB*) genannt. Ein ORB muß die Kommunikation in einem Netzwerk, zwischen Betriebssystemen und Programmiersprachen ermöglichen. Damit eine Implementierung mit allen anderen kommunizieren kann, wird das *Internet Inter-ORB Protocol* (*IIOP*) verwendet. Dieses Protokoll geht aus dem *General Inter-ORB Protocol* (*GIOP*) hervor, das als Nachrichten-Format über das *Transmission*

[14] Alt-Systeme.

Control Protocoll/Internet Protocol (*TCP/IP*) gesendet wird. Die über IIOP versendeten Packete enthalten die Identifikation des Zielobjektes, den Namen der auszuführenden Methode sowie die Parameter. Damit ist es möglich, daß jeder *CORBA-Client* mit jedem *CORBA-Objekt* kommunizieren kann.

Eine Eigenschaft des objekt-orientierten Paradigmas, das in CORBA-Systemen verwendet wird, ist die Kommunikation zwischen Objekten über Schnittstellen (*Interfaces*). Ein solches *Interface* definiert alle Dienste, die ein Objekt für andere Objekte bereitstellt. Implementierungsdetails sind damit Interna jedes Objektes und ermöglichen die saubere Kapselung.
CORBA läßt es zu, verschiedene Implementierungssprachen für die einzelnen Objekte zu verwenden. Diese können z.B. C, C++, COBOL oder Java sein. Deshalb erfolgt die Kapselung nicht nur bezüglich des implementierten Algorithmus, auch die Implementierungssprache wird als interne Eigenschaft eines Objektes gekapselt. Dadurch wurde es notwendig, eine allgemeine Sprache für die *Schnittstellen-Definition* festzulegen. Diese im CORBA-Standard verwendete Sprache ist die *Interface Definition Language* (*IDL*). IDL enthält gemäß ihres Verwendungsbereiches keine Konstrukte wie *Variable, Statements* oder *Kontrollstrukturen*. Mit Hilfe einfacher Konstruktionsmechanismen können unter Verwendung einfacher CORBA-Datentypen (z.B. *unsigned long*) beispielsweise *Aggregations*- (*Struct*-) und *Mengen*- (*Array, Sequence*) Typen definiert werden. Wichtigster Bestandteil einer Definition in IDL ist jedoch die Interface-Konstruktion. Sie entspricht grundsätzlich einer Schnittstellen-Definition in C++ (public) oder Smalltalk. Alle angegebenen Merkmale werden als Schnittstelle allen anderen CORBA-Objekten im System über ein *Interface Repository* bekannt gemacht. Die Schnittstellen-Definition kann nun in die *Implementierungssprache* eines Client-Objektes (z.B. Java) und die Implementierungssprache eines Server-Objektes (z.B. C++) übersetzt werden. Dadurch wird es möglich, im Methodenaufruf-Format der jeweiligen Implementierungssprache mit den CORBA-Objekten zu kommunizieren.
Die Aufrufe von Methoden eines Objektes einer in einer Interface-Definition deklarierten Klasse werden über ORB und IIOP vom Client zum Server-Objekt übertragen und Ergebnisse an den Client zurückgeliefert. Gegenwärtig kann IDL in Objekt-Klassen der Implementierungssprachen *C++, C, Smalltalk, Ada, OLE (Visual Basic, Delphi), COBOL* und *Java* übersetzt werden.

Im folgenden sollen neben einer groben Darstellung des CORBA-Standards besonders die *CORBA-IDL-Datentypen und –Methoden* im Mittelpunkt stehen.

2.2.2.2 Der CORBA-Standard

Ein CORBA-System setzt sich aus verschiedenen technischen Komponenten zusammen. Ähnlich wie bei WfMS sollen diese auch in diesem Zusammenhang in *Buildtime*- und *Runtime*-Komponenten eingeteilt werden.
Um ein CORBA-System aufzubauen, ist es notwendig, neben der Schaffung der technischen Voraussetzungen wie *Server, Tools und Kommunikationskomponenten* auch die eigentlichen *Anwendungen* zu erstellen. Gemäß dem Konzept von CORBA geschieht dies in einer stark gekapselten Form. Zunächst erfolgt die *Definition von Schnittstellen* für die Kommunikation zwischen den einzelnen CORBA-Objekten. Nach ihrer Definition sind die Schnittstellen mittels eines Tools auf syntaktische und semantische Korrektheit (z.B. bei Referenzierung anderer Datentypen) zu überprüfen. Ergibt die Überprüfung ein positives Ergebnis, so werden die Definitionen „kompiliert“, d.h. sie werden in das *Interface Repository*, das ebenfalls ein CORBA-Objekt ist, eingetragen. Damit wird die Deklaration für alle CORBA-Objekte des Systems verfügbar. Gleichzeitig mit der Kompilierung der Schnittstellen-Definition können für beliebige Implementierungssprachen *Code-Fragmente* erzeugt werden.

> Die Code-Fragmente werden in zwei Kategorien eingeteilt (Abbildung 2-1): *IDL stub* (Code-Fragment für einen CORBA-Client) und *IDL skeleton* (Code-Fragment für ein CORBA-Server-Objekt). Die Code-Fragmente enthalten die für eine transparente Kommunikation von CORBA-Objekten nötige Information (Netzwerk-Kommunikation, gekapselt in Hilfs-Datentypen usw.).

Mit Hilfe dieser Code-Fragmente und weniger, speziell auf eine CORBA-Kommunikation ausgerichteter Methoden können nun CORBA-Anwendungen erstellt werden. Dies kann einfach geschehen, indem ebenfalls mit dem IDL-Compiler ein zusätzliches Code-Fragment für die Klasse mit der definierten Schnittstelle generiert wird.

> Unter Verwendung dieses einfachen Code-Fragments kann die Applikationslogik nun direkt implementiert werden. Methodisch sauberer ist die Trennung von Applikations- und Kommunikationslogik. Das läßt sich durch die Implementierung zweier Klassen realisieren, wobei die eine als CORBA-Objekt ein Objekt der Klasse mit der Applikationslogik enthält.

Die erstellte Applikation wird schließlich mit einem Compiler der Implementierungssprache unter zusätzlicher Verwendung von CORBA-Bibliotheken in ausführbaren Code übersetzt. Das ebenfalls

zu erstellende (Server-)Programm ist lediglich für die *Instantierung* des CORBA-Objektes zuständig.

Die Erstellung eines CORBA-Clients verläuft ähnlich.

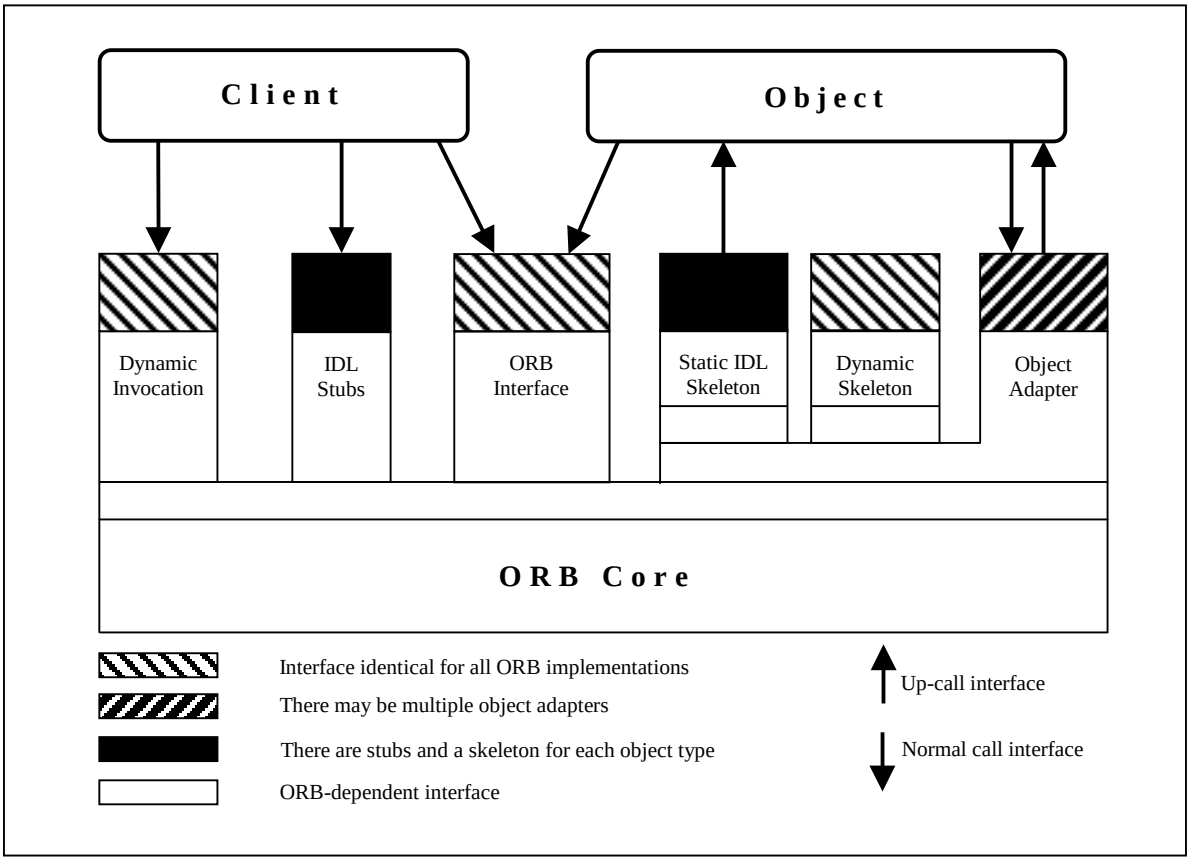


Abbildung 2-1: CORBA: Basiselemente. Nach [OMG98].

Anstelle der Instantierung tritt hierbei die über spezielle CORBA-Objekte und –Methoden erfolgende Identifizierung und Anwendung eines bereits existierenden CORBA-Objektes. Das kann über CORBA-weit eindeutige Referenzen erfolgen.

Ein anderer Weg ist die Verwendung eines Name-Service, mit dessen Hilfe CORBA-Objekten eindeutige Namen vergeben werden können.

Der *Aufruf* einer Methode eines CORBA-Objektes durch einen Client kann auf zwei verschiedene Weisen erfolgen:

- statisch: Die Schnittstellen-Deklaration des aufgerufenen CORBA-Objektes ist als *IDL stub* Bestandteil des Clients (*Static Invocation Interface*; *SII*). Methodenaufrufe von Objekten, die dieser Schnittstellen-Definition entsprechen, können in der Implementierungssprache fest kodiert werden.

  Nachteil: Sollen vom Client Objekte neuer Klassen, die erst nach seiner eigenen Kompilierung erstellt wurden,

angesprochen werden, erfordert das eine Änderung des Clients und dessen Neukompilierung.

Vorteil: Es können bereits während der Kompilierung alle Syntax- und Semantik-Tests durch den Compiler vorgenommen werden.

- dynamisch: Die Schnittstellen-Deklaration des aufgerufenen CORBA-Objektes ist nicht (als *IDL stub*) Bestandteil des Client.
Mittels eines anderen CORBA-Objektes, des *Dynamic Invocation Interface* (*DII*), können Methodenaufrufe zur Laufzeit spezifiziert und durch das *DII* ausgeführt werden. Unter Verwendung des *Interface Repository* kann das *DII*, aber auch der Client, die Schnittstellen-Deklaration feststellen. Mit Hilfe dieser Information kann der konkrete Methodenaufruf vom Client generiert und vom *DII* getestet und ausgeführt werden.

Nachteil: Der Syntax- und Semantik-Test kann während der Kompilierung nicht vorgenommen werden und wird deshalb in den Client verlagert. Das bedeutet insbesondere einen höheren Verarbeitungsaufwand, der sich im Zeit-, aber auch im Ressourcenverbrauch niederschlagen kann.

Vorteil: Um vom Client Objekte neuer CORBA-Klassen ansprechen zu können, ist keine Neukompilierung nötig.

Die *Ausführung* einer Methode auf der Server-Seite kann ebenfalls statisch über das *Static Skeleton Interface* (*SSI*) und dynamisch über das *Dynamic Skeleton Interface* (*DSI*) erfolgen. Dabei wird in der Regel die statische Methode (*SSI*) unter Verwendung des *IDL skeletons* befolgt ([Ba97]). Ein Server-Programm, das ein CORBA-Objekt instanziert, verfügt dabei über die Schnittstellen- und die Klassen-Deklaration und –Definition des zu instanzierenden Objektes.
Die Deklarationen und Definitionen sind jedoch bei der Verwendung des *DSI* nicht nötig.

Über das *DSI* können Objekt-Implementierungen verwendet werden, deren Schnittstellen zur Kompilierung eines CORBA-Servers nicht bekannt sind. Gemäß [Ba97] wurde das *DSI* als Zusatz des CORBA-Standards für den Aufbau von *Gateways*, als Schnittstelle zwischen CORBA und Nicht-CORBA-Systemen (z.B. DCOM), entwickelt. Weitere Informationen über die statischen und dynamischen Kommunikationsmöglichkeiten sind in [Ba97] enthalten.

Die *Runtime-Umgebung* eines CORBA-Systems ist ebenfalls in Abbildung 2-1 dargestellt. CORBA-Client und CORBA-Objekt kommunizieren über einen (Client und Objekt befinden sich auf dem gleichen Rechner) oder zwei *Object Request Broker* (*ORB*). Zentrale Aufgabe eines ORB ist es, die technischen Laufzeit-Voraussetzungen für die Erstellung verteilter Systeme unter Verwendung verschiedener Implementierungssprachen und Betriebssysteme bereitzustellen. Dazu wird mittels eines Objekt-Adapters, meist des *Basic Object Adapters* (*BOA*), eine Übersetzung zwischen den Methodenaufrufen und den Datentypen der einzelnen Implementierungssprachen vorgenommen. Sind mehrere ORB's an einer Client/Server-Kommunikation beteiligt, so werden die übersetzten Daten im IIOP-Format zwischen den involvierten ORB ausgetauscht.

Gemeinsam mit den *CORBAservices* und den *CORBAfacilities* bildet der ORB die *Object Management Architecture* (*OMA*, Abb. 2-2).

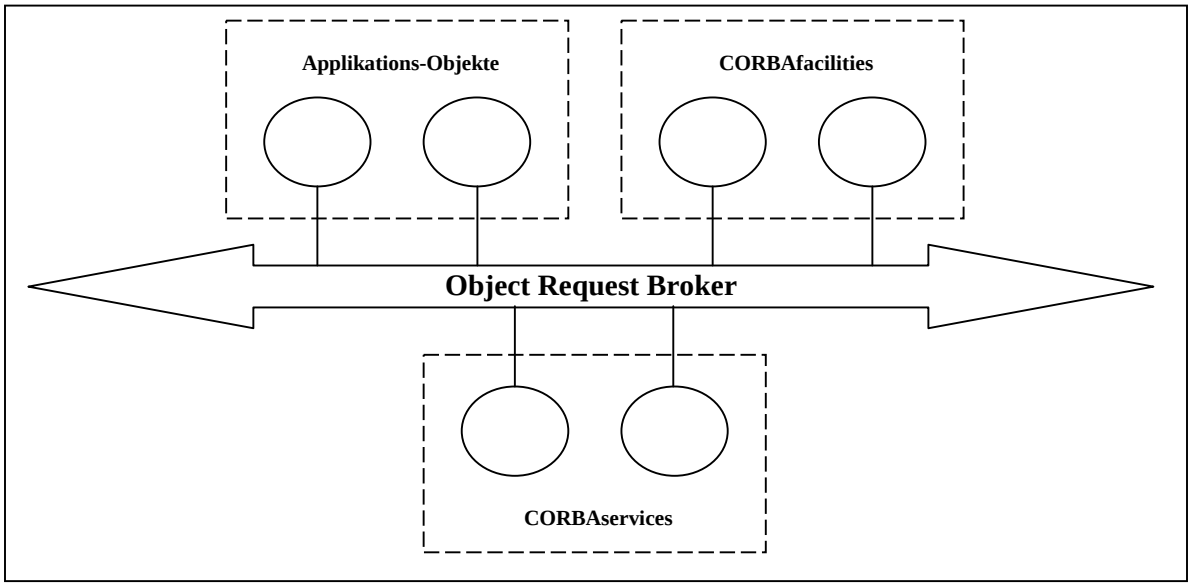


Abbildung 2-2: Die Object Management Architecture (OMA).

CORBAservices und CORBAfacilities erleichtern die Erstellung von Anwendungen unter Nutzung von CORBA, sind jedoch als zusätzliche Elemente nicht notwendige Voraussetzung. Während CORBAservices *technische* oder *objekt-bezogene* Aspekte abdecken, werden durch CORBAfacilities mehr *anwendungsbezogene* Aufgaben wahrgenommen. Gemäß [OMG98] und [Ba97] können die CORBAservices in die in Tabelle 2-4 dargestellten Kategorien eingeteilt werden.

CORBAfacilities als anwendungsnahe CORBA-Ergänzung können nach [Ba97] in zwei Dimensionen aufgeteilt werden:

- horizontal: Unterstützung für Applikationen beliebiger Anwendungsbereiche (Tabelle 2-5)

- vertikal: Unterstützung von Aufgaben individueller, domain-spezifischer Anwendungsbereiche: Gesundheitswesen (z.B. CORBAmed), Telekommunikation, Finanzwesen, Industrie, Handel und Transport.

Services zur Unterstützung verteilter Systeme	
Naming Service	unterstützt die Identifizierung von Objekten durch einen Namen
Event Service	asynchrone Versendung (Entkopplung von Sender und Empfänger) von Nachrichten als Ereignisse
Security Service	Schutzmechanismen in Form von Zugriffsrechten auf Objekte und Verschlüsselung von Netzwerk-Kommunikation.
Trading Service	Aufsuchen von Objekten, die einer Bedingung genügen
Datenbank-bezogene Services	
Concurrency Service	Locking-Mechanismen zur Zugriffskontrolle von Objekten bei mehreren Nutzern
Property Service	Eigenschaften für Objekte als Name-Wert-Paare.
Transaction Service	Kontrolle von Beendigung und Abbrüchen von Transaktionen, die über mehrere Datenbanken hinweg initiiert werden, Unterstützung des 2-Phasen-Commit-Protokolls (2PC) zwischen Datenbanken (Agenten) und Koordinator. Auch als *Object Transaction Service* (*OTS*) bezeichnet.
Relationship Service	Konstruktion und Verwaltung von Beziehungen zwischen Objekten
Query Service	Unterstützung von Queries über Kollektionen von Objekten
Persistence Object Service	Definition der Kommunikation von Datenbanken und Objekten, um ein Objekt in einer Datenbank zu speichern bzw. es wieder zu rekonstruieren.
Externalization Service	Konvertierung von Objekt-Daten von und in Byte-Sequenzen zum Kopieren zu anderen Lokationen.
Allgemeine Services	
Life Cycle Service	definiert Interfaces zur Erzeugung, Bewegung und Kopie von Objekten
Licensing Service	ermöglicht dem Server festzustellen, ob eine Lizenz zum Ausführen der Software vorhanden ist
Time Service	Bestimmung der Tageszeit, Erzeugung von Ereignissen nach einer bestimmten Zeit oder innerhalb eines gegebenen Intervalls.

Tabelle 2-4: Die CORBAservices. Nach [OMG98] und [Ba97].

Nutzer-Schnittstelle	Präsentation von Objekten und zusammengehörigen Dokumenten Hilfeaufgaben, Rechtschreibungs-und Grammatik-Überprüfung
Informations-Management	Informations-Modellierung Speicherung und Retrieval Verschlüsselung und Übersetzung Zeitbezogene Aspekte
System-Management	Verwalten von ORB und CORBA-Anwendungen
Task-Management	Workflow Agenten Regeln

Tabelle 2-5: Die horizontalen CORBAfacilities. Nach [Ba97].

2.2.2.3 Die CORBA-Datentypen

CORBA-Objekte tauschen über ihre Schnittstelle Daten aus. Die Definition und Spezifikation der Datentypen erfolgt ebenso wie die Definition von Interfaces in der Schnittstellen-Definitionssprache IDL. Die Typen der ausgetauschten Daten sind wie die Interface-Deklaration eines Objektes unabhängig von der Implementierungssprache. Um die CORBA-Datentypen (auch als IDL-Datentypen bezeichnet) innerhalb der Implementierungssprachen verwenden zu können, erfolgt zur Schnittstellen-Übersetzung eine Abbildung der IDL-Datentypen auf Datentypen dieser Sprachen. IDL-Datentypen lassen sich allgemein in Basis-Datentypen, konstruierte Typen und Interfaces einteilen. Eine Übersicht über die Basis-Datentypen geben die Tabellen 2-6, 2-7 und 2-8.

Boolean-Typ	
boolean	{TRUE;FALSE}
Octet-Typ	
octet	8-Bit-Größe, unterliegt bei der Übertragung keiner Konvertierung.
Aufzählungs-Typ	besteht aus einer geordneten Liste von Bezeichnern
Any-Typ	
any	enthält Werte eines beliebigen OMG-IDL-Types

Tabelle 2-6: OMG IDL: Basis-Datentypen (1). Nach [OMG98].

Integer-Typen	
short	$[-2^{15};2^{15}-1]$
long	$[-2^{31};2^{31}-1]$
long long	$[-2^{63};2^{63}-1]$
unsigned short	$[0;2^{16}-1]$
unsigned long	$[0;2^{32}-1]$
unsigned long long	$[0;2^{64}-1]$
Float-Typen	[IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985]
float	IEEE single-precision floating point number (32 Bit)
double	IEEE double-precision floating point number (64 Bit)
long double	IEEE double-extended floating point number, Exponent von mindestens 15 Bits, vorzeichenbehaftete Fraktion von mindestens 64 Bits
Fixed Type	
fixed	Fest-Komma-Zahl mit bis zu 31 signifikanten Stellen.

Tabelle 2-7: OMG IDL: Basis-Datentypen (2). Nach [OMG98].

Character-Typ	
char	8-Bit-Größe, die - ein 1-Byte-Zeichen eines 1-Byte-orientierten Codes oder, - wenn sie in einem Array genutzt wird, ein Zeichen eines Multi-Byte-Codes darstellt. unterliegt bei der Übertragung Konvertierungen.
Wide-Character-Typ	
wchar	Größe ist implementationsabhängig, kodiert Zeichen einer beliebigen Zeichenmenge. unterliegt bei der Übertragung Konvertierungen.
String-Typ	
string	Anzahl der Werte von 8-Bit-Größen (Länge des Strings) wird zur Laufzeit festgelegt. Untertypen: - längenbeschränkte und - unbeschränkte Strings.
Wide-Character-String-Typ	
wstring	wie String, aber für Wide-Character

Tabelle 2-8: OMG IDL: Basis-Datentypen (3). Nach [OMG98].

Mit Hilfe einfacher Datentypen und Konstruktionsvorschriften lassen sich die konstruierten Datentypen definieren. Die Konstruktionsmechanismen sind in Tabelle 2-9 wiedergegeben.

Strukturen (Aggregation)	geordnete Menge von (Name,Wert)-Paaren
discriminated unions	besteht aus einem Diskriminator, dessen exakter Wert abgefragt werden kann und einer Instanz eines Typs, der vom Wert des Diskriminators in eineindeutiger Weise abhängt. Für unterschiedliche Diskriminator-Werte variiert der Typ der in der Union enthaltenen Instanz
Sequenzen	enthalten Werte eines zur Übersetzungszeit festgelegten beliebigen IDL-Typs, Anzahl der Werte (Länge der Sequenz) wird zur Laufzeit festgelegt. Untertypen: - längenbeschränkte und - unbeschränkte Sequenzen.
Array-Typ	festdefinierte mehrdimensionale Matrix mit Werten eines zur Übersetzungszeit festgelegten beliebigen IDL-Typs
Interface-Typ	spezifiziert eine Menge von Operationen, die eine Instanz dieses Types unterstützen muß.

Tabelle 2-9: OMG IDL Datentypen. Nach [OMG98].

Schließlich existiert noch der Native-Typ, dessen Übersetzung in die Implementierungssprachen vom Anwender selbst definiert werden muß.
Die Werte der einzelnen Typen sind bis auf den Interface-Typ Instanzen, die den beschriebenen Restriktionen (Wertebereich) genügen ([OMG98], S. 1-5).

Gemäß [OMG98] ist ein Interface die Beschreibung einer Menge möglicher Operationen, die ein Client über einem Objekt ausführen darf. Ein Objekt genügt einem Interface, wenn

es als Ziel-Objekt für jede mögliche, durch das Interface beschriebene Anfrage spezifiziert werden kann.

> „An object satisfies an interface if it can be specified as the target object in each potential request decribed by the interface“ ([OMG98]).

Daraus folgt unmittelbar, daß mehrere verschiedene CORBA-Objekt-Klassen der gleichen Interface-Beschreibung genügen können. Dieser Aspekt wird im folgenden jedoch vernachlässigt und der Begriff des Interfaces mit dem Begriff des Objekt-Typs bzw. der Klasse synonym verwendet.

Ein Interface-Typ ist ein Typ, der eine Referenz auf ein beliebiges Objekt, das der Interface-Beschreibung genügt, enthalten kann ([OMG98]).

Aus der Spezifikation der IDL-Datentypen ergeben sich zwei wichtige Merkmale von Datentypen. Datentypen sind

- statisch.
  Ihre Instanzen besitzen zur Übersetzungszeit bereits eine festdefinierte Größe und einen festdefinierten Typ.
- dynamisch.
  Ihre Instanzen besitzen eine zur Laufzeit spezifizier- und ermittelbare variable Größe (*Sequenzen*), sind von einem Typ, der erst zur Laufzeit festgelegt wird (*discriminated unions*) oder besitzen einen Element-Typ, der erst zur Laufzeit ermittelt wird (Instanzen des *Any-Typs*).

Eine weitere wichtige Unterscheidung von Datentypen muß bezüglich ihrer Instanzen getroffen werden. Eine Variable eines Datentyps kann

- eine Instanz des Typs oder
- eine Referenz auf eine Instanz des Typs[15]

bezeichnen.
Hervorzuheben ist, daß im ersten Fall die Variable eines Typs auf die Instanz einstufig abgebildet wird.

Diese Abbildung wird im Adreßraum des konkreten CORBA-Moduls (CORBA-Client oder –Server) generiert und gibt dem CORBA-Modul die Möglichkeit, die Instanz vollständig zu manipulieren.

[15] Die Form der referenziellen Darstellung eines Datentyps findet ausschließlich bei Objekt-Typen Anwendung.

Ferner besitzt die Instanz des Typs nur eine Lebensdauer, die maximal der Laufzeit des CORBA-Moduls entspricht.

Im Gegensatz dazu wird im zweiten Fall eine zweistufige Abbildung erzeugt. Eine Variable wird einer Referenz zugeordnet, die ihrerseits auf eine Instanz verweist.

Während die erste Abbildung im Adreßraum des CORBA-Moduls stattfindet und damit in ihrer Lebensdauer direkt von der Lebensdauer des CORBA-Moduls abhängig ist, verweist die zweite Abbildung auf eine Instanz, die außerhalb des CORBA-Moduls existiert und damit nicht dessen direkter Kontrolle unterliegt. Durch CORBA-Kommunikations-Mechanismen kann zwar gewährleistet werden, daß die Lebensdauer der externen Instanz – eines CORBA-Objektes – mindestens der Laufzeit eines als Client darauf zugreifenden CORBA-Moduls entspricht. Das CORBA-Objekt kann jedoch auch von anderen CORBA-Clients, möglicherweise gleichzeitig, angesprochen werden. Deshalb muß dieses Objekt mindestens während der gesamten Laufzeit aller darauf zugreifenden Clients, sofern diese es noch referenzieren, existieren. Durch den parallelen Zugriff der CORBA-Clients auf ein CORBA-Objekt kann zudem nicht davon ausgegangen werden, daß sich der Zustand zwischen zwei Methodenaufrufen des gleichen Clients nicht verändert hat.

Damit ergibt sich die Frage der persistenten Speicherung von IDL-Datentyp-Instanzen.

Diese Speicherung, die die Lebensdauer einer Instanz über die Beendigung eines Clients bzw. aller Clienten hinaus gewährleistet, muß im Falle eines Objekt-Typs zweistufig erfolgen. Einerseits muß der Zustand (der Wert aller Variablen und referenzierten Werte) des Objektes persistent sein. Andererseits muß für eine spätere Referenzierung des Objektes die Referenz und die Zuordnung zur Instanz selbst der Persistenz unterliegen. Während für alle anderen Datentypen die persistente Speicherung von Werten in einem einzigen System, z.B. in einem beliebigen DBMS, erfolgen kann, müssen an der persistenten Speicherung von CORBA-Objekten mindestens zwei Systeme partizipieren. Das ist einerseits das System, in welchem die Referenz gespeichert wird, und andererseits ein Mechanismus, der die Speicherung des Objekt-Zustandes gewährleistet. Daraus folgt unmittelbar, daß ein CORBA-Objekt an der persistenten Speicherung seines Zustandes teilnehmen muß, da seine Interna wie Algorithmus, aber auch private Variable und Implementierungssprache im CORBA-Netzwerk transparent sind. Die persistente Speicherung eines CORBA-Objektes kann auf mehrere Weisen erfolgen:

- Das Objekt besitzt eine Schnittstelle für die Ausgabe von privaten Werten in ein vom CORBA-Netzwerk verwaltetes Speicherungssystem, z.B. ein DBMS.
  Das CORBA-Objekt gewährt einen vollständigen Zugriff auf seinen Zustand, einerseits lesend für die persistente Speicherung und andererseits verändernd für die Wiederherstellung des Objektes.
- Das CORBA-Objekt gewährleistet seine Persistenz gegenüber dem Netzwerk. Das bedeutet, daß es für seine persistente Speicherung alleinverantwortlich ist. Gleichzeitig genügt es damit dem Anspruch der maximalen Kapselung von CORBA-Objekten. Das CORBA-Objekt enthält einen eigenen Persistenz-Mechanismus. Es könnte beispielsweise Bestandteil einer objekt-orientierten Datenbank oder umgekehrt auch Client eines OODBMS oder sogar eines relationalen DBMS sein.

Eine Kombination der beiden Möglichkeiten zur persistenten Speicherung wurde in der *Persistence Object Service* (*POS*) *Specification* der OMG gewählt ([OMG97]; Abbildung 2-3).

Dabei werden dem CORBA-Netzwerk Objekte (*Persistent Objects*) zur Verfügung gestellt, die von einem *Persistence Object Manager* (*POM*) verwaltet werden. Dieser sowie weitere Module gewährleisten in Kooperation mit den *Persistent Objects* (*PO*) deren Persistenz. Damit ist der Persistenz-Mechanismus einerseits für das CORBA-Netzwerk transparent, andererseits müssen die *Persistent Objects* dem *Persistence Object Service* den Zugriff auf persistent zu speichernde Daten gewährleisten. Diese Daten können schließlich durch *Persistence Data Services* (*PDS*) in beliebiger Weise abgelegt werden. Speicherungsmedien können Dateien, RDBMS, OODBMS und andere Systeme sein. Eine Implementierung des *POS* bietet die Firma *Secant*.

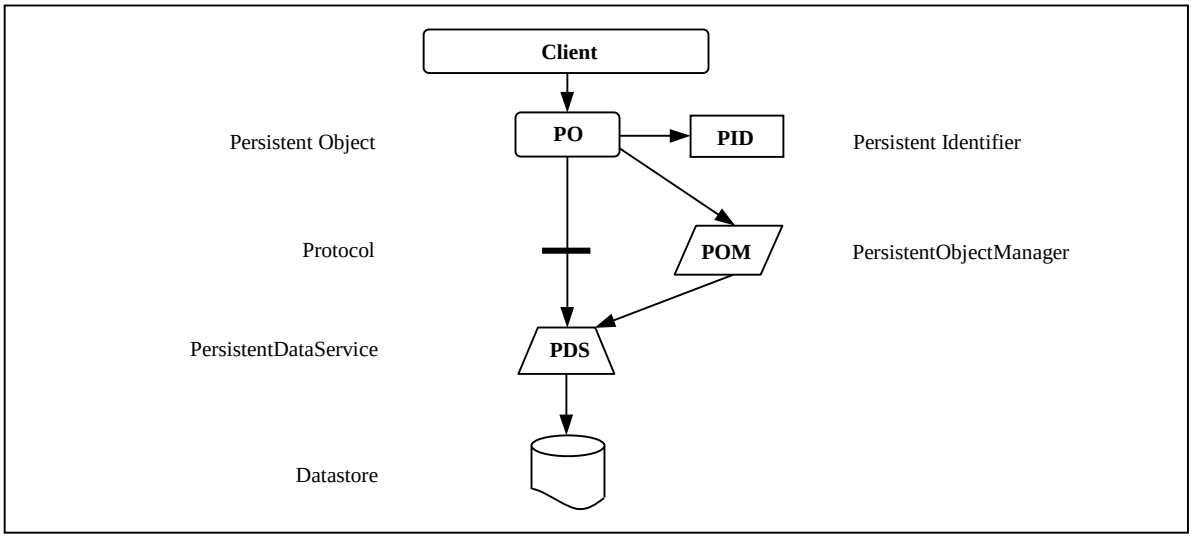


Abbildung 2-3: Der Persistence Object Service. Nach [OMG97].

Ein Beispiel für die zweite Art der Speicherung ist der Ansatz, der vom Hersteller des OODBMS O_2 mit der *O_2CORBA*-Bibliothek verfolgt wird.

> Das OODBMS übernimmt dabei die Rolle eines Servers für CORBA-Objekte, der diese durch Instantierung dem CORBA-Netzwerk zur Verfügung stellt. Als CORBA-Objekte können beliebige Datenbank-Objekte, sofern sie den IDL-Schnittstellen-Definitionen entsprechen, verwendet werden. Für die Definition von CORBA-Objekten im Datenbanksystem steht ein Übersetzer von IDL in die *Object Definition Language* (*ODL*) zur Verfügung. Umgekehrt kann auch für bereits in der Datenbank enthaltene Objekte eine IDL-Deklaration aus der entsprechenden ODL-Deklaration generiert werden.

2.2.2.4 Die CORBA-IDL-Methode

Die Kommunikation zwischen CORBA-Client und –Server-Objekt erfolgt über die in der Interface-Definition deklarierten Methoden. Diese Deklaration wird gemeinsam mit den Parametern der Methoden in die jeweiligen Implementierungssprachen abgebildet. Die Methoden können dann im Aufruf-Format dieser Sprachen angesprochen werden.
Eine IDL-Methoden-Deklaration erfolgt immer innerhalb einer IDL-Interface-Definition. Dazu hat eine Methode die folgende Signatur:

[oneway] <op_type_spec> <identifier> ($param_1$,...,$param_L$) [raises($except_1$,...,$except_N$)] [context($name_1$,...,$name_M$)]

Die Signatur enthält gemäß [OMG98] Spezifikationen

- der Parameter, die für einen Methodenaufruf verlangt werden ($param_1$,...,$param_L$),
- des Ergebnisses der Methode (<op_type_spec>),
- der Ausnahmebehandlungen ($except_1$,...,$except_N$),
- einer zusätzlichen Kontext-Information, die den Methodenaufruf beeinflußt ($name_1$,...,$name_M$) sowie
- der Ausführungssemantik, die der Client erwartet (oneway[16]).

Der Methodenname wird mit <identifier> definiert. Die Übergabe von Parameterwerten als Methoden-Parameter kann auf 3 Arten erfolgen:

[16] Durch Angabe des Schlüsselwortes *oneway* wird ein nichtblockierender Methodenaufruf festgelegt, d.h. die Methode wird asynchron aufgerufen.

- *in-Parameter*: Ein Parameterwert wird der Methode übergeben. Er wird durch die Methode nicht verändert.
- *inout-Parameter*: Ein Parameterwert wird der Methode übergeben. Er kann von der Methode aktualisiert werden.
- *out-Parameter*: Ein Parameter wird der Methode übergeben. Sein Wert wird von der Methode festgelegt.

Zusätzlich zu den 3 genannten Arten der Parameter-Übergabe existiert der *Rückgabewert*. Dieser ist eine spezielle Form des *out-Parameters*.

Die Gliederung der Parameter, die einen Methodenaufruf beschreibt, muß um eine technische, laufzeitrelevante Sicht ergänzt werden. Diese Sicht wird in [Ba97] auch als *Memory Management* bezeichnet. Zwei Varianten sind möglich:

- *Kopieren von Werten*:
  Der vom Client an die Methode übergebene Wert wird vor der Ausführung der Methode in den Adreßraum des Server-Objektes kopiert (*in*- und *inout-Parameter*). Wurde der Wert modifiziert (*inout-Parameter*) oder ein neuer Wert erzeugt (*out-Parameter; Rückgabewert*), wird er nach Beendigung der Methode in gleicher Weise an den Clienten zurückgegeben. Für einfache Datentypen wird ihr Wert, bei komplexen Datentypen wie String wird der durch einen *Pointer* bezeichnete Speicherbereich kopiert. Diese Art der Wertübertragung wird dann angewandt, wenn die Daten unabhängig von der konkreten Programmiersprache sind, d.h. keine Applikationslogik besitzen. Objekt-Typen werden nicht auf diese Weise übergeben.
- *Instanzieren eines Proxy-Objektes*:
  Ist ein CORBA-Objekt Parameter einer Methode, so wird der Methode eine *Referenz* auf dieses Objekt übergeben. Für diese vom Client an die Methode übergebene Referenz wird vor Ausführung der Methode im Adreßraum des Servers ein Proxy-Objekt (Hilfsdatentyp, der Kommunikation im CORBA-Netzwerk dienend) instanziert. Dieses Objekt kommuniziert mit der tatsächlichen, im CORBA-Netzwerk beliebig lokalisierten, durch die Referenz bezeichneten Instanz. Die Kommunikation erfolgt transparent. Wird von der Methode eine Instanz eines solchen Objekt-Typs erzeugt, so wird auf der Seite des Clients ein *Proxy-Objekt* instanziert.

Die in diesem Abschnitt behandelten Aspekte von CORBA beziehen sich ausschließlich auf die Schnittstellen- und Kommunikationsaspekte von IDL-Datentypen und CORBA-Objekten.
Für weitere Informationen über CORBA wird auf [OMG98], [OMG97] und [Ba97] verwiesen.

2.3 Technische Anforderungen

Um CORBA in die Ablauforganisation von IBM FlowMark integrieren zu können, sind verschiedene technische Bedingungen zu erfüllen. Dabei müssen die Eigenschaften von CORBA und IBM FlowMark beachtet werden.

> Eine CORBA-Integration kann in IBM FlowMark nur durch Programm-Aktivitäten und API's erfolgen.
>
> In CORBA erfolgt die Kommunikation zwischen CORBA-Modulen über Schnittstellen, die durch Methoden und deren Parameter realisiert werden.

Es müssen Forderungen gestellt werden, die sich an den Schnittstellen und an den in diesem Kapitel skizzierten Eigenschaften der Systeme orientieren. Aufgrund der festdefinierten Schnittstellen von IBM FlowMark und CORBA muß eine Abbildung der Schnittstellen vorgenommen werden. Sie muß einer Anzahl von technischen Bedingungen genügen. Diese Bedingungen können in Bedingungen für den Aufruf von Methoden und Bedingungen für Datenaustausch und -management unterteilt werden.

(A) Bedingungen für den Methodenaufruf:

(1) Der Aufruf einer CORBA-Methode durch das WfMS ist möglich.
(2) Der Ursprung des Methodenaufrufes ist für CORBA transparent, insbesondere müssen im CORBA-Netzwerk keine Regeln, WfMS betreffend, beachtet werden.
(3) Eine Methode kann im Workflow als Aktion, die verändernd auf einen Datenbestand einwirkt, oder
(4) in einer Bedingung ausgeführt werden.
(5) Tritt während der Ausführung einer Methode eine Ausnahme-Bedingung auf, so sind Mechanismen für deren Behandlung zu berücksichtigen.

(B) Bedingungen für Datenaustausch und –management sind:

(1) In Methoden können Instanzen von beliebigen IDL-Datentypen verwendet werden.
(2) Diese Instanzen können zur Ablaufsteuerung eines Workflows verwendet und
(3) beim Aufruf weiterer Methoden von diesen genutzt werden.
(4) Eine Instanz eines IDL-Datentyps muß mindestens so lange existieren, wie sie vom Workflow noch zur Ablaufsteuerung benötigt wird
(5) oder als Parameter für einen Methodenaufruf verwendet werden kann.
(6) Eine Datenreplikation im WfMS soll weitgehend vermieden werden.

(7) Für die Steuerung eines Workflows in Abhängigkeit von IDL-Datentypen müssen Vergleichsoperatoren vorhanden sein.

Die Bedingungen (A) und (B) zeigen, daß ihre Erfüllung Auswirkungen auf die Modellierung (Buildtime; IDL-Datentypen, Methodenaufruf, Vergleichsoperatoren in Bedingungen) und die Ausführung (Runtime; Verwaltung von Instanzen von IDL-Datentypen, Lebensdauer der Instanz eines IDL-Datentyps, Vergleiche von IDL-Datentypen) von Workflows hat.

2.4 Diskussion

Nicht allen in 2.1 und 2.3 aufgeführten Forderungen kann in dieser Arbeit entsprochen werden. Das liegt an der großen Komplexität des Themas, der Abhängigkeit von weiteren Modulen, die im Projekt derzeit und in Zukunft bearbeitet werden, und an bereits stattfindenden Standardisierungsbemühungen durch internationale Organisationen. Die in dieser Arbeit fokussierten Forderungen können daher nur eine Basis für weitere Entwicklungen im Projekt „HematoWork“ sein. Insbesondere werden hauptsächlich technische Anforderungen, aber auch Anforderungen der onkologischen Station erfüllt. Das sind die folgenden Forderungen:

- Beibehaltung des gegenwärtigen Betriebssystems der onkologischen Station (*MS Windows95*)
- Einfache Integration bestehender Anwendungen
- Ermöglichung des Methodenaufrufs durch das WfMS
- einfache Modellierung von Workflows unter Verwendung von CORBA-Methodenaufrufen
- Unterstützung von *Treatment Functionality, Intra-* und *Inter-hospital Communication Functionality*
- Transparenz bzw. Unabhängigkeit beider Systeme (IBM FlowMark, CORBA) voneinander
- Ausführung einer CORBA-Methode als Anwendung oder zur Steuerung von Workflows in Bedingungen
- Verwendung von Instanzen beliebiger IDL-Datentypen als Parameter in Methoden und zur Ablaufsteuerung von Workflows

Für weitere, erst später im Projekt erfüllbare Forderungen werden grundlegende Voraussetzungen geschaffen:

- Ausnahmebehandlung für Methoden und Workflows,
- Lebensdauer der Instanz eines IDL-Datentyps so lange, wie sie von Workflow zur Ablaufsteuerung oder in Methodenaufrufen benötigt wird,
- Vergleichsoperatoren für IDL-Datentypen,
- Einfache Verwaltung des Gesamtsystems,
- Stabilität des Systems und
- Keine Verzögerung durch die Ausführung von Workflows im Behandlungsablaufes von Patienten, d.h. es wird eine „Real-Time-Prozeßsteuerung“ ([Hö81]) angestrebt.

3 Konzeptioneller Entwurf

3.1 Einführung

Das in dieser Arbeit erarbeitete Konzept betrachtet die Integration von CORBA in IBM FlowMark in zwei verschiedenen Dimensionen:

- Die Integration muß in *Build*- und *Runtime* erfolgen.

  Die Notwendigkeit, bereits zur *Buildtime* eine CORBA-Integration vorzunehmen, folgt unmittelbar aus den Forderungen nach der Verwendung von IDL-Datentypen zur Ablaufsteuerung von Workflows und der einfachen Modellierung von Workflows, in denen Methodenaufrufe als Anwendung und zur Ablaufsteuerung spezifiziert werden.
  Eine CORBA-Integration zur *Runtime* muß z.B. im Rahmen einer Anpassung der Schnittstellen der beiden verwendeten Systeme erfolgen.

- Sowohl zur Buildtime als auch zur Runtime müssen als weitere Dimension die modellier- und verwendbaren *Elemente von Business-Prozessen* betrachtet werden. Wie bereits in dieser Arbeit ausgeführt, sind das sowohl die verwendeten Datentypen als auch die modellierbaren Prozesse.

Die in den folgenden Abschnitten beschriebenen konzeptionellen Aspekte der CORBA-Integration ergeben sich aus der Notwendigkeit der Erfüllung der grundlegenden Forderungen. Sie erfüllen jedoch gleichzeitig weitere Forderungen oder bilden dafür eine wichtige Grundlage.

3.2 Buildtime

3.2.1 Grundlagen

Die Unterstützung der CORBA-Integration zur Buildtime soll besonders die Modellierung von Prozessen unter Verwendung von CORBA-Datentypen und Methodenaufrufen ermöglichen bzw. vereinfachen.
Wie im Abschnitt 2.2 (Systemanalyse) ausführlich beschrieben, gibt es zwischen CORBA und IBM FlowMark keine direkte Unterstützung der Kommunikation in Form von Modellierungskonstrukten und Schnittstellen.

Die Ausführung von CORBA-Methoden ist jedoch in Aktivitäten und in Bedingungen eines Workflows für Vergleiche nötig.

> Um IBM FlowMark den Aufruf externer Applikationen (z.B. CORBA-Methoden) zu ermöglichen, steht lediglich das Modellierungskonstrukt der *Programm-Aktivität* zur Verfügung. In dieser Aktivität können nur externe Programme bzw. Skripte festgelegt werden. Eine Spezifizierung von CORBA-Objekt-Methoden als auszuführende Applikation ist nicht möglich.

Um IBM FlowMark trotzdem den Aufruf von CORBA-Methoden zu ermöglichen, muß dieser Aufruf *außerhalb* des WfMS, unter Verwendung einer *Programm-Aktivität* erfolgen. Das kann nur geschehen, indem das WfMS innerhalb einer Programm-Aktivität direkt oder indirekt einen CORBA-Client zur Ausführung bringt.

> Dieser CORBA-Client kann eine Methode eines CORBA-Objektes aufrufen. Eine Spezifizierung von CORBA-Objekt und -Methode erfolgt im Client statisch oder dynamisch (über Parameter). Die Dynamik wird vom WfMS gesteuert, indem dem Programm in der Programm-Aktivität, das der CORBA-Client selbst oder ein Zwischenglied ist, Parameter übergeben werden. Diese Parameter können das CORBA-Objekt, seine Methode sowie zu übergebende Methodenparameter sein. Die Übergabe muß durch die in Kapitel 2.2.1 beschriebenen Mechanismen erfolgen.

Der Methodenaufruf ohne Methodenparameter ist in IBM FlowMark leicht unter Verwendung adäquater Modellierungskonstrukte zu realisieren.
Sollen einer Methode Parameter übergeben werden oder ist es nötig, eine Methode in einer Bedingung anzugeben, so müssen zusätzliche, technische Aktivitäten in den Workflow eingefügt werden.

> Wenn eine Methode in einer Bedingung auftritt, muß in IBM FlowMark die Methode ausgeführt werden, d.h. eine Programm-Aktivität sowie zusätzliche Elemente, wie Datenstrukturen, müssen in den Workflow eingefügt werden. Damit ist es nicht mehr möglich, Workflows in einfacher, problemnaher Weise zu modellieren. Es müssen viele technische Elemente beachtet werden, die die Übersichtlichkeit des Workflows vermindern und damit die Möglichkeit von Fehlern während der Modellierung vergrößern. Dadurch wird die Modellierung komplexer.

Die Modellierung von Methoden in Aktivitäten und in Bedingungen ist in IBM FlowMark möglich, aber manuell aufwendig und damit fehleranfällig. Dieser Vorgang wiederholt sich immer in gleicher Weise. Das führt zu der Überlegung, solche Aufgaben automatisch ausführen zu lassen. Dies ist möglich, indem eine Prozeß-Modellierungssprache entwickelt wird, die eine Modellierung von Workflows unter Abstrahierung von den Modellierungsmöglichkeiten des WfMS anbietet. Diese Sprache wird im folgenden als *CORBA Workflow Definition Language* (*CWDL*) bezeichnet.

> Mit Hilfe eines zu entwickelnden Übersetzers soll diese Sprache in die Modellierungssprache FDL übersetzt werden. Dabei müssen zusätzlich erstellte, später unter dem Runtime-Aspekt betrachtete, Module vorhanden sein. Während die Module für den mit CWDL arbeitenden Prozeß-Modellierer transparent sind, muß der Übersetzer Workflows in FDL generieren, die die entsprechenden technischen Elemente wie Programm-Aktivitäten, Datenstrukturen, Programm-Definitionen, Rollen-Definitionen usw. enthalten.

CWDL eignet sich neben der stark vereinfachten Modellierung von Methodenaufrufen auch zur Einführung von weiteren, neuen Modellierungskonstrukten.

> Diese dienen der Modellierung von Zeitangaben, dynamischen Workflows sowie der Kommunikation zwischen Workflows des gleichen WfMS oder verschiedener WfMS.

Da sowohl in CWDL als auch in FDL erstellte Prozesse Workflows sind, muß im folgenden zwischen dem *CWDL-Workflow* und dem *FDL-Workflow* unterschieden werden. Während der *CWDL-Workflow* lediglich der semantischen Workflow-Definition dient, damit von vielen Eigenschaften des verwendeten WfMS abstrahiert und nicht ausführbar ist, kann der durch eine Übersetzung erzeugte operationale FDL-Workflow direkt im WfMS zur Ausführung gebracht werden.

3.2.2 Die Modellierungssprache CWDL

Im folgenden wird die Modellierungssprache *CORBA Workflow Definition Language* (*CWDL*) beschrieben. Diese Sprache dient der CORBA-Integration zur *Buildtime*.
Nach einer Übersicht über die Modellierungskonstrukte wird deren Semantik betrachtet. Im Anhang dieser Arbeit befindet sich eine Darstellung der Syntax von CWDL in Form einer kontextfreien Grammatik.

Das Konzept der Modellierungssprache CWDL ist verwandt mit der Modellierungssprache FDL von IBM FlowMark. Ausgehend von FDL wurden Elemente hinzugefügt, die einerseits der CORBA-Integration dienen und die andererseits gerade für die Anwendung von CWDL im Projekt „HematoWork“ (*Refinement-Aktivität, Aktivitäten zur Kommunikation*) sinnvoll sind. Aufgrund der Verbindung mit FDL sollen die Elemente von CWDL, die FDL-Konstrukten entsprechen, nur kurz genannt werden, für genauere Informationen wird auf [IBM96a] und die Darstellung im Anhang verwiesen.

3.2.2.1 Die Modellierungselemente

Die Modellierungskonstrukte von CWDL können in Konstrukte zur

- *Aufbaumodellierung*,
- *Ablaufmodellierung* und
- *Datenmodellierung*

gegliedert werden.

Wie bereits in Kapitel 2 für IBM FlowMark wird auch für CWDL zunächst ein Überblick über die Elemente von Business-Prozessen, die in CWDL verwendet werden können, gegeben
(Tabelle 3-1).

Da die *Aufbaumodellierung* kein Gegenstand dieser Arbeit ist (siehe Kapitel 1), wird diese nicht weiter beschrieben. Insbesondere werden keine Konstrukte benötigt, die über die von IBM FlowMark angebotenen Elemente *Mitarbeiter*, *Rolle*, *Level* und *Organisation* hinausgehen. Deshalb entspricht die Aufbaumodellierung von CWDL der Aufbaumodellierung von IBM FlowMark und ist vollständig im Anhang bzw. in [IBM96a] enthalten.

Beginnend mit der Datenmodellierung werden die CWDL-Elemente beschrieben und die Anknüpfungspunkte für die CORBA-Integration aufgezeigt.

Aufbauorganisation		Mitarbeiter (People) Rolle (Role) Level Organisation
Ablauforganisation		
	Aktivitäten (und von diesen benötigten Konstrukte)	Applikations-Aktivität Skript-Aktivität Manuelle Aktivität Nachrichten-Sender-Aktivität Nachrichten-Empfänger-Aktivität Prozeß-Aktivität Call-Aktivität Refinement-Aktivität Logische Programme (Program) Logische Refinement-Manager
	Elemente des Kontroll- und Datenflusses	Or-Split Or-Join And-Split And-Join Kontroll-Konnektor (Control Connector) Daten-Konnektor (Data Connector)
Produkte / Daten		Produkte bzw. Daten werden durch Variable repräsentiert, die im Prozeß global oder in einer Aktivität lokal vereinbart werden. Variable können Instanzen beliebiger, im CORBA Interface Repository definierter IDL-Datentypen bezeichnen.

Tabelle 3-1: Elemente von Business-Prozessen in CWDL.

3.2.2.2 Die Datenmodellierung

Um einen Workflow sinnvoll modellieren zu können, muß sowohl ein Datenaustausch zwischen Aktivitäten als auch eine Verwendung der von den Aktivitäten erzeugten Daten in Bedingungen ermöglicht werden. Dafür besitzen Workflow-Modellierungssprachen ein Daten- bzw. Variablen-Konzept. Dabei können Datenstrukturen vom Nutzer konstruiert werden, die mittels einer Variablenvereinbarung instanziert werden können.

Während einige WfMS, wie z.B. COSA, ein Variablen-Konzept besitzen, das fast ausschließlich *globale Variable* erlaubt, die von allen Aktivitäten eines Workflows gelesen und modifiziert werden können, gibt es andere WfMS, wie z.B. IBM FlowMark, die ein Variablen-Konzept haben, das als *lokales Variablen-Konzept* zu bezeichnen ist. IBM FlowMark bietet den Datenaustausch lokaler Art zwischen zwei im Workflow bezüglich des Datenflusses benachbarten Aktivitäten. Allerdings können in *Bedingungen* von *Kontroll-Konnektoren* Elemente von *Output-Containern* beliebiger Aktivitäten enthalten sein.

Zwei Aktivitäten sind *benachbart* im Workflow, wenn die Ausführung der einen Aktivität unmittelbar notwendige Voraussetzung für die Ausführung der anderen Aktivität ist. Beide

Aktivitäten sind mittels eines Konnektors (*Daten*- oder *Kontroll-Konnektor*) miteinander verbunden.

(1) Zwei Aktivitäten sind im Workflow *benachbart bezüglich des Kontrollflusses*, wenn sie gemäß der Definition des Kontrollflusses direkt nacheinander auszuführen sind.
(2) Zwei Aktivitäten sind im Workflow *benachbart bezüglich des Datenflusses*, wenn die Ergebnisse einer Aktivität von der anderen weiterverarbeitet werden.

Das Variablen-Konzept von CWDL erlaubt es, sowohl *globale* Variable als auch *lokale* Variable zu verwenden. Variable bezeichnen Instanzen beliebiger, im CORBA Interface Repository deklarierter IDL-Datentypen.

Globale Variable können von allen Aktivitäten des Workflows, in dem sie definiert wurden, gelesen und modifiziert werden. Lese– und Schreibvorgänge werden dabei als atomar betrachtet. Damit können auch gleichzeitige Zugriffe von Aktivitäten auf Variablen realisiert werden. Eine (für den Nutzer transparente) Serialisierung der Zugriffe liegt in der Verantwortung der später beschriebenen *Datenbank für IDL-Datentypen*, die die CORBA-Integration zur Build- und Runtime unterstützt. Zusätzlich müssen CORBA-Objekte, beispielsweise die onkologische Patientendatenbank, ihrerseits selbst Mechanismen zum parallelen Zugriff bieten.
Globale Variable können als schreibgeschützt deklariert werden. Sie können allerdings beim Start eines Workflows einmal initialisiert werden. Danach dürfen nur noch Lesezugriffe erfolgen.

Lokale Variable besitzen lediglich Bedeutung für zwei benachbarte Aktivitäten, d.h. eine lokale Variable kann entweder von einer im Datenfluß benachbarten Aktivität in Form einer Kopie weiterverarbeitet werden oder beeinflußt den Kontrollfluß zu der bezüglich des Kontrollflusses benachbarten Aktivität. Eine lokale Variable wird in einer Aktivität definiert. Ähnlich wie bei globalen Variablen kann eine lokale Variable einen Schreibschutz besitzen. Dessen Verwendung ist jedoch nur dann sinnvoll, wenn eine Skript- oder eine Prozeß-Aktivität (im Abschnitt 3.2.2.3 beschrieben) definiert wird. Außerhalb einer Aktivität kann nur lesend auf deren lokale Variablen zugegriffen werden.

Eine globale Variable kann eine Information sein, die im Verlauf des Prozesses nicht oder nur selten geändert, aber in vielen Aktivitäten und Bedingungen verwendet wird. In einem Workflow des Projektes „HematoWork“ kann ein Identifikator, der einem Patienten zugeordnet ist, durch eine globale Variable repräsentiert werden. Es ist sinnvoll, die den Patienten bezeichnende Variable mit einem Schreibschutz zu versehen. Damit kann der Identifikator nicht verändert werden, d.h. die Spezifikation eines Patienten ist immer

eindeutig möglich. Unter Verwendung des Identifikators können Patientendaten gelesen und ggf. modifiziert werden.

Eine lokale Variable besitzt nur in Teilen eines Prozesses Bedeutung. Das liegt daran, daß sie von den meisten anderen Aktivitäten nicht verwendet wird. In einem Workflow des Projektes „HematoWork“ könnte eine lokale Variable den Leukozytenwert[17] eines Patienten repräsentieren, in dessen Abhängigkeit der weitere Behandlungsverlauf steht.

Mechanismen zur Konstruktion von Datentypen aus bereits vorhandenen IDL-Datentypen werden in CWDL nicht angeboten.

Mit CORBA-Modulen werden Daten ausschließlich über bereits im Interface Repository von CORBA definierte IDL-Datentypen ausgetauscht.
Aufgabe des Variablen-Konzeptes ist es lediglich, Instanzen von IDL-Datentypen zu verwalten.

Eine abschließende Übersicht über das Variablen-Konzept gibt Tabelle 3-2.

	Lokale Variablen	**Globale Variablen**
Gültigkeitsbereich	innerhalb einer Aktivität, Kontrollfluß zur benachbarten Aktivität, durch Datenfluß für benachbarte Aktivität lesbar	Workflow, in dem die Variable deklariert wurde, nicht in einem Subworkflow
Lebensdauer	innerhalb einer Aktivität, Kontrollfluß zur benachbarten Aktivität, durch Datenfluß für benachbarte Aktivität lesbar	Workflow, in dem die Variable deklariert wurde
Lesezugriff	innerhalb einer Aktivität, Kontrollfluß zur benachbarten Aktivität, durch Datenfluß für benachbarte Aktivität lesbar	Workflow, in dem die Variable deklariert wurde, nicht in einem Subworkflow
Schreibzugriff	innerhalb einer Aktivität	Workflow, in dem die Variable deklariert wurde, nicht in einem Subworkflow, wenn Variable mit Schreibsperre markiert, ist nur ein einmaliger Schreibvorgang möglich
Datentypen	IDL-Datentypen des Interface Repository von CORBA	IDL-Datentypen des Interface Repository von CORBA
Verwendung einer Variablen	durch Angabe des Namen, gibt es eine globale Variable mit gleichem Namen, wird in Form einer Kontextspezifikation der Name der Aktivität, gefolgt von einem zweifachen Doppelpunkt, der lokalen Variablen vorangestellt	durch Angabe des Namens

Tabelle 3-2: CWDL: Das Variablen-Konzept.

3.2.2.3 Die Ablaufmodellierung

Kennzeichnend für einen Workflow sind die Aktivitäten.
Die kleinste Arbeitseinheit, die im WfMS nicht mehr unterteilt werden kann oder deren Unterteilung nicht mehr sinnvoll ist, ist die *produktive Aktivität*. Kennzeichnend für produktive Aktivitäten ist die Veränderung oder Erzeugung von Produkten oder Daten. Das geschieht außerhalb des WfMS, welches lediglich die Reihenfolge der produktiven Aktivitäten steuert.

Produktive Aktivitäten sind meistens Aktivitäten, bei deren Bearbeitung Applikationen ausgeführt werden (oft als *Programm-Aktivität* bezeichnet). Diese Applikationen werden entweder vollautomatisch oder in Interaktion mit dem Benutzer ausgeführt. Vollautomatische Applikationen können Anfragen an die onkologische Patientendatenbank sein, in deren Abhängigkeit der weitere Ablauf des Workflows gesteuert werden muß (z.B. Leukozytenwert). Interaktive Applikationen sind beispielsweise die in [Be97] und [Jö97] Beschriebenen.

Eine andere Form der produktiven Aktivität ist die *manuelle Aktivität*, die allerdings als spezielle Ausprägung der Programm-Aktivität nur in manchen WfMS explizit als Modellierungskonstrukt zur Verfügung steht. Eine manuelle Aktivität dient der Modellierung von Arbeitsschritten, die nicht vom WfMS gesteuert werden können. Sollen beispielsweise einem Patienten die Medikamente seiner Chemotherapie zu einem bestimmten Termin gegeben werden, so besitzt das WfMS keinen Einfluß auf die tatsächliche Applizierung der Medikamente. Um dem WfMS trotzdem die Information der Applikation zu geben, kann eine manuelle Aktivität im Workflow definiert werden. Diese kann durch das medizinische Personal zur expliziten Bestätigung der Applikation ausgeführt werden.

Die Aktivitäts-Arten, die in CWDL verwendet werden, sind in Tabelle 3-3 dargestellt.

Produktive Aktivitäten	Applikations-Aktivität Skript-Aktivität Manuelle Aktivität Nachrichten-Sender-Aktivität Nachrichten-Empfänger-Aktivität
Aktivitäten zur Modellierung von hierarchischen Prozessen	Prozeß-Aktivität Call-Aktivität Refinement-Aktivität

Tabelle 3-3: CWDL: Aktivitäten.

[17] Ist der Leukozytenwert unmittelbar vor Beginn einer Chemotherapie unterhalb des Mindestwertes, müssen einerseits Medikamente (G-CSF) für die Erhöhung dieses Wertes gegeben und andererseits muß die Chemotherapie verschoben werden

Die Aktivitäts-Typen werden im folgenden erläutert.

Die *Applikations-Aktivität* ermöglicht die Ausführung von Anwendungen. Neben der Spezifizierung von CORBA-Methodenaufrufen ist es nötig, auch durch CORBA-Objekte nicht kapselbare Legacy-Programme als Applikation verwenden zu können.

> Es reicht nicht aus, lediglich CORBA-Methodenaufrufe als Anwendung zuzulassen. Insbesondere Anwendungen, die ein Nutzer-Interface besitzen, können nicht im CORBA-Netzwerk als CORBA-Objekte gekapselt werden. Die Ausführung eines solchen Programmes kann aufgrund technischer Unterschiede zwischen Rechnersystemen (z.B. Bildschirmausgabe über *X-Windows* versus Ausgabe unter *Microsoft Windows* oder *OS/2*) nicht mehr transparent im Netzwerk erfolgen, sondern muß speziell für die physischen Bedingungen, die der Nutzer-Rechner besitzt, konfiguriert sein.
> Ferner besitzt die CORBA-Spezifikation ([OMG98]) keine Unterstützung für die Umleitung von Bildschirmausgaben, wie es beispielsweise für *X-Windows* und dessen Anwendungen möglich ist.

Ein CORBA-Methodenaufruf wird durch die Referenz des CORBA-Objektes, den Methodennamen und die übergebenen Parameter spezifiziert. Als Parameter können lokale und globale Variablen, aber auch Konstanten von IDL-Basistypen angegeben werden.

Die Definition des Aufrufs von Legacy-Applikationen erfolgt durch Spezifizierung des Namens von *logischen Programmen*.

> Ein logisches Programm ist eine CWDL-Definition, die es erlaubt, in einer Aktivität unter Abstraktion von den physischen Eigenschaften eines konkreten Rechnersystems Anwendungen zu spezifizieren.
>
> Als physische Eigenschaften werden die Art des Betriebssystems sowie Netzwerkaspekte (z.B. die Adresse des Systems, auf dem eine Applikation auszuführen ist) betrachtet.

Die Definition eines logischen Programms kann gleichzeitig für unterschiedliche Betriebssysteme erfolgen, ähnlich zu IBM FlowMark. Einziger Unterschied ist die Art der Übergabe von Daten.

> Bei IBM FlowMark werden Daten durch einen *Input-Container* und *Kommandozeilen-Parameter* an ein Programm übergeben. Eine Rückgabe von Daten wird in Form eines *Return-Codes* oder von Einträgen in den *Output-Container* vorgenommen.

Die Spezifizierung der Datenübergabe erfolgt dagegen in CWDL prozedural.

Ein spezifiziertes Programm wird als Prozedur betrachtet, für deren Ausführung Parameter übergeben werden. Dabei kann für diese Parameter mittels der Schlüsselwörter „IN“ eine Übergabe als Kopie (d.h. der Wert der Variablen darf nicht verändert werden) oder mittels „INOUT“ eine Übergabe als Referenz realisiert werden. Übergebene Parameter können *Konstanten* einfacher, aber auch *Namen* lokaler und globaler Variabler beliebiger IDL-Datentypen sein.
Um einem Programm, etwa einer Legacy-Anwendung, auch *Werte* von Variablen übergeben zu können, wird dem Variablennamen das Schlüsselwort „VALUE“ vorangestellt. Die Übergabe als Wert kann nur für einfache IDL-Datentypen (*short, long, long long, unsigned short, unsigned long, unsigned long long, float, double, long double, fixed, boolean, char* und *string*) erfolgen.

Sowohl bei der Angabe einer CORBA-Methode als auch eines logischen Programmes als Anwendung erfolgt eine Abstraktion von physischen Aspekten.

Bei der Spezifizierung einer CORBA-Methode wird die Transparenz bereits durch das CORBA-Netzwerk erreicht (Kommunikation ausschließlich über Schnittstellen), während die physische Transparenz für eine Legacy-Applikation erst in der Workflow-Modellierungssprache erfolgen kann.

Sowohl für den Aufruf einer CORBA-Methode als auch für den Aufruf eines Programmes können in der Applikations-Aktivität Übergabe-Parameter festgelegt werden. Tabelle 3-4 faßt die Eigenschaften der Applikations-Aktivität zusammen.

	CORBA-Objekte und -Methoden	Ausführbare Programme
Art der Applikations-Spezifikation	Angabe einer Variablen des Typs des CORBA-Objektes, Angabe des Methodennamens	Angabe eines logischen Programmnamens
Art der Datenübergabe	Angabe einer geordneten Liste von Parametern, Parameter sind mit den Schlüsselwörtern „IN“ für lesenden und „INOUT“ für lesenden und schreibenden Zugriff auf Variable des Workflows versehen	Angabe einer geordneten Liste von Parametern, Parameter sind mit den Schlüsselwörtern „IN“ für lesenden, „INOUT“ für lesenden und schreibenden Zugriff auf Variable des Workflows und „VALUE“ für die Übergabe als Wert auf der Kommandozeile versehen
Deklaration der Applikation	Interface Repository von CORBA	Definition eines logischen Programms in CWDL
Ort der physischen Implementierung	beliebig, daher keine Änderung des Workflows bei Veränderung des physischen Systems nötig.	im in der logischen Programm-Definition festgelegten Zugriffspfad, daher kann Änderung in der Workflow-Definition bei Veränderung des physischen Systems erforderlich sein
Ausnahmebehandlung	Angabe von Workflows zur Ausnahme-behandlung für jede Ausnahme möglich	Im wesentlichen im Programm selbst, trotzdem in Abhängigkeit vom Wert einer zu spezifizierenden Variablen Angabe von Workflows zur Ausnahmebehandlung möglich

Tabelle 3-4: CWDL: Applikations-Aktivität.

Die *Skript-Aktivität* umfaßt mehrere Teilarbeitsschritte, die stark zusammenhängen.

> Ein starker Zusammenhang liegt dann vor, wenn eine Sequenz von automatischen Teilschritten in unmittelbarer Folge zu bearbeiten ist, die Anzahl der Teilschritte so gering ist, daß deren Modellierung als Subprozeß nicht sinnvoll ist, oder alle Teilschritte die gleichen lokalen Variablen verwenden.

In einer Skript-Aktivität können CORBA-Methoden und logische Programme ausgeführt werden. Es ist möglich, Variable zu deklarieren.

> Diese Variablen dienen neben der Kommunikation mit anderen Aktivitäten und der Auswertung in Bedingungen zusätzlich der Kommunikation zwischen den Teilarbeitsschritten der Skript-Aktivität.
> Ähnlich wie in einem Prozeß können die Variablen bei der Deklaration mit einer Schreibsperre gekennzeichnet werden. Dadurch kann nach einer Initialisierung nur noch lesend auf die entsprechende Variable zugegriffen werden.

Die Festlegung der einzelnen Teilschritte erfolgt wie in der Applikations-Aktivität.

Die *Manuelle Aktivität* soll als besondere Form der Applikations-Aktivität als eigenes Konstrukt angeboten werden. Es ist zu erwarten, daß im Rahmen des Projektes „HematoWork“ eine solche Aktivität häufig auszuführen ist. Um den Prozeß-Modellierer von der aufwendigeren Modellierung

einer manuellen Aktivität als Applikations-Aktivität zu entlasten und diese gleichzeitig übersichtlicher zu gestalten, ist es sinnvoll, einen solchen Arbeitsschritt gesondert modellieren zu können.

Der Kommunikation mit externen Systemen und Institutionen dienen die *Nachrichten-Aktivitäten*. Diese ermöglichen es einem Workflow einerseits, Nachrichten an einen anderen Workflow innerhalb des gleichen WfMS oder eines anderen WfMS zu senden (Abbildung 3-1). Andererseits können solche Aktivitäten auch in Abhängigkeit von anderen Workflows oder externen Systemen ausgeführt werden.

Ein der Nachrichten-Aktivität ähnliches Konzept findet sich auch im WfMS *COSA*. Hier werden Aktivitäten von externen Programmen getriggert. Den getriggerten Aktivitäten werden Daten einfachen Typs übergeben.

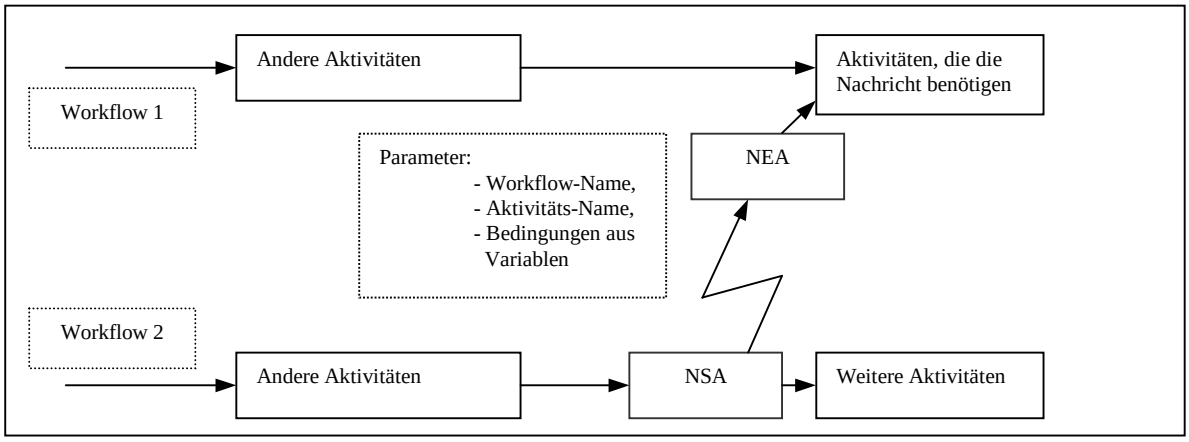


Abbildung 3-1: Nachrichten-Sender-Aktivität (NSA) und Nachrichten-Empfänger-Aktivität (NEA).

Eine Kommunikation zwischen Workflows kann asynchron erfolgen, d.h. eine abgesandte Nachricht kann einen anderen Workflow auch erst zeitlich verzögert, vom weiteren Verlauf des Sender-Workflows entkoppelt, beeinflussen.

Die *Nachrichten-Sender-Aktivität* ist für das *Senden* von Nachrichten zuständig. Soll eine Kommunikation zwischen zwei Workflows stattfinden, so muß der Empfänger-Workflow eindeutig identifizierbar sein. Für die Identifizierung einer Workflow-Klasse (Workflow-Template, modellierter Workflow) und einer Aktivität ist die Angabe des Prozeßnamens sowie des Namens der angesprochenen Aktivität notwendig. Da jedoch ein modellierter Workflow mehrfach instanziert werden kann, ist zudem eine Identifizierung der Prozeß-Instanz erforderlich. Diese kann auf zwei Wegen erfolgen:

- Durch eine Aktivität des aufrufenden Workflows wurde bereits die Instanz identifiziert (z.B. als Ergebnis eines Methodenaufrufs). Deshalb kann diese Instanz direkt angesprochen werden.
- Eine angesprochene Instanz wird durch die Erfüllung einer Bedingung identifiziert. Diese Bedingung wird aus globalen Variablen des Empfänger-Workflows gebildet. Qualifizieren sich mehrere Instanzen einer Workflow-Klasse, so sind alle anzusprechen.
  Eine Bedingung könnte zum Beispiel durch Verwendung eines Patienten-Identifikators formuliert werden:

  Patient.Name()==“Littmann“

Falls die kommunizierenden Workflows in zwei verschiedenen WfMS ausgeführt werden, muß zusätzlich ein symbolischer Identifikator für das andere WfMS verwendet werden. Ist dieses andere WfMS ebenfalls Bestandteil des CORBA-Netzwerkes, so kann es als CORBA-Objekt einfach durch eine Referenz angegeben werden.

Wie bereits ersichtlich, ist eine Spezifikation eines externen Systems abhängig von seinen physischen Eigenschaften. Im folgenden werden als *andere WfMS* nur noch CORBA-Objekte betrachtet. Soll die Kommunikation mit einem anderen System auf eine andere Weise als über ein CORBA-Netzwerk erfolgen, so kann dieses WfMS im CORBA-Netzwerk ohne Probleme durch ein Objekt gekapselt werden. Das kapselnde Objekt repräsentiert das WfMS, während es als Gateway die Kommunikation mit dem möglicherweise entfernten WfMS über beliebige, im CORBA-Netzwerk transparente Mechanismen durchführen kann.

Der Inhalt einer Nachricht ist immer der Wert einer Variablen eines IDL-Datentyps.

Für die Übermittlung von Nachrichten können Objekt-Klassen von Nachrichten eingeführt werden. Eine Nachricht wird dann immer durch eine Variable des Nachrichten-Typs ausgedrückt.

Der Empfang einer Nachricht erfolgt durch eine *Nachrichten-Empfänger-Aktivität*. Diese wird extern durch die eingehende Nachricht getriggert. Um eine Nachricht empfangen zu können, muß in einer Nachrichten-Empfänger-Aktivität ein Sender festgelegt werden. Wie bei der Nachrichten-Sender-Aktivität erfolgt dies über die Angabe der Workflow-Klasse (Workflow-Template) und des

Namens der sendenden Aktivität. Die Identifizierung einer Workflow-Instanz, deren Nachricht erwartet wird, erfolgt auf die gleiche Weise wie bei der Nachrichten-Sender-Aktivität.

> Die Instanz wurde bereits in einer anderen Aktivität des Workflows identifiziert oder qualifiziert sich durch Definition einer Bedingung unter Verwendung globaler Variablen des Sender-Workflows. Qualifizieren sich mehrere potentielle Sender, so wird die zuerst eingegangene Nachricht verarbeitet.

Die empfangene Nachricht ist ein Wert eines IDL-Datentyps.

Die *Prozeß-Aktivität* ermöglicht die hierarchische Modellierung. In einer Prozeß-Aktivität wird der Prozeß-Name eines bereits modellierten Prozesses spezifiziert. Dieser Subprozeß wird synchron ausgeführt, d.h. die Prozeß-Aktivität wird erst dann beendet, wenn der Subprozeß beendet wurde. Werden in einer Prozeß-Aktivität lokale Variable definiert, so sind sie für den aufgerufenen Subprozeß global.

> Werden in einer Prozeß-Aktivität lokale Variable vereinbart, die den gleichen Namen und den gleichen Typ wie globale Variable des Subprozesses besitzen, so werden im Subprozeß die lokalen Variablen der Prozeß-Aktivität verwendet.
>
> Eine Deklaration lokaler Variabler in der Prozeß-Aktivität, die mit globalen Variablen des Subprozesses in ihrem Bezeichner, aber nicht in ihrem IDL-Datentyp übereinstimmen, ist nicht erlaubt.

Wie die Prozeß-Aktivität, ermöglicht auch die *Call-Aktivität* den Aufruf eines anderen Prozesses. Im Gegensatz zur Prozeß-Aktivität erfolgt der Aufruf jedoch asynchron, d.h. der aufrufende Prozeß wird weiter abgearbeitet, während der neue Prozeß (z.B. Befund-Anforderung) parallel zu ihm verläuft. Beide Prozesse können bei Bedarf unter Verwendung von Nachrichten-Aktivitäten wieder synchronisiert werden. Vorteil der Call-Aktivität gegenüber der Prozeß-Aktivität ist die Möglichkeit, die Fehlerbehandlung (im Falle eines Abbruchs des Subprozesses oder anderer Ausnahmesituationen) vom Ablauf des aufrufenden Prozesses weitgehend zu entkoppeln.

Den Prozeß- und Call-Aktivitäten ähnlich ist die *Refinement-Aktivität*. Diese Aktivität ist ein Konstrukt, das aufgrund der inhaltlichen Anforderungen im Projekt „HematoWork“ nötig wird.
Die Refinement-Aktivität ermöglicht wie die Prozeß-Aktivität eine hierarchische Modellierung von Workflows. Anders als bei der Prozeß-Aktivität ist der Subprozeß, der die Aktivität verfeinert,

zur Modellierungszeit der Refinement-Aktivität nicht bekannt. Die Zuordnung oder Generierung eines Subprozesses zur Refinement-Aktivität erfolgt erst während der Laufzeit (Abbildung 3-2).

Das geschieht mit Hilfe eines Werkzeuges, dem *Refinement-Manager*. Der Refinement-Manager generiert zur Laufzeit mit der Information übergebener Parameter einen Subworkflow und gewährleistet dessen Ausführung. Die Art der Generierung des Subworkflows ist für das WfMS transparent.

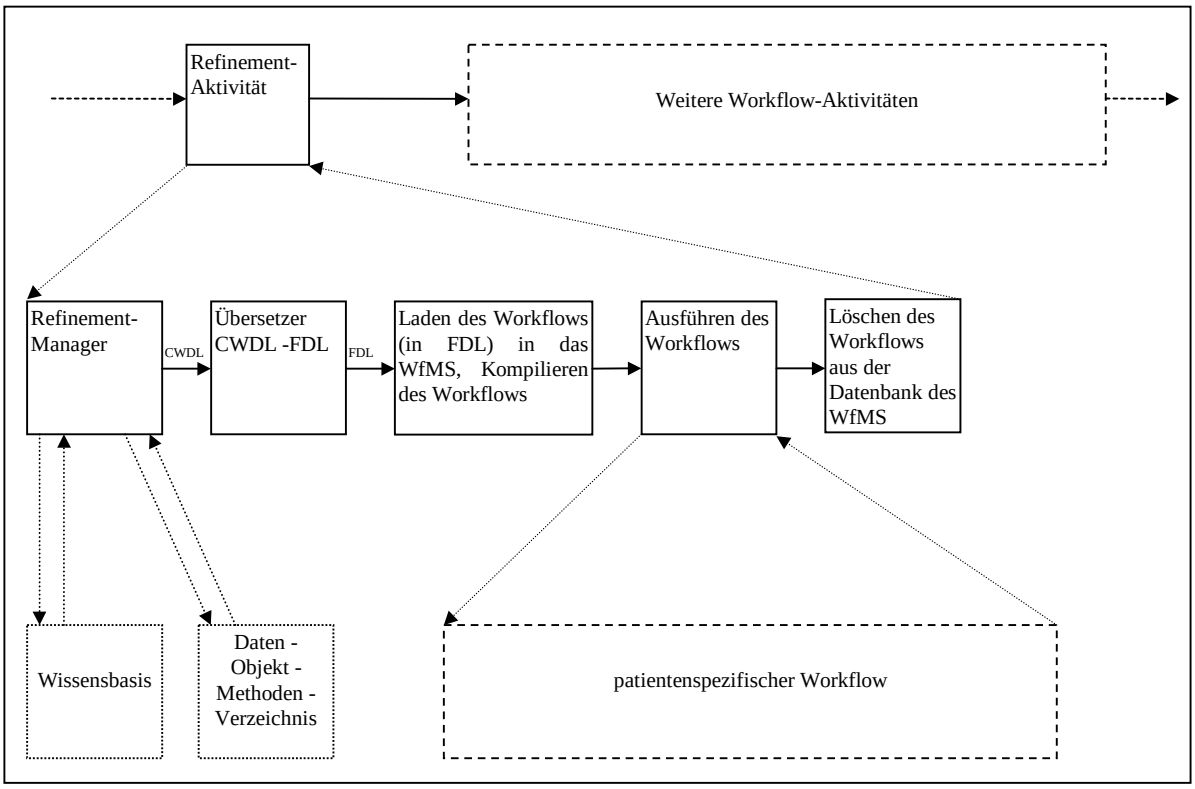


Abbildung 3-2: Refinement-Aktivität und –Manager.

Der im Projekt „HematoWork“ benötigte Refinement-Manager generiert die Subworkflows in Abhängigkeit der übergebenen Parameter in Kooperation mit der *Wissensbasis*. In diesem Modul ist die für die Generierung von Workflows relevante Information u.a. in Form von semantischen Netzen abgelegt. Mit Hilfe der dem Refinement-Manager übergebenen patientenspezifischen Parameter können aus diesen Netzen den konkreten Patienten betreffende Teilstrukturen erzeugt werden. Diese Teilnetze werden in eine operationale und vom WfMS interpretierbare Form übersetzt und ausgeführt.

Der Refinement-Manager wird, wie bereits für logische Programme und Methodenaufrufe in Applikations-Aktivität beschrieben, durch Angabe eines CORBA-Objektes, einer Methode und

von Parametern festgelegt (gekapselt als *logischer Refinement-Manager*). Folgende Arten von Parametern sind nötig:

(1) Parameter technischer Art:
Ein Refinement-Manager muß für die Übersetzung und Ausführung von Workflows Information über die Art des ausführenden WfMS (Modellierungssprache des WfMS; bei IBM FlowMark FDL) und Instanz des WfMS (CORBA-Referenz) besitzen.
Werden keine Parameter technischer Art bei der Definition von CWDL-Workflows angegeben, so werden Default-Werte verwendet. Das bedeutet, daß der Refinement-Manager so konfiguriert wird, daß die von ihm erzeugten Subprozesse für das gleiche Ziel-System wie der CWDL-Workflow generiert werden. Eine Ausführung der Prozesse erfolgt ebenfalls in diesem System.

(2) Für den verwendeten Refinement-Manager spezifische Parameter:
Im Rahmen des Projektes „HematoWork“ sind diese Parameter medizinischer, patientenspezifischer Art. Die Parameter besitzen bei der Identifikation patientenspezifischer semantischer Teilnetze in der Wissensbasis für diese qualifizierenden Charakter und beeinflussen dadurch bei der Generierung des Subworkflows dessen Struktur. Die Parameterwerte haben im erzeugten Subworkflow den Charakter globaler Variabler.

Übergebene technische Parameter sind Konstante oder Variable des Typs *string*. Patientenspezifische Parameter sind Variable eines beliebigen IDL-Datentyps.

Mit dem Konstrukt der Refinement-Aktvität kann ein Prozeß durch einen vom Prozeß-Modellierer anzugebenden Refinement-Manager verfeinert werden. Das bedeutet, daß das Konstrukt in beliebigen Anwendungsgebieten zum Einsatz kommen kann. Es können im gleichen Anwendungsgebiet bei Bedarf auch mehrere Refinement-Manager eingesetzt werden.

Durch Anwendung eines Refinement-Managers können alle Funktionalitäten der Prozeß-Aktivität, allerdings mit größerem Modellierungsaufwand, erfüllt werden. In diesem Fall würde ein spezifizierter Refinement-Manager lediglich die Ausführung eines bereits vordefinierten Workflows veranlassen.

Ein Refinement-Manager kann als Schnittstelle zwischen WfMS verschiedener Hersteller arbeiten. Es wird zwar von der WfMC im Rahmen einer Architekturbeschreibung von WfMS für diesen Zweck eine Schnittstelle definiert (Schnittstelle 4; [WfMC94]). Sie wird

jedoch nicht von allen kommerziellen WfMS unterstützt. Teilweise bieten kommerzielle WfMS auch eine eigene Schnittstelle an, die eine Kommunikation mit dem Ziel der Ausführung von Subprozessen auf einem anderen WfMS ausschließlich zwischen WfMS des gleichen Herstellers erlaubt. Ein WfMS auf der Basis von IBM FlowMark kann beispielsweise nur mit anderen FlowMark-Systemen kooperieren.

Bisher wurde nur die hierarchische Verfeinerung bzw. *vertikale Dynamisierung* gezeigt. Demgegenüber kann mit einem Refinement-Manager auch eine *horizontale Dynamisierung* erreicht werden. Dies bedeutet, daß ein Workflow in seiner gesamten Struktur nicht festgelegt wird (ähnlich wie bei Groupware). Es erfolgt lediglich eine Prozeß-Definition, die obligatorische Aktivitäten, die immer ausgeführt werden müssen, enthält. Abgeschlossen wird der Prozeß durch eine Refinement-Aktivität. Durch diese wird gemäß der von den obligatorischen Aktivitäten erhaltenen Information der weitere Workflow generiert.

Die folgenden Ausführungen gelten für alle Arten von Aktivitäten.
Für eine Aktivität können Zeitangaben festgelegt werden. Diese Zeitangaben definieren in Form eines Zeitintervalls, wann eine Aktivität bearbeitet werden kann. Sie geben damit einerseits den Zeitpunkt an, ab dem die Aktivität *frühestens* ausführbar ist. Andererseits kann ein Zeitpunkt für die erwartete Beendigung einer Aktivität definiert werden. Die Zeitangaben erfolgen durch Angabe von Tagen, Stunden und Minuten. Sie sind relativ, d.h. der Zeitpunkt der frühestmöglichen Bearbeitung einer Aktivität wird relativ zu einer oder mehreren im Kontrollfluß benachbarten Aktivitäten definiert. Relative Zeitangaben werden immer in Bezug auf den Abschlußzeitpunkt der unmittelbar im Kontrollfluß vorangegangenen Aktivitäten angegeben. Die Zeitangabe für den Startzeitpunkt einer Aktivität ist Bestandteil des Kontrollflusses.

Die Definition von Zeitpunkten für die Bearbeitung einzelner Aktivitäten kann vielfach genutzt werden. So kann beispielsweise die Durchführung von Chemotherapien termingerecht gesteuert werden[18].

Eine andere Anwendungsmöglichkeit ist die Überwachung der fristgerechten Einsendung von Studienbögen durch das WfMS der Studienleitung. Wird ein Studienbogen nicht rechtzeitig eingesandt, könnte dieser als Ausnahmebehandlung von der Studienleitung explizit angefordert werden. Als Ausnahmebehandlung einer Nachrichten-Empfänger-

[18] Chemotherapien werden in Form von Chemotherapie-Blöcken durchgeführt. Zwischen diesen Blöcken liegen im Studienprotokoll definierte Zeiträume. Diese Zeiträume sind in der NHL-Studie B 14 oder 21 Tage. Erst am 15. bzw. 22. Tag nach Ende des letzten Chemotherapie-Blocks darf die Chemotherapie fortgesetzt werden.

Aktivität kann ein Subprozeß gestartet werden, der eine Erinnerung, Nachfrage oder Mahnung an das WfMS der onkologischen Station schickt.

Als weiteres Beispiel soll die Ausführung von automatischen Aufgaben durch das WfMS betrachtet werden. Das sind Arbeitsschritte, die ohne Interaktion mit dem Nutzer stattfinden. In Form von automatischen Aufgaben kann ein WfMS Datenbankoperationen und Berechnungsmodule wie etwa das in [Be97] Beschriebene starten. Aufgrund von Fehlern dieser Programme oder anderer, für ihre Ausführung wichtiger Systeme kann es zu einer wiederholten Ausführung dieser Aktivitäten durch das WfMS kommen[19], auch wenn der Systemfehler weiterhin besteht und das Programm erneut erfolglos beendet wird. Es ist sinnvoll, einen Zeitpunkt festzulegen, zu dem die Ausführung einer Aktivität erfolgreich abgeschlossen sein muß. Bei Verletzung dieser Vorgabe ist durch eine eingeleitete Ausnahmebehandlung die Behebung des Systemfehlers zu veranlassen.

Wird ein Abschluß-Zeitpunkt für eine Aktivität nicht eingehalten, können bei der Prozeß-Definition festgelegte Personen von dieser Situation benachrichtigt werden. Ihre Aufgabe ist es, die Ausführung der Aktivität zu veranlassen. Die Benachrichtigung verläuft zweistufig.

Zunächst kann eine beliebige Person benachrichtigt werden. Wird von ihr innerhalb einer definierten Zeit die Ausführung der Aktivität nicht veranlaßt, so wird der Administrator des Systems informiert.

Da die bisher beschriebene Art der Ausnahmebehandlung vom Benutzer eines WfMS Kenntnisse verlangen, die WfMS-spezifisch bzw. prozeßspezifisch sind und über das medizinische Wissen des Personals hinausgehen, wird eine weitere Möglichkeit der Ausnahmebehandlung eingeführt. Wird eine bestimmte Aktivität nicht termingerecht ausgeführt, so wird an ihrer Stelle ein Subprozeß gestartet.

Dieser Prozeß muß in Form einer Prozeß-Definition wie bei der Prozeß-Aktivität vorliegen. Durch Ausführen des Subprozesses wird der korrekte Behandlungsverlauf des Patienten wieder gewährleistet.

Die erste Möglichkeit der Ausnahmebehandlung erfolgt äquivalent zur Ausnahmebehandlung des WfMS IBM FlowMark, während die zweite Form eine echte Erweiterung ist.
Eine Übersicht über die Eigenschaften von Aktivitäten enthalten die Tabellen 3-5 und 3-6.

[19] In IBM FlowMark wird jedes nicht erfolgreich beendete Programm sofort erneut ausgeführt.

Allgemeine Definitionen		Name der Aktivität, obligatorisch Beschreibung der Aktvität, optional	
Aktivitätsspezifische Definitionen			
	Applikations-Aktivität (es kann nur genau eine der beiden Möglichkeiten angewandt werden)	Definition eines Methodenaufrufs durch Spezifizierung - einer Variablen, die ein CORBA-Objekt repräsentiert, - eines Methodennames sowie - zu übergebender Parameter. Parameter sind lokale Variablen der Aktivität oder globale Variablen des Prozesses	Spezifizierung eines logischen Programmes sowie dessen Parameter. Das logische Programm wird, wie in [IBM96a] beschrieben, durch eine Programm-Definition festgelegt mit zwei Unterschieden: - Kommandozeilen-Parameter werden erst in der Applikations-Aktivität angegeben. - In der Programm-Definition (physische Kapselung) sind keine Aspekte des Datenmanagements enthalten.
	Skript-Aktivität (es können beide Möglichkeiten in beliebiger Reihenfolge und Anzahl angewandt werden)		
	Manuelle Aktivität	*Keine zusätzlichen Angaben nötig*	
	Nachrichten-Sender-Aktivität	- CORBA-Referenz des Empfänger-WfMS - Prozeßname des Empfänger-Prozesses - Aktivitätsname der Nachrichten-Empfänger-Aktivität - Bedingung für die Identifizierung der Empfänger-Prozeß-Instanz oder Angabe einer Variablen mit der entsprechenden Information, qualifizieren sich mehrere Empfänger, so erhalten alle die Nachricht - Angabe der Variablen mit der zu übermittelnden Nachricht	
	Nachrichten-Empfänger-Aktivität	- CORBA-Referenz des Sender-WfMS - Prozeßname des Sender-Prozesses - Aktivitätsname der Nachrichten-Sender-Aktivität - Bedingung für die Identifizierung der Sender-Prozeß-Instanz oder Angabe einer Variablen mit der entsprechenden Information. Es können sich mehrere potentielle Sender qualifizieren. Die erste erhaltene Nachricht wird verarbeitet. - Angabe der Variablen für die zu empfangene Nachricht	
	Prozeß-Aktivität	Angabe eines Prozeßnamens; der bezeichnete Prozeß muß bereits definiert worden sein.	
	Refinement-Aktivität	Angabe eines logischen Refinement-Managers. Dieser kapselt CORBA-Objekt, Methode und technische Parameter. Parameter in der Refinement-Aktivität sind patientenspezifischer Art.	

Tabelle 3-5: CWDL: Definition von Aktivitäten (1).

Diese sind mit Ausnahme der aktivitätsspezifischen Semantik, Variablen-Definition und der zweiten Art der Ausnahmebehandlung (Start eines Subprozesses) alle äquivalent mit den entsprechenden Konstrukten von IBM FlowMark. Deshalb wird auf eine detaillierte Beschreibung der Syntax und Semantik verzichtet und auf [IBM96a] verwiesen.

Ende einer Aktivität	Wenn die angegebene Abschluß-Bedingung erfüllt ist	
Lokale Variable	Variable beliebiger IDL-Datentypen. Es können Schreibsperren für Variable definiert werden (sinnvoll in Skript- und Prozeß-Aktivität).	
Ausführungsberechtigungen (Definition entweder dynamisch oder statisch)	Dynamische Zuweisung von Berechtigungen zur Ausführungszeit des Workflows. Zuweisung in Form von Inhabern und Koordinatoren von Rollen (Koordinator: z.B. Fachlicher Leiter), Organisationen (Angabe des Namens) und Levels (Angabe eines Intervalls). Die Nutzer, für die alle Kriterien gelten, sind für die Ausführung einer Aktivität qualifiziert.	Statische Zuweisung von Berechtigungen in Form von Eigenschaften durch Angabe genau eines der folgenden Punkte: - Prozeßadministrator - Prozeßstarter - Manager des Prozeßstarters (Manager: Leiter einer Organisation) - Starter einer bestimmten (zur Buildtime festzulegenden) Aktivität - Manager des Starters einer bestimmten Aktivität - Nicht der Starter einer bestimmten Aktivität - Einzeln angegebener Nutzer des WfMS
Ausnahmebehandlung (es kann nur eine der beiden Möglichkeiten spezifiziert werden)	Wird der Termin für das Aktivitätsende nicht eingehalten, so erfolgt eine Ausnahmebehandlung. Der Termin ist eine relative Zeitangabe und bezieht sich immer auf den Zeitpunkt, ab dem alle Voraussetzungen für die Ausführung der Aktivität erfüllt waren.	
	Ausnahmebehandlung durch spezifizierte Nutzer: (1) Erste Stufe: - Prozeßadministrator - Manager - Koordinator - spezifizierte Person Zeitangabe in Tagen, Stunden und Minuten (2) Zweite Stufe: Wird von der in der ersten Stufe informierten Person innerhalb einer definierten Zeit das Problem nicht behoben, so wird der Prozeßadministrator informiert. Zeitangabe in Tagen, Stunden und Minuten.	Ausnahmebehandlung durch einen Subprozeß. Definition siehe Prozeß-Aktivität.
Dokumentation	Beschreibung der Aktivität, optional	

Tabelle 3-6: CWDL: Definition von Aktivitäten (2).

Die Aktivitäten können manuell oder automatisch ausgeführt werden (Tabelle 3-7).

Arten der Ausführung	**Automatisch** (transparent für den Nutzer) : - Applikations-Aktivität, - Skript-Aktivität, - Nachrichten-Sender-Aktivität, - Nachrichten-Empfänger-Aktivität, - Prozeß-Aktivität - Call-Aktivität - Refinement-Aktivität	**Manuell** (in Interaktion mit dem Nutzer): - Applikations-Aktivität - Skript-Aktivität - Manuelle Aktivität

Tabelle 3-7: CWDL: Ausführung von Aktivitäten.

Um Applikations- und Skript-Aktivitäten definieren zu können, ist es notwendig, die Definition von logischen Programmen zu ermöglichen. Logische Programme werden wie in Tabelle 3-8 angegeben definiert. Die Programm-Definition wurde aus dem entsprechenden Konstrukt von IBM FlowMark abgeleitet.

Allgemeine Definitionen	- Name des logischen Programmes, obligatorisch - Beschreibung des Programmes, optional.
Netzwerk	- Angabe eines Rechners, auf dem das Programm auszuführen ist. Die Adresse kann im APPC- oder im TCP/IP-Format angegeben werden. - Ein Programm kann ohne Nutzerinteraktion ablaufen. Das kann ebenfalls spezifiziert werden.
Spezifikation des Programmaufrufs (teilweise betriebssystem-abhängig)	- Pfad und Dateiname des physischen Programmes - Entry-Point, falls als physisches Programm eine DLL definiert wurde - Arbeitsverzeichnis - Betriebssystem-Umgebung. Es können Umgebungsvariablen definiert werden. - Vererbung der Umgebung des Betriebssystems möglich (in Kombination mit der Umgebung, falls definiert) - Startart des Programms (1) sichtbar (2) unsichtbar (ohne Nutzer-Interface) (3) minimiert (als Icon) (4) maximiert (5) im Vordergrund
Nur für OS/2	- Ende des Programmes automatisch manuell (Nutzer muß Kommando-Interpreter des Betriebssystems, in dem das Programm lief, selbst beenden)

Tabelle 3-8: CWDL: Definition von logischen Programmen.

Die Konstrukte unterscheiden sich in einem Punkt:

> Im Gegensatz zu FDL werden in CWDL keine Daten-Container angegeben. Parameter werden in Applikations- und Skript-Aktivitäten festgelegt.

Eine Aufstellung über die unterstützten ausführbaren Programme gibt [IBM96a], S.84. Für die Syntax von Namen und Konstanten wird ebenfalls auf [IBM96a] verwiesen.

Die Definition einer Refinement-Aktivität erfordert die Definition eines logischen Refinement-Managers. Tabelle 3-9 gibt einen Überblick.

Allgemeine Definitionen	- Name des logischen Refinement-Managers, obligatorisch - Beschreibung des Refinement-Managers, optional.
Spezifikation des Aufrufs des Refinement-Managers	- Spezifikation eines CORBA-Objektes in Form einer Referenz - Spezifikation einer Methode durch ihren Namen - Spezifikation von technischen Parametern des Refinement-Managers (Modellierungssprache des Ziel-WfMS, Spezifikation des Ziel-WfMS durch eine CORBA-Referenz)

Tabelle 3-9: CWDL: Definition von logischen Refinement-Managern.

Die Reihenfolge der Ausführung der Aktivitäten wird durch *logische Operatoren, den Kontrollfluß* und den *Datenfluß* festgelegt.

Um in einem Workflow Strukturen zu modellieren, die Parallelisierungen und alternative Pfade repräsentieren, werden logische Operatoren eingeführt (Tabelle 3-10).

Operator	Bedeutung	Dualer Operator
Or-Split	- Mindestens ein im Or-Split beginnender Pfad wird durchlaufen. - Die Bedingung für den Split wird im Kontroll-Konnektor angegeben. - In einer Bedingung können die lokalen Variablen aller dem Operator (oder der Operatorsequenz) direkt vorausgehenden Aktivitäten ebenso wie Methodenaufrufe verwendet werden.	Or-Join
And-Split	- Alle im And-Split beginnenden Pfade werden durchlaufen. - Die aus dem And-Split auslaufenden Kontroll-Konnektoren dürfen keine Bedingungen enthalten.	And-Join
Or-Join	- Kennzeichnung des Endes alternativer Pfade. - Mindestens ein in einen Or-Join einlaufender Kontroll-Konnektor muß durchlaufen werden	Or-Split
And-Join	- Kennzeichnung des Endes paralleler Pfade. - Alle in einen And-Join einlaufenden Kontroll-Konnektoren müssen durchlaufen werden	And-Split

Tabelle 3-10: CWDL: Logische Operatoren im Kontrollfluß.

Die Ausgangs-Arität von CWDL-Splits und die Eingangs-Arität von CWDL-Joins ist nicht beschränkt. Deshalb müssen im Falle des Or-Splits alle CWDL-Kontroll-Konnektoren darauf getestet werden, ob mindestens ein Pfad durchlaufen wird. Werden in einer Bedingung Variablen verwendet, so ist eine entsprechende Überprüfung zur Buildtime nicht möglich, d.h. ein modellierter Workflow ist möglicherweise nicht korrekt. Es können mehrere im Or-Split beginnende Pfade oder kein einziger Pfad durchlaufen werden.
Würde man anstelle des Or-Splits einen Exclusive-Or-Split mit einer Ausgangs-Arität von 2 verwenden, so kann ebenfalls ein Or-Split simuliert werden (Abbildung 3-3). Zusätzlich könnte bereits syntaktisch zur Buildtime erzwungen werden, daß ein Or-Split korrekt modelliert ist. Durch die Angabe einer Exclusive-Or-Bedingung in einem Exclusive-Or-Split können im Split beginnende Pfade mit den Labels „TRUE“ oder „FALSE“ markiert werden.
Das bedeutet, daß immer genau ein Pfad durchlaufen wird.

Im Rahmen des Projektes „HematoWork“ wird jedoch für eine *kompakte Modellierung* das Konstrukt des Or-Splits verwendet, dessen *Arität nicht beschränkt* ist.

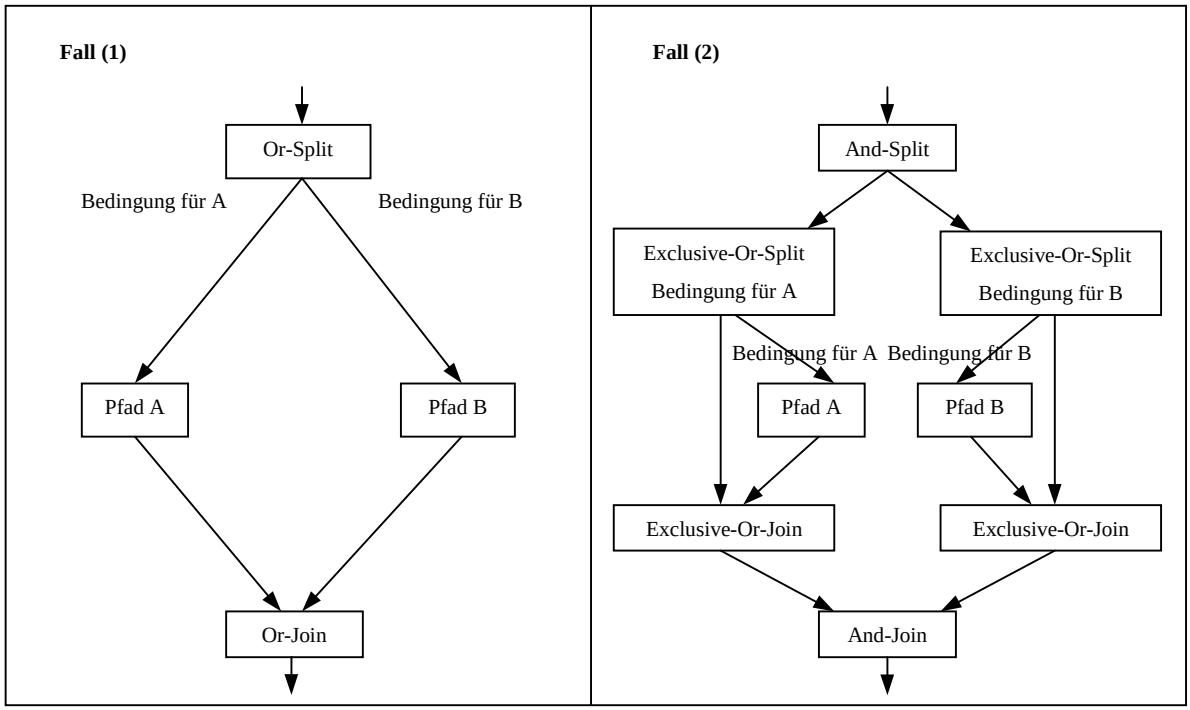


Abbildung 3-3: Logische Operatoren im Kontrollfluß.

Mit Hilfe der logischen Split- und Join-Operatoren können ausschließlich balancierte Workflows modelliert werden.
Ein balancierter Workflow ist ein Workflow, der der folgenden rekursiven Definition genügt:

(1) Enthält der Workflow einen Or-Split, so enthält er auch einen Or-Join.
(2) Enthält der Workflow einen And-Split, so enthält er auch einen And-Join.
(3) Jedem Or-Split ist ein Or-Join zugeordnet, d.h. alle alternativen Pfade, die im gleichen Or-Split beginnen, enden im gleichen Or-Join.
(4) Jedem And-Split ist ein And-Join zugeordnet, d.h. alle parallelen Pfade, die im gleichen And-Split beginnen, enden im gleichen And-Join.
(5) Ein Split und ein zugehöriger Join können in einem Pfad, der wiederum in einem Split beginnt, enthalten sein.

Der *Kontrollfluß* wird in CWDL ebenso wie in IBM FlowMark durch *Kontroll-Konnektoren* modelliert. Diese verbinden zwei Aktivitäten, die dadurch bezüglich des Kontrollflusses benachbart sind. Dabei ist die erfolgreiche Ausführung der Aktivität, an der der Konnektor beginnt, Voraussetzung für den Ausführungsbeginn der Aktivität, an der der Konnektor endet.

Falls ein CWDL-Kontroll-Konnektor in einem Or-Split beginnt, so ist er mit einer Übergangsbedingung behaftet.

Diese Übergangsbedingungen sind aussagenlogische Formeln.
Definiert ein Kontroll-Konnektor den direkten Übergang zwischen zwei Aktivitäten oder beginnt er in einem And-Split, so besitzt er keine Übergangsbedingung.

In einem CWDL-Workflow werden Bedingungen in Form von Abschluß-Bedingungen in Aktivitäten und als Bedingungen in Kontroll-Konnektoren angegeben. Diese Bedingungen setzen sich aus Termen zusammen.

Ein Term ist:

- eine Variable oder eine Konstante eines einfachen IDL-Datentyps.
- ein Methodenaufruf eines CORBA-Objektes.
- eine Verknüpfung zweier Terme durch
  (1) die boolschen Operatoren „&&“ (Konjunktion) und „||“ (Disjunktion)
  (2) die arithmetischen Operatoren „+“ (Addition), „-“ (Subtraktion), „*“ (Multiplikation), „/“ (Division), „%“ (Rest der Division beider Operatoren)
  (3) die Vergleichsoperatoren „<“, „>“, „<=“, „=>“, „==“ und „!=“ .

- ein durch „(“ und „)“ geklammerter Term.
- ein Term mit vorangestelltem unären Operator.
  (1) „!“ (boolsche Negation)
  (2) „-“ (arithmetische Negation)

Operatoren können auf bestimmte IDL-Basistypen angewandt werden (Tabellen 3-11 und 3-12). Konstruierte IDL-Datentypen und *Instanzen* von Objekt-Klassen können nur dann miteinander verglichen werden, wenn in der Interface-Definition ein Vergleichsoperator definiert ist.

Operator	Anwendbar auf IDL-Datentypen
&&, ||	Integer-Typen Float-Typen Fixed Type (Festkomma-Zahl) Boolean-Typ Octet-Typ
+, -, *, /	Integer-Typen Float-Typen Fixed Type Boolean-Typ Octet-Typ
%	Integer-Typen
!, -, + (unär)	Integer-Typen Float-Typen Fixed Type Boolean-Typ Octet-Typ

Tabelle 3-11: CWDL: Anwendung von Operatoren in Bedingungen (1).

Operator	Anwendbar auf IDL-Datentypen
<, >, <=, >=	Integer-Typen Float-Typen Fixed Type Character-Typ Wide-Character-Typ Boolean-Typ Octet-Typ Aufzählungstyp
!=, ==	Integer-Typen Float-Typen Fixed Type Character-Typ Wide-Character-Typ Boolean-Typ Octet-Typ Aufzählungstyp Strukturen Discriminated unions Sequenzen Array-Typen Objekt-Typen (für den Vergleich von Objekt-Referenzen, Vergleiche von Instanzen müssen durch Methoden der Objekte ermöglicht werden)

Tabelle 3-12: CWDL: Anwendung von Operatoren in Bedingungen (2).

Die Syntax der Terme ist aus der C++ -Syntax abgeleitet. Sie wird im Rahmen der gesamten CWDL-Syntax im Anhang dargestellt.

Wurde für eine Aktivität ein Startzeitpunkt definiert, so ist diese Information ebenfalls Bestandteil eines CWDL-Kontroll-Konnektors, der die Aktivität als Ziel besitzt.

> Eine CWDL-Aktivität kann einen Startzeitpunkt besitzen, der vom Abschluß-Zeitpunkt mehrerer anderer CWDL-Aktivitäten abhängig ist. Erst wenn alle im Startzeitpunkt benannten CWDL-Aktivitäten abgeschlossen und alle relativen Zeitangaben erfüllt wurden, kann die CWDL-Aktivität ausgeführt werden.

Daten-Konnektoren steuern die Datenübergabe zwischen zwei bezüglich des Datenflusses benachbarten Aktivitäten.
Eine Aktivität, die mit einer anderen Aktivität bezüglich des Datenflusses benachbart ist und Ziel des Datenflusses ist, besitzt Lese-Zugriff auf die lokalen Variablen der Quell-Aktivität.
Ein Daten-Konnektor bietet in Form eines *Mappings* die Möglichkeit, die Werte einzelner lokaler Variabler von der Quelle des Datenflusses in Variable der Ziel-Aktivität zu übertragen. Damit kann die Ziel-Aktivität in der Quell-Aktivität erzeugte Werte modifizieren, ohne sie über globale Variable des Prozesses zu übertragen.

Die Darstellung der CWDL-Syntax erfolgt in Form einer kontextfreien Grammatik.

> Aufgrund von Definitionen und Verwendung von Variablen, Personen, Organisationen, logischen Programmen und Refinement-Managern kann die korrekte Syntax einer CWDL-Workflow-Definition nicht ausschließlich durch die Anwendung der kontextfreien Grammatik überprüft werden. Die nötigen zusätzlichen Prüfungen (z.B. die korrekte Verwendung von Variablennamen) müssen mittels eines Semantik-Tests vorgenommen werden.

Workflows, die mit der in diesem Abschnitt beschriebenen Modellierungssprache erzeugt wurden, können auf keinem kommerziellen WfMS abgearbeitet werden. Deshalb wird ein Übersetzer benötigt, der Definitionen aus CWDL in die Workflow-Modellierungssprache von IBM FlowMark, FDL, übersetzt.

Im WfMS IBM FlowMark, aber auch in COSA Workflow können keine IDL-Datentypen verwendet werden. Deshalb werden bei einer Übersetzung von CWDL in FDL zusätzliche Module zur IDL-Datentyp-Verwaltung (z.B. Datenbank für IDL-Datentypen, im Abschnitt 3.3.2 beschrieben) benutzt. Diese Module besitzen hauptsächlich während der Laufzeit eines in FDL übersetzten Workflows Bedeutung.

3.3 Buildtime und Runtime

3.3.1 Grundlagen

Die Workflow-Modellierungssprache CWDL wurde ohne Bezug auf ein konkretes WfMS entwickelt. Um Prozesse, die in CWDL modelliert wurden, ausführen zu können, ist es nötig, die CWDL-Definitionen in die Modellierungssprache eines WfMS zu übersetzen. Um CWDL-Prozesse in IBM FlowMark ausführen zu können, müssen die relevanten Definitionen in FDL übersetzt werden.
CWDL unterscheidet sich in mehreren Punkten von FDL, nämlich bezüglich

- der unterstützten Datentypen,
- des Variablen-Konzeptes und
- der Prozeß-Modellierung, d.h. bezüglich der angebotenen Modellierungskonstrukte.

Um CWDL in FDL zu übersetzen, ist eine Abbildung zu definieren. Diese wird für jedes CWDL-Konstrukt einzeln bestimmt.

Die Abbildung der *Aufbauorganisation von CWDL* in die *Aufbauorganisation von FDL* erfolgt in eineindeutiger Weise dadurch, daß jedem Modellierungskonstrukt in CWDL genau ein Konstrukt in FDL zugeordnet ist (z.B. entspricht dem Rollen-Konstrukt in CWDL das Rollen-Konstrukt in FDL). Das folgt aus der Voraussetzung, daß die Aufbauorganisation von CWDL ohne Änderungen von FDL übernommen wurde.

Wesentlich komplexer ist die Abbildung der unterstützten Datentypen, des Variablen-Konzeptes und der Prozeß-Modellierung (*Ablauforganisation*) auf eine in IBM FlowMark realisierbare Weise.
Diese wird in den folgenden Abschnitten genauer betrachtet.

3.3.2 Die Datenbank für IDL-Datentypen

Wie bereits in Kapitel 2 beschrieben, können in IBM FlowMark keine IDL-Datentypen verwaltet werden. Insbesondere betrifft dies Instanzen dynamischer IDL-Datentypen (Sequenzen als Listentyp, dessen Elemente wiederum Instanzen eines beliebigen IDL-Datentyps sind; Strings) und CORBA-Objekte.

Die Verwaltung eines Datentyps in IBM FlowMark bedeutet, daß Instanzen dieses Datentyps zwischen zwei Aktivitäten ausgetauscht oder innerhalb von Bedingungen verwendet werden können.

Um die in CWDL angebotene Möglichkeit der Verwendung von IDL-Datentypen mit IBM FlowMark realisieren zu können, müssen die Instanzen der IDL-Datentypen (in der technischen Realisierung) außerhalb des WfMS verwaltet werden. Zusätzlich sind Mechanismen zu schaffen, die es IBM FlowMark ermöglichen, den Kontrollfluß in Abhängigkeit von den Werten der Instanzen zu realisieren und Instanzen zwischen zwei Aktivitäten auszutauschen.
Zunächst soll jedoch die Verwaltung der Instanzen beschrieben werden. Die Verwaltung von IDL-Datentyp-Instanzen muß folgenden Anforderungen gerecht werden:

(1) Es müssen IDL-Datentypen instanziert werden.
(2) Die Instanzen müssen über längere Zeit zur Verfügung stehen.

Zunächst soll Forderung (2) betrachtet werden. Um zwischen Aktivitäten eine Instanz eines IDL-Datentyps auszutauschen, muß die Instanz auch nach Beendigung der Quell-Aktivität weiterhin zur Verfügung stehen.

Das geschieht in IBM FlowMark bei den FlowMark-Datentypen durch Abspeicherung der (FlowMark-Datentyp-)Instanz in der internen ObjectStore-Datenbank des WfMS.
Im Falle der Verwendung von IDL-Datentypen muß dies ebenso zuverlässig, aber außerhalb des WfMS geschehen.

Die Instanzen von IDL-Datentypen, die für einen Workflow relevant sind, werden in einer externen Datenbank gespeichert.

Wichtigster Grund hierfür ist die Verwendung von IDL-Datentypen mit dynamischen Aspekten (Sequenzen, any usw.), die keinesfalls durch eine Abbildung auf FlowMark-Datentypen erreicht werden kann.

Die externe Datenbank enthält die Instanzen aller im Workflow vereinbarten Variablen. Zu beachten ist dabei, daß der Wert einer Variablen eines IDL-Interface-Typs immer eine Referenz auf ein CORBA-Objekt, das dem IDL-Interface entspricht, ist.
Das Datenbank-Schema in Form eines Entity-Relationship-Diagramms enthält Abbildung 3-4.
Dieses Schema, das an dieser Stelle zunächst nur als Übersichtsdarstellung wiedergegeben wird, soll im Kapitel 4 für die Implementierung weiter verfeinert werden.

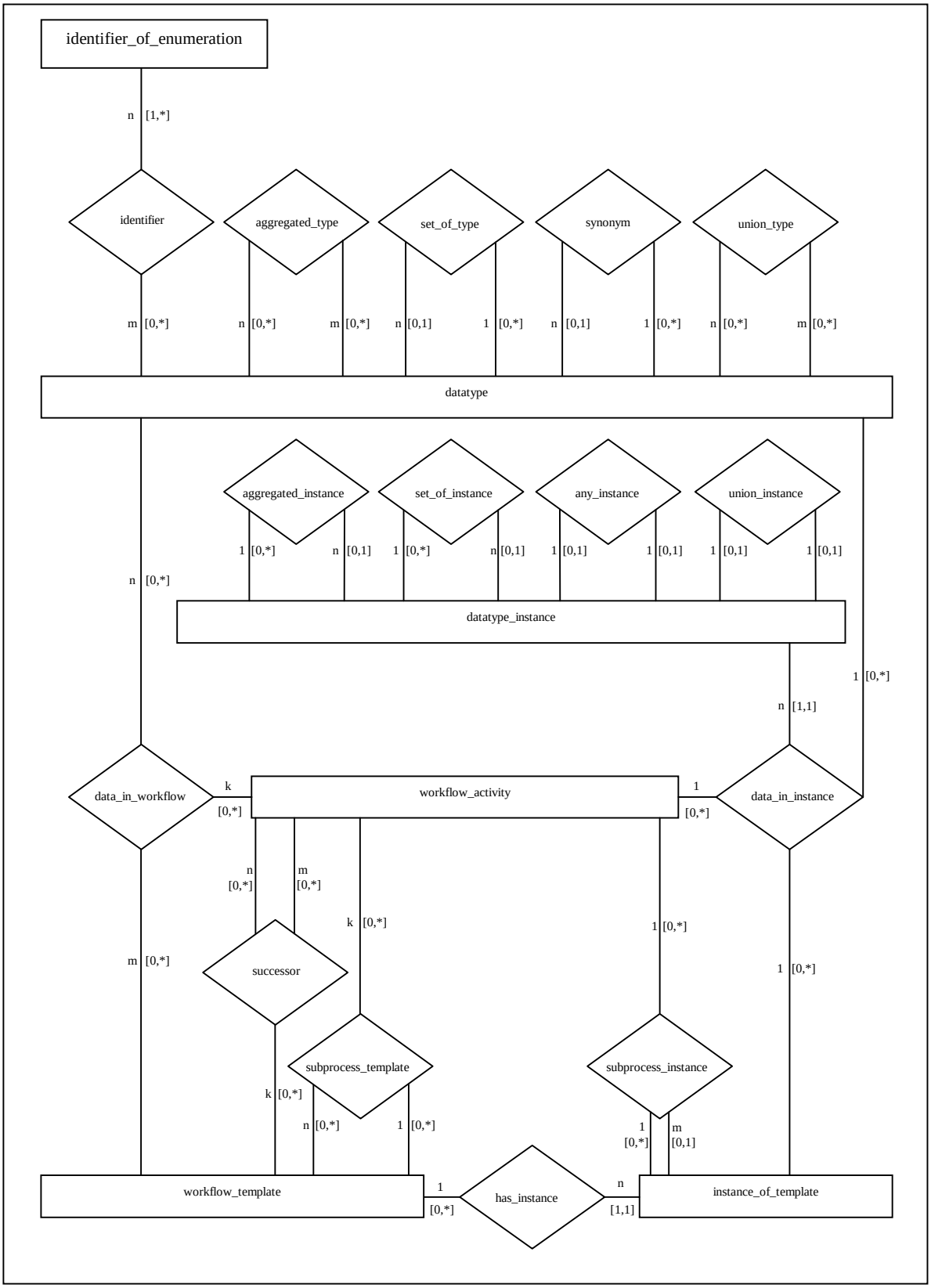


Abbildung 3-4: Die Darstellung von IDL-Datentypen in einer Datenbank.

Die Darstellung von IDL-Datentypen erfolgt generisch.

Das bedeutet, daß ein Datenbank-Schema erarbeitet wurde, in dessen Implementierung auch IDL-Datentypen verwaltet werden können, die *nach* der Generierung der Datenbank in einer IDL-Definition festgelegt wurden.

Damit besitzt das Datenbank-Schema zwei Aspekte. Es enthält einerseits als Meta-Information die Information über den konstruktiven Aufbau von IDL-Datentypen. Andererseits können diese Datentypen unter Verwendung dieser Meta-Information instanziert werden.

Die *Verwaltung der Meta-Information* erfolgt durch das Entity *datatype*. Hier werden zunächst die einzelnen IDL-Basistypen mit ihrem Namen sowie ihrem Typ (einfacher Typ, konstruierter Typ) angegeben. Die Konstruktion von IDL-Datentypen wird mittels der Relationships *identifier, aggregated_type, set_of_type, synonym* und *union_type* realisiert. Konstruierte Datentypen sind Aufzählungen (*identifier*), Strukturen (Aggregation; durch *aggregated_type* modelliert), Sequenzen (Mengenbildung; *set_of_type*), Arrays (Mengenbildung; *set_of_type*), Synonyme (*synonym*) und discriminated unions (*union_type*).

Das Relationship *synonym* repräsentiert zwei Konzepte von Datentypen. Dies sind einerseits einfache Synonyme, d.h. Typ-Namen, die mittels einer expliziten Namensdefinition in einer IDL-Definition für bereits vorhandene Typen festgelegt wurden. Andererseits können mit dieser Relation auch Vererbungshierarchien dargestellt werden.

Dabei interessiert besonders, welcher IDL-Interface-Typ die gleichen Dienste wie ein anderer IDL-Interface-Typ anbietet. Auf die vollständige Darstellung von Vererbungshierarchien wird verzichtet, da diese für eine Verwaltung von IDL-Datentypen nicht notwendig ist[20]. Vererbungshierarchien besitzen Einfluß auf die Schnittstelle und die damit verbundenen Dienste von CORBA-Objekten. Damit zusammenhängende Überprüfungen erfolgen jedoch an anderer Stelle (CWDL-Übersetzer).

Die *Verwaltung von Instanzen* von IDL-Datentypen erfolgt im Entity *datatype_instance*. Dabei gehört jede Instanz einem Typ an (im Relationship *data_in_instance* dargestellt).

Im Entity *datatype_instance* werden für alle Elemente eines IDL-Datentyps Instanzen angelegt.

[20] Es interessiert lediglich, welche Variable Instanzen welcher Typen enthalten kann.

Wird beispielsweise eine Sequenz instanziert, so wird eine Instanz des Typs Sequenz angelegt, die durch das Relationship *aggregated_instance* mit Instanzen ihres Element-Typs assoziiert ist. Dieser Element-Typ kann wiederum ein beliebiger IDL-Datentyp sein.

Die Elemente einer Instanz eines IDL-Datentyps sind durch die Relationships *aggregated_instance*, *set_of_instance*, *any_instance* und *union_instance* mit der konstruierten Instanz assoziiert.

Die *Darstellung von Werten* von Instanzen von IDL-Datentypen und IDL-Interface-Typen erfolgt durch die Abbildung auf Datentypen der verwendeten Datenbank. Die Abbildung wird in dieser Arbeit auf Datentypen des DBMS DB2, Version 5, vorgenommen. Die detaillierte Darstellung dieser Abbildung besitzt auf das Konzept dieser Arbeit keinen Einfluß und wird daher erst in Kapitel 4 behandelt. Trotzdem sollen an dieser Stelle zwei Beispiele herausgegriffen werden:

Es ist sinnvoll, einen IDL-String durch einen DB2-String und einen IDL-Integer durch einen DB2-Integer darzustellen[21].
Auf diese Weise können Typ-Konvertierungen beim Schreiben in oder Lesen aus der Datenbank vermieden oder zumindest verringert werden. Zusätzlich können für Integritätsbedingungen vorhandene Datenbank-Mechanismen besser genutzt werden (z.B. Überprüfung des Wertebereichs eines IDL-Integers als Constraint innerhalb einer Tabelle, da der IDL-Integer durch einen DB2-Integer dargestellt wird, dessen Wertebereich mindestens so viele Elemente wie der Wertebereich des IDL-Integers umfaßt).

Wesentlich für die Abbildung von IDL-Datentypen auf die Datentypen einer Datenbank ist an dieser Stelle, daß verschiedene Datenbank-Datentypen Verwendung finden.

Für die Darstellung von Werten der Instanzen einfacher IDL-Datentypen[22] oder IDL-Interface-Typen in Tabellen gibt es zwei Möglichkeiten:

- Wert als Attribut im Entity *datatype_instance* oder
- Einführung von Entities und Relationships zur Repräsentation der Werte für jeden IDL-Basis-Datentyp und für die Darstellung von Objekt-Referenzen von IDL-Interface-Typen.

[21] z.B. kann *short* (IDL) auf *smallint* (DB2) abgebildet werden.
[22] Mit Ausnahme des Types *any*.

Beide Darstellungsvarianten besitzen Vor- und Nachteile.

Die erste Variante vermeidet die Einführung weiterer Entities und Relationships, was sich positiv auf die Antwortzeit eines DBMS auswirken kann.

Betrachtet man aber die oben skizzierte Darstellung von IDL-Datentypen durch die Datentypen der Datenbank, so bedeutet das, daß das Attribut zur Wert-Repräsentation entweder zusammengesetzt sein muß, ein String ist oder, etwa in Form eines Union-Konstruktes der Datenbank, Werte beliebiger Datenbank-Typen zuläßt. Dadurch wird jedoch eine Überprüfung von Integritätsbedingungen wesentlich erschwert. Die Typ-Überprüfung wird komplex und muß durch Trigger erfolgen. Werden zusammengesetzte Attribute verwendet, so wird immer nur ein Element dieses Attributes verwendet, d.h. die Mehrzahl der Elemente des zusammengesetzten Attributes wird nicht verwendet.
Zusätzlich ist eine Implementierung dieser ersten Variante sehr stark vom konkreten DBMS abhängig. Bei einem Versions- oder Produktwechsel müssen unter Umständen weitreichende Änderungen vorgenommen werden.

Demgegenüber bietet die zweite Variante durch ihr modulares Konzept Vorteile, nimmt man den Nachteil zusätzlicher Entities und Relationships und damit evtl. verbundener Verlängerungen der Reaktionszeiten des DBMS in Kauf.
Gleichzeitig kann aber die Verwaltung der die Entities repräsentierenden Datenbank-Elemente (RDMBS: Tabellen) durch Übersichtlichkeit stark vereinfacht werden. Constraints und Trigger sind von einer geringeren Komplexität. Dadurch wird durch eine bessere Wartbarkeit eine geringere Fehlerrate bei der Implementierung erreicht. In Abhängigkeit von der Menge der verwalteten Daten könnten sogar positive Laufzeit-Effekte im Vergleich zur ersten Variante eintreten, d.h. wenn die Verknüpfung von kleinen Tabellen (RDBMS) während einer Datenbank-Abfrage weniger Zeit als das Suchen von Datensätzen innerhalb von großen Tabellen benötigt.
Die Abhängigkeit vom DBMS ist gering; bei Versions- und Produktwechsel sind notwendige Änderungen ohne großen Aufwand durchführbar.

Deshalb wird das Datenbank-Schema um Entities für die Darstellung von Werten von IDL-Datentypen ergänzt.
Die so entstandenen Beziehungen stellt Abbildung 3-5 am Beispiel des IDL-Datentyps *short* dar. Die Entity-Relationship-Diagramme der anderen, strukturell gleichen Beziehungen sind Bestandteil des Anhangs.

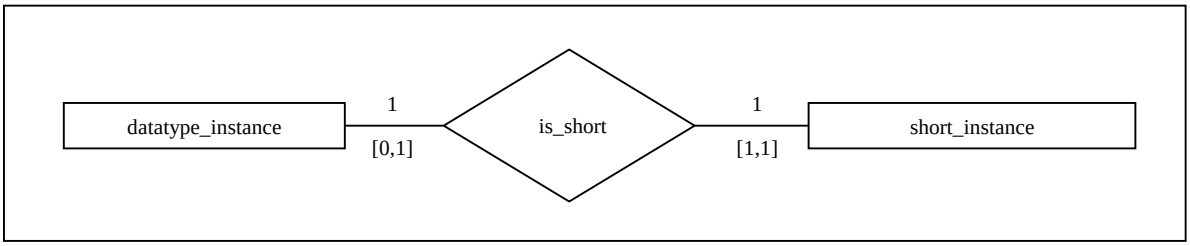


Abbildung 3-5: Die Darstellung von Werten des IDL-Datentyps *short*.

Instanzen vom *any-Typ* werden mittels des Relationships *any_instance* und des Entities *datatype_instance* repräsentiert. Die Dynamik dieses Typs (enthält zur Laufzeit eine Instanz eines beliebigen IDL-Datentyps) wird dabei durch sich verändernde Einträge im Relationship *any_instance* (Verweis auf Instanz des Element-Typs) sowie im Entity *datatype_instance* (Eintrag des jeweiligen Element-Typs) wiedergegeben.

Der Native-Typ fehlt im Datenbank-Schema, da seine Übersetzung aus der IDL-Definition in die Implementierungssprache eines CORBA-Objektes vom Implementierer festgelegt wird. Es kann keine Aussage über seine Eigenschaften getroffen werden.

> In Abhängigkeit von der konkreten Definition kann dieser Typ jedoch als konstruierter Datentyp in der Datenbank repräsentiert werden.

Während sich die bisherigen Überlegungen auf die Modellierung und Verwaltung von Instanzen von IDL-Datentypen in einer Datenbank bezogen, wird im folgenden auf die Zuordnung von Instanzen zu Workflows eingegangen.

Die Definition eines Arbeitsprozesses ergibt ein *Workflow-Template*.

> Ein Workflow-Template ist zur Definition eines Datentyps analog. Das Workflow-Template repräsentiert einen Workflow-Typ. Ein Workflow-Template besitzt einen eindeutigen Namen.

Wird ein Workflow gestartet, so erfolgt eine Instantierung des Workflows. Das Resultat wird im folgenden als *Workflow-Instanz* bezeichnet. Die Instanz besitzt ebenso wie das Workflow-Template einen eindeutigen Namen.
Ein Workflow enthält *Aktivitäten*. Jede Aktivität besitzt innerhalb eines Workflows einen eindeutigen Namen.

Um gemäß dem Variablen-Konzept der Sprache CWDL Variable beliebigen IDL-Typs lokal oder global im Workflow-Template definieren und in der Workflow-Instanz verwenden zu können, müssen für Verwaltung von IDL-Datentypen in einer Datenbank weitere Aspekte betrachtet werden.
Um eine Variable, die in einer Workflow-Instanz verwendet wird, in der Datenbank ordnungsgemäß zu instanzieren, muß sie der Workflow-Instanz zugeordnet werden. Die Workflow-Instanz wird dabei durch das Entity *instance_of_template*, ihr Template als *workflow_template* repräsentiert.

Es ist zu beachten, daß ein Workflow-Template in einer CWDL-Definition vorliegt. Das bedeutet, daß es nicht ausführbar ist. Dieses CWDL-Workflow-Template wird jedoch in FDL-Definitionen übersetzt. Nach der Übersetzung (im Abschnitt 3.3.3 beschrieben) ist die Ausführung des FDL-Workflows in IBM FlowMark möglich. Eine (technische) Workflow-Instanz ist daher immer eine Instanz eines FDL-Workflows. Aufgrund der eindeutigen Zuordnung eines FDL-Workflow-Template zum CWDL-Workflow-Template wurde auf ein Entity zur Darstellung von FDL-Workflow-Templates verzichtet.
Das Entity, das Aktivitäten darstellt, repräsentiert deren CWDL-Semantik. Wie im folgenden Abschnitt beschrieben, werden CWDL-Aktivitäten immer in eine oder mehrere FDL-Aktivitäten übersetzt. Dabei können die FDL-Aktivitäten ihrer CWDL-Aktivität immer eindeutig zugeordnet werden. Somit bezeichnet das Entity *workflow_activity* im Sinne von CWDL eine Aktivität, im Kontext von FDL aber eine Gruppe von Aktivitäten, die auf die entsprechenden Instanzen der IDL-Datentypen zugreifen können.
Es gilt also:

- Workflow-Template=CWDL-Workflow-Template
- Workflow-Instanz=FlowMark-Workflow-Instanz
- Aktivität=CWDL-Aktivität und
- Aktivität=Gruppe von FDL-Aktivitäten

Jeder Workflow kann eine oder mehrere globale Variable besitzen. Unter Angabe des Namens der Workflow-Instanz und des Namens der Variablen kann in eindeutiger Weise ein Zugriff auf globale Variable eines Workflows realisiert werden.
Um lokale Variable von Aktivitäten in der Datenbank darstellen zu können, muß neben einer Darstellung von Workflow-Instanzen ebenfalls eine Darstellung der Aktivitäten, die zu einem Workflow gehören, erfolgen. Jede Aktivität kann eine oder mehrere lokale Variablen besitzen.

Die Identifizierung einer lokalen Variablen einer Aktivität erfolgt durch die Angabe des Namens der Workflow-Instanz, des Aktivitätsnamens und des Variablennamens.

Globale und lokale Variable eines CWDL-Workflows werden durch das Relationship *data_in_workflow* modelliert. Für globale Variable wird dabei im Attribut der Aktivität ein Leerstring eingetragen.
Globale und lokale Variable einer Workflow-Instanz werden durch das Relationship *data_in_instance* modelliert. Für globale Variable wird ebenfalls im Attribut der Aktivität ein Leerstring eingetragen.

Um die Lebensdauer lokaler Variabler von Aktivitäten in der Datenbank zu modellieren, wird das Relationship *successor* eingeführt. Es repräsentiert den Datenfluß innerhalb eines CWDL-Workflows.

Zur Modellierung der Gültigkeit lokaler Variabler von Prozeß-Aktivitäten in Subprozessen dienen die Relationships *subprocess_template* und *subprocess_instance*.

Die Realisierung der Datenübergabe von IDL-Instanzen zwischen zwei Aktivitäten sowie die Steuerung eines FDL-Workflows geschieht durch zusätzliche Programme. Gemäß ihrer Aufgabe können sie in drei Klassen eingeteilt werden:

- Lesen und Schreiben eines Wertes aus der Datenbank in das WfMS:
  Programme dieser Klasse ermöglichen es, Werte der Integer-, Float-, Fixed-, Character-, Boolean- und Stringtypen aus der Datenbank in Input-Container von IBM FlowMark zu kopieren. Sie dienen einerseits der Realisierung einfacher Vergleiche zwischen Werten dieses Typs innerhalb von IBM FlowMark.
  Andererseits können derart eingelesene Werte als Kommandozeilen-Parameter von Legacy-Programmen dienen (Datenübergabe zwischen Aktivitäten).
- Vergleichen von Instanzen:
  Diese Programme realisieren einen Vergleich von Variablen von IDL-Datentypen, deren Instanzen nicht im WfMS dargestellt werden können. Sie liefern den Ergebniswert das Vergleiches an das WfMS.
- Ausführen von Methoden eines CORBA-Objektes:
  Derartige Programme können als ausführbare Programme vom WfMS aufgerufen werden und sind CORBA-Clients. Für die Übergabe von Parametern an die aufzurufende Methode werden Werte der Datenbank gelesen und modifiziert.

3.3.3 Die Abbildung von CWDL-Prozeß-Definitionen auf FDL-Prozeß-Definitionen

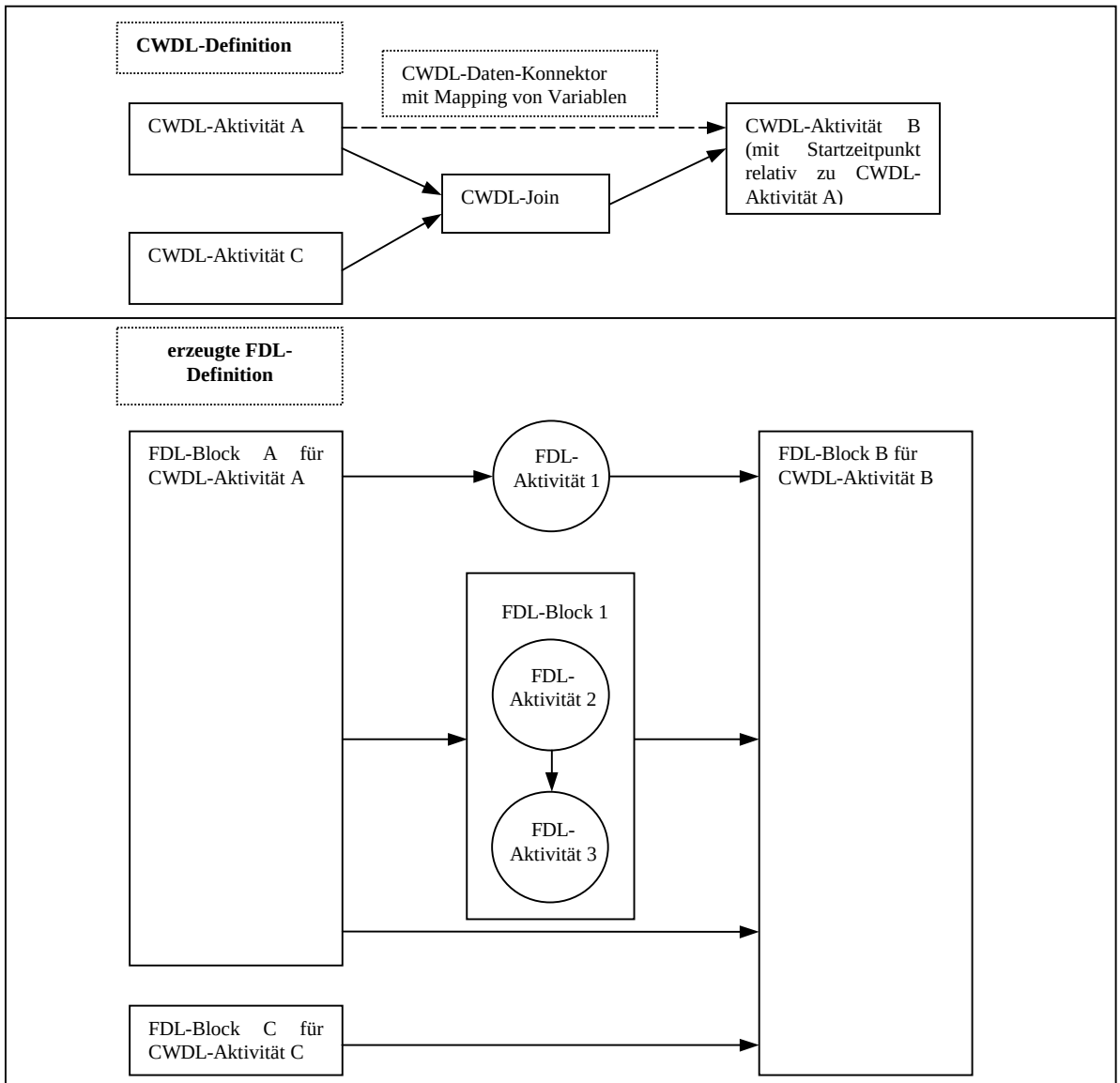
Wie bereits am Anfang dieses Kapitels beschrieben, unterscheidet sich CWDL von FDL nicht nur im Daten-Management, sondern auch in der Prozeß-Modellierung.
Dabei besitzen die Aktivitäten, die von CWDL angeboten werden, eine komplexere Semantik als FDL-Aktivitäten. Das bedeutet, daß eine CWDL-Aktivität als eine Folge von FDL-Aktivitäten implementiert werden muß.
Die Implementierung bzw. Abbildung von CWDL-Prozessen auf FDL-Prozesse zeigt dieser Abschnitt. Zunächst werden die Aspekte, die alle Aktivitäten betreffen, im Mittelpunkt stehen, danach folgen die aktivitätsspezifischen Aspekte.
Nach einer Darstellung der Übersetzung von CWDL-Split- und –Join-Konstrukten wird die Beschreibung mit der Übersetzung von CWDL-Kontroll- und –Daten-Konnektoren abgeschlossen.
Den schematischen Aufbau der Übersetzung von CWDL-Aktivitäten, -Daten- und -Kontroll-Konnektoren in FDL-Aktivitäten und –Kontroll-Konnektoren zeigen die Abbildungen 3-6 (Übersetzung von Prozessen) und 3-7 (Übersetzung von Aktivitäten).

Ergebnis der Übersetzung eines CWDL-Workflows ist immer genau ein FDL-Workflow.

Zunächst soll kurz die Realisierung eines CORBA-Methodenaufrufs in IBM FlowMark skizziert werden.
Gemäß der in Kapitel 2 beschriebenen Eigenschaften von IBM FlowMark und CORBA muß ein CORBA-Methodenaufruf durch einen CORBA-Client (CORBA-Eigenschaft) erfolgen, der ein ausführbares Programm ([IBM96a]) ist. Daher wurde im Projekt „HematoWork“ der Ansatz des *Wrappers* gewählt.

> Ein Wrapper ist ein Modul, das zwei oder mehrere Systeme bezüglich ihrer Eigenschaften kapselt und eine Schnittstelle zwischen den beteiligten Systemen bereitstellt.
>
> Parametrisiert durch Angabe des Namens der FlowMark-Prozeß-Instanz, des CWDL-Aktivitäts-Namens, der Referenz des CORBA-Objektes, des Methodennamens sowie der benötigten Methodenparameter wird eine festgelegte Methode vom Wrapper ausgeführt und die Ergebniswerte zurückgeliefert.

Wesentlich für die folgenden Betrachtungen ist, daß der Wrapper als ausführbares Programm konzipiert ist.



→	CWDL-Kontroll-Konnektor (CWDL-Definition) FDL-Kontroll-Konnektor (erzeugte FDL-Definition)
- - - →	CWDL-Daten-Konnektor (CWDL-Definition)
FDL-Aktivität 1	FDL-Aktivität zur Übersetzung der Abbildungsvorschrift des CWDL-Daten-Konnektors (Kopieren von Werten innerhalb der IDL-Datentyp-Datenbank)
FDL-Block 1	FDL-Block zur Übersetzung der Startbedingung der CWDL-Aktivität B
FDL-Aktivität 4	FDL-Aktivität für die Übersetzung von CWDL-And- und- Or-Join-Konstrukten

Abbildung 3-6: Überblick über die Übersetzung der Ablauforganisation (1)[23].

[23] FDL-Kontroll-Konnektoren ohne Angabe einer Übergangs-Bedingung stellen eine Parallelisierung dar (Siehe Abschnitt 2.2.1).

Jeder Aufruf einer CORBA-Methode wird in den Aufruf des Wrappers mit der Angabe von Parametern übersetzt. Konkrete Anforderungen an das Aufruf-Format des Wrappers werden im Kapitel 4 beschrieben.

Die Implementierung des Wrappers ist nicht Gegenstand dieser Arbeit, sondern wird im Rahmen einer anderen Arbeit behandelt.

Verwendung findet der Wrapperansatz für die Übersetzung der *Applikations*- und *der Skript-Aktivität* in FDL-Konstrukte.

Das Schema der Übersetzung von CWDL-Prozess-Definitionen zeigt Abbildung 3-6. Dieses Schema muß für die Übersetzung von CWDL-Aktivitäten weiter verfeinert werden.

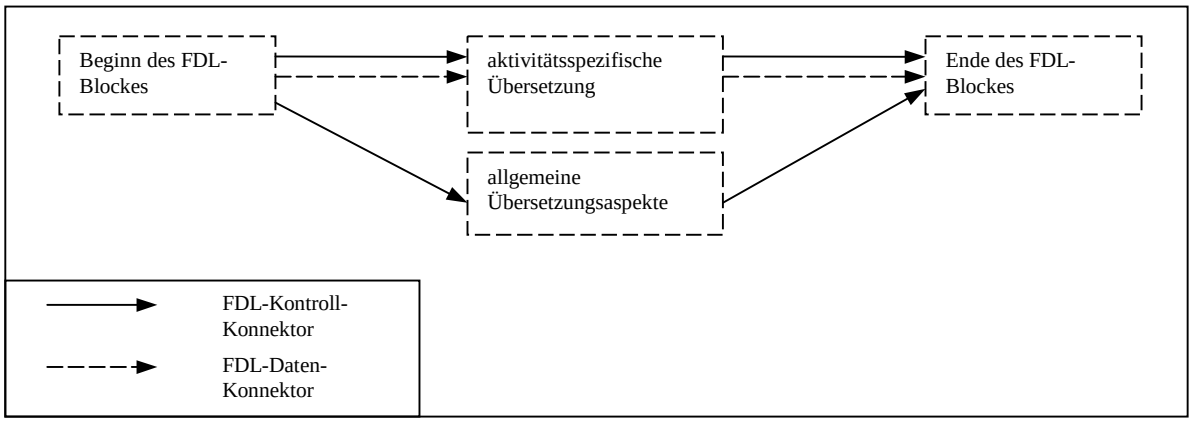


Abbildung 3-7: Überblick über die Übersetzung der Ablauforganisation (2)[24].

Für die Übersetzung von CWDL in FDL wird eine Aktivität zweigeteilt (Abbildung 3-7). Einerseits sind Eigenschaften zu übersetzen, die alle Aktivitäten besitzen (Nutzer-Berechtigung, Abschluß-Bedingung, Definition lokaler Variabler). Andererseits gibt es aktivitätsspezifische Aspekte (Programm-Definitionen für Applikations-Aktivität, Realisierung der Kommunikation von Nachrichten-Aktivitäten, Prozesse in Prozeß-Aktivitäten).

3.3.3.1 Allgemeine Übersetzungsaspekte

Bei der Übersetzung von CWDL in FDL gibt es Mechanismen, die für alle CWDL-Aktivitäts-Arten gültig sind.

[24] FDL-Kontroll-Konnektoren ohne Angabe einer Übergangs-Bedingung stellen eine Parallelisierung dar (Siehe Abschnitt 2.2.1).

Eine CWDL-Aktivität muß im allgemeinen in FDL durch mehrere Aktivitäten repräsentiert werden. Einerseits muß die aktivitätsspezifische Funktionalität modelliert werden. Andererseits existieren in jeder CWDL-Aktivität Abschluß-Bedingungen.

> Die aktivitätsspezifische Funktionalität und die Abschluß-Bedingungen können nicht innerhalb der gleichen FDL-Aktivität modelliert werden.
> Durch die Möglichkeit, in Bedingungen CORBA-Methoden angeben zu können, wird es notwendig, daß für die Evaluierung von Bedingungen eine FDL-Programm-Aktivität zum Aufruf eines externen Modules zur Bedingungsauswertung generiert wird.

Deshalb wird jede CWDL-Aktivität in FDL durch einen FDL-Block[25] dargestellt.

> Dadurch kann der resultierende FDL-Workflow einfach und übersichtlich gestaltet werden. Seine Struktur entspricht im wesentlichen der Struktur des CWDL-Workflows, es werden jedoch weitere, tiefere Hierarchie-Ebenen im FDL-Prozeß sowie FDL-Blöcke und FDL–Aktivitäten für die Realisierung des Startzeitpunktes und zur Auswertung von Übergangsbedingungen von CWDL-Kontroll-Konnektoren generiert.
> Dadurch wird das erzeugte *FDL-Workflow-Template* übersichtlich. Ferner läßt sich vermeiden, daß während der Übersetzung neue FDL-Prozesse generiert werden, die im WfMS eindeutig sein müssen. Es muß nicht getestet werden, ob FDL-Subworkflows (anstelle des Blocks für eine CWDL-Aktivität generiert) im WfMS eindeutig sind.

Mit diesen Voraussetzungen wird nun die Übersetzung von Eigenschaften aller CWDL-Aktivitäten beschrieben. Die Ausführungen orientieren sich an den Tabellen 3-5 und 3-6.

Allgemeine Definitionen:

- Der generierte FDL-Block erhält den Namen der CWDL-Aktivität. FDL-Aktivitäten innerhalb des Blockes werden fortlaufend numeriert.
- Die Beschreibung der CWDL-Aktivität wird in den FDL-Block und in alle FDL-Aktivitäten des Blockes übernommen.

Ende einer CWDL-Aktivität:

- Wurde in einer Aktivität eine Abschluß-Bedingung angegeben, so muß für deren Auswertung eine Programm-Aktivität in den FDL-Block der CWDL-Aktivität eingefügt werden.

[25] FDL-Konstrukt zur Modellierung von hierarchischen Prozessen (siehe Abschnitt 2.2.1).

Das zugehörige FDL-Programm wertet die Bedingung aus. Der bei der Auswertung des Vergleiches erhaltene boolsche Wert wird in den Output-Container der FDL-Programm-Aktivität eingetragen. Er kann dann innerhalb des WfMS in Form der Abschluß-Bedingung des FDL-Blockes, der die CWDL-Aktivität repräsentiert, ausgewertet werden.

Lokale Variable[26]:

- Für die lokalen Variablen einer CWDL-Aktivität erfolgen während der CWDL-FDL-Übersetzung in der IDL-Datentyp-Datenbank die entsprechenden Einträge[27].
  Für die Instantierung und das Löschen von Instanzen von Variablen werden in den Block der CWDL-Aktivität zwei weitere FDL-Blöcke eingefügt. Im ersten Block werden sämtliche lokalen Variablen der CWDL-Aktivität in der IDL-Datentyp-Datenbank durch die Ausführung eines Programmes instanziert, im zweiten Block erfolgt das Löschen der Instanzen.

Ausführungsberechtigungen:

- Die Ausführungsberechtigungen der CWDL-Aktivität werden mit Ausnahme der Skript-Aktivität (siehe weiter unten) für alle generierten FDL-Aktvitäten übernommen.

Ausnahmebehandlung (bei Termin-Überschreitung):

- Von den angebotenen Möglichkeiten der Ausnahmebehandlung für die Überschreitung der Bearbeitungsfrist einer CWDL-Aktivität kann immer nur eine gewählt werden.
- Die Ausnahmebehandlung durch festgelegte Personen wird im FDL-Workflow für den Teil der aktivitätsspezifischen Übersetzung direkt aus der CWDL-Definition übernommen. Für technische FDL-Aktivitäten wird als Ausnahmeberechtigter der Administrator des Systems angegeben.
- Die Ausnahmebehandlung durch einen FDL-Subprozeß wird ebenfalls nur für den aktivitätsspezifischen Teil der Übersetzung vorgenommen. Für die Ausnahmebehandlung aller technischen Aktivitäten ist wiederum der Administrator zuständig.

  Die Ausnahmebehandlung wird durch einen FDL-Block modelliert (Abbildung 3-8; Block 1).

[26] Die Ausführungen gelten auch für globale Variable des CWDL-Workflows.
[27] Es erfolgt unter Verwendung des CWDL-Prozeß-Namens, des CWDL-Aktivitätsnamens, des IDL-Typ-Namens und des Variablen-Namens für jede Variable ein Eintrag in die (im Abschnitt 4.3 beschriebenen) Tabellen *workflow_activity*, *workflow_template*, *data_in_workflow*, *successor* und *subprocess_template*.

In IBM FlowMark wird nach dem Abschluß einer Aktivität sofort die nächste ausführbar, sofern die entsprechenden Daten vorliegen und alle Kontroll-Konnektoren evaluiert wurden. Auf diese Weise kann aber auch die Ausführung einer Aktivität verzögert werden. Zu diesen Zweck werden zwei Programm-Aktivitäten in den *FDL-Block 1* eingefügt.

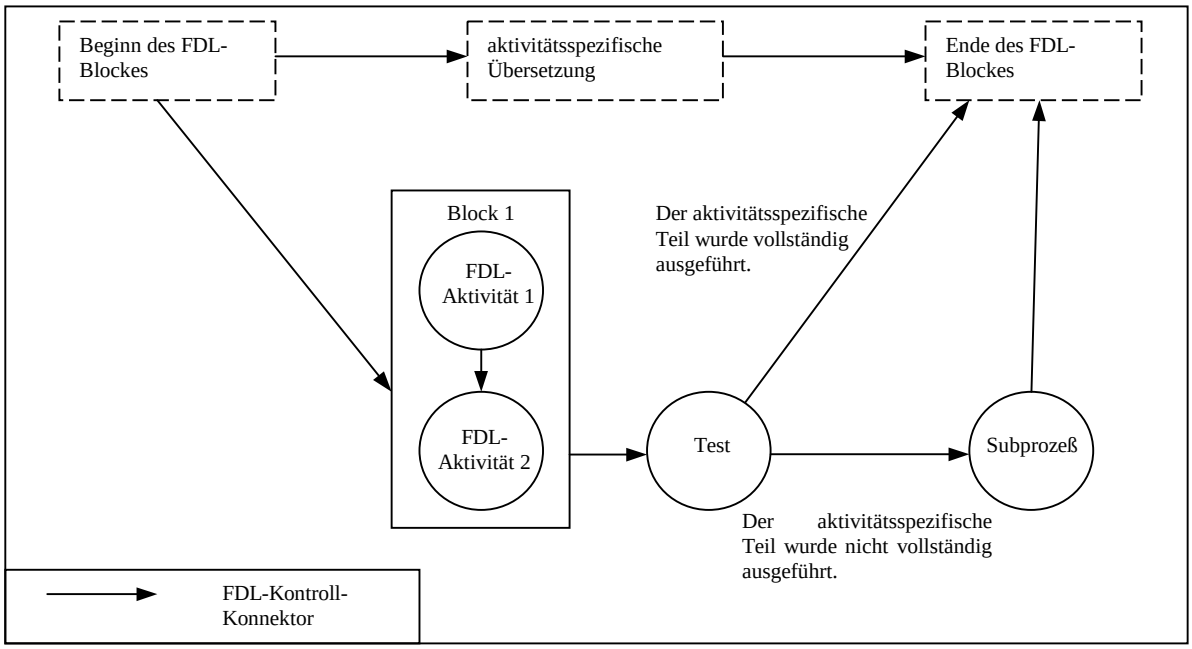


Abbildung 3-8: Übersetzung des spätestmöglichen Abschlußzeitpunktes einer Aktivität (innerhalb eines Blockes einer CWDL-Aktivität)[28].

Mit Hilfe eines externen Moduls können Ereignisse generiert werden. Das kann mit dem CORBA Time Service erfolgen. Dieser ermöglicht es, nach einer festgelegten Zeit ein Ereignis zu erzeugen und ein anderes CORBA-Objekt zu benachrichtigen.

Durch die *FDL-(Programm-)Aktivität 1* wird dem externen Modul der Zeitpunkt für die Generierung des Ereignisses mitgeteilt. Gleichzeitig kann der Empfänger[29] des später generierten Ereignisses benannt werden.

Der Status einer Aktivität kann modifiziert werden[30]. Das erfolgt über die FlowMark-API.

[28] FDL-Kontroll-Konnektoren ohne Angabe einer Übergangs-Bedingung stellen eine Parallelisierung dar (Siehe Abschnitt 2.2.1).
[29] Methode des als CORBA-Objekt im CORBA-Netzwerk repräsentierten WfMS IBM FlowMark.
[30] Die Stati einer Aktivität wurden in Abschnitt 2.2.1 beschrieben.

Die *FDL-(Programm-)Aktivität 2* wird derart konfiguriert, daß sie sofort nach Beendigung der *FDL-(Programm-)Aktivität 1* ausführbar (*ready*) wird. Als Ausführungsberechtigter wird eine einzige Person festgelegt, die allerdings kein Nutzer des WfMS ist.

Die Person wird nur symbolisch eingerichtet, um ihre Arbeitsliste als Puffer für derart ausführbare FDL-Aktivitäten zu nutzen. In diese Arbeitsliste wird die zweite FDL-Aktivität als ausführbar eingetragen.

Wird ein Ereignis durch das externe Modul generiert (zum Zeitpunkt, an dem die CWDL-Aktivität beendet sein muß), so kann über die FlowMark-API bewirkt werden, daß die ausführbare zweite Aktivität den Status beendet (*finished*) erhält. Dadurch wird die FDL-Aktivität *Test* ausführbar.

Bei der Ausführung der FDL-Aktivität *Test* wird überprüft, ob der Bereich der aktivitätsspezifischen Übersetzung vollständig durchlaufen wurde.

Dieser Test erfolgt durch ein Programm, das über die FlowMark-API den Status der im FDL-Kontrollfluß letzten aktivitätsspezifischen FDL-Aktivität(en) abfragt. Wurden diese erfolgreich ausgeführt, so besitzt der Status den Wert *finished*.
Ist das nicht der Fall, so wird der entsprechende Subprozeß gestartet.

Dokumentation:

- Die Dokumentation dient der ausführlichen Beschreibung der CWDL-Aktivität und wird in alle Aktivitäten des FDL-Blocks übernommen.

3.3.3.2 Aktivitätsspezifische Übersetzungsaspekte

Eine *Applikations-Aktivität* kann sowohl den Aufruf einer CORBA-Methode als auch eine Legacy-Applikation repräsentieren.

Enthält die Applikations-Aktivität einen CORBA-Methodenaufruf, so muß bei ihrer Übersetzung ein Wrapper-Aufruf generiert werden. Es ist also, wenn noch nicht früher generiert, eine FDL-Programm-Definition zu erzeugen, die den Wrapper im WfMS repräsentiert.

Neben der Angabe der physischen Eigenschaften des Wrappers ist besonders die Information über das involvierte CORBA-Objekt, seine aufzurufende Methode und die zu übergebenden Parameter von großer Bedeutung. Es ist zu erwarten, daß innerhalb einer CWDL-Workflow-Definition mehrfach Methoden von CORBA-Objekten aufzurufen sind. Deshalb muß die FDL-Programm-Definition für den Wrapper so generiert werden, daß das FDL-Programm an allen Stellen der Übersetzung eines Methodenaufrufs verwendet werden kann. Dies bedeutet, daß die Kommunikation zwischen Wrapper und WfMS durch Verwendung von Variablen erfolgt.

Das kann einerseits geschehen, indem der Wrapper lediglich auf seinen Input-Container zugreift, in dem alle Werte enthalten sind. Andererseits kann dem Wrapper auch in Form von (FDL-)Variablennamen auf der Kommandozeile die entsprechende Information übergeben werden. Diese Variablennamen bezeichnen Elemente des Input-Containers des FDL-Programms.

Im ersten Fall muß der Wrapper mittels der FlowMark-API auf seinen Input-Container zugreifen, während das im zweiten Fall nicht nötig ist. Trotzdem besitzt die erste Variante besonders bei einer großen Menge von Übergabeparametern Vorteile, da die Kommandozeile in IBM FlowMark einer Längenbeschränkung unterworfen ist.

Die Programm-Definition, die den Wrapper kapselt, wird so konfiguriert, daß die Ausführung des Wrappers für den Nutzer transparent ist, d.h. es erfolgt zwischen Nutzer und Wrapper keine Interaktion.

Soll eine Legacy-Applikation zum Einsatz kommen, so wird in der Applikations-Aktivität die Definition eines *logischen CWDL-Programms* referenziert, dem innerhalb der Applikations-Aktivität Parameter übergeben werden.

Die Definition des logischen Programms erfolgt in CWDL analog der entsprechenden Definition in FDL mit der Ausnahme, daß im Gegensatz zu FDL in CWDL keine Daten-Container innerhalb der Definition angegeben werden.

Bei der Übersetzung eines logischen Programms von CWDL nach FDL wird zunächst die CWDL-Definition ohne Änderungen syntaktisch in das entsprechende FDL-Konstrukt übersetzt. Es ist jedoch zu beachten, daß innerhalb einer Applikations-Aktivität Parameter für das logische Programm festgelegt wurden. Diese Parameter sind Konstanten oder Variable eines IDL-Basis-Datentyps. Deshalb muß die im ersten Übersetzungsschritt erhaltene FDL-Programm-Definition um die zusätzlichen Parameter erweitert werden.

Wird das logische Programm in der CWDL mehrfach und mit unterschiedlichen Parametern verwendet, so muß eine Übersetzung in eine FDL-Programm-Definition vorgenommen werden, die diese „dynamischen“ Aspekte berücksichtigt. Das bedeutet, daß in der FDL-Programm-Definition keine CWDL-Konstanten auf der Kommandozeile angegeben werden dürfen.

Werden in CWDL einem logischen Programm Parameter in Form von Variablen einfacher IDL-Datentypen übergeben, denen das Schlüsselwort „VALUE“ vorangestellt ist, so muß in der Übersetzung der Applikations- und Skript-Aktivitäten (FDL-Block) zusätzlich eine FDL-Programm-Aktivität eingefügt werden, durch deren Ausführung die entsprechenden Werte in einen Daten-Container von IBM FlowMark geschrieben werden.

Die durch die FDL-Programm-Definition, die für das entsprechende CWDL-Konstrukt erzeugt wurde, gekapselte Applikation muß ihre Übergabe-Parameter aus ihrem Input-Container erhalten. Falls die Applikation im Rahmen des Projektes „HematoWork“ erstellt wurde, so ist es möglich, daß sie direkt auf ihren Input-Container zugreift. Im allgemeinen ist jedoch davon auszugehen, daß das nicht der Fall ist. Deshalb müssen die Werte des Input-Containers auf der Kommandozeile in Form von FDL-Variablennamen angegeben werden. Diese Variablennamen bezeichnen die entsprechenden Elemente des Input-Containers, die die Information enthalten[31].

Der aktivitätsspezifische Teil einer Applikations-Aktivität wird immer in mindestens eine FDL-Programm-Aktivität übersetzt.

In dieser Programm-Aktivität wird entweder der *Wrapper* für die Ausführung von CORBA-Methoden oder ein Legacy-Programm aufgerufen.
Müssen zusätzlich Werte aus der IDL-Datentyp-Datenbank gelesen werden, sind, wie beschrieben, weitere FDL-Programm-Aktivitäten nötig.

Zusätzlich werden FDL-Datenstrukturen und FDL-Programm-Definitionen generiert, falls dies noch nicht bei der Übersetzung anderer CWDL-Aktivitäten geschehen ist.

[31] Soll zum Beispiel eine Text-Datei mit dem Programm Microsoft Word bearbeitet werden, so kann man das Programm mit dem Aufruf „WinWord <Dateiname>“ starten. Um den Datei-Namen einem Input-Container von IBM FlowMark zu entnehmen, muß der Parameter als %<InputContainerElementMitDateiNamen>% angegeben werden.

Der aktivitätsspezifische Teil der *Skript-Aktivität* ist in seiner Konzeption der Applikations-Aktivität ähnlich. Im Gegensatz zur Applikations-Aktivität werden in der *Skript-Aktivität* jedoch mehrere Applikationen aufgerufen. Deshalb wird der aktivitätsspezifische Teil der Skript-Aktivität in eine Folge von FDL-Programm-Aktivitäten (ggf. auch in FDL-Programm-Definitionen und FDL-Datenstrukturen) übersetzt. Da eine Skript-Aktivität nur durch einen einzigen Nutzer auszuführen ist, werden die FDL-Programm-Aktivitäten sequentiell angeordnet. Der Name desjenigen Nutzers, der die Ausführung der ersten FDL-Programm-Aktivität (also der Skript-Aktivität) startet, wird in die Input-Container der folgenden FDL-Aktivitäten eingetragen. Diese werden so konfiguriert, daß die Information über die Nutzerberechtigung zur Laufzeit durch den Input-Container erfolgt. Damit ist gewährleistet, daß bei Aufteilung einer CWDL-Aktivität in mehrere, möglicherweise mit dem Nutzer kommunizierende FDL-Aktivitäten, alle FDL-Aktivitäten, wie in CWDL modelliert, vom gleichen Nutzer bearbeitet werden.

> Da eine Skript-Aktivität häufig nur die Ausführung automatischer Aufgaben repräsentiert, besitzt dieser Aspekt der Nutzerverwaltung eine untergeordnete Bedeutung.

Der aktivitätsspezifische Teil der *Manuellen Aktivität* wird ebenfalls auf eine FDL-Programm-Aktivität abgebildet.

> Innerhalb dieser Programm-Aktivität wird ein Programm ausgeführt, welches vom Nutzer lediglich eine Bestätigung der manuell ausgeführten Tätigkeit verlangt.

Der aktivitätsspezifische Teil der *Nachrichten-Sender-Aktivität* wird in FDL als einfache FDL-Programm-Aktivität modelliert. Dem aufgerufenen Programm werden die nötigen Werte als Kommandozeilen-Parameter übergeben.

> Das WfMS, in dem der Empfänger-Workflow abgearbeitet wird, wird als CORBA-Objekt im CORBA-Netzwerk vorausgesetzt. Dieses CORBA-Objekt bietet eine Methode für den Nachrichten-Empfang an. Innerhalb dieser Methode wird unter Verwendung der eigenen IDL-Datentyp-Datenbank ermittelt, welcher Workflow sich für den Empfang der Nachricht qualifiziert. Die Empfänger-Aktivität überprüft die Berechtigung des Senders. Falls dieser Test positiv beendet wird, werden die empfangenen Daten in die IDL-Datentyp-Datenbank eingetragen. Im negativen Fall wartet die Aktivität weiterhin auf eine eingehende Nachricht.

Der aktivitätsspezifische Teil der *Nachrichten-Empfänger-Aktivität* wird ähnlich wie die Ausnahmebehandlung einer Aktivität modelliert (Abbildung 3-9).

Im FDL-Block der Nachrichten-Empfänger-Aktivität wird eine FDL-Aktivität eingefügt, die mit Beginn des Blockes sofort ausführbar wird (*FDL-Aktivität 1*). Der zur Ausführung berechtigte Nutzer ist wie bei der Übersetzung von Abschlußzeitpunkten von CWDL-Aktivitäten ein symbolisch definierter Nutzer, in dessen Arbeitsliste die Aktivität als ausführbar gekennzeichnet wird. Wird eine Nachricht empfangen, so erhält diese FDL-Aktivität über die API den Status *finished.* Damit kann eine folgende Aktivität ausgeführt werden, die überprüft, ob der Sender qualifiziert war (*Test*). Ist das der Fall, so können die übergebenen Werte in die IDL-Datentyp-Datenbank eingetragen werden.

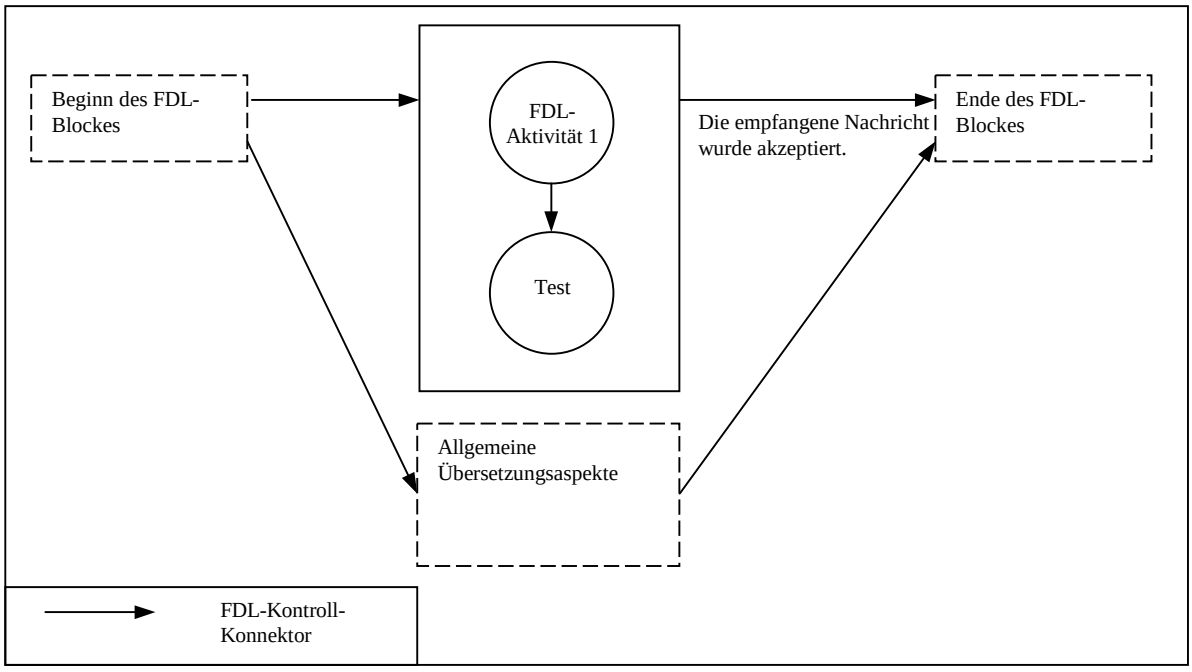


Abbildung 3-9: Die Nachrichten-Empfänger-Aktivität in FDL[32].

Der aktivitätsspezifische Teil der *CWDL-Prozeß-Aktivität* wird direkt in eine FDL-Prozeß-Aktivität übersetzt.

Der aktivitätsspezifische Teil der *Call-Aktivität* wird in eine FDL-Programm-Aktivität sowie zugehörige Datenstrukturen und eine symbolische FDL-Programm-Definition (falls noch nicht vorhanden) übersetzt. Im angegebenen Programm wird über die FlowMark-API eine Instanz des spezifizierten Workflows asynchron gestartet.

[32] FDL-Kontroll-Konnektoren ohne Angabe einer Übergangs-Bedingung stellen eine Parallelisierung dar (Siehe Abschnitt 2.2.1).

Der aktivitätsspezifische Teil der *Refinement-Aktivität* wird wie der Aufruf von CORBA-Methoden behandelt. Der Refinement-Manager, als CORBA-Objekt implementiert, wird mit seinen nötigen Parametern unter Verwendung des Wrappers ausgeführt.

Ein *CWDL-Kontroll-Konnektor* wird in FDL wiederum durch einen *Kontroll-Konnektor* repräsentiert. Dieser besitzt im allgemeinen keine Übergangsbedingungen und wird damit nach positivem Abschluß der vorangegangenen FDL-Aktivität durchlaufen. Er verbindet zwei FDL-Blöcke, die CWDL-Aktivitäten in FDL simulieren. Falls eine CWDL-Aktivität allerdings mit einer Startbedingung (Startzeitpunkt) definiert ist oder ein CWDL-Kontroll-Konnektor in einem Or-Split beginnt, so wird der CWDL-Kontroll-Konnektor in mehrere FDL-Kontroll-Konnektoren übersetzt. Abbildung 3-10 zeigt die Übersetzung eines mit einer Übergangsbedingung bewerteten CWDL-Kontroll-Konnektors.

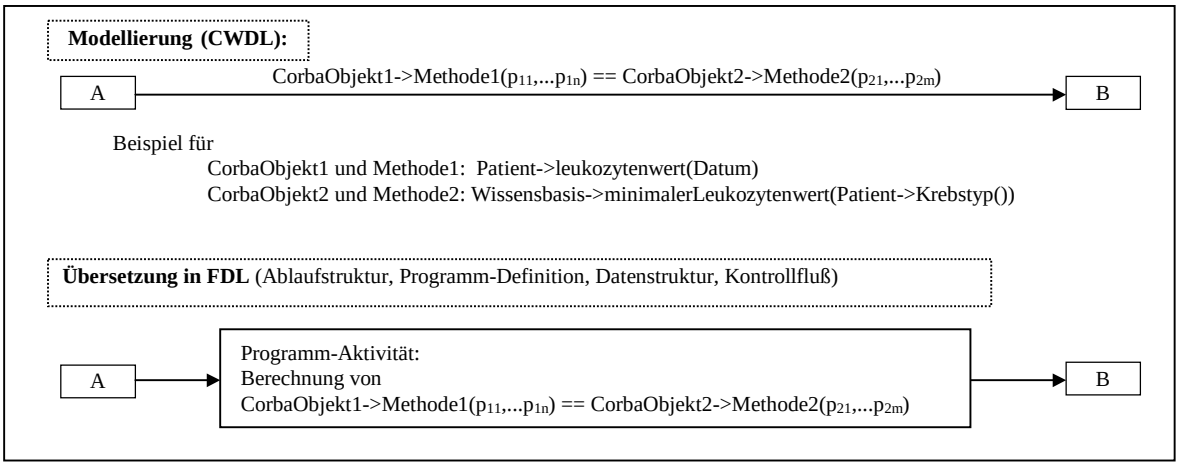


Abbildung 3-10: Die Übersetzung eines CWDL-Kontroll-Konnektors.

Die Übersetzung einer Startbedingung zeigt Abbildung 3-11.

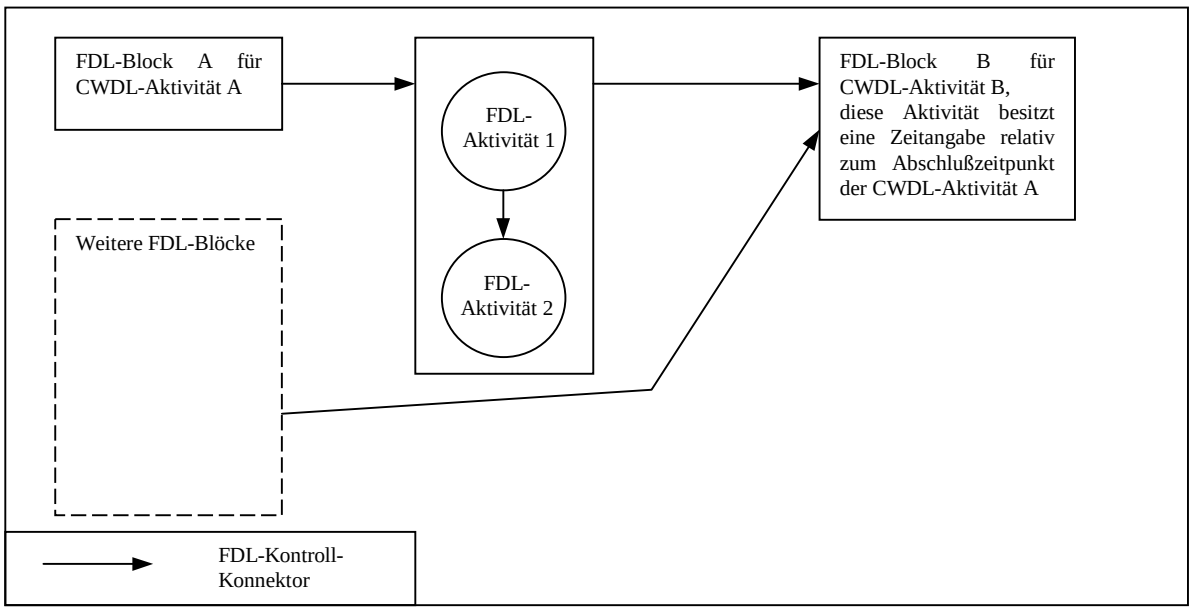


Abbildung 3-11: Übersetzung des Startzeitpunktes von CWDL in FDL.

Besitzt ein CWDL-Kontroll-Konnektor eine Übergangsbedingung und eine Startbedingung, so wird er in 4 FDL-Konnektoren, eine FDL-Programm-Aktivität (Übergangsbedingung) und einen FDL-Block übersetzt. Das zeigte bereits Abbildung 3-6.
Da für eine CWDL-Aktivität der Startzeitpunkt relativ zu *mehreren* anderen Aktivitäten definiert werden kann, werden für jede weitere Startbedingung 2 FDL-Kontroll-Konnektoren und ein FDL-Block in den FDL-Prozeß eingefügt.

> Damit kann eine CWDL-Aktivität erst dann ausgeführt werden, das zeitliche Maximum aller Startzeitpunkte erreicht ist.

Ein *CWDL-Daten-Konnektor* wird in FDL durch einen FDL-Kontroll-Konnektor zwischen den FDL-Blöcken, die CWDL-Aktivitäten modellieren, repräsentiert. Das besagt, daß die Daten der einen Aktivität zur Ausführung der folgenden vorliegen müssen.
In der IDL-Datentyp-Datenbank wird der CWDL-Daten-Konnektor durch einen Eintrag in der Beziehung *successor* abgebildet. Enthält ein CWDL-Daten-Konnektor eine Abbildungsvorschrift zwischen zwei CWDL-Aktivitäts-Variablen, so muß in den CWDL-Workflow eine Programm-Aktivität zwischen den beiden CWDL-Aktivitäten eingefügt und mit diesen mittels eines Kontroll-Konnektors verbunden werden. In der Programm-Aktivität wird die Übersetzungsvorschrift realisiert. Ein Programm wird automatisch ausgeführt, das den Inhalt der entsprechenden lokalen CWDL-Variablen innerhalb der IDL-Datentyp-Datenbank in die spezifizierte lokale Variable der folgenden CWDL-Aktivität kopiert.

> Wurden mehrere Daten-Konnektoren definiert, so wird der Kopiervorgang mehrfach ausgeführt.

Der *Or-Split* wird in FDL-Kontroll-Konnektoren und Programm-Aktivitäten übersetzt (Abbildung 3-12). Jeder CWDL-Kontroll-Konnektor wird, wie bereits beschrieben, in zwei FDL-Kontroll-Konnektoren aufgeteilt, die eine FDL-Programm-Aktivität einschließen.

Die Vereinigung derart entstehender Pfade erfolgt durch einen *Or-Join*. Dieser wird in FDL als Startbedingung von FDL-Aktivitäten und FDL-Blöcken angeboten.

> Bei der Modellierung eines FDL-Workflows kann in jeder FDL-Aktivität und jedem FDL-Block angegeben werden, daß sie zu starten sind, wenn mindestens einer oder alle eingehenden Kontroll-Konnektoren positiv evaluiert wurden. Damit können Or- und And-Joins modelliert werden. Wie bereits Abbildung 3-6 zeigte, wird der CWDL-Or-Join in den FDL-Or-Join übersetzt. Da ein FDL-Block bzw. eine FDL-Aktivität nur dann gestartet

werden kann, wenn *alle* eingehenden FDL-Kontroll-Konnektoren evaluiert wurden (wahr *oder* falsch), besitzen die in Abbildung 3-6 zusätzlich in den FDL-Block B endenden FDL-Kontroll-Konnektoren keinen Einfluß auf die Auswertung des Joins.

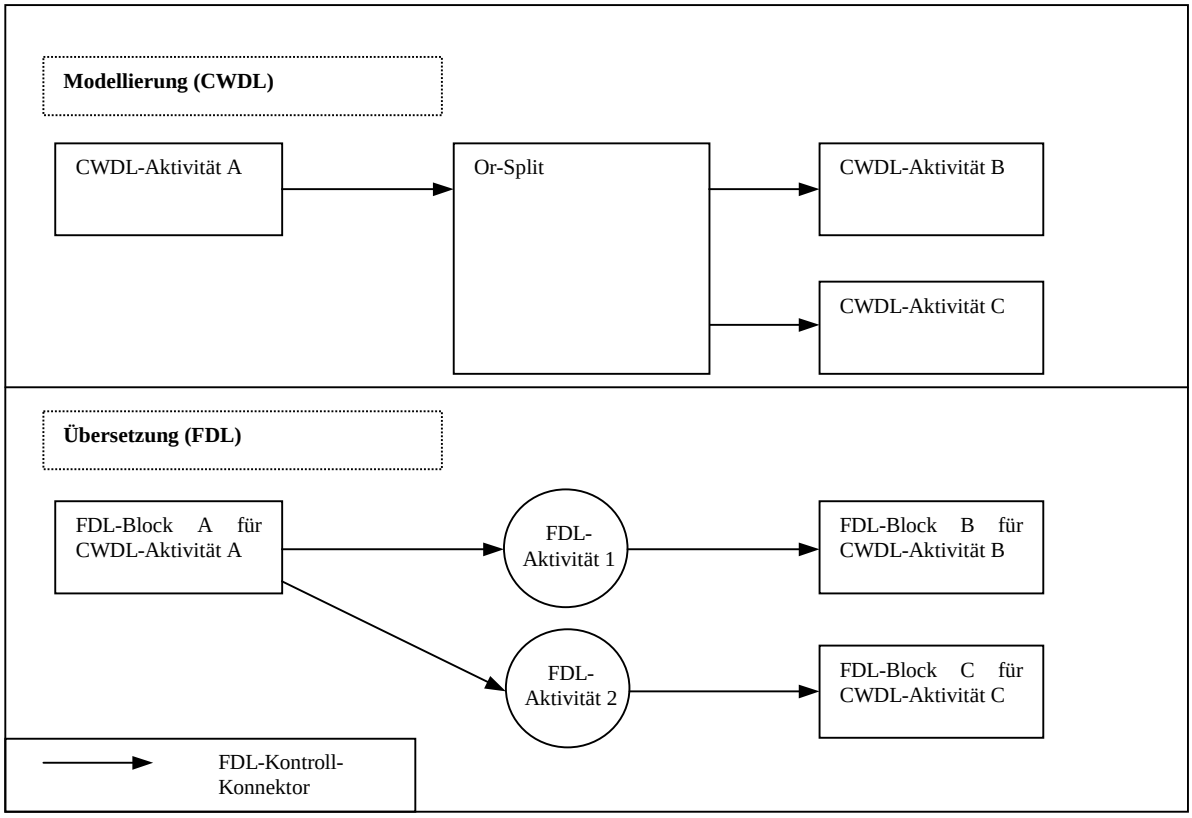


Abbildung 3-12: Die Übersetzung eines CWDL-Or-Splits.

Die Übersetzung eines *And-Splits* erfolgt ebenfalls durch FDL-Kontroll-Konnektoren, allerdings ohne zusätzliche FDL-Programm-Aktivität. Alle FDL-Kontroll-Konnektoren müssen für einen And-Split positiv evaluiert werden. In FDL besitzen sie deshalb keine Übergangsbedingung.

Folgen innerhalb eines CWDL-Workflows mehrere And-Splits aufeinander, so können diese in FDL zusammengefaßt werden.
Dafür werden für jeden And-Split weitere Kontroll-Konnektoren ohne Übergangsbedingung in den FDL-Workflow eingefügt.

Die Anzahl der zusätzlich einzufügenden FDL-Kontroll-Konnektoren richtet sich nach der Ausgangs-Arität des CWDL-And-Splits. Für eine Ausgangs-Arität von n werden n-1 zusätzlich Kontroll-Konnektoren in den FDL-Workflow eingefügt.

Ein *And-Join* wird ebenso wie der Or-Join durch die Startbedingung eines FDL-Blockes realisiert.

Da ein FDL-Block oder eine FDL-Aktivität nur dann gestartet werden kann, wenn *alle* eingehenden FDL-Kontroll-Konnektoren evaluiert wurden, kann der CWDL-And-Join ebenfalls in den FDL-Or-Join übersetzt werden.

Die Übersetzung eines CWDL-Workflows in einen FDL-Workflow erfolgt unter Verwendung eines Übersetzers und zusätzlicher Module. Diese dienen einerseits der Ablaufsteuerung eines Prozesses (Bedingungsauswertung). Andererseits können mit Hilfe dieser Module IDL-Datentypen und –Instanzen verwaltet werden. Die damit erfüllten technischen Forderungen werden um die Verwendung semantisch komplexer Modellierungskonstrukte (Refinement-Aktivität) ergänzt. Für deren Realisierung müssen weitere Module verwendet werden (Refinement-Manager; Ereignisgenerierung für den Startzeitpunkt; Nachrichtenübertragung).

Der Module werden im folgenden Abschnitt in einer Übersicht dargestellt und im Kapitel 4 näher beschrieben.

3.4 Runtime

Für die Ausführung eines Workflows, der in CWDL modelliert wurde, sind zusätzliche Module, aber auch der Übersetzer notwendig. Die Module sind in Form von CORBA-Objekten oder einfachen Programmen zu erstellen. Ferner muß eine IDL-Datentyp-Datenbank vorhanden sein.
Eine Übersicht über die zur Runtime verwendeten Module gibt Tabelle 3-13.

Übersetzer	Übersetzen eines CWDL-Workflows in einen FDL-Workflow (im Kombination mit dem Refinement-Manager), Kommunikation mit der IDL-Datentyp-Datenbank (Einträge für Variable), Kommunikation mit dem Interface Repository (Typ-Überprüfung von Methodenaufrufen), Kommunikation mit dem WfMS IBM FlowMark (Eindeutigkeit von erzeugten FDL-Definitionen), ausführbares Programm oder CORBA-Objekt
Wrapper	Ausführen von Methoden auf CORBA-Objekten, dabei lesender und schreibender Zugriff auf die IDL-Datentyp-Datenbank, ausführbares Programm
IDL-Datentyp-Datenbank	Verwaltung von Instanzen von IDL-Datentypen
Refinement-Manager	Generierung und Ausführung von Subprozessen, Kommunikation mit IDL-Datentyp-Datenbank, ausführbares Programm oder CORBA-Objekt
Modul zum Auswerten von Bedingungen	Wertet Bedingungen aus und gibt boolschen Wert an WfMS zurück, ausführbares Programm
Modul zum Einlesen aus der Datenbank	Dient zum Einlesen von Instanzen einfacher IDL-Datentypen aus der Datenbank, ausführbares Programm
Kapselung eines WfMS als CORBA-Objekt	Beeinflussen des Prozeß-Ablaufs in diesem WfMS durch externe Module, Ändern des Status von Aktivitäten, Starten von Subprozessen (asynchron für Call-Aktivität, synchron für Refinement-Manager), Kommunikation mit IDL-Datentyp-Datenbank, CORBA-Objekt
Statusabfrage und –änderung von Aktivitäten	Abfrage erfolgt direkt über API oder Methoden des CORBA-Objektes, das dasWfMS kapselt, Rückgabewert als Integer, ausführbares Programm
Generierung von Ereignissen	Zeitangabe nötig, Angabe des zu benachrichtigenden CORBA-Objektes, Methode und Parameter nötig. Generiert nach relativer Zeit ein Ereignis, Ereignis bewirkt Aufruf der Methode des spezifizierten Objektes Ereignis kann unter Verwendung des CORBA Time Service und des Event Service erzeugt werden, ausführbares Programm oder CORBA-Objekt
Instantierung von Prozeß- und Aktivitäts-Variablen	Generierung von Daten-Instanzen in der IDL-Datentyp-Datenbank für Prozeß- und Aktivitäts-Variable, ausführbares Programm
Löschen von Instanzen von Prozeß- und Aktivitäts-Variablen	Löschen von Daten-Instanzen in der IDL-Datentyp-Datenbank , ausführbares Programm

Tabelle 3-13: Zur Ausführung eines in CWDL modellierten Prozesses benötigte Module.

Von den aufgeführten Modulen werden in dieser Arbeit in Ergänzung zum Übersetzer die IDL-Datentyp-Datenbank, das Instanzieren und Löschen von Variablen in der IDL-Datentyp-Datenbank sowie die Statusabfrage und -änderung von Aktivitäten wurden implementiert.
Eine Beschreibung der implementierten Module folgt im Kapitel 4. Zusätzlich werden die Schnittstellen der noch zu implementierenden Module definiert.

4 Implementierung

4.1 Grundlagen

Um Prozesse, die in CWDL modelliert wurden, in eine ausführbare Form zu bringen, müssen diese in die Workflow-Modellierungssprache eines konkreten WfMS übersetzt werden. In der vorliegenden Arbeit wird IBM FlowMark verwendet. Die Workflow-Modellierungssprache dieses Systems ist FDL.
In Abhängigkeit von den Eigenschaften des verwendeten WfMS müssen zusätzliche Module geschaffen werden, um die von CWDL angebotenen Modellierungsmöglichkeiten mit dem WfMS realisieren zu können. Die für IBM FlowMark zu realisierenden Module wurden bereits in den Abschnitten 3.3.3 und 3.4 genannt.

4.2 Der Übersetzer

Wichtigstes Buildtime-Modul für eine Umsetzung von CWDL-Definitionen in FDL ist der *Übersetzer*. Der in dieser Arbeit erstellte Übersetzer ist ein *Compiler*.

> Gemäß [LJO87] bedeutet Compilation
>
> „...allgemein Programmübersetzung; das gesamte Quellprogramm wird vor seiner Abarbeitung in eine äquivalente Maschinensprachversion übersetzt“.
>
> Demgegenüber sind
>
> „...bei der Interpretation [...] Programmübersetzung und –abarbeitung unmittelbar miteinander verbunden: entsprechend der dynamischen Programmabarbeitung wird ein zusammenhängendes Quelltextstück (z.B. eine Anweisung) übersetzt und sofort ausgeführt. Ein geschlossenes Zielprogramm wird nicht erzeugt.“

Die Wahl, einen Compiler für die Übersetzung zu verwenden, wurde aus technischen und benutzerergonomischen Gründen getroffen.

> Im Gegensatz zum *Interpreter* erzeugt der Compiler ein *geschlossenes Zielprogramm*. Damit ist es möglich, fehlerhafte CWDL-Definitionen bereits vor ihrer Abarbeitung zu erkennen. Fehler können bereits zur Buildtime vom Prozeß-Modellierer behoben werden und treten nicht erst bei der Workflow-Ausführung auf. Das bedeutet, daß die Zuverlässigkeit eines Workflows erhöht und gleichzeitig das medizinische Personal nicht mit Ausführungsfehlern, die bereits zur Buildtime feststellbar sind, konfrontiert wird.
> Ein weiterer Grund für die Wahl eines Compilers als Übersetzer ist das Ziel, ein bereits vorhandenes kommerzielles WfMS (IBM FlowMark) als *Workflow Engine* einzusetzen. So

können alle Funktionen des WfMS genutzt werden. Im Gegensatz dazu werden durch die Wahl eines Interpreters wesentliche Elemente eines WfMS redundant im Interpreter implementiert (beispielsweise dynamische Rollenauflösung zur Laufzeit; Ablaufsteuerung des Workflows zusätzlich im Interpreter).

Die Aufgaben eines Compilers zeigt Abbildung 4-1.

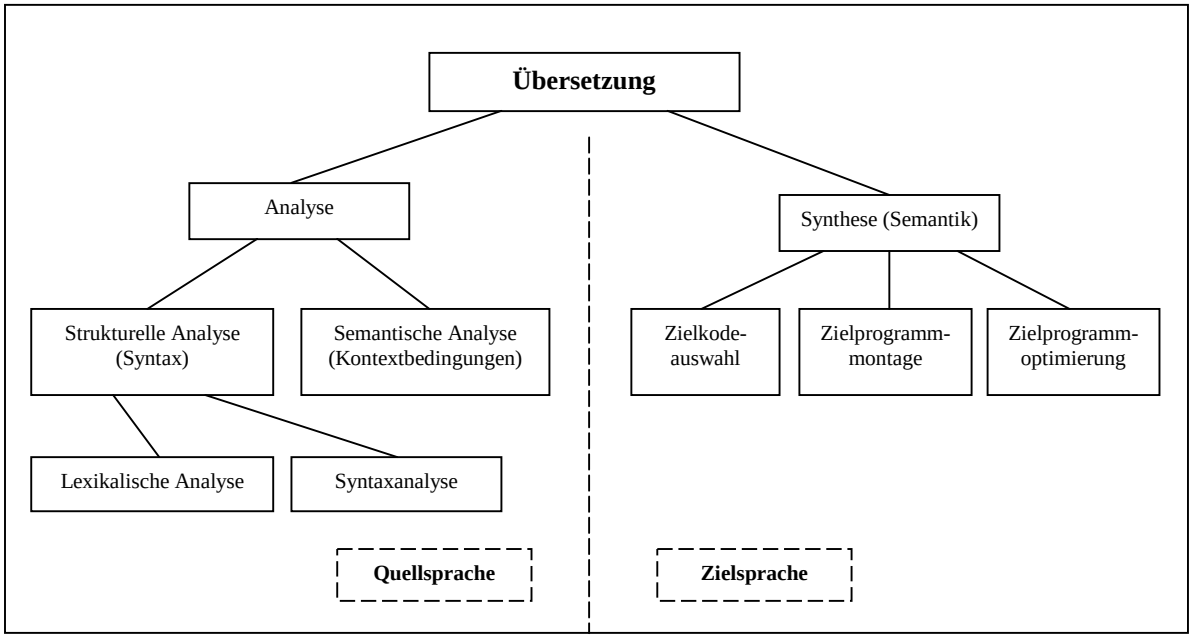


Abbildung 4-1: Teilaufgaben bei der Programmübersetzung. Nach [LJO87].

Die logische Struktur eines Compilers zeigt Abbildung 4-2.

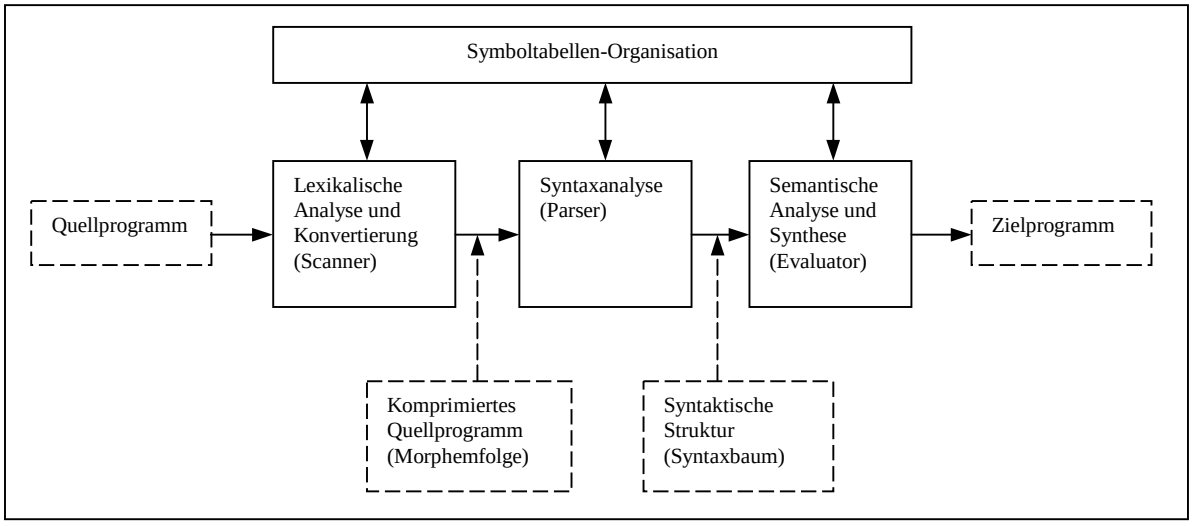


Abbildung 4-2: Logische Struktur eines Compilers. Nach [LJO87].

Wie in [LJO87] beschrieben, wird die logische Struktur eines Compilers nicht immer in seiner physischen Struktur wiedergegeben:

> „...Das Zielprogramm ist [...] das Ergebnis eines oder mehrerer Transformationsschritte. Man spricht von einer Einpaßübersetzung (Einpaßcompiler), wenn die Programmtransformation einstufig ist. Sonst spricht man von einer Mehrpaßübersetzung (Mehrpaßcompiler).“

Der hier erstellte Compiler ist ein *Mehrpaßcompiler*. Seine einzelnen *Transformationsschritte* (*Pässe*) sind:

(1) die *Syntaxanalyse*, verbunden mit der von ihr gesteuerten *lexikalischen Analyse*,
(2) eine *semantische Analyse* (beispielsweise Test auf korrekte Variablennamen),
(3) die *Synthese des Zielprogramms* und
(4) *Ausgabe* des Zielprogramms.

Die lexikalische Analyse erfolgt durch einen Scanner, der durch den Scanner-Generator *lex* unter Eingabe der lexikalischen Grammatik von CWDL (*Token*) erzeugt wurde.
Die Syntaxanalyse wird mit einem Parser, der durch den Parser-Generator *yacc* unter Verwendung der Grammatik von CWDL generiert wurde, vorgenommen. Dabei wird der Scanner vom Parser in Form eines Unterprogrammes gesteuert.

Die Grammatik, die die Sprache CWDL erzeugt, kann durch einen *LALR-Parser*[33] [34] syntaktisch analysiert werden.

Ergebnis der Analyse ist ein *Syntaxbaum*.

Für jede Regel der Quell-Grammatik CWDL wurde für den Übersetzer eine Objekt-Klasse (*Regel-Klasse*) implementiert.
Bei positiver Evaluierung einer Regel wird eine Instanz der zugehörigen Regel-Klasse erzeugt und in den Syntaxbaum, d.h. in die Objekt-Instanz der Regel, die zur aktuellen Regel führte, eingefügt.

Ein Syntaxbaum, der vom Parser erzeugt wurde, besteht aus Instanzen von Regel-Klassen .

Mit Hilfe des Syntax-Baumes kann der erste Teil der semantischen Analyse erfolgen.

Im ersten Teil der semantischen Analyse werden Bedingungen überprüft, die *lokal* für einzelne Regeln gelten müssen. Das betrifft vor allem Namen oder Werte von Variablen.

[33] LALR: Lookahead-Left-to-Right.
[34] Die Vorschau um ein Token reicht aus, um dem Parser eine eindeutige Auswahl der Analyse-Aktion zu ermöglichen.

Beispielsweise werden Adressen, die im TCP/IP-Format in CWDL angegeben sind, auf ihre *syntaktische* („$Zahl_1.Zahl_2.Zahl_3.Zahl_4$“) und ihre *semantische* ($0 \leq Zahl_x \leq 255$) *Korrektheit* überprüft. Ferner werden IDL-Datentypen, die als Typ für Variable angegeben wurden, ebenso wie Methoden-Namen und -Parameter auf ihre Korrektheit unter Verwendung des *CORBA Interface Repository* evaluiert.
Unter Verwendung der IBM FlowMark-Datenbank können alle FlowMark-Nutzer-, Rollen-, Level-, Organisations-, Datenstruktur-, logische Programm- und Prozeßnamen eingelesen und auf ihre Eindeutigkeit überprüft werden. Diese Information ist auch dann wichtig, wenn vom Compiler Datenstrukturen und andere FDL-Konstrukte generiert werden müssen, denen ein eindeutiger Name gegeben werden muß.

An den ersten Teil der semantischen Analyse anschließend wird deren zweiter Teil durchgeführt.

Im zweiten Teil der semantischen Analyse wird die *globale* Korrektheit evaluiert. Die globale Korrektheit betrifft nicht nur einen Knoten, sondern mehrere Knoten bzw. ganze Teilbäume des Syntaxbaumes.
Sie bezieht sich beispielsweise auf die Definition und Verwendung von Variablen oder von logischen Programmen.

Die semantische Analyse wird rekursiv ausgeführt, d.h. das Objekt, das die (einzige) Regel mit dem Startsymbol der CWDL-Grammatik repräsentiert, führt die semantische Analyse über seinen Daten (einschließlich seiner Teilbäume) durch. Bei einem positiven Ergebnis wird für alle Objekte, die Regeln repräsentieren, die in einem Schritt vom Startsymbol erreicht werden können, die entsprechende Methode aufgerufen.

Unter anderem bei der semantischen Analyse kann der objekt-orientierte Aspekt der Vererbung eingesetzt werden. Alle Klassen, die Regeln der CWDL-Grammatik repräsentieren, können Eigenschaften einer gemeinsamen Klasse erben. Diese gemeinsame Klasse (*CommonCWDLObject*) besitzt Eigenschaften, die auch jede andere Klasse besitzen muß.
Beispielsweise enthält sie eine Implementierung der Methode für den ersten Teil der semantischen Analyse, die lediglich rekursiv für alle enthaltenen Objekte die Methode ausführt. Bei Bedarf wird die Implementierung dieser Methode in einzelnen Klassen durch eine speziellere ersetzt.

Nach positiver semantischer Analyse erfolgt die Generierung des Zielprogramms (FDL-Definitionen).

Wie für die Sprache CWDL existiert auch bei FDL für jede einzelne Regel eine Objekt-Klasse, die diese Regel repräsentiert. Diese Klassen besitzen ebenfalls gemeinsame Eigenschaften, die sie von der Klasse *CommonFDLObject* erben.

Die Synthese des Zielprogramms ergibt einen Syntaxbaum der Workflow-Modellierungssprache FDL.

Besitzt ein CWDL-Konstrukt eine Entsprechung in FDL, so kann ein ganzer CWDL-Teilbaum in einen FDL-Teilbaum übersetzt werden. Die Übersetzung erfolgt dabei wieder rekursiv, d.h. jeder Knoten des CWDL-Teilbaums übersetzt seine eigene Information und initiiert dabei schrittweise die Übersetzung der Teilbäume, deren Wurzel er ist.
Eine solche Übersetzung erfolgt bei Definitionen von Personen, Rollen, Organisationen und Levels.

Besitzt ein CWDL-Konstrukt keine Entsprechung in FDL, so wird es durch die Übersetzung in FDL konstruiert. Diese Konstruktion erfolgt für alle Teilbäume, deren Wurzel eine CWDL-Prozeß-Definition ist. Das ist notwendig, da bereits die *Variablen* der CWDL-Prozesse keine Entsprechung in FDL besitzen. Die Definition einer lokalen Variablen in einem CWDL-Prozeß wird daher in FDL-Programm-Aktivitäten übersetzt. Aufgabe dieser Programm-Aktivitäten ist es, eine im Input-Container angegebene Variable eines ebenfalls spezifizierten Typs in der IDL-Datentyp-Datenbank zu instanzieren.
Für die Definition der Programm-Aktivitäten müssen daher weitere Konstrukte generiert werden. Das in den Programm-Aktivitäten spezifizierte logische FDL-Programm ist im Rahmen einer FDL-Definition zu generieren. Diese Definition kann automatisch erfolgen. Die Information über die Eigenschaften des physischen Programms kann dem Compiler als Parameter, etwa als Datei, mitgeteilt werden. Angaben über die Ablaufform des Programms (z.B. für den Nutzer unsichtbar; im Hintergrund) werden automatisch, ohne weiteren Informationsbedarf, erzeugt.
Um das Programm mit Daten versorgen zu können, werden zusätzlich FDL-Datenstrukturen generiert. Diese Strukturen dienen der Übergabe von CWDL-Variablennamen , -typ und –wert an logische Programme. Ferner wird in ihnen der Name des CWDL-Prozesses und der CWDL-Aktivität eingetragen.

Die Erzeugung des logischen FDL-Programms und der FDL-Datenstrukturen erfolgt nur dann, wenn diese nicht bereits bei der Übersetzung eines anderen CWDL-Prozesses generiert wurden.

Schließlich werden FDL-Daten- und –Kontroll-Konnektoren erzeugt, die festlegen, daß die Initialisierung (bzw. das Löschen der Daten-Instanzen aus der IDL-Datentyp-Datenbank; die entsprechenden Definitionen werden ebenso generiert) vor (oder nach) allen anderen Aktivitäten des FDL-Prozesses oder allen Teilkonstrukten der übersetzten CWDL-Aktivität ausgeführt wird.
Müssen mehrere Variable eines CWDL-Prozesses oder einer CWDL-Aktivität instanziert oder gelöscht werden, so wird für jede Variable eine FDL-Programm-Aktivität erzeugt.
Zusammengehörige CWDL-Variablen-Definitionen (eines CWDL-Prozesses oder einer CWDL-Aktivität) werden innerhalb eines FDL-Blockes zusammengefaßt.

Die Übersetzung aller CWDL-Aktivitäten in FDL-Aktivitäten erfolgt auf ähnliche konstruktive Weise durch das Objekt, das die Wurzel einer CWDL-Prozeß-Definition repräsentiert. Existieren innerhalb einer CWDL-Prozeß-Definition jedoch Teilbäume, die eine Entsprechung in FDL besitzen (beispielsweise die Definition von Nutzerberechtigungen), so werden diese rekursiv, wie oben beschrieben, übersetzt.

CWDL-Refinement-Manager-, logische CWDL-Programm- und CWDL-WfMS-Definitionen werden nicht direkt in FDL-Definitionen übersetzt. Wird ein solches CWDL-Konstrukt in einer CWDL-Aktivität referenziert, so wird dessen Definition als „Skelett“ für eine FDL-Programm-Aktivität und –Definition verwendet. Unter zusätzlicher Verwendung der in der CWDL-Aktivität angegebenen Information (z.B. Übergabeparameter an ein CWDL-Programm) werden schließlich eine FDL-Programm-Definition sowie ggf. Datenstruktur-Definitionen erzeugt.

Als letzter Schritt erfolgt die Ausgabe des Zielprogrammes, d.h. des FDL-Syntaxbaumes.

Die Ausgabe erfolgt rekursiv. Dazu besitzt die Klasse *CommonFDLObject* eine Methode, die rekursiv für jedes enthaltene Wurzel-Objekt eines Teilbaumes die Ausgabe initiiert. Klassen, die Regeln der FDL-Grammatik repräsentieren, die keine terminalen Symbole enthalten, können zur Ausgabe die entsprechende Methode von *CommonFDLObjekt* verwenden. Um jedoch auch eine Formatierung der FDL-Ausgabe erreichen zu können, wird die Methode in den meisten FDL-Klassen durch eine eigene Implementierung überschrieben.

Die Ausgabe der FDL-Definitionen erfolgt in eine Datei, deren File-Handle der Methode als Parameter übergeben wurde.

Ergebnis der Übersetzung von CWDL nach FDL ist eine Text-Datei, die FDL-Definitionen enthält.

Die Definitionen können mittels des IBM FlowMark-*Buildtime-Clients* oder des *Import-Tools* (*exmpfimp* auf WindowsNT) in die FlowMark-interne Datenbank eingeladen und in eine (FlowMark-interne) ablauffähige Form übersetzt werden.

Damit kann der in CWDL modellierte Workflow ausgeführt werden.

Die Übersetzung kann optimiert werden.

Beispielsweise kann durch Kommandozeilen-Parameter dem Übersetzer mitgeteilt werden, daß ein Semantik-Test nicht stattfinden soll. Das bedeutet, daß sowohl mit dem CORBA Interface Repository als auch mit IBM FlowMark während der Übersetzung keine Kommunikation erfolgt.
Diese Optimierung kann allerdings nur dann angewandt werden, wenn generierte FDL-Definitionen, die in ihrem vom Compiler erzeugten Namen synonym zu bereits existierenden FDL-Definitionen sind, aufgrund ihrer gleichen Bedeutung beim Import in die FlowMark-Datenbank überschrieben werden können oder Prozesse mit gleichem Namen überschrieben werden sollen.
Für die Eindeutigkeits-Überprüfung von Prozeßnamen kann ein eigener Parameter angegeben werden.

Um die den CWDL-Definitionen entsprechenden FDL-Definitionen wieder aus der FlowMark-Datenbank entfernen zu können, ist zusätzlich eine Liste zu führen, die die Zuordnung der CWDL-Definitionen zu FDL-Definitionen enthält.

Das kann innerhalb der IDL-Datentyp-Datenbank (mit Schema-Erweiterung) oder in einer anderen Form erfolgen.

4.3 Die Datenbank für IDL-Datentypen

In diesem Abschnitt wird die Implementierung der Datenbank für IDL-Datentypen beschrieben. Diese Datenbank wurde bereits konzeptionell in Form von Entity-Relationship-Diagrammen beschrieben. Das DBMS, in dem diese Datenbank implementiert werden soll, ist das System DB2 Version 5.0 von IBM.
Die Datenbank wird deshalb in eine relationale Struktur übertragen.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name des Datentyps
type	varchar(8)	not null, type ∈ {simple, sequence, array, struct, object, enum, synonym, any, union}	Angabe der Art des Datentyps

Tabelle 4-1: Die Repräsentation des Entities *datatype* in der Relation *data_type.*

Ein IDL-Datentyp besitzt einen Namen, der eindeutig im gesamten CORBA-Netzwerk ist. Im Gegensatz zur Verwendung des Typ-Namens in CORBA soll ein Typ-Name der Relation *data_type* nicht nur einfache Typen, sondern auch Typen innerhalb von *Modulen* (*Namensräume*) bezeichnen. Der Typ-Name setzt sich daher aus einzelnen Modul-Namen zusammen, die durch „::“ getrennt werden. Letztes Element eines solchen zusammengesetzten Typ-Namens ist ein einfacher Typ-Name.

> Die Länge eines Typ-Namens (und der meisten anderen Attribute) wird in der Datenbank auf 50 Zeichen begrenzt. Grund dafür ist, daß der Typ-Name als Primärschlüssel verwendet wird. Das DBMS DB2 besitzt eine Restriktion bezüglich der Primärschlüssel-Länge. Sollen längere Typ-Namen verwendet werden, so kann in den entsprechenden Relationen der Primärschlüssel durch ein neues Attribut (künstlich generierter Primärschlüssel; siehe auch Relation *datatype_instance*) ersetzt werden. Der Typ-Name ist dann als „normales“ Attribut nicht mehr Bestandteil des Primärschlüssels.

Das Attribut *type* gibt die Art des Datentyps an, d.h. ob ein Datentyp einfach ist („*simple*“; alle Basistypen), eine Aufzählung ist („*enum*“), aus anderen Datentypen konstruiert ist („*sequence*“, „*array*“, „*struct*“), seine Instanzen Instanzen anderer Datentypen enthalten können („*any*“, „*union*“) oder ob er ein Objekt-Typ ist („*object*“). Ferner kann ein Typ-Name lediglich ein Synonym für einen anderen Typ sein („*synonym*“).

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
enum_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Datentyps
member_name	varchar(50)	not null, Teil des Primärschlüssels	Name des Aufzählungselementes
member_number	smallint	not null	Nummer des Aufzählungselementes

Tabelle 4-2: Die Repräsentation des Entities *identifier_of_enumeration* und des Relationships *identifier* in der Relation *identifier_of_enum.*

Neben der einfachen *Deklaration* eines Datentyps, die durch die Relation *data_type* ausgedrückt wird, muß ein Datentyp auch *definiert* werden. Die für seine Verwaltung in der IDL-Datentyp-Datenbank relevanten Aspekte, wie sein konstruktiver Aufbau bzw. seine besonderen Eigenschaften, sind in der Datenbank zu modellieren. Dazu dienen die Relationen *identifier_of_enum, aggregated_type, set_of_type, synonym, union_type, aggregated_inst, set_of_instance, any_instance, union_instance* sowie die Relationen zur Modellierung von Instanzen der IDL-Basistypen.

Durch die Relation *identifier_of_enum* können Elemente von Aufzählungstypen (*member_name*) sowie deren Ordnung (*member_number*) definiert werden.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des aggregierten Typs (z.B. Struct)
contd_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs eines aggregierten Typs (z.B. Struct)
member_name	smallint	not null, Teil des Primärschlüssels	Element-Name des aggregierten Typs (z.B. Struct)

Tabelle 4-3: Die Repräsentation des Relationships *aggregated_type* in der Relation *aggregated_type.*

Die Relation *aggregated_type* dient der Definition von Aggregations-Typen (*contng_type_name*), d.h. von Structs. Diese bestehen aus Element-Typen (*contd_type_name*), die durch einen Bezeichner (*member_name*) im Aggregations-Typ eindeutig identifiziert werden.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Primärschlüssel, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Mengentyps (Array, Sequence)
contd_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs des Mengentyps (Array, Sequence)
member_count	smallint	not null	Größe des Mengentyps (Array, Sequence)

Tabelle 4-4: Die Repräsentation des Relationships *set_of_type* in der Relation *set_of_type.*

Unter Verwendung der Relation *set_of_type* werden Typen definiert (*contng_type_name*), die Elemente gleichen Typs (*contd_type_name*) in einer bestimmtem Anzahl (*member_count*) enthalten (Arrays, Sequenzen).

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
syn_type_name	varchar(50)	not null, Primärschlüssel, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Synonym eines bestehenden Typ-Namens
orig_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des bestehenden Typs

Tabelle 4-5: Die Repräsentation des Relationships *synonym* in der Relation *synonym.*

Synonyme Typ-Namen, die etwa durch eine *typdef*-Deklaration entstehen, werden durch die Relation *synonym* repräsentiert. Ein bestehender Typ-Name (*orig_type_name*) kann dabei beliebig viele Synonyme (*syn_type_name*) besitzen.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Union-Typs
contd_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs
discriminator	varchar(50)	not null	Diskriminator des Union-Typs

Tabelle 4-6: Die Repräsentation des Relationships *union_type* in der Relation *union_type.*

Ein Union-Typ, der durch die Relation *union_type* ausgedrückt wird, wird mit dem Namen des Union-Typs (*contng_type_name*), dem Namen des von ihm repräsentierten Typs (*contd_type_name*) sowie durch einen Diskriminator (*discriminator*) definiert.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
key	smallint	not null, Primärschlüssel	für die Identifizierung einer Dateninstanz, bei der Instantierung von der Applikation erzeugt
contng_inst_key	smallint	Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	für die Identifizierung von Komponenten einer Dateninstanz,

Tabelle 4-7: Die Repräsentation des Entities *datatype_instance* in der Relation *data_instance.*

Instanzen von Daten-Typen, die in der Relation *data_type* deklariert wurden, werden in der Relation *data_instance* verwaltet. Ist ein instanzierter Typ ein zusammengesetzter Typ, d.h. kein IDL-Basistyp und kein Objekt-Typ, so werden für alle Komponenten Instanzen des Komponenten-Typs erzeugt. Alle Instanzen besitzen einen eindeutigen Schlüssel (*key*), der von der instanzierenden Applikation erzeugt wird. Komponenten-Instanzen werden mit der Instanz, deren Komponente sie sind, durch das Attribut *contng_inst_key* verknüpft. Neben dieser rein technischen Verknüpfung werden semantische Aspekte des *Enthaltenseins* durch die Relationen *aggregated_instance*, *set_of_instance*, *any_instance* und *union_instance* ausgedrückt.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Bezeichnet Instanz des Element-Typs eines aggregierten Typs (z.B. Struct)
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz des Element-Typs eines aggregierten Typs (z.B. Struct)
member_name	varchar(50)	not null, Teil des Primärschlüssels	Name des Elementes eines aggregierten Typs (z.B. Struct)

Tabelle 4-8: Die Repräsentation des Relationships *aggregated_instance* in der Relation *aggregated_inst.*

Die Relation *aggregated_instance* stellt eine Verknüpfung von Instanzen aggregierter Typen (*contng_inst_key*) mit Instanzen ihrer Komponenten (*contd_inst_key*) dar.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Bezeichnet Instanz des Mengentyps (Array, Sequence)
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz eines Elementes des Mengentyps (Array, Sequence)
member_number	smallint	not null, Teil des Primärschlüssels	Nummer des Elementes des Mengentyps (Array, Sequence)

Tabelle 4-9: Die Repräsentation des Relationships *set_of_instance* in der Relation *set_of_instance.*

Durch die Relation *set_of_instance* werden Instanzen von Mengen-Typen repräsentiert. Das erfolgt durch die Angabe der Instanz des Mengentyps (*contng_inst_key*), der Instanz des Komponenten-Typs (*contd_inst_key*) sowie des Indexes (*member_number*) der Komponente.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet any-Instanz
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz des im any enthaltenen Typs

Tabelle 4-10: Die Repräsentation des Relationships *any_instance* in der Relation *any_instance.*

Die Beziehung von any-Instanzen zu ihren enthaltenen Instanzen eines beliebigen Datentyps enthält die Relation *any_instance*. Eine durch *contng_inst_key* bezeichnete Instanz enthält eine Instanz, die durch *contd_inst_key* benannt wird.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Union-Instanz
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz des durch die Union dargestellten Typs

Tabelle 4-11: Die Repräsentation des Relationships *union_instance* in der Relation *union_instance.*

Die Beziehung von Union-Instanzen zu den von ihnen repräsentierten Instanzen eines Datentyps enthält die Relation *union_instance*. Eine durch *contng_inst_key* bezeichnete Instanz stellt eine Instanz, die durch *contd_inst_key* benannt wird, dar.

Um Variable von Datentypen einer CWDL-Aktivität eines CWDL-Workflows zuordnen zu können, müssen zusätzlich Informationen über die Struktur von CWDL-Workflows in der IDL-Datentyp-Datenbank enthalten sein.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name des CWDL-Workflows

Tabelle 4-12: Die Repräsentation des Entities *workflow_template* in der Relation *workflow_template.*

Ein CWDL-Workflow wird durch die Relation *workflow_template* unter Angabe seines Namens (*name*) definiert.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name der CWDL-Aktivität

Tabelle 4-13: Die Repräsentation des Entities *workflow_activity* in der Relation *workflow_activity*.

CWDL-Aktivitätsnamen (*name*) werden in die Relation *workflow_activity* eingetragen.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
pred_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der Quell-CWDL-Aktivität eines CWDL-Daten-Konnektors
succ_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der Ziel-CWDL-Aktivität eines CWDL-Daten-Konnektors
template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows, der beide Aktivitäten enthält

Tabelle 4-14: Die Repräsentation des Relationships *successor* in der Relation *successor*.

Durch die Relation *successor* werden CWDL-Daten-Konnektoren modelliert. Diese Information dient der Darstellung der Lebensdauer einer Variablen bzw. der Zugriffsberechtigung von Aktivitäten auf die Variablen.

Attribut	Wertebereich (Datentyp des DBMS)	Integritätsbedingung	Bemerkung
proc_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Prozeß-Aktivität
template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des Prozesses mit der CWDL-Prozeß-Aktivität
sub_template_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Subprozesses

Tabelle 4-15: Die Repräsentation des Relationships *subprocess_template* in der Relation *subprocess_template.*

Die globale Gültigkeit von lokalen Variablen einer Prozeß-Aktivität (*proc_activity_name*) in ihrem Subprozeß *sub_template_name* wird in der Tabelle *subprocess_template* wiedergegeben. Die Aktivität gehört zum Prozeß *template_name*.

Attribut	Wertebereich (Datentyp des DBMS)	Integritätsbedingung	Bemerkung
activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Aktivität, in deren CWDL-Definition eine Variable des durch *data_name* bezeichneten Datentyps vereinbart wurde
workflow_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows, in dessen CWDL-Definition (bzw. in der CWDL-Definition einer seiner Aktivitäten) eine Variable des durch *data_name* bezeichneten Datentyps vereinbart wurde
variable_name	varchar(50)	not null, Teil des Primärschlüssels	Name der Variable der CWDL-Aktivität oder des CWDL-Workflows
data_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Datentyps der Variablen
mode	smallint	not null, mode $\in$ {0,1}	Zugriffsmodus für Variable 0 = READ_WRITE 1 = READ_ONLY

Tabelle 4-16: Die Repräsentation des Relationships *data_in_workflow* in der Relation *data_in_workflow.*

Die Zuordnung von Variablen zu CWDL-Aktivitäten und CWDL-Workflows erfolgt in der Relation *data_in_workflow*. Eine Variable (*variable_name*) des Typs *data_name* wird einem

Workflow (*workflow_name*) und einer Aktivität (*acitivity_name*) zugeordnet. Dabei gilt zur Vereinfachung von Anfragen an die Datenbank die Konvention, daß eine Variable, die in einem CWDL-Workflow global definiert wurde, durch einen Leerstring im Feld *activity_name* ausgedrückt wird. Das Attribut *mode* enthält die Information über die Zugriffsart für eine Variable.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name der IBM FlowMark-Workflow-Instanz
template_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows

Tabelle 4-17: Die Repräsentation des Entities *instance_of_template* und des Relationships *has_instance* in der Relation *inst_of_template.*

Die Instantierung eines Workflows ergibt immer eine FlowMark-Workflow-Instanz. In der Relation *inst_of_template* wird eine FlowMark-Workflow-Instanz (*name*) einem CWDL-Workflow (*template_name*) zugeordnet.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Aktivität
inst_template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der IBM FlowMark-Workflow-Instanz
variable_name	varchar(50)	not null, Teil des Primärschlüssels	Name der Variablen der CWDL-Aktivität oder des CWDL-Workflows
data_instance_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
mode	smallint	not null, mode $\in$ {0,1}	Zugriffsmodus für Variable 0 = READ_WRITE 1 = READ_ONLY
init	smallint	not null, init $\in$ {0,1}	Initialisierungsstatus für Variable 0 = noch nicht initialisiert 1 = initialisiert

Tabelle 4-18: Die Repräsentation des Relationships *data_in_instance* in der Relation *data_in_instance.*

Um die Verbindung von Daten-Instanzen, FlowMark-Workflow-Instanzen und CWDL-Aktivitäten herzustellen, werden Einträge in der Relation *data_in_instance* vorgenommen. Einer möglicherweise als Leerstring (siehe *data_in_workflow*) angegebenen Aktivität (*activity_name*) und einer FlowMark-Workflow-Instanz (*inst_template_name*) wird eine Daten-Instanz (*data_instance_key*) unter Angabe eines Variablennamens (*variable_name*) zugeordnet.
Das Attribut *mode* dient wiederum der Spezifikation der Zugriffsart einer Variable. Wurde eine Variable als *READ_ONLY* definiert, so kann sie nur einmal initialisiert werden. Die Information über die Initialisierung enthält das Attribut *init*.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
proc_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Prozeß-Aktivität
instance_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der Prozess-Instanz mit der CWDL-Prozeß-Aktivität
sub_instance_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der Subprozeß-Instanz

Tabelle 4-19: Die Repräsentation des Relationships *subprocess_instance* in der Relation *subprocess_instance.*

Die globale Gültigkeit von lokalen Variablen einer Prozeß-Aktivität (*proc_activity_name*) in ihrer Subprozeß-Instanz *sub_instance_name* wird in der Tabelle *subprocess_instance* wiedergegeben. Die Aktivität gehört zur Prozeß-Instanz *instance_name*.

Die Werte eines IDL-Datentyps werden unabhängig von seiner Art (einfach, konstruiert) immer durch Werte einfacher Typen oder Objekt-Referenzen ausgedrückt. Deshalb erfolgt für jede Daten-Instanz der Relation *data_instance*, die einen einfachen Daten-Typ repräsentiert, ein Eintrag in genau einer der folgenden Relationen. In diesen ist im Attribut *value* der tatsächliche Wert der Daten-Instanz enthalten. Der Typ des Attributes wurde jeweils so festgelegt, daß der Wert mit geringstem Typ-Konversions- und Werte-Bereichs-Überprüfungs-Aufwand in die entsprechende Relation übernommen werden kann. Für die strukturell gleichen Relationen wird an dieser Stelle lediglich die Relation *short_instance* aufgeführt. Alle anderen Relationen sind Bestandteil des Anhangs.

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	smallint	not null	Wert der Daten-Instanz

Tabelle 4-20: Die Repräsentation des Entities *short_instance* und des Relationships *is_short* in der Relation *short_instance.*

Die Einträge in die Relationen der IDL-Datentyp-Datenbank erfolgen:

- durch den Übersetzer (Relationen *workflow_template*, *workflow_activity*, *data_in_workflow*, *successor* und *subprocess_template*),
- durch ein Modul, das den Namen einer FlowMark-Workflow-Instanz in die Datenbank einträgt (Relation *inst_of_template* und *subprocess_instance*),
- durch ein Programm, das am Anfang der Bearbeitung jeder FlowMark-Workflow-Instanz die entsprechenden Variablen instanziert bzw. nach Ende der Lebensdauer einer Variablen die Instanzen löscht (Relationen *data_instance, aggregated_instance*, *set_of_instance*, *any_instance*, *union_instance*, *data_in_instance* sowie die Relationen, in denen die tatsächlichen Werte der einfachen Datentypen oder der Objekt-Typen eingetragen werden) und
- durch Programme, die mit den Variablen arbeiten (Relationen *set_of_instance* für Sequenzen, *any_instance*, *union_instance* sowie alle die Relationen, in denen die tatsächlichen Werte der einfachen Datentypen oder der Objekt-Typen eingetragen werden).

Diese Programme werden im letzten Abschnitt des Kapitels skizziert.

4.4 Weitere Module

Für die CORBA-Integration müssen neben dem Übersetzer und der Datenbank für IDL-Datentypen weitere Module geschaffen werden. Diese werden kurz skizziert.

Die Zuordnung einer FDL-Prozeß-Instanz zu einem CWDL-Workflow in der Datenbank erfolgt durch ein einfaches ausführbares Programm, das als Parameter lediglich den Namen des CWDL-Workflows sowie, aus seinem *Input-Container*, den Namen der FlowMark-Prozeß-Instanz erhält.

Nach Beendigung des Workflows wird der entsprechende Datenbank-Eintrag durch den Aufruf eines weiteren Programms gelöscht.

Um Variable, die in CWDL-Aktivitäten und –Workflows definiert wurden, verwenden zu können, müssen diese vor ihrer Verwendung in der IDL-Datentyp-Datenbank instanziert bzw. nach Ablauf ihrer Lebensdauer auch wieder gelöscht werden.

> Dafür existieren zwei ausführbare Programme, deren Aufruf bei der CWDL-FDL-Übersetzung als FDL-Programm-Aktivität für jede CWDL-Aktivität und jeden CWDL-Prozeß generiert wird. Diese Programme erhalten als Übergabe-Parameter den Namen der Prozeß-Instanz, ggf. den Namen der CWDL-Aktivität und den Namen der Variablen. Unter Verwendung der bereits vom Übersetzer in der Datenbank abgelegten und der durch die Zuordnung der Prozeß-Instanz zu ihrem Template generierten Information können die einzelnen Variablen instanziert oder gelöscht werden.

Aufgabe des *Wrappers* ist die Ausführung von CORBA-Methoden.

> Der *Wrapper* wurde als ausführbares Programm konzipiert und kommuniziert mit IBM FlowMark mittels der *FlowMark-API* über die *Daten-Container*. Zusätzlich erhält er *Kommandozeilen-Parameter*. Die Daten, die dem Wrapper übergeben werden, enthalten die Information über das CORBA-Objekt, dessen Methode ausgeführt werden soll (*CORBA-Objektreferenz* als String), über die Methode, die auszuführen ist (*Methodenname*) sowie über die Parameter (*Variablennamen* und *Konstanten*), die der Methode übergeben werden sollen. Zusätzlich wird dem Wrapper der *Name der Prozeß-Instanz* des Workflows sowie der *Name der CWDL-Aktivität*, in deren Kontext der Methodenaufruf stattfindet, übergeben.
>
> Unter Verwendung der Information über die Prozeß-Instanz und die CWDL-Aktivität und der Variablennamen rekonstruiert der Wrapper die Werte der Variablen aus der IDL-

Datentyp-Datenbank. Mit diesen Werten erfolgt unter Verwendung des *DII (Dynamic Invocation Interface*) der Methodenaufruf. Ergebniswerte des Methodenaufrufs werden vom Wrapper in die Datenbank der IDL-Datentypen eingetragen.

Die Aufgabe des Refinement-Managers ist die Erzeugung eines Subworkflows.

Der Subworkflow wird in Abhängigkeit von Patientendaten erzeugt. Der Refinement-Manager kann als CORBA-Objekt implementiert werden. Sein Aufruf kann somit einheitlich wie bei anderen CORBA-Objekten unter Verwendung des Wrappers erfolgen.
Eine Implementierung des Refinement-Managers, Gegenstand einer anderen Arbeit, existierte zum Abschlußzeitpunkt dieser Arbeit noch nicht.

Um einen Workflow in Abhängigkeit von IDL-Datentyp-Instanzen steuern zu können, ist ein Modul zur Auswertung von Bedingungen erforderlich.

Während der Übersetzung von CWDL in FDL werden Bedingungen, die im CWDL-Workflow angegeben wurden, syntaktisch überprüft. Da Teilbedingungen immer außerhalb des WfMS überprüft werden müssen (IDL-Datentypen können nicht in IBM FlowMark behandelt werden), ist es sinnvoll, die gesamte Bedingung in einem einzigen Modul auszuwerten. Dieses Modul kann ebenfalls als CORBA-Objekt implementiert werden. Die auszuwertende Bedingung wird als String übergeben, zusätzlich erhält das Auswertungsmodul Kontextinformation in Form des Prozeß-Instanz-Namens sowie ggf. des Aktivitätsnamens.
Unter Verwendung der lexikalischen und eines Teils der syntaktischen Analyse, die bereits im Übersetzer implementiert wurde, wird im Modul ein Syntax-Baum der zu überprüfenden Bedingung erzeugt. In einem zweiten Schritt werden ggf. Typ-Überprüfungen durchgeführt. Daran anschließend wird der Wert von Teilbedingungen bzw. der gesamten Bedingung ermittelt. Das Ergebnis dieser Auswertung (*wahr* oder *falsch*) wird an das WfMS in ein Element des *Output-Containers* zurückgegeben. Mittels dieses Wertes kann das WfMS den weiteren Verlauf des Kontrollflusses auf einfache Weise ermitteln.

Um Legacy-Anwendungen, die ausschließlich durch Kommandozeilen-Parameter parametrisierbar sind, ausführen zu können, ist ein Modul nötig, das Werte einfacher IDL-Basis-Datentypen aus der IDL-Datenbank liest und sie in einen *Output-Container* des WfMS einträgt. Diese Werte werden dann der Legacy-Anwendung von WfMS direkt übergeben.

Die Bezeichnung der einzulesenden Werte erfolgt ebenso wie die Spezifizierung der Methoden-Parameter für einen Wrapper-Aufruf. Einzige Ausnahme ist, daß die Typen der einzulesenden Instanzen (auch Teilkomponenten von konstruierten Typen) lediglich einelementige Basis-Datentypen *(short, long, long long, unsigned short, unsigned long, unsigned long long, float, double, long double, fixed, char, string und boolean*) sind.

Um das WfMS als Modul im CORBA-Netzwerk zu repräsentieren, muß seine API in IDL definiert werden. Dafür gibt es verschiedene Möglichkeiten:

- Die API des WfMS wird auf die bereits von der WfMC definierten Schnittstellen eines WfMS abgebildet. Das bedeutet, daß ggf. zusätzliche Abbildungsfunktionen von API-Methoden und -Funktionen auf die WfMC-Schnittstellen implementiert werden müssen.
- Die API des WfMS wird direkt als Schnittstelle in IDL deklariert. Dadurch müssen keine Konversionsfunktionen erstellt werden. Gleichzeitig ist die definierte Schnittstelle jedoch nicht WfMC-konform. Dieser Nachteil zeigt sich jedoch erst, wenn WfMS verschiedener Hersteller miteinander kommunizieren sollen.
- Von der *OMG* (*Object Management Group*) werden derzeit in Form von *RFP*'s (*Request for proposal*) Möglichkeiten einer objekt-orientierten Modellierung und Verwaltung von Workflows diskutiert. Die eingereichten Vorschläge reichen von Vorschlägen der Definition von Schnittstellen, die Workflows im CORBA-Netzwerk repräsentieren, bis zu einer Verwaltung von Workflow-Instanzen als CORBA-Objekte durch CORBAfacilities. In Abhängigkeit von der Verabschiedung eines solchen Standards kann das WfMS auch durch derartige Schnittstellen im CORBA-Netzwerk repräsentiert werden.

Um die Einhaltung des Abschlußzeitpunktes einer Aktivität zu überprüfen, muß im Rahmen eines weiteren Moduls der Status (*ready, finished* usw.) einer FlowMark-Aktivität festgestellt werden können.

Dieses Modul kann einfach in Form eines ausführbaren Programms unter Verwendung der API von IBM FlowMark eine entsprechende Abfrage durchführen und deren Ergebnis in einen *Output-Container* eintragen. In Abhängigkeit von diesem Wert wird der weitere Kontrollfluß festgelegt.

Um den frühesten Startzeitpunkt einer CWDL-Aktivität in IBM FlowMark realisieren zu können, wird dieser so in FlowMark übersetzt, daß eine Aktivität aufgrund eines externen Ereignisses ausführbar wird[35].

> Dieses externe Ereignis wird durch einen Zeitgeber erzeugt. Der Zeitgeber kann der *CORBA Time Service* sein, der die Möglichkeit bietet, zu definierten Zeitpunkten Nachrichten zu versenden.

Um den Eingang von Nachrichten für Nachrichten-Aktivitäten zu realisieren, kann als weiteres Modul der *CORBA Event Service* zum Einsatz kommen.

> Dieser versendet aufgrund eingehender Nachrichten Nachrichten an weitere Empfänger. Zieht man die Arbeitsweise des KIS in Betracht, so können eingehende Nachrichten von beliebigen Einrichtungen des Krankenhauses kommen. Insbesondere arbeitet das KIS ereignis-orientiert, d.h. Nachrichten werden nicht auf Anforderung versandt, sondern immer dann, wenn eine Information erzeugt wurde (z.B. „Patient wurde aufgenommen“).
> Auf diese Weise kann unter Verwendung des CORBA-Event-Service sogar eine Anpassung der unterschiedlichen Arbeitsweisen von KIS (ereignis-orientiert) und WfMS (anfrage-orientiert) erfolgen.

[35] Das geschieht ebenfalls durch die Status-Änderung einer Aktivität.

5 Zusammenfassung

5.1 Diskussion der Ergebnisse

Das Ergebnis dieser Arbeit ist eine Integration von CORBA und IBM FlowMark. Es wurde gezeigt, daß eine Integration dieser beiden Systeme möglich ist.
Unter Verwendung der neuen Modellierungssprache CWDL ist es möglich, von den technischen Einzelheiten einer Integration bereits zu Buildtime eines Workflows zu abstrahieren. Im Hinblick auf Anforderungen des Projektes „HematoWork“ wurden neben der Integration zusätzliche komplexe Funktionen in die Modellierungssprache aufgenommen (z.B. Refinement-Aktivität).
Die durch CWDL-Elemente repräsentierten Modellierungskonstrukte fassen verschiedene Aufgaben als *Aufgaben eines CWDL-Workflows* zusammen, während die Übersetzung diese in *Aufgaben des FDL-Workflows* (bzw. des WfMS IBM FlowMark) und *Aufgaben von Applikationen* (z.B. des Wrappers) aufgliedert.
Durch Module, die zur Runtime eines Workflows die korrekte Ausführung eines in CWDL definierten Workflows gewährleisten, wird das Ziel der CORBA-Integration erreicht.

Die Methode, die entwickelt wurde, ist mit dem Ausgangspunkt der Integration von CORBA in IBM FlowMark stark FlowMark-spezifisch. Das spiegelt sich besonders in der Übernahme von Syntax und Semantik der Aufbauorganisation wider. Ungeachtet dessen ist es allerdings ohne weiteres denkbar, die gleiche Methode auch bei anderen WfMS, wie z.B. COSA anzuwenden. Dann müssen allerdings die Aspekte der Aufbauorganisation von COSA verwendet werden. Falls eine CWDL-Version für IBM FlowMark und COSA geschaffen werden soll, so muß ein Organisationsmodell für CWDL erarbeitet werden, das beiden Systemen Rechnung trägt.

Die Anforderungen an diese Arbeit lagen jedoch im Bereich der Prozeß-Definition und damit in der Ablauforganisation. Das Konzept der Ablauforgansiation von CWDL abstrahiert vollständig von den Eigenschaften eines speziellen WfMS (mit Ausnahme der Verknüpfungen zur Aufbauorganisation in Form von Berechtigungen zum Ausführen von Aktivitäten). Einzige Voraussetzungen sind, daß das WfMS Applikationen ausführen kann, die Modellierung des Kontrollflusses in Abhängigkeit von Daten zuläßt und eine API anbietet, die es einer Applikation erlaubt, mit dem WfMS Daten auszutauschen. Diese Voraussetzungen sind aber gleichbedeutend mit den Eigenschaften eines WfMS und sind daher in jedem WfMS gegeben. Lediglich Details sind unterschiedlich realisiert. So erfolgt in COSA die Modellierung eines Workflows durch eine Form von Petri-Netzen, d.h. der Kontrollfluß wird durch Token gesteuert, bei IBM FlowMark erfolgt die Modellierung des Kontrollflusses durch Kontroll-Konnektoren.

Die Arbeit trägt ausschließlich der technischen Realisierbarkeit einer CORBA-Integration in IBM FlowMark Rechnung. Eine juristische Zulässigkeit wurde nicht betrachtet. Die *Sensibilität von Patienten-Daten* muß besonders bei der Zwischenspeicherung in der IDL-Datentyp-Datenbank berücksichtigt werden. Eine Kommunikation von CORBA-Objekten kann nur unter Verwendung von sicheren Übertragungs-Protokollen erfolgen (z.B. bei der Kommunikation zwischen der Studienleitung und den behandelnden Kliniken).

Die Nachteile des WfMS IBM FlowMark, wie das Fehlen eines verteilten *Commit-Protokolls* oder das *Recovery* (des FlowMark-internen DBMS ObjectStore) im Zeitscheibenverfahren wurden durch die Auslagerung von Workflow-relevanten Daten in die IDL-Datentyp-Datenbank gemildert, aber keineswegs behoben, da die Steuerung von Workflows immer noch durch das WfMS erfolgt. Lediglich die Datenmenge in ObjectStore, die der Workflow-Steuerung dient, wurde verringert.

5.2 Ausblick

Die Ergebnisse dieser Arbeit orientieren sich am gegenwärtigen Stand der Entwicklung von CORBA und WfMS. Wie bereits im Kapitel 4 am Beispiel der Kapselung von WfMS im CORBA-Netzwerk erwähnt, gibt es Tendenzen, Workflows in CORBA als CORBA-Objekte zu integrieren. Das könnte zu einen Standard führen, der im Gegensatz zum Standard der WfMC neben der syntaktischen Vereinheitlichung von Workflows auch die Semantik von Aufbau-, Ablauf- und Datenorganisation definiert. Dieser Standard kann dazu führen, daß entsprechende CORBAfacilities unabhängig vom Hersteller Eingang in CORBA-Netzwerke finden. Durch den Standard auch in der Semantik von Workflows werden solche Vereinbarungen auch ihren Niederschlag in einer einheitlichen Prozeß-Modellierungssprache finden, die ebenfalls eine CORBA-Integration zur Ausführung von Methoden als Applikation und zur Steuerung des Kontrollflusses bietet.
Neben den Bestrebungen der OMG gehen immer mehr Hersteller von kommerziellen WfMS dazu über, die CORBA-Integration bereits im WfMS selbst vorzunehmen. Beispielsweise besitzt IBM bereits Laborversionen des FlowMark-Nachfolgers *MQSeries Workflow*, die diese CORBA-Integration besitzen.

Unabhängig von den konkreten WfMS, selbst den WfMS mit CORBA-Integration, kann das Konzept von CWDL bei Bedarf an neue WfMS angepaßt werden. Neben einer Änderung der Aufbauorganisation ist lediglich eine Änderung des Übersetzers sowie der Module, die die FlowMark-API verwenden, notwendig.

Um die Modellierungssprache CWDL auch für komplexe Workflows einsetzen zu können, ist es sinnvoll, ein graphisches Buildtime-Tool zu implementieren, mit dessen Hilfe die CWDL-Definitionen einfach und übersichtlich als Graphen spezifiziert werden können. Mit Hilfe dieses Tools könnten bereits zur Buildtime unter Verwendung des CORBA-Interface-Repository's Methoden korrekt spezifiziert werden. Diesem Buildtime-Tool kann als Teilmodul der Übersetzer hinzugefügt werden. Der Übersetzer könnte auch ein Teilmodul des Refinement-Managers werden. Denkbar ist auch die Implementierung des Übersetzers als CORBA-Objekt. Derzeit ist der Übersetzer nur als ausführbares Programm verfügbar, eine Kapselung ist jedoch ohne weiteres möglich.

Literaturverzeichnis

Ba97 Baker, S.: *CORBA distributed objects*. Harlow: Addison Wesley, 1997.

Be97 Berger, H.: *Konzeption und Implementierung eines Konsultationssystems zur Therapieplanung in der Erwachsenenonkologie*. Diplomarbeit, Universität Heidelberg 1997.

Br97 Brümmer, F.: *Entwicklung einer Patienten-Datenbank zur Repräsentierung onkologischer Behandlungsverläufe*. Diplomarbeit, Universität Leipzig 1997.

COSA98a COSA Solutions GmbH: *COSA-Administrator-Handbuch*. Pulheim 1998.

COSA98b COSA Solutions GmbH: *COSA-Benutzer-Handbuch*. Pulheim 1998.

COSA98c COSA Solutions GmbH: *COSA-Referenz-Handbuch*. Pulheim 1998.

COSA98d COSA Solutions GmbH: *COSA-Programmierer-Handbuch*. Pulheim 1998.

Du81 Dudeck, J.: *DV-Unterstützung klinischer Studien*. In [Vi81].

Fi99 Fiebig, F.: *Ein Prozeßmodell für das Dokumentenmanagement in einer onkologischen Studienzentrale*. Diplomarbeit, Universität Leipzig 1999.

He95 Herold, H.: lex und yacc: *Lexikalische und syntaktische Analyse*. Bonn: Addison Wesley, 1995

Hö81 Hölzel, D.: *Anforderungen an Softwareinstrumente für kontrollierte klinische Studien*. In [Vi81].

IBM96a International Business Machines Corporation (IBM): *IBM FlowMark: Modeling Workflow, Version 2 Release 3*. IBM 1996.

IBM96b International Business Machines Corporation (IBM): *IBM FlowMark: Programming Guide, Version 2 Release 3*. IBM 1996.

IBM96c International Business Machines Corporation (IBM): *IBM FlowMark: Managing Your Workflow, Version 2 Release 3*. IBM 1996.

IBM96d International Business Machines Corporation (IBM): *IBM FlowMark: Installation and Maintenance, Version 2 Release 3*. IBM 1996.

Jö97 Jödecke, E.: *Konzept und Implementierung eines datenbankgestützten Dokumentenstruktur-Editors und -Generators*. Diplomarbeit, Universität Leipzig 1997.

LJO87 Loeper, H.; Jäkel, H.-J.; Otter, W.: *Compiler und Interpreter für höhere Programmiersprachen*. Berlin, Akademie-Verlag 1987.

MH98 Müller, R.; Heller, B.: *A Petri Net-based Model for Knowledge-based Workflows* in *Distributed Cancer Therapy*. Universität Leipzig 1998.

NHL94 Löffler, M.; Pfreundschuh, M.: *Integratives Konzept zur Behandlung hochmaligner Non-Hodgkin-Lymphome*. Studienprotokoll der Studie B. Leipzig 1994.

O2T97 O_2 Technology: *O_2CORBA User Manual. Release 5.0*. Oktober 1997.

OMG97 Object Management Group, Inc. (OMG): *CORBAservices: Common Object Services Spezification*. November 1997.

OMG98 Object Management Group, Inc.(OMG): *The Common Object Request Broker: Architecture and Specification*. Revision 2.2, Februar 1998.

Orb97a IONA Technologies Ltd: *Interface Repository Supplement*. Februar 1997.

Orb97b IONA Technologies Ltd: *Orbix Programmer's Reference*. Oktober 1997.

Ra94 Rahm, E.: *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung*. Bonn, Addison-Wesley 1994.

Sch95 Schäfer, H.: *Methodik kontrollierter klinischer Therapiestudien*. Schriftenreihe der Abteilung Medizinische Biometrie. Universität Heidelberg, 1995.

Schu97a Schulze, W.: *Evaluation of the Submissions to the Workflow Management Facility RFP*. OMG Document bom/97-09-02.

Schu97b Schulze, W.: *The OMG Workflow Working Group's Workflow Facility RFP*. Presentation to the OMG Architecture Board (AB). OMG Document cf/97-05-08. OMG Technical Meeting, Stresa (Italy), 1997.

Schu97c Schulze, W.: *Workflow Facility RFP*. Presentation to the OMG BODTF and the CFTF. OMG Technical Meeting, Austin (Texas), 1997.

Schm97 Schmidt, M.-T.: *jFlow – Joint Submission to the OMG Workflow Management Facility RFP*. Presentation to BODTF 09/25/97.

Üb81 Überla, K. K.: *Therapiestudien: Indikation, Erkenntniswert und Herausforderung*. In: [Vi81].

Vi81 Victor, N.: *Therapiestudien*: 21.-23. September 1981, Gießen. Medizinische Informatik und Statistik, 33. Berlin, Springer 1981.

Vi81a Victor, N.: *Therapiestudien: Herausforderung für den Biometriker*. In: [Vi81].

Wa97 Warne, J.: *Workflow Management Facility Initial Submission*. bom/97-08-04.

WfMC94 Workflow Management Coalition: *The Workflow Reference Model*. Document Number TC00-1003, Document Status – Issue 1.1, November 1994.

WfMC96 Workflow Management Coalition: *Terminology & Glossary*. Document Number WFMC-TC-1011, Document Status - Issue 2.0, Juni 1996.

WfMC97 Workflow Management Coalition: *Workflow Facility Specification*. Document Number WFMC-TC-2101, Document Status – Draft 0.6, März 1997.

Wi94 Winter, A.: *Beschreibung, Bewertung und Planung heterogener Krankenhausinformationssysteme*. Habilitationsschrift, Heidelberg 1994.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A Die Datenbank für IDL-Datentypen

A.1 Entity-Relationship-Diagramme

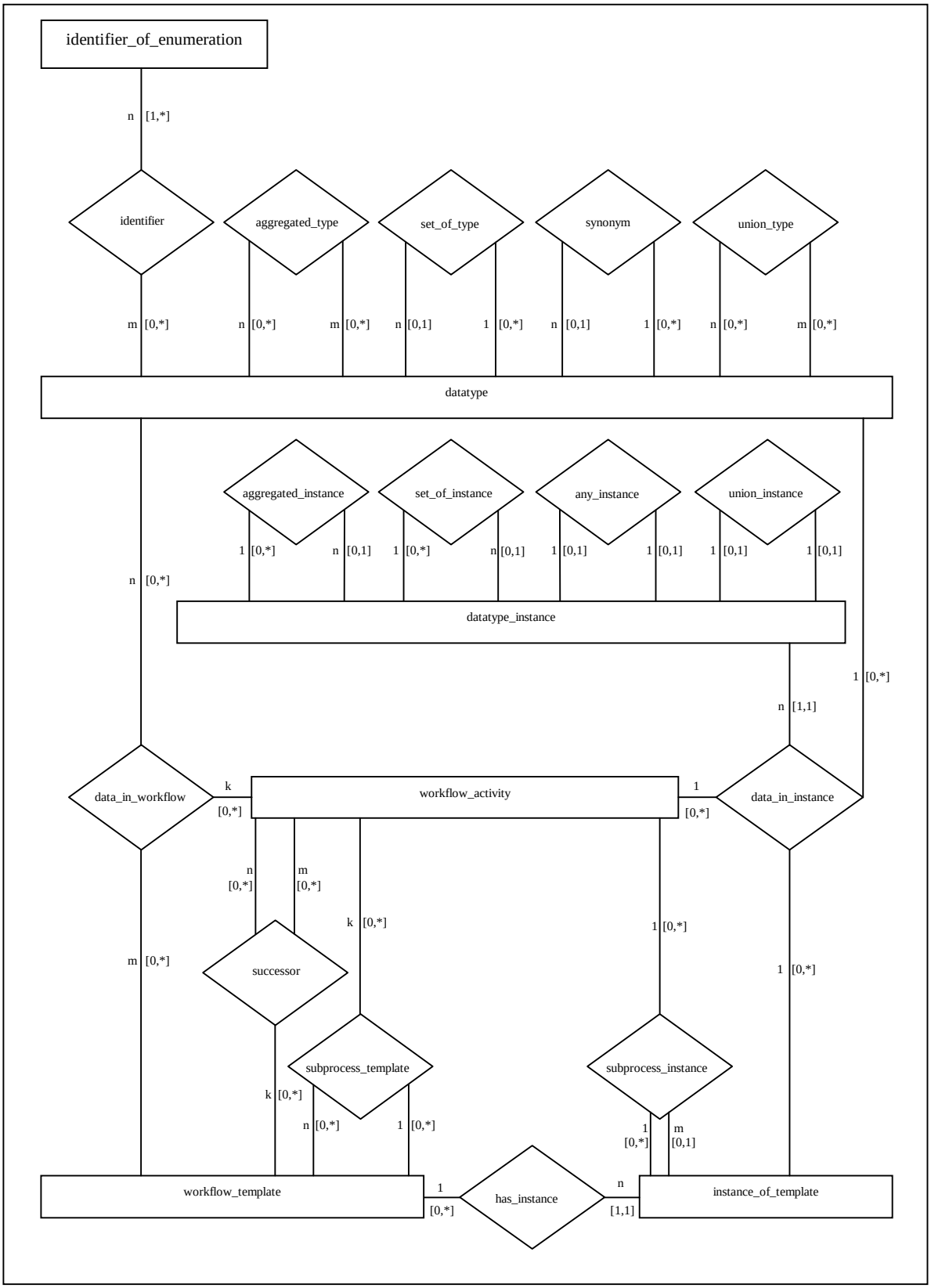


Abbildung A-1: Entity-Relationship-Diagramm für die Darstellung von IDL-Datentypen.

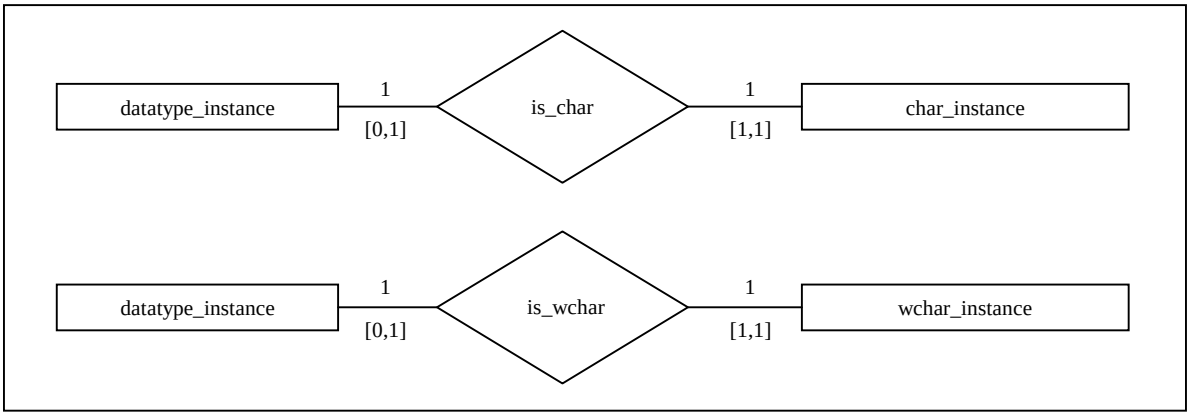


Abbildung A-2: Entities und Relationships zur Darstellung von Werten von IDL-Charakter-Typen.

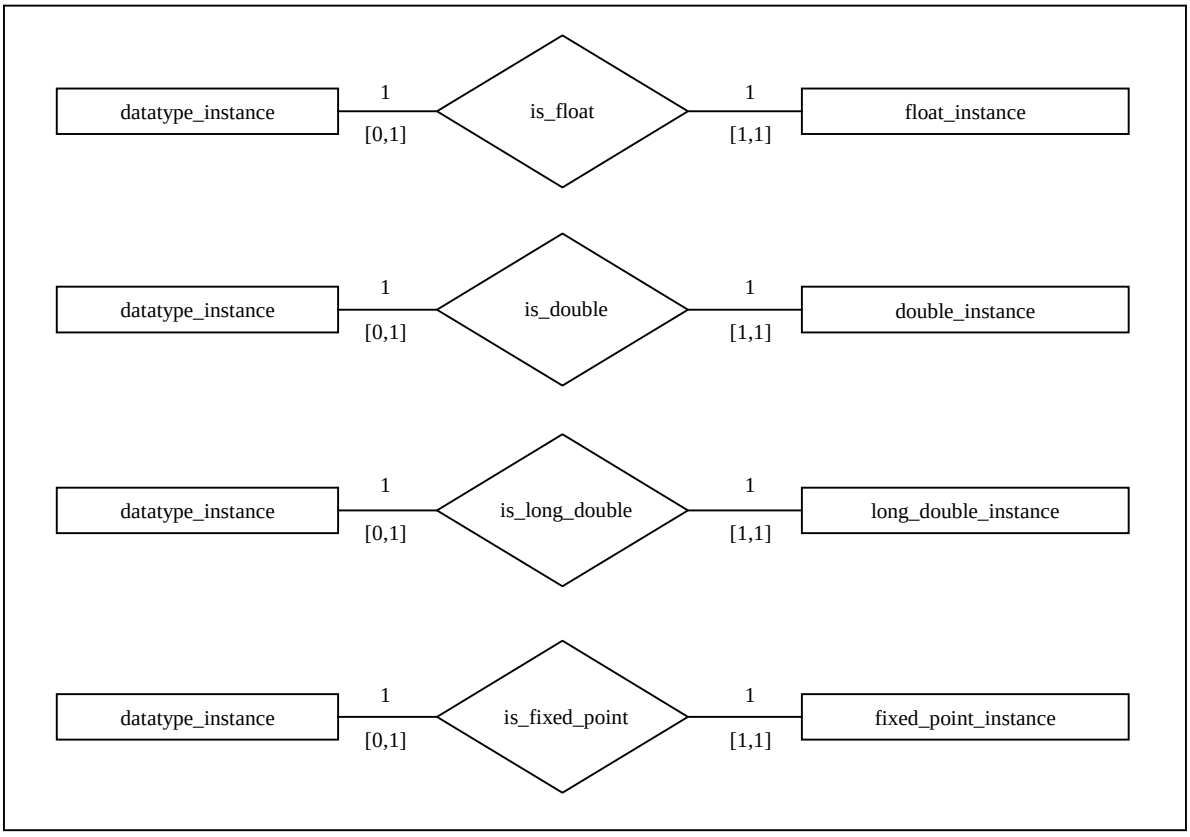


Abbildung A-3: Entities und Relationships zur Darstellung von Werten von numerischen, nicht-ganzzahligen IDL-Typen.

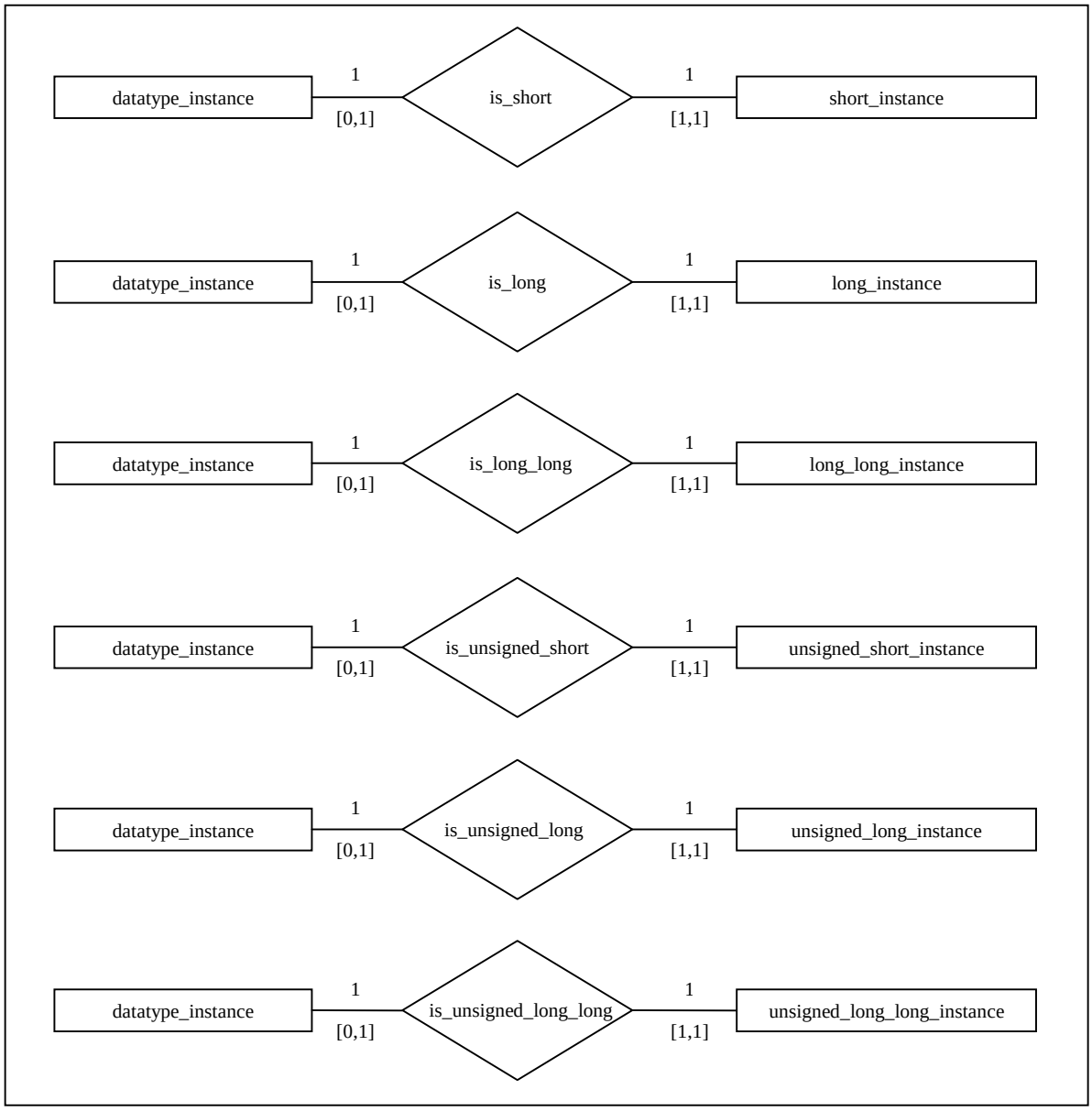


Abbildung A-4: Entities und Relationships zur Darstellung von Werten von IDL-Integer-Typen.

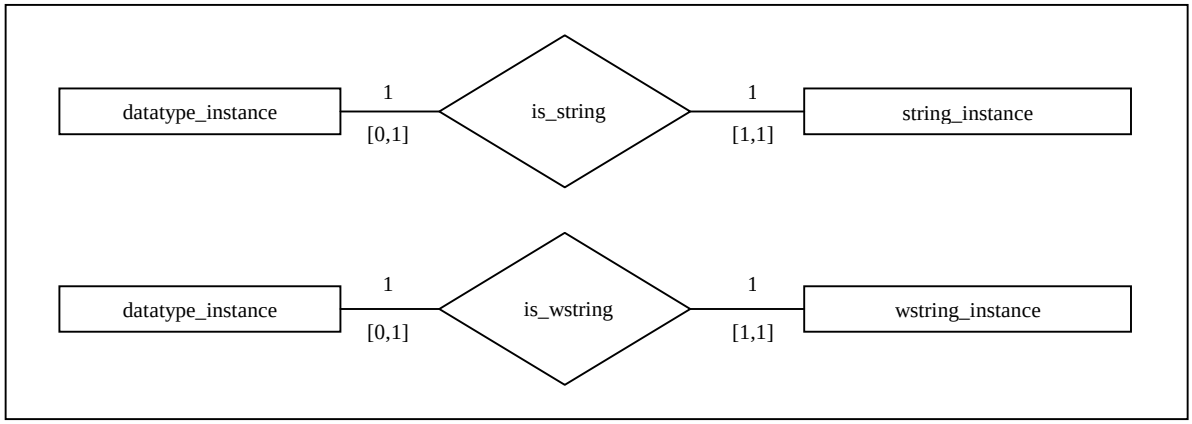


Abbildung A-5: Entities und Relationships zur Darstellung von Werten von IDL-String-Typen.

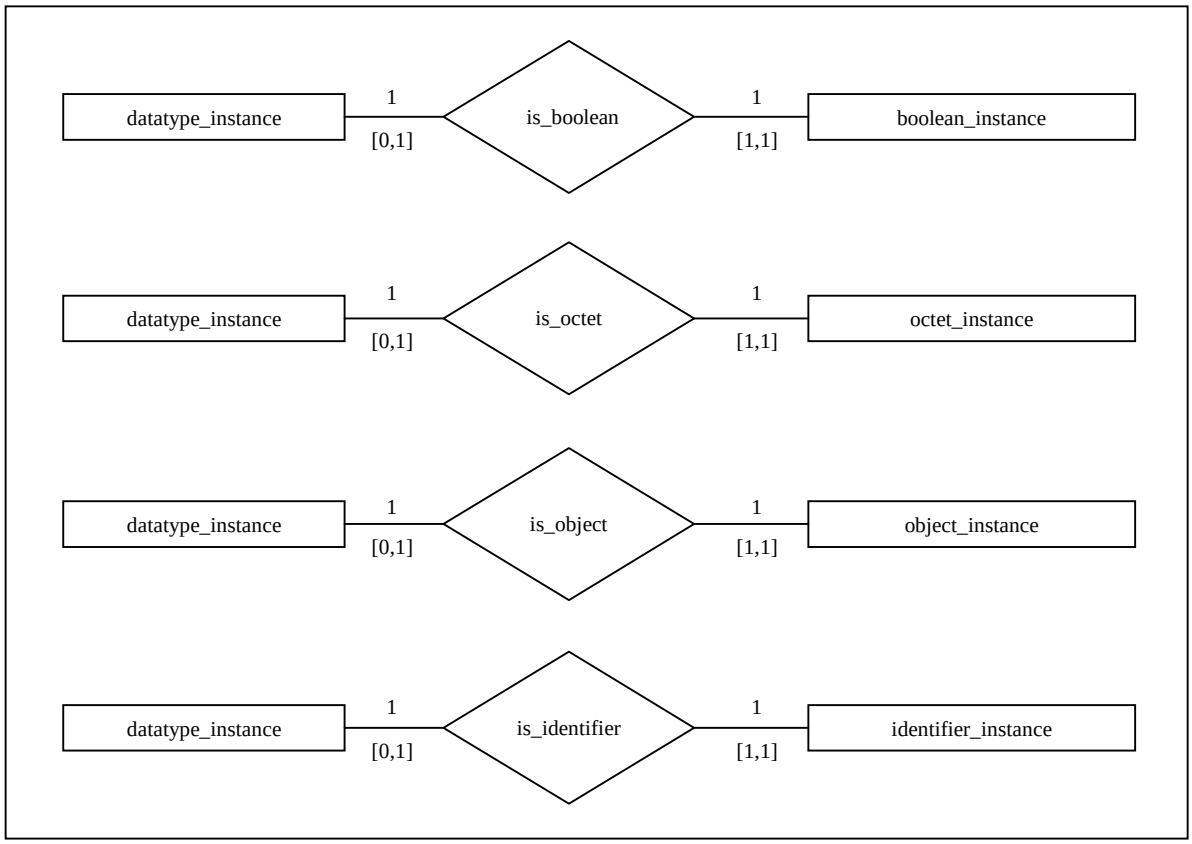


Abbildung A-6: Entities und Relationships zur Darstellung von Werten anderer IDL-Basistypen.

A.2 Relationales Schema

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name des Datentyps
type	varchar(8)	not null, type ∈ {simple, sequence, array, struct, object, enum, synonym, any, union}	Angabe der Art des Datentyps

Tabelle A-1: Die Repräsentation des Entities *datatype* in der Relation *data_type.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
enum_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Datentyps
member_name	varchar(50)	not null, Teil des Primärschlüssels	Name des Aufzählungselementes
member_number	smallint	not null	Nummer des Aufzählungselementes

Tabelle A-2: Die Repräsentation des Entities *identifier_of_enumeration* und des Relationships *identifier* in der Relation *identifier_of_enum.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des aggregierten Typs (z.B. Struct)
contd_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs eines aggregierten Typs (z.B. Struct)
member_name	smallint	not null, Teil des Primärschlüssels	Element-Name des aggregierten Typs (z.B. Struct)

Tabelle A-3: Die Repräsentation des Relationships *aggregated_type* in der Relation *aggregated_type.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Primärschlüssel, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Mengentyps (Array, Sequence)
contd_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs des Mengentyps (Array, Sequence)
member_count	smallint	not null	Größe des Mengentyps (Array, Sequence)

Tabelle A-4: Die Repräsentation des Relationships *set_of_type* in der Relation *set_of_type.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
syn_type_name	varchar(50)	not null, Primärschlüssel, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Synonym eines bestehenden Typ-Namens
orig_type_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des bestehenden Typs

Tabelle A-5: Die Repräsentation des Relationships *synonym* in der Relation *synonym.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Union-Typs
contd_type_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Element-Typs
discriminator	varchar(50)	not null	Diskriminator des Union-Typs

Tabelle A-6: Die Repräsentation des Relationships *union_type* in der Relation *union_type.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
key	smallint	not null, Primärschlüssel	für die Identifizierung einer Dateninstanz, bei der Instantierung von der Applikation erzeugt
contng_inst_key	smallint	Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	für die Identifizierung von Komponenten einer Dateninstanz,

Tabelle A-7: Die Repräsentation des Entities *datatype_instance* in der Relation *data_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Bezeichnet Instanz des Element-Typs eines aggregierten Typs (z.B. Struct)
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz des Element-Typs eines aggregierten Typs (z.B. Struct)
member_name	varchar(50)	not null, Teil des Primärschlüssels	Name des Elementes eines aggregierten Typs (z.B. Struct)

Tabelle A-8: Die Repräsentation des Relationships *aggregated_instance* in der Relation *aggregated_inst.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Bezeichnet Instanz des Mengentyps (Array, Sequence)
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz eines Elementes des Mengentyps (Array, Sequence)
member_number	smallint	not null, Teil des Primärschlüssels	Nummer des Elementes des Mengentyps (Array, Sequence)

Tabelle A-9: Die Repräsentation des Relationships *set_of_instance* in der Relation *set_of_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet any-Instanz
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz im any enthaltenen Typs

Tabelle A-10: Die Repräsentation des Relationships *any_instance* in der Relation *any_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
contng_inst_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Union-Instanz
contd_inst_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	bezeichnet Instanz des durch die Union dargestellten Typs

Tabelle A-11: Die Repräsentation des Relationships *union_instance* in der Relation *union_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name des CWDL-Workflows

Tabelle A-12: Die Repräsentation des Entities *workflow_template* in der Relation *workflow_template.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name der CWDL-Aktivität

Tabelle A-13: Die Repräsentation des Entities *workflow_activity* in der Relation *workflow_activity.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
pred_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der Quell-CWDL-Aktivität eines CWDL-Daten-Konnektors
succ_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der Ziel-CWDL-Aktivität eines CWDL-Daten-Konnektors
template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows, der beide Aktivitäten enthält

Tabelle A-14: Die Repräsentation des Relationships *successor* in der Relation *successor.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
proc_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Prozeß-Aktivität
template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des Prozesses mit der CWDL-Prozeß-Aktivität
sub_template_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Subprozesses

Tabelle A-15: Die Repräsentation des Relationships *subprocess_template* in der Relation *subprocess_template.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Aktivität, in deren CWDL-Definition eine Variable des durch *data_name* bezeichneten Datentyps vereinbart wurde
workflow_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows, in dessen CWDL-Definition (bzw. in der CWDL-Definition einer seiner Aktivitäten) eine Variable des durch *data_name* bezeichneten Datentyps vereinbart wurde
variable_name	varchar(50)	not null, Teil des Primärschlüssels	Name der Variable der CWDL-Aktivität oder des CWDL-Workflows
data_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *data_type*), on delete cascade	Name des Datentyps der Variablen
mode	smallint	not null, mode $\in$ {0,1}	Zugriffsmodus für Variable 0 = READ_WRITE 1 = READ_ONLY

Tabelle A-16: Die Repräsentation des Relationships *data_in_workflow* in der Relation *data_in_workflow.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
name	varchar(50)	not null, Primärschlüssel	Name der IBM FlowMark-Workflow-Instanz
template_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *workflow_template*), on delete cascade	Name des CWDL-Workflows

Tabelle A-17: Die Repräsentation des Entities *instance_of_template* und des Relationships *has_instance* in der Relation *inst_of_template.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Aktivität
inst_template_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der IBM FlowMark-Workflow-Instanz
variable_name	varchar(50)	not null, Teil des Primärschlüssels	Name der Variablen der CWDL-Aktivität oder des CWDL-Workflows
data_instance_key	smallint	not null, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
mode	smallint	not null, mode $\in$ {0,1}	Zugriffsmodus für Variable 0 = READ_WRITE 1 = READ_ONLY
init	smallint	not null, init $\in$ {0,1}	Initialisierungsstatus für Variable 0 = noch nicht initialisiert 1 = initialisiert

Tabelle A-18: Die Repräsentation des Relationships *data_in_instance* in der Relation *data_in_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
proc_activity_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *workflow_activity*), on delete cascade	Name der CWDL-Prozeß-Aktivität
instance_name	varchar(50)	not null, Teil des Primärschlüssels, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der Prozess-Instanz mit der CWDL-Prozeß-Aktivität
sub_instance_name	varchar(50)	not null, Fremdschlüssel (Attribut *name* der Relation *inst_of_template*), on delete cascade	Name der Subprozeß-Instanz

Tabelle A-19: Die Repräsentation des Relationships *subprocess_instance* in der Relation *subprocess_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	smallint	not null	Wert der Daten-Instanz

Tabelle A-20: Die Repräsentation des Entities *short_instance* und des Relationships *is_short* in der Relation *short_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	integer	not null	Wert der Daten-Instanz

Tabelle A-21: Die Repräsentation des Entities *long_instance* und des Relationships *is_long* in der Relation *long_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(20)	not null	Wert der Daten-Instanz

Tabelle A-22: Die Repräsentation des Entities *long_long_instance* und des Relationships *is_long_long* in der Relation *long_long_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	integer	not null	Wert der Daten-Instanz

Tabelle A-23: Die Repräsentation des Entities *unsigned_short_instance* und des Relationships *is_unsigned_short* in der Relation *unsigned_short_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(20)	not null	Wert der Daten-Instanz

Tabelle A-24: Die Repräsentation des Entities *unsigned_long_instance* und des Relationships *is_unsigned_long* in der Relation *unsigned_long_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(20)	not null	Wert der Daten-Instanz

Tabelle A-25: Die Repräsentation des Entities *unsigned_long_long_instance* und des Relationships *is_unsigned_long_long* in der Relation *unsigned_long_long_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	real	not null	Wert der Daten-Instanz

Tabelle A-26: Die Repräsentation des Entities *float_instance* und des Relationships *is_float* in der Relation *float_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	double	not null	Wert der Daten-Instanz

Tabelle A-27: Die Repräsentation des Entities *double_instance* und des Relationships *is_double* in der Relation *double_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(20)	not null	Wert der Daten-Instanz

Tabelle A-28: Die Repräsentation des Entities *long_double_instance* und des Relationships *is_long_double* in der Relation *long_double_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(1)	not null	Wert der Daten-Instanz

Tabelle A-29: Die Repräsentation des Entities *char_instance* und des Relationships *is_char* in der Relation *char_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(1)	not null	Wert der Daten-Instanz

Tabelle A-30: Die Repräsentation des Entities *wchar_instance* und des Relationships *is_wchar* in der Relation *wchar_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	smallint	not null	Wert der Daten-Instanz

Tabelle A-31: Die Repräsentation des Entities *boolean_instance* und des Relationships *is_boolean* in der Relation *boolean_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	smallint	not null	Wert der Daten-Instanz

Tabelle A-32: Die Repräsentation des Entities *octet_instance* und des Relationships *is_octet* in der Relation *octet_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(3000)	not null	Wert der Daten-Instanz

Tabelle A-33: Die Repräsentation des Entities *string_instance* und des Relationships *is_string* in der Relation *string_instance*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(3000)	not null	Wert der Daten-Instanz

Tabelle A-34: Die Repräsentation des Entities *wide_string_instance* und des Relationships *is_wide_string* in der Relation *wide_string_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value_1	smallint	not null	Vorkomma-Stellen
value_2	smallint	not null	Nachkomma-Stellen

Tabelle A-35: Die Repräsentation des Entities *fixed_point_instance* und des Relationships *is_fixed_point* in der Relation *fixed_point_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(50)	not null	Wert der Daten-Instanz

Tabelle A-36: Die Repräsentation des Entities *object_instance* und des Relationships *is_object* in der Relation *object_instance.*

Attribut	**Wertebereich** (Datentyp des DBMS)	**Integritätsbedingung**	**Bemerkung**
data_instance_key	smallint	not null, Primärschlüssel, Fremdschlüssel (Attribut *key* der Relation *data_instance*), on delete cascade	Schlüssel der Daten-Instanz
value	varchar(50)	not null	Wert der Daten-Instanz

Tabelle A-37: Die Repräsentation des Entities *identifier_instance* und des Relationships *is_identifier* in der Relation *identifier_instance.*

Anhang B CWDL

B.1 Token

CORBA-Token:

comment_begin	("/*")
comment_end	("*/")
condition_White_Space	[" "]|\t|\r|\n
condition_Other_Char	[^" "^"\t"^"\r"^\n]
corba_id_expression	({corba_unqualified_id})|({corba_qualified_id})
corba_unqualified_id	({corba_identifier})
corba_qualified_id	({corba_nested_name_specifier})({corba_unqualified_id})
corba_nested_name_specifier	(({corba_class_name})("::"))+
corba_class_name	({corba_identifier})
corba_nondigit	"_" |a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
corba_identifier	(({corba_nondigit})({corba_digit}|{corba_nondigit})*)
corba_boolean_literal	(("t"|"T")("r"|"R")("u"|"U")("e"|"E"))|(("f"|"F")("a"|"A")("l"|"L")("s"|"S")("e"|"E"))
corba_s_char	([^\\"\n])|{corba_escape_sequence}
corba_s_char_sequence	({corba_s_char})+
corba_string_literal	(L)?["]({corba_s_char_sequence})?["]
corba_octal_digit	0|1|2|3|4|5|6|7
corba_octal_escape_sequence	\\{corba_octal_digit}{1,3}
corba_simple_escape_sequence	\\\']\\\"|\\\?|\\\|\\"a"|\\"b"|\\"f"|\\"n"|\\"r"|\\"t"|\\"v"
corba_escape_sequence	({corba_simple_escape_sequence})|({corba_octal_escape_sequence})
corba_c_char	([^\\'\n])|({corba_escape_sequence})
corba_c_char_sequence	({corba_c_char})+
corba_character_literal	(L)?[']({corba_c_char_sequence})[']
corba_string	({corba_character_literal})|({corba_string_literal})
corba_digit	0|1|2|3|4|5|6|7|8|9
corba_nonzero_digit	1|2|3|4|5|6|7|8|9
corba_sign	"+"|"-"
corba_decimal_literal	"0"|({corba_sign})?({corba_nonzero_digit})({corba_digit})*
corba_integer_literal	({corba_decimal_literal})
corba_exponent_part	(e|E)({corba_sign})?({corba_digit_sequence})
corba_digit_sequence	({corba_digit})+
corba_fractional_constant	({corba_sign})?((({corba_digit})"."({corba_digit_sequence})?)|"."{corba_digit_sequence})
corba_floating_literal	({corba_fractional_constant})({corba_exponent_part})?
corba_char_literal	[']({corba_c_char})[']

CWDL-Schlüsselwörter(die bereits in FDL verwendet werden):

ACTIVITIES_OF ADDRESS AIX ALL APPC APPLICATION_TYPE AS_ADMINISTRATOR AUDIT AUTHORIZED_FOR AUTOCLOSE AUTOMATIC

CATEGORY CODEPAGE CONTROL COORDINATOR

DATA DATABASE DAYS DEFAULT DEFINED_IN DELIVERY_SERVER DESCRIPTION DO DOCUMENTATION DONE_BY DO_NOT_DELETE_FINISHED_ITEMS DURATION DURATION_FOR_DECISION

END ENTRY_POINT ENVIRONMENT EXIT

FIRST_NAME FOREGROUND FROM

GIVEN

HOSTNAME HOURS HPUX

INCLUDE INCLUDE_CHILD_ORGANIZATIONS INHERIT INPUT_CONTAINER INVISIBLE ISABSENT

LAST_NAME LAYOUT LEVEL LOCAL LOCATION

MANAGER MANUAL MAP MAXIMIZED MEMBER MIDDLE_NAME MINIMIZED

NAME NO NOT NOTIFY

OF ORGANIZATION OS2

PARAMETER PARENT_ORGANIZATION PATH_AND_FILENAME PERSON PERSON_ID PHONE PRIORITY PROCESS PROCESS_ACTIVITY PROCESS_ADMINISTRATOR PROCESS_CATEGORY PROCESS_STARTER PROGRAM PROMPT_AT_PROCESS_START PROTOCOL

RELATED_ORGANIZATION RELATED_PERSON RELATED_ROLE REMOTE_STARTER ROLE

SECOND_PHONE SERVER SINK SOURCE STAFF STAFF_INHERITED START STARTER_OF_ACTIVITY STYLE SUBSTITUTE

TCPIP TERMINATE_ON_ERROR THEN TO UNATTENDED USER_ACCOUNT USER_ID

VISIBLE

WHEN WINDOWS WINNT WORKING_DIRECTORY WORKLIST_OF

XWIN

CWDL-Schlüsselwörter(neu):

ACTIVITY AND APPLICATION_ACTIVITY

CALL_ACTIVITY CORBA_OBJECT

DEPENDS_ON

IN INOUT

JOIN JOINS

MANUAL_ACTIVITY METHOD MESSAGE_RECEIVER_ACTIVITY MESSAGE_SENDER_ACTIVITY MINUTES

OR

READ_ONLY READ_WRITE REFINEMENT_ACTIVITY REFINEMENT_MANAGER

SCRIPT_ACTIVITY SPLIT START_TIME SUBPROCESS

VALUE VARIABLE

WITH WORKFLOW_MANAGEMENT_SYSTEM

Weitere CWDL-Token:

CONDITION_DEREF_ELEMENT_OP CONDITION_DOPPELPUNKT

CONDITION_KLAMMER_AUF_UND_ZU_RUND

CONDITION_GLEICH_GLEICH CONDITION_GROESSER_GLEICH CONDITION_KLEINER_GLEICH CONDITION_LOGIC_AND CONDITION_LOGIC_OR CONDITION_UNGLEICH

FLOAT_LITERAL_FLOAT FLOAT_LITERAL_DOUBLE FLOAT_LITERAL_LONG_DOUBLE

INTEGER_LITERAL_SHORT INTEGER_LITERAL_LONG INTEGER_LITERAL_LONG_LONG

INTEGER_LITERAL_UNSIGNED_SHORT INTEGER_LITERAL_UNSIGNED_LONG INTEGER_LITERAL_UNSIGNED_LONG_LONG

CHAR_LITERAL_CHAR
STRING_LITERAL_STRING

BOOLEAN_LITERAL

B.2 Grammatik

```
condition_constant_primary_expression  : '(' condition_constant_primary_expression ')'
                                       | STRING_LITERAL_STRING
                                       | INTEGER_LITERAL_SHORT
                                       | INTEGER_LITERAL_LONG
                                       | INTEGER_LITERAL_LONG_LONG
                                       | INTEGER_LITERAL_UNSIGNED_SHORT
                                       | INTEGER_LITERAL_UNSIGNED_LONG
                                       | INTEGER_LITERAL_UNSIGNED_LONG_LONG
                                       | FLOAT_LITERAL_FLOAT
                                       | FLOAT_LITERAL_DOUBLE
                                       | FLOAT_LITERAL_LONG_DOUBLE
                                       | CHAR_LITERAL_CHAR
                                       | BOOLEAN_LITERAL
                                       ;


condition_no_constant_primary_expression      : '(' condition_no_constant_primary_expression ')'
                                              | CONDITION_DOPPELPUNKT CORBA_IDENTIFIER
                                              | CORBA_ID_EXPRESSION
                                              | CORBA_IDENTIFIER
                                              ;


condition_no_constant_postfix_expression_brackets    : condition_no_constant_primary_expression
                                                            '(' condition_conditional_expression_list ')'
                                                     | condition_no_constant_primary_expression CONDITION_KLAMMER_AUF_UND_ZU_RUND
                                                     ;

condition_no_constant_postfix_expression_without_brackets  : condition_no_constant_primary_expression
                                                           | condition_no_constant_postfix_expression
                                                                '[' condition_logical_or_expression ']'
                                                           ;

condition_no_constant_postfix_expression      : condition_no_constant_postfix_expression_brackets
                                              | condition_no_constant_postfix_expression_without_brackets
                                              ;
```

```
condition_no_function_no_constant_postfix_expression_term_field   : condition_no_function_no_constant_postfix_expression_field
                                                                  '[' condition_logical_or_expression ']'
                                                                  | '('
                                                                     condition_no_function_no_constant_postfix_expression_term_field
                                                                    ')'
                                                                  ;

condition_no_function_no_constant_postfix_expression_field : condition_no_function_no_constant_postfix_expression_term_field
                                                           | condition_no_constant_primary_expression
                                                           ;



condition_no_function_no_constant_postfix_expression : condition_no_function_no_constant_postfix_expression_field
                                                     ;

condition_postfix_term_expression      : condition_postfix_expression '.' condition_no_constant_postfix_expression
                                       | condition_postfix_expression  CONDITION_DEREF_ELEMENT_OP
                                            condition_no_constant_postfix_expression
                                       | '(' condition_postfix_term_expression ')'
                                       ;

condition_postfix_expression     : condition_postfix_term_expression
                                 | condition_no_function_no_constant_postfix_expression
                                 ;

condition_conditional_expression_list   : condition_conditional_expression
                                        | condition_conditional_expression_list ',' condition_conditional_expression
                                        ;


condition_arithmetical_atom      : condition_postfix_expression
                                 | condition_constant_primary_expression
                                 ;

condition_unary_term_expression_term    : condition_unary_expression
                                        | '(' condition_multiplicative_term_expression ')'
                                        | '(' condition_additive_term_expression ')'
                                        | '(' condition_logical_relational_term_expression ')'
                                        | '(' condition_logical_equality_term_expression ')'
                                        | '(' condition_logical_and_term_expression ')'
                                        | '(' condition_logical_or_term_expression ')'
                                        ;
```

```
condition_arithmetical_unary_operator   : '+'
                                        | '-'
                                        | '!'
                                        ;

condition_unary_term_expression  : condition_arithmetical_unary_operator condition_unary_term_expression_term
                                 | '(' condition_unary_term_expression ')'
                                 ;

condition_unary_expression: condition_unary_term_expression
                          | condition_arithmetical_atom
                          ;

condition_cast_expression : condition_unary_expression
                          ;

condition_pm_expression   : condition_cast_expression
                          ;

condition_multiplicative_term_expression_right      : condition_pm_expression
                                                    | '(' condition_multiplicative_term_expression ')'
                                                    | '(' condition_additive_term_expression ')'
                                                    | '(' condition_logical_relational_term_expression ')'
                                                    | '(' condition_logical_equality_term_expression ')'
                                                    | '(' condition_logical_and_term_expression ')'
                                                    | '(' condition_logical_or_term_expression ')'
                                                    ;

condition_multiplicative_term_expression_left : condition_multiplicative_expression
                                              | '(' condition_additive_term_expression ')'
                                              | '(' condition_logical_relational_term_expression ')'
                                              | '(' condition_logical_equality_term_expression ')'
                                              | '(' condition_logical_and_term_expression ')'
                                              | '(' condition_logical_or_term_expression ')'
                                              ;

condition_multiplicative_term_expression      : condition_multiplicative_term_expression_left '*'
                                                    condition_multiplicative_term_expression_right
                                              | condition_multiplicative_term_expression_left '/'
                                                    condition_multiplicative_term_expression_right
                                              | condition_multiplicative_term_expression_left '%'
                                                    condition_multiplicative_term_expression_right
                                              | '(' condition_multiplicative_term_expression ')'
                                              ;
```

```
condition_multiplicative_expression     : condition_multiplicative_term_expression
                                        | condition_pm_expression
                                        ;

condition_additive_term_expression_term_right : condition_multiplicative_expression
                                              | '(' condition_additive_term_expression ')'
                                              | '(' condition_logical_relational_term_expression ')'
                                              | '(' condition_logical_equality_term_expression ')'
                                              | '(' condition_logical_and_term_expression ')'
                                              | '(' condition_logical_or_term_expression ')'
                                              ;

condition_additive_term_expression_term_left  : condition_additive_expression
                                              | '(' condition_logical_relational_term_expression ')'
                                              | '(' condition_logical_equality_term_expression ')'
                                              | '(' condition_logical_and_term_expression ')'
                                              | '(' condition_logical_or_term_expression ')'
                                              ;

condition_additive_term_expression      : condition_additive_term_expression_term_left '+'
                                              condition_additive_term_expression_term_right
                                        | condition_additive_term_expression_term_left '-'
                                              condition_additive_term_expression_term_right
                                        | '(' condition_additive_term_expression ')'
                                        ;

condition_additive_expression    : condition_multiplicative_expression
                                 | condition_additive_term_expression
                                 ;

condition_shift_expression: condition_additive_expression
                          ;

condition_logical_relational_term_expression_term_right    : condition_shift_expression
                                                           | '(' condition_logical_relational_term_expression ')'
                                                           | '(' condition_logical_equality_term_expression ')'
                                                           | '(' condition_logical_and_term_expression ')'
                                                           | '(' condition_logical_or_term_expression ')'
                                                           ;
```

```
condition_logical_relational_term_expression_term_left     : condition_logical_relational_expression
                                                           | '(' condition_logical_equality_term_expression ')'
                                                           | '(' condition_logical_and_term_expression ')'
                                                           | '(' condition_logical_or_term_expression ')'
                                                           ;

condition_logical_relational_term_expression  : condition_logical_relational_term_expression_term_left '<'
                                                     condition_logical_relational_term_expression_term_right
                                              | condition_logical_relational_term_expression_term_left '>'
                                                     condition_logical_relational_term_expression_term_right
                                              | condition_logical_relational_term_expression_term_left CONDITION_KLEINER_GLEICH
                                                     condition_logical_relational_term_expression_term_right
                                              | condition_logical_relational_term_expression_term_left CONDITION_GROESSER_GLEICH
                                                     condition_logical_relational_term_expression_term_right
                                              | '(' condition_logical_relational_term_expression ')'
                                              ;

condition_logical_relational_expression       : condition_logical_relational_term_expression
                                                     | condition_shift_expression
                                              ;

condition_logical_equality_term_expression_term_right      : condition_logical_relational_expression
                                                           | '(' condition_logical_equality_term_expression ')'
                                                           | '(' condition_logical_or_term_expression ')'
                                                           | '(' condition_logical_and_term_expression ')'
                                                           ;

condition_logical_equality_term_expression_term_left : condition_logical_equality_expression
                                                     | '(' condition_logical_or_term_expression ')'
                                                     | '(' condition_logical_and_term_expression ')'
                                                     ;

condition_logical_equality_term_expression    : condition_logical_equality_term_expression_term_left CONDITION_GLEICH_GLEICH
                                                     condition_logical_equality_term_expression_term_right
                                              | condition_logical_equality_term_expression_term_left CONDITION_UNGLEICH
                                                     condition_logical_equality_term_expression_term_right
                                              | '(' condition_logical_equality_term_expression ')'
                                              ;

condition_logical_equality_expression  : condition_logical_equality_term_expression
                                       | condition_logical_relational_expression
                                       ;
```

```
condition_logical_and_term_expression_term_right    : condition_logical_equality_expression
                                                    | '(' condition_logical_and_term_expression ')'
                                                    | '(' condition_logical_or_term_expression ')'
                                                    ;

condition_logical_and_term_expression_term_left     : condition_logical_and_expression
                                                    | '(' condition_logical_or_term_expression ')'
                                                    ;

condition_logical_and_term_expression  :  condition_logical_and_term_expression_term_left CONDITION_LOGIC_AND
                                              condition_logical_and_term_expression_term_right
                                       | '(' condition_logical_and_term_expression ')'
                                       ;

condition_logical_and_expression : condition_logical_and_term_expression
                                 | condition_logical_equality_expression
                                 ;

condition_logical_or_term_expression_term_right     : condition_logical_and_expression
                                                    | '(' condition_logical_or_term_expression ')'
                                                    ;

condition_logical_or_term_expression_term_left      : condition_logical_or_expression
                                                    ;

condition_logical_or_term_expression   : condition_logical_or_term_expression_term_left CONDITION_LOGIC_OR
                                             condition_logical_or_term_expression_term_right
                                       | '(' condition_logical_or_term_expression ')'
                                       ;

condition_logical_or_expression  : condition_logical_or_term_expression
                                 | condition_logical_and_expression
                                 ;

condition_conditional_expression : condition_logical_or_expression
                                 ;


Integer      : INTEGER_LITERAL_SHORT
             | INTEGER_LITERAL_UNSIGNED_SHORT
             ;


Identifier   : CORBA_IDENTIFIER
             ;
```

```
String : STRING_LITERAL_STRING
       ;

File_Name    : String
             ;

Name   : Identifier
       | String
       ;

Network_Node_Name   : Name
                    ;

Network_Node_Names  : Network_Node_Name
                    | Network_Node_Names Network_Node_Name
                    ;

Control_Name : Name
             ;

Program_Name : Name
             ;

Object_Name  : Name
             ;

Method_Name  : Name
             ;

WfMS_Name    : Name
             ;

Person_Name  : Name
             ;

Person_Names : Person_Name
             | Person_Names Person_Name
             ;


Server_Name  : Name
             ;

Process_Name : Name
             ;
```

```
Refinement_Manager_Name   : Name
                          ;

Variable_Name: Name
             ;

IDL_Type_Name: Name
             ;

Organization_Name   : Name
                    ;

Role_Name    : Name
             ;

Role_Names   : Role_Name
             | Role_Names Role_Name
             ;

Condition    : START condition_conditional_expression END
             ;

OR_Condition : Condition
             ;

Exit_Condition      : Condition
                    ;

Map_From_To  : MAP Variable_Name TO Variable_Name
             ;

Data_To      : Network_Node_Name
             | SINK
             ;

Data_From    : Network_Node_Name
             | SOURCE
             ;
Data_From_To : FROM Data_From TO Data_To
             ;
Map_From_Tos : Map_From_To
             | Map_From_Tos Map_From_To
             ;
```

```
Data_Flow    : DATA Data_From_To Map_From_Tos
             | DATA Data_From_To
             ;

Control_From_To     : FROM Network_Node_Name TO Network_Node_Name
                    | FROM SOURCE TO Network_Node_Name
                    | FROM Network_Node_Name TO SINK
                    ;

Control_Settings    : DESCRIPTION String
                    | NAME Control_Name
                    ;

Start_Definition    : START_TIME Time_Definition DEPENDS_ON Network_Node_Name
                    ;

Start_Definitions   : Start_Definition
                    | Start_Definitions Start_Definition
                    ;

Control_Flow : CONTROL Control_Settings Control_From_To WHEN OR_Condition
             | CONTROL Control_Settings Control_From_To
             | CONTROL Control_From_To WHEN OR_Condition
             | CONTROL Control_From_To
             | CONTROL Control_Settings Control_From_To Start_Definitions WHEN OR_Condition
             | CONTROL Control_Settings Control_From_To Start_Definitions
             | CONTROL Control_From_To Start_Definitions WHEN OR_Condition
             | CONTROL Control_From_To Start_Definitions
             ;

Done_By_Role : COORDINATOR OF ROLE Role_Names
             | MEMBER OF ROLE Role_Names
             | ROLE Role_Names
             ;
```

```
Done_By_Specification     : PERSON Person_Names
                          | ORGANIZATION Organization_Name INCLUDE_CHILD_ORGANIZATIONS
                          | ORGANIZATION Organization_Name
                          | Done_By_Role
                          | LEVEL Integer '..' Integer
                          | PROCESS_ADMINISTRATOR
                          | PROCESS_STARTER
                          | MANAGER OF PROCESS_STARTER
                          | STAFF DEFINED_IN INPUT_CONTAINER
                          | STARTER_OF_ACTIVITY Network_Node_Names
                          | MANAGER OF STARTER_OF_ACTIVITY Network_Node_Names
                          | NOT STARTER_OF_ACTIVITY Network_Node_Names
                          ;

Common_Node_Start_Setting : MANUAL
                          | AUTOMATIC
                          ;

Common_Node_Exit_Setting  : MANUAL WHEN Exit_Condition
                          | AUTOMATIC WHEN Exit_Condition
                          | MANUAL
                          | AUTOMATIC
                          ;

Common_Node_Priority_Setting     : Integer
                                 | DEFINED_IN VARIABLE Variable_Name
                                 ;

Common_Node_Setting : DESCRIPTION String
                    | START Common_Node_Start_Setting
                    | EXIT Common_Node_Exit_Setting
                    | DOCUMENTATION String
                    | PRIORITY Common_Node_Priority_Setting
                    | DONE_BY Done_By_Specification
                    ;
```

```
Program_Parameter   : IN Identifier
                    | IN String
                    | IN Integer
                    | INOUT Identifier
                    | INOUT String
                    | INOUT Integer
                    | VALUE Identifier
                    | VALUE String
                    | VALUE Integer
                    ;

Program_Parameters  : Program_Parameter
                    | Program_Parameters Program_Parameter
                    ;

Activity_Program_Setting  : PROGRAM Program_Name
                          | PROGRAM Program_Name Parameter_Setting
                          ;

Activity_Method_Setting   : CORBA_OBJECT Object_Name METHOD Method_Name Parameter_Setting
                          ;

Application_Activity_Setting     : APPLICATION_ACTIVITY Activity_Program_Setting
                                 | APPLICATION_ACTIVITY Activity_Method_Setting
                                 ;

Script_Activity_Setting   : SCRIPT_ACTIVITY Script_Activity_Settings
                          ;

Script_Activity_Settings  : Activity_Program_Setting
                          | Activity_Method_Setting
                          | Script_Activity_Settings Activity_Program_Setting
                          | Script_Activity_Settings Activity_Method_Setting
                          ;

Manual_Activity_Setting   : MANUAL_ACTIVITY
                          ;


Message_Sender_Activity_Setting  : MESSAGE_SENDER_ACTIVITY WORKFLOW_MANAGEMENT_SYSTEM WfMS_Name WITH Condition
                                 ;


Message_Receiver_Activity_Setting      : MESSAGE_RECEIVER_ACTIVITY WORKFLOW_MANAGEMENT_SYSTEM WfMS_Name WITH Condition
                                       ;
```

```
Process_Activity_Starter_Setting : REMOTE_STARTER DEFINED_IN Variable_Name
                                 | REMOTE_STARTER Person_Name
                                 ;

Process_Activity_Server_Setting  : SERVER DEFINED_IN Variable_Name
                                 | SERVER Server_Name
                                 ;

Subprocess_Settings : SUBPROCESS Process_Name Process_Activity_Server_Setting Process_Activity_Starter_Setting
                    | SUBPROCESS Process_Name Process_Activity_Server_Setting
                    | SUBPROCESS Process_Name Process_Activity_Starter_Setting
                    | SUBPROCESS Process_Name
                    ;

Process_Activity_Settings  : PROCESS_ACTIVITY Subprocess_Settings
                           ;

Call_Activity_Settings     : CALL_ACTIVITY Subprocess_Settings
                           ;

Refinement_Activity_Setting      : REFINEMENT_ACTIVITY REFINEMENT_MANAGER Refinement_Manager_Name Parameter_Setting
                                 ;

Spezific_Activity_Setting  : Application_Activity_Setting
                           | Script_Activity_Setting
                           | Manual_Activity_Setting
                           | Message_Sender_Activity_Setting
                           | Message_Receiver_Activity_Setting
                           | Process_Activity_Settings
                           | Call_Activity_Settings
                           | Refinement_Activity_Setting
                           ;

Days   : Integer DAYS
       ;

Hours  : Integer HOURS
       ;


Minutes      : Integer MINUTES
             ;

Time_Definition_3   : Minutes
                    ;
```

```
Time_Definition_2   : Hours Time_Definition_3
                    | Time_Definition_3
                    | Hours
                    ;

Time_Definition     : Days Time_Definition_2
                    | Time_Definition_2
                    | Days
                    ;

Process_Notification_Setting     : Subprocess_Settings
                                 ;

Notify_type  : PROCESS_ADMINISTRATOR
             | MANAGER
             | COORDINATOR
             | Name
             ;

Notification_1      : NOTIFY Notify_type
                    | Notification_1 NOTIFY Notify_type
                    ;

Person_Notification_2     : Notification_1
                          | Notification_1 DURATION_FOR_DECISION  Time_Definition
                          ;

Person_Notification_Setting_1    : DURATION Time_Definition
                                 | DURATION Time_Definition THEN Person_Notification_2
                                 ;

Exception_Setting   : Person_Notification_Setting_1
                    | Process_Notification_Setting
                    ;



Common_Activity_Settings  : Variable_Definitions Common_Node_Setting Exception_Setting
                          | Variable_Definitions Common_Node_Setting
                          | Variable_Definitions Exception_Setting
                          | Variable_Definitions
                          | Common_Node_Setting Exception_Setting
                          | Common_Node_Setting
                          | Exception_Setting
                          ;
```

```
Logic_Split  : OR
             | AND
             ;

Logic_Join   : OR
             | AND
             ;

Network_Node : ACTIVITY Network_Node_Name Spezific_Activity_Setting Common_Activity_Settings END Network_Node_Name
             | ACTIVITY Network_Node_Name Spezific_Activity_Setting END Network_Node_Name
             | SPLIT Network_Node_Name Logic_Split END Network_Node_Name
             | JOIN Network_Node_Name Logic_Join JOINS Network_Node_Name END Network_Node_Name
             | SOURCE
             | SINK
             ;

Definitions  : Definition
             | Definitions Definition
             ;


Definition   : Network_Node
             | Control_Flow
             | Data_Flow
             ;

Process_Setting     : STAFF_INHERITED
                    | DURATION Time_Definition THEN NOTIFY
                    | ORGANIZATION Organization_Name
                    | ROLE Role_Name
                    | DOCUMENTATION String
                    | PROCESS_ADMINISTRATOR Person_Name
                    | AUDIT
                    | TERMINATE_ON_ERROR
                    | PROMPT_AT_PROCESS_START
                    | CATEGORY Name
                    | DESCRIPTION String
                    ;

Process_Settings    : Process_Setting
                    | Process_Settings Process_Setting
                    ;
```

```
Access_Spezifier    : READ_ONLY
                    | READ_WRITE
                    ;

Variable_Definition : VARIABLE Access_Spezifier IDL_Type_Name Variable_Name END
                    | VARIABLE IDL_Type_Name Variable_Name END
                    ;

Variable_Definitions      : Variable_Definition
                          | Variable_Definitions Variable_Definition
                          ;

Process_Definition  : PROCESS Process_Name Variable_Definitions Process_Settings Definitions END Process_Name
                    | PROCESS Process_Name Process_Settings Definitions END Process_Name
                    | PROCESS Process_Name Variable_Definitions Definitions END Process_Name
                    | PROCESS Process_Name Definitions END Process_Name
                    ;

WINNT_Setting: PATH_AND_FILENAME File_Name
             | ENTRY_POINT String
             | WORKING_DIRECTORY String
             | ENVIRONMENT String
             | DO NOT INHERIT ENVIRONMENT
             | INHERIT ENVIRONMENT
             | STYLE VISIBLE
             | STYLE INVISIBLE
             | STYLE MINIMIZED
             | STYLE MAXIMIZED
             | START FOREGROUND
             ;

HPUX_Setting : PATH_AND_FILENAME File_Name
             | WORKING_DIRECTORY String
             | ENVIRONMENT String
             | DO NOT INHERIT ENVIRONMENT
             | INHERIT ENVIRONMENT
             | APPLICATION_TYPE XWIN
             | USER_ACCOUNT Name
             | USER_ID Name
             ;
```

```
WIN_Setting  : PATH_AND_FILENAME File_Name
             | ENTRY_POINT String
             | WORKING_DIRECTORY String
             | ENVIRONMENT String
             | DO NOT INHERIT ENVIRONMENT
             | INHERIT ENVIRONMENT
             | STYLE VISIBLE
             | STYLE INVISIBLE
             | STYLE MINIMIZED
             | STYLE MAXIMIZED
             | START FOREGROUND
             ;

AIX_Setting  : PATH_AND_FILENAME File_Name
             | WORKING_DIRECTORY String
             | ENVIRONMENT String
             | DO NOT INHERIT ENVIRONMENT
             | INHERIT ENVIRONMENT
             | APPLICATION_TYPE XWIN
             | USER_ACCOUNT Name
             | USER_ID Name
             ;

OS2_Setting  : PATH_AND_FILENAME File_Name
             | ENTRY_POINT String
             | WORKING_DIRECTORY String
             | ENVIRONMENT String
             | DO NOT INHERIT ENVIRONMENT
             | INHERIT ENVIRONMENT
             | STYLE VISIBLE
             | STYLE INVISIBLE
             | STYLE MINIMIZED
             | STYLE MAXIMIZED
             | START FOREGROUND
             | NO AUTOCLOSE
             ;

Parameter_Setting   : PARAMETER Program_Parameters
                    ;
```

```
Platform_Specific_Setting  : OS2 OS2_Setting
                           | AIX AIX_Setting
                           | WINDOWS WIN_Setting
                           | HPUX HPUX_Setting
                           | WINNT WINNT_Setting
                           ;

Protocol_Specific_Setting  : LOCATION String
                           | HOSTNAME String
                           | TCPIP ADDRESS String
                           | APPC ADDRESS String
                           ;

Common_Program_Setting     : DESCRIPTION String
                           | UNATTENDED
                           ;

Program_Setting     : Common_Program_Setting
                    | Protocol_Specific_Setting
                    | Platform_Specific_Setting
                    | Parameter_Setting
                    ;

Program_Settings    : Program_Setting
                    | Program_Settings Program_Setting
                    ;

Program       : PROGRAM Program_Name Program_Settings END Program_Name
              ;

Level_Setting: DESCRIPTION String
              | NAME String
              ;

Level_Settings      : Level_Setting
                    | Level_Settings Level_Setting
                    ;

Level  : LEVEL Integer Level_Settings END Integer
       | LEVEL Integer END Integer
       ;
```

```
Organization_Entry  : DESCRIPTION String
                    | MANAGER Person_Name
                    | PARENT_ORGANIZATION Organization_Name
                    | RELATED_PERSON Person_Names
                    ;

Organization_Entries      : Organization_Entry
                          | Organization_Entries Organization_Entry
                          ;

Organization : ORGANIZATION Organization_Name Organization_Entries END Organization_Name
             | ORGANIZATION Organization_Name END Organization_Name
             ;

Role_Entry   : DESCRIPTION String
             | COORDINATOR Person_Name
             | RELATED_PERSON Person_Names
             ;

Role_Entries : Role_Entry
             | Role_Entries Role_Entry
             ;

Role   : ROLE Role_Name Role_Entries END Role_Name
       | ROLE Role_Name END Role_Name
       ;

Authorize_process_categories     : Person_Name AS_ADMINISTRATOR
                                 | Person_Name
                                 | Authorize_process_categories Person_Name AS_ADMINISTRATOR
                                 | Authorize_process_categories Person_Name
                                 ;

Authorize    : STAFF
             | PROCESS
             | PROCESS_CATEGORY Authorize_process_categories
             | PROCESS_CATEGORY ALL
             | ACTIVITIES_OF Person_Names
             | ACTIVITIES_OF ALL
             | WORKLIST_OF Person_Names
             | WORKLIST_OF ALL
             ;
```

```
Authorizes   : Authorize
             | Authorizes Authorize
             ;

Person_Entry : DESCRIPTION String
             | PERSON_ID Person_Name
             | LAST_NAME Name
             | FIRST_NAME Name
             | MIDDLE_NAME Name
             | PHONE String
             | SECOND_PHONE String
             | LEVEL Integer
             | SUBSTITUTE Person_Name
             | RELATED_ROLE Role_Names
             | RELATED_ORGANIZATION Organization_Name
             | AUTHORIZED_FOR Authorizes
             | ISABSENT
             | DO_NOT_DELETE_FINISHED_ITEMS
             ;

Person_Entries      : Person_Entry
                    | Person_Entries Person_Entry
                    ;

Person : PERSON Person_Name Person_Entries END Person_Name
       | PERSON Person_Name END Person_Name
       ;

Staff  : Person
       | Role
       | Organization
       | Level
       ;

Server_Setting      : PROTOCOL APPC
                    | PROTOCOL LOCAL
                    | PROTOCOL TCPIP
                    | ADDRESS String
                    | DATABASE String
                    | DELIVERY_SERVER PROTOCOL APPC
                    | DELIVERY_SERVER PROTOCOL TCPIP
                    | DELIVERY_SERVER ADDRESS String
                    ;
```

```
Server_Settings     : Server_Setting
                    | Server_Settings Server_Setting
                    ;

Server : SERVER Server_Name Server_Settings END Server_Name
       | SERVER Server_Name END Server_Name
       ;

WfMS   : WORKFLOW_MANAGEMENT_SYSTEM WfMS_Name Activity_Method_Setting END WfMS_Name
       ;

Refinement_Manager  : REFINEMENT_MANAGER Refinement_Manager_Name Activity_Method_Setting END Refinement_Manager_Name
                    ;

Declaration  : Refinement_Manager
             | Program
             | Server
             | Staff
             | WfMS
             ;

FileDetails  : CODEPAGE Integer
             ;

Source_2     : Declaration
             | Process_Definition
             | Source_2 Declaration
             | Source_2 Process_Definition
             ;

Source : FileDetails Source_2
       | FileDetails
       | Source_2
       ;
```

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, Februar 1999

.....

Unterschrift