UNIVERSITÄT LEIPZIG

Fakultät für Mathematik und Informatik

Institut für Informatik

KONZEPTION UND PROTOTYPISCHE
IMPLEMENTIERUNG EINER WORKFLOW ENGINE
ZUR DYNAMISCHEN WORKFLOW-ADAPTATION

DIPLOMARBEIT

Betreuung durch:

Prof. Dr. habil. E. Rahm,

Dipl.-Math. R. Müller

vorgelegt von

Alexander Dietzsch

Leipzig, 10. Juli 2000

Danksagung

Bei den folgenden Personen möchte ich mich ganz besonders für die Unterstützung der vorliegenden Arbeit bedanken:

- Herrn Prof. Dr. habil. E. Rahm für die Vergabe der Aufgabenstellung und die freundliche Unterstützung der Arbeit,

- Herrn Prof. Dr. habil. A. Winter für die hilfreichen Anregungen in bezug auf Krankenhausinformationssysteme (KIS),

- Herrn Dipl. Math. R. Müller für die intensive Betreuung der Arbeit,

- meinen Kommilitonen Ulrike Greiner und Rainer Böhme für die gute Zusammenarbeit.

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Arbeit ist im Rahmen des Projektes AGENTWORK entstanden, welches eine Kooperation zwischen dem Institut für Informatik und dem Institut für Medizinische Informatik, Statistik und Epidemiologie der Universität Leipzig ist. Ziel des Projektes AGENTWORK ist die rechnergestützte Durchführung von Therapiestudien in der Hämato-Onkologie. Zur Realisierung dieses Zieles soll ein Workflow-Management-System verwendet werden, von dem man sich eine wesentliche Verbesserung der Dokumentation und der Kommunikation zwischen den beteiligten Institutionen verspricht.

Das Workflow-Management-System soll hierbei zur Unterstützung der Behandlung von Patienten verwendet werden. Im Gegensatz zu den meisten anderen Bereichen, in denen Workflow-Management-Systeme eingesetzt werden, gibt es im medizinischen Anwendungsbereich während der Therapie eines Patienten eine Vielzahl von Situationen, in denen ein laufender Workflow an geänderte Verhältnisse angepasst werden muss. Für das Workflow-Management-System bedeutet dies, dass es auf Ereignisse reagieren und gegebenenfalls den Workflow zur Laufzeit umbauen muss.

Die zur Zeit verfügbaren kommerziellen Workflow-Management-Systeme besitzen diese Möglichkeit in der Regel nicht. Meistens ist es ihnen nur möglichen einen Workflow abzubrechen oder bis zum Ende abzuarbeiten. Gerade im medizinischen Anwendungsbereich ist dies eine unzureichende Lösung. Aus diesem Grund wird im Rahmen des Projektes AGENTWORK ein Workflow-Management-System entwickelt, das es ermöglicht Workflows zur Laufzeit zu adaptieren.

1.1 Workflow-Management-Systeme

1.1.1 Konzepte und Begriffe

Ausgangspunkt für jedes Workflow-Management-System bilden die sogenannten Geschäftsprozesse.

Ein *Geschäftsprozess* stellt einen Arbeitsvorgang in einem Unternehmen oder einer Organisation dar, der ein bestimmtes Ziel verfolgt, wie zum Beispiel die Therapie eines Patienten. Der Geschäftsprozess besteht dazu aus einer Reihe von Aktivitäten, die zum erreichen des Ziels notwendig sind. Eine *Aktivität* ist die Beschreibung einer bestimmten Arbeit, die einen logischen Schritt innerhalb eines Prozesses darstellt. Man unterscheidet zwischen manuellen und automatischen Aktivitäten. Manuelle Aktivitäten sind Aktivitäten, die nicht ohne die Interaktion mit einem Benutzer auskommen. Automatische Aktivitäten hingegen lassen sich ohne menschliche Interaktion ausführen und benötigen für die Ausführung nur maschinelle Ressourcen.

> *Definition des Begriffes Workflow-Definition laut Workflow Management Coalition [WMC99]:*
> *„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."*

Um einen Geschäftsprozess teilweise oder vollständig zu automatisieren, muss aus dem Geschäftsprozess eine *Workflow-Definition*[1] modelliert werden. Die Modellierung einer Workflow-Definition erfordert allerdings, dass der zugrundeliegende Geschäftsprozess stark strukturiert ist. Für die Reihenfolge der Aktivitäten im zugrundeliegenden Geschäftprozess bedeutet dies, dass sie gewissen Regeln unterworfen ist.

Die Workflow-Definition setzt sich aus dem *Kontroll-* und *Datenfluss* zusammen. Der Kontrollfluss definiert die Reihenfolge der Arbeitsschritte und die Beziehungen zwischen den Aktivitäten. Aufgabe des Datenflusses ist es, den Weg von Informationen zwischen den Aktivitäten zu beschreiben.

[1] Workflow-Definition und Workflow-Modell werden hier als äquivalente Begriffe verwendet.

In der Folge werden Aktivitäten häufig auch als *Knoten* und die Beziehungen zwischen den Aktivitäten als *Kanten* oder *Transitionen* bezeichnet. Wichtig wird dies, wenn es um die Modellierung des Kontroll- und Datenflusses gehen wird.

Eine *Workflow-Instanz* oder nur Workflow ist schließlich eine konkrete Ausführung einer Workflow-Definition, wobei die Workflow-Definition als Vorlage für den konkreten Workflow dient.

> *Definition des Begriffes Workflow-Management-System laut Workflow Management Coalition [WMC99]:*
> *„A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.“*

Das System, das die Definition, Erzeugung, Verwaltung und Ausführung von Workflow-Definitionen und Workflow-Instanzen übernimmt, wird als *Workflow-Management-System* bezeichnet. Das System besteht aus verschiedenen Komponenten, die auf einem oder mehreren Rechnern laufen können. Die wichtigsten Komponenten eines solchen Systems sind der *Workflow Editor* und die *Workflow Engine*.

Die Architektur eines Workflow-Management-Systems wird im Abschnitt 1.1.2 "Architektur von Workflow-Management-Systemen" genauer vorgestellt. An dieser Stelle werden dann auch die verschiedenen Komponenten des Systems diskutiert.

Bei der zeitlichen Betrachtung eines Workflows[2] unterscheidet man zwischen *Buildtime* und *Runtime*. Während der Buildtime wird die Workflow-Definition modelliert und zur Runtime wird eine Workflow-Instanz, aus der Workflow-Definition, erzeugt und ausgeführt.

Während der Verarbeitung einer Workflow-Instanz werden von den Aktivitäten sogenannte *Aktivitäts-Instanzen* erzeugt. Diese Aktivitäts-Instanzen werden durch menschliche Benutzer und maschinelle Ressourcen bearbeitet. Die Benutzer des Workflow-Management-Systems werden häufig auch als *Workflow-Teilnehmer* bezeichnet. Die Interaktion mit den Benutzern erfolgt dabei über sogenannte *Worklists*,

[2] Häufig werden Workflow-Definitionen und Workflow-Instanzen einfach als Workflows bezeichnet.

in die das Workflow-Management-System alle Arbeiten einträgt, die der Nutzer ausführen soll.

1.1.2 Architektur von Workflow-Management-Systemen

Die allgemeine Architektur eines Workflow-Management-Systems lässt sich in zwei Komponenten teilen.

Die eine Komponente ist die sogenannte *Buildtime-Komponente*. Ziel dieser Komponente ist die Definition, Modellierung und der Test von Workflow-Definitionen. Es ist dabei durchaus möglich, dass die Buildtime-Komponente aus verschiedenen Werkzeugen besteht, die auch von verschiedenen Herstellern stammen können. Das Ergebnis der Komponente sind die Workflow-Definitionen.

Die andere Komponente ist die *Runtime-Komponente*. Aufgabe der Runtime-Komponente ist die Verwaltung und die Ausführung der Workflow-Instanzen, die aus den Workflow-Definitionen der Buildtime-Komponente erzeugt wurden. Diese Komponente stellt auch die Verbindung zu den Benutzern des Workflow-Management-Systems und den Anwendungsprogrammen her, die für die Verarbeitung eines Workflows benötigt werden.

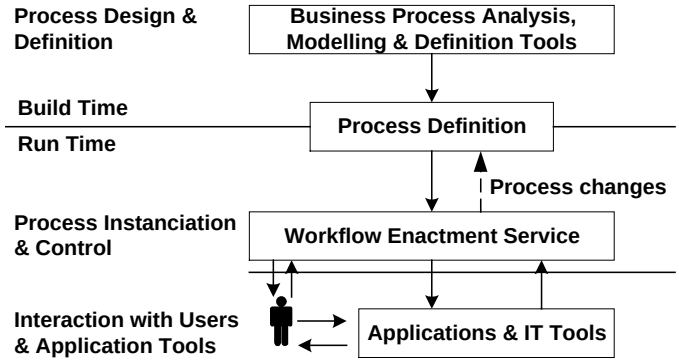


Abbildung 1 : Struktur der Workflow-Management-Systeme nach [WMC99]

Bei der Vielzahl von Workflow-Produkten auf dem Markt hat sich ein grundsätzliches Modell für die Implementierung eines Workflow-Management-Systems entwickelt,

dass von den meisten Produkten verwendet wird. Das hier vorgestellte Modelle beruht auf dem Konzept, welches von der Workflow Management Coalition [WMC99] entworfen worden ist. Dieses ist in Fachkreisen durchaus sehr umstritten[JAB97]. An dieser Stelle soll das Modell aber nur dazu verwendet werden, um die Architektur eines Workflow-Management-Systems allgemein vorzustellen. Für eine detailliertere Diskussion würde sich das Konzept der Workflow Management Coalition allerdings nicht unbedingt eignen.

Das Modell soll zur Identifikation von Komponenten und Schnittstellen eines Workflow-Management-Systems dienen. Bei der praktischen Implementierung weichen allerdings die verschiedenen Produkte in einzelnen Teilen vom Modell ab.

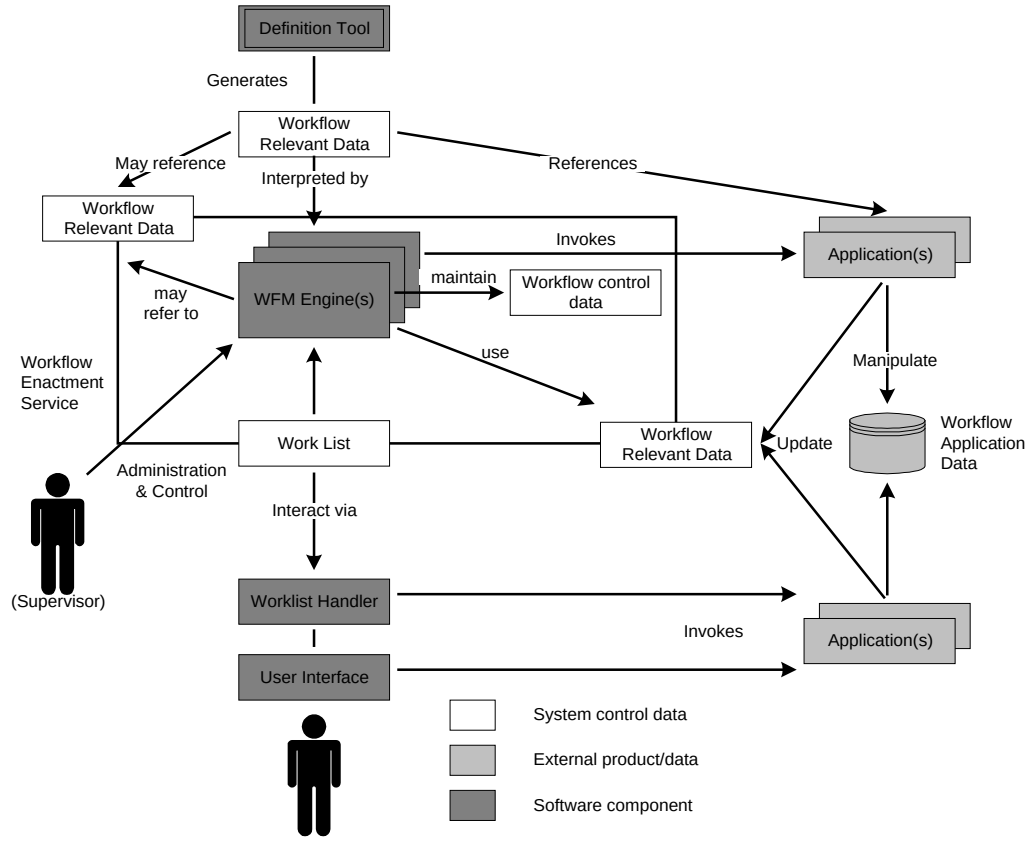


Abbildung 2 : Struktur der Workflow-Management-Systeme nach [WMC99]

Die wichtigsten Komponenten des abstrakten Architekturmodells (Abbildung 2) werden in der Folge vorgestellt:

- **Workflow Editor (Process Definition Tool)**

  Der Workflow Editor wird benutzt, um die Workflow-Definitionen zu modellieren. Dieses Werkzeug stellt die Buildtime-Komponente des Workflow-Management-Systems dar. Die Buildtime-Komponente muss nicht zwangsläufig nur ein Werkzeug umfassen, sondern kann auch aus einer Reihe von Werkzeugen bestehen, die verschiedene Aufgaben der Workflow-Definition übernehmen. Es ist zum Beispiel möglich, dass es eigene Werkzeuge für die Modellierung und den Test von Workflow-Definitionen gibt. Die verschiedenen Werkzeuge müssen nicht unbedingt nur von einem Hersteller stammen, dann ist es allerdings zwingend notwendig, dass die Schnittstellen und Austauschformate zwischen den Werkzeugen und der Runtime-Komponente des Workflow-Management-Systems kompatibel sind. Für eine ausführlichere Diskussion der Schnittstellen und Interoperabilität empfiehlt sich [WMC99].

- **Workflow Runtime Server (Workflow Enactment Service)**

  Der Workflow Runtime Server umfasst in der Hauptsache die Workflow Engine und die Worklist und stellt damit die Runtime-Komponente des Workflow-Management-Systems dar. Hauptaufgabe ist die Instanziierung, Verwaltung und Ausführung der Workflow-Instanzen und die Interaktion mit den Benutzern.

  Der Workflow Runtime Server kann aus einer oder mehreren kooperierenden Workflow Engines bestehen, die auf verschiedenen Rechnern laufen. Auf diese Weise ist es möglich die Skalierbarkeit und Verfügbarkeit des Workflow-Management-Systems zu verbessern und die Last besser zu balancieren.

  Die Workflow Engines sind für die Verarbeitung der aktiven Workflow-Instanzen zuständig. Sie besitzen Mechanismen, um Anwendungsprogramme zu starten, die für die Ausführung der Workflows notwendig sind. Bei Aktivitäten, die einen menschlichen Benutzer benötigen, erzeugen die Workflow Engines Einträge in die Worklist, die die Schnittstelle zu den Benutzern darstellt.

  Während der Ausführung eines Workflows werden durch die Workflow Engine neben den Workflow-Instanzen auch die notwendigen Daten des Datenflusses und die Statusinformationen über den internen Zustand von Workflow-Instanzen und Aktivitäts-Instanzen verwaltet. Diese Daten werden im Fehlerfall

dazu verwendet, um eine geeignete Fehlerbehandlung durchführen zu können.

- **Worklist**

  Die Worklist stellt die Kommunikation zwischen Workflow-Teilnehmern und den Workflow Engines dar. Die Workflow Engines generieren für alle Aktivitäten, die eine Interaktion mit den Benutzern benötigen, in der Worklist einen entsprechenden Eintrag. In vielen Fällen werden mehrere Benutzer den gleichen Eintrag in der Worklist erhalten, weil sie die gleiche Funktion im Unternehmen oder Organisation besitzen. Wenn der erste Benutzer die zugrundeliegende Aktivität ausführt, wird der Eintrag in der Worklist gesperrt, so dass keiner der anderen Benutzer diese Aktivität erneut starten kann. Wenn die Aktivität beendet ist, so wird der entsprechende Eintrag aus der Worklist entfernt. In der Regel ist die Worklist Teil des Workflow Runtime Servers und ist zentral angeordnet.

- **Worklist Handler und User Interface**

  Der Worklist Handler und das User Interface sind in der Regel eine gemeinsame Komponente, die auf Seiten des Nutzers angesiedelt ist. Beides zusammen bildet den Workflow Client. Auf jedem Rechner, an dem ein Workflow-Teilnehmer arbeitet, muss ein solcher Workflow Client installiert sein, so dass der Benutzer mit dem System arbeiten kann.

  Der Worklist Handler hat die Aufgabe alle Einträge aus der Worklist abzufragen, die für den konkreten Nutzer des Systems angefallen sind. Wenn der Benutzer die Aktivität, die einem Eintrag zugrunde liegt, abarbeitet, dann sperrt der Worklist Handler diesen Eintrag in der Worklist, damit kein anderer Benutzer diese Aktivität erneut ausführt. Nach Beendigung der Aktivität löscht der Worklist Handler schließlich auch den Eintrag aus der Worklist und informiert somit die Workflow Engine darüber, dass die Aktivität beendet ist.

  Das User Interface ist die Schnittstelle, über die der Nutzer mit dem Systems interagiert. In den meisten Fällen ist dies eine graphische Oberfläche, in der die Einträge der Worklist dargestellt werden, ähnlich einem E-Mail Client.

1.1.3 Middleware

Ein wichtiger Aspekt der Runtime-Komponente ist, dass die Benutzer des Workflow-Management-Systems von den verschiedenen Rechnern aus Zugriff auf das Workflow-System haben. Damit dies möglich ist benötigt das System eine Kommunikationsschicht oder *Middleware*[3], mittels derer die verschiedenen Komponenten des Systems miteinander kommunizieren und Daten austauschen.

Eine ganz wichtige Aufgabe der Middleware ist die Unterstützung des Datenflusses und der verteilten Ausführung von Aktivitäten. Dies kann unter Umständen bedeuten, dass Aktivitäten und Daten zwischen verschiedenen Rechnern ausgetauscht werden müssen. Ein solches Szenario wird in der folgenden Abbildung (Abbildung 3) gezeigt. Jeder Schritt des Workflows findet auf einem anderen Arbeitsplatz statt, so dass die Aktivitäten und Daten von einer Arbeitsstation zur nächsten transferiert werden müssen.

Die Abbildung zeigt außerdem, dass die Middleware dazu verwendet werden kann, um auf Anwendungsprogramme und Datenquellen zuzugreifen, die ebenfalls über ein Netzwerk verteilt sein können.

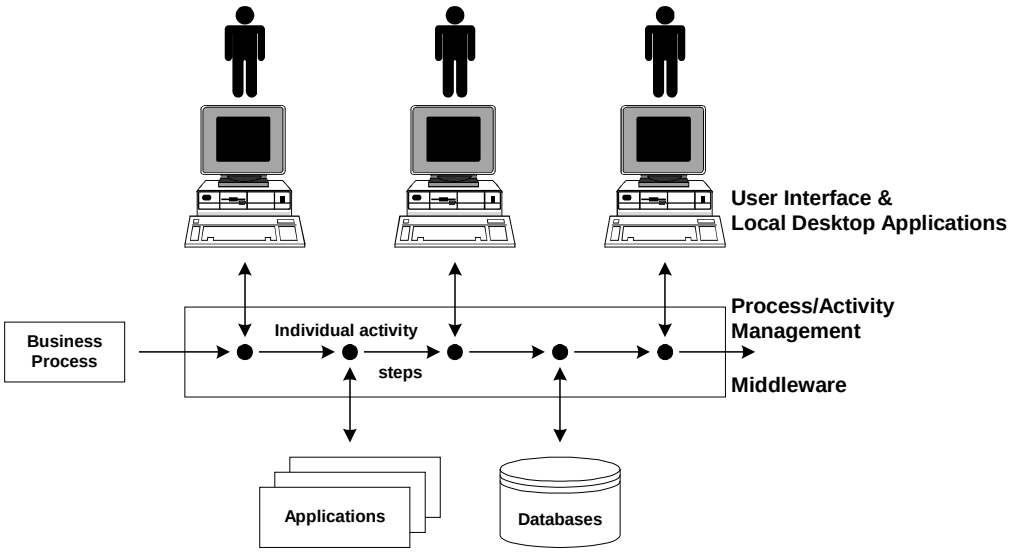


Abbildung 3 : Middleware eines Workflow-Management-Systems nach [WMC99]

[3] Häufig verwendete Middlewares sind CORBA oder DCOM. Einzelne Hersteller verwenden aber auch eigene proprietäre Systeme, z.B. IBM MQSeries Workflow verwendet IBM MQSeries.

1.2 Zielsetzung der Arbeit

Im Rahmen der vorliegenden Diplomarbeit soll ein Konzept für eine Workflow Engine entwickelt werden, die eine Adaptation der Workflows zur Laufzeit ermöglicht. Das Konzept der Workflow Engine soll dazu in das Gesamtkonzept eines Workflow-Management-Systems für dynamische Workflow-Adaptation integriert werden können. Außerdem soll die Workflow Engine auf dem globalen Datenmodell des Workflow-Management-Systems aufsetzen.

Die wichtigste Aufgabe der Workflow Engine besteht darin, einen Workflow zur Laufzeit jederzeit unterbrechen zu können, um ihn anschließend zu adaptieren. Grundlage für die Unterbrechung eines Workflows bilden die sogenannten *logischen Fehler*.

> *Logische Fehler sind Ereignisse, die während der Ausführung der Workflows auftreten und die dazu führen, dass der Kontroll- und Datenfluss der betroffenen Workflows geändert werden muss, um die Workflows an die geänderten Bedingungen anzupassen.*

Nachdem ein logischer Fehler aufgetreten ist, werden die Aktivitäten und Workflows identifiziert, die davon betroffen sind. Die Workflow Engine hat dann die Aufgabe die betroffenen Workflows zu unterbrechen, so dass keine Aktivitäten der Workflows mehr ausgeführt werden und die Ausführung der Workflows damit angehalten ist.

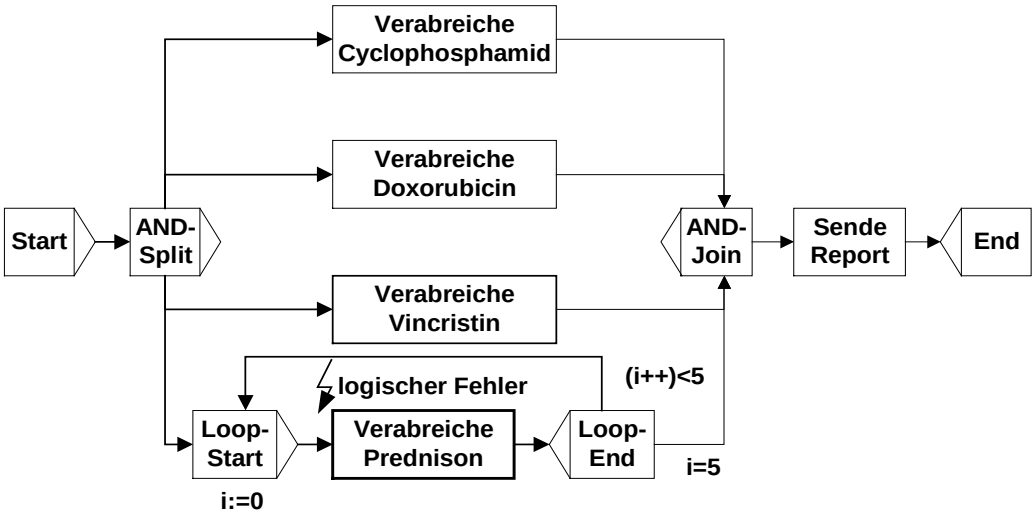


Abbildung 4 : Zu adaptierende Workflow

Nachdem alle von dem logischen Fehler betroffenen Workflows unterbrochen sind, beginnt die Adaptation des Workflows (siehe Abschnitt 2.2.2 "Agentenbasierte Schicht"). Wenn schließlich alle Änderungen an den Workflows erfolgreich durchgeführt wurden und somit eine Anpassung an die geänderten Bedingungen erfolgt ist, wird die Workflow Engine informiert und setzt die Ausführung der Workflows an der Stelle fort, an der sie unterbrochen worden sind.

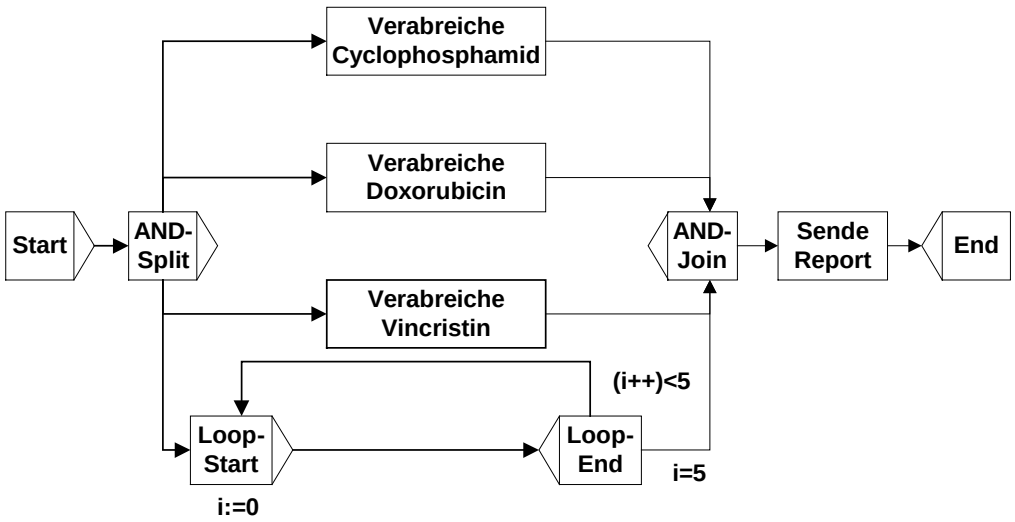


Abbildung 5 : Adaptierter Workflow

Das erarbeitete Konzept der Workflow Engine soll anschließend in einer prototypischen Implementierung umgesetzt werden. Hierbei sollen speziell die Algorithmen zur Erzeugung, Ausführung und Unterbrechung von Workflows getestet werden.

Im Vorfeld der Realisierung der Workflow Engine müssen die Buildtime-Datenbank, die Runtime-Datenbank und die Database Access Layer entworfen und implementiert werden. Die Datenbanken gewährleisten die persistente Speicherung von Workflow-Definitionen bzw. Workflow-Instanzen. Die Database Access Layer soll schließlich allen Komponenten des Workflow-Management-Systems den Zugriff auf die benötigten Daten ermöglichen und die Speicherung der Daten in den Datenbanken vor den höherliegenden Schichten verbergen.

1.3 Übersicht

Im zweiten Kapitel wird das Projekt AGENTWORK vorgestellt und auf die einzelnen Komponenten des Workflow-Management-Systems eingegangen.

Die drei anschließenden Kapitel befassen sich mit der Buildtime und Runtime des Workflow-Management-Systems. Zuerst wird die Definition und Modellierung eines Workflows beschrieben. Im Anschluss werden die Aufgaben der Runtime vorgestellt und auf die Verarbeitung und Adaptation der Workflows zur Laufzeit eingegangen. Dazu werden die wichtigsten Bestandteile der Runtime diskutiert.

In Kapitel sechs wird die Database Access Layer vorgestellt, deren Ziel der Zugriff auf die Buildtime- und Runtime-Datenbank des Workflow-Management-Systems ist. Dabei werden die Konzepte und Probleme besprochen, die durch diese Schicht adressiert werden. Diese Schicht ist eine wichtige Voraussetzung für die Implementierung der Workflow Engine.

Die prototypische Implementierung einer Workflow Engine zur dynamischen Adaptation wird im siebten Kapitel dargestellt.

Zum Abschluss werden die Ergebnisse der prototypischen Implementierung diskutiert und ein Ausblick auf die weitere Entwicklung des Workflow-Systems gegeben.

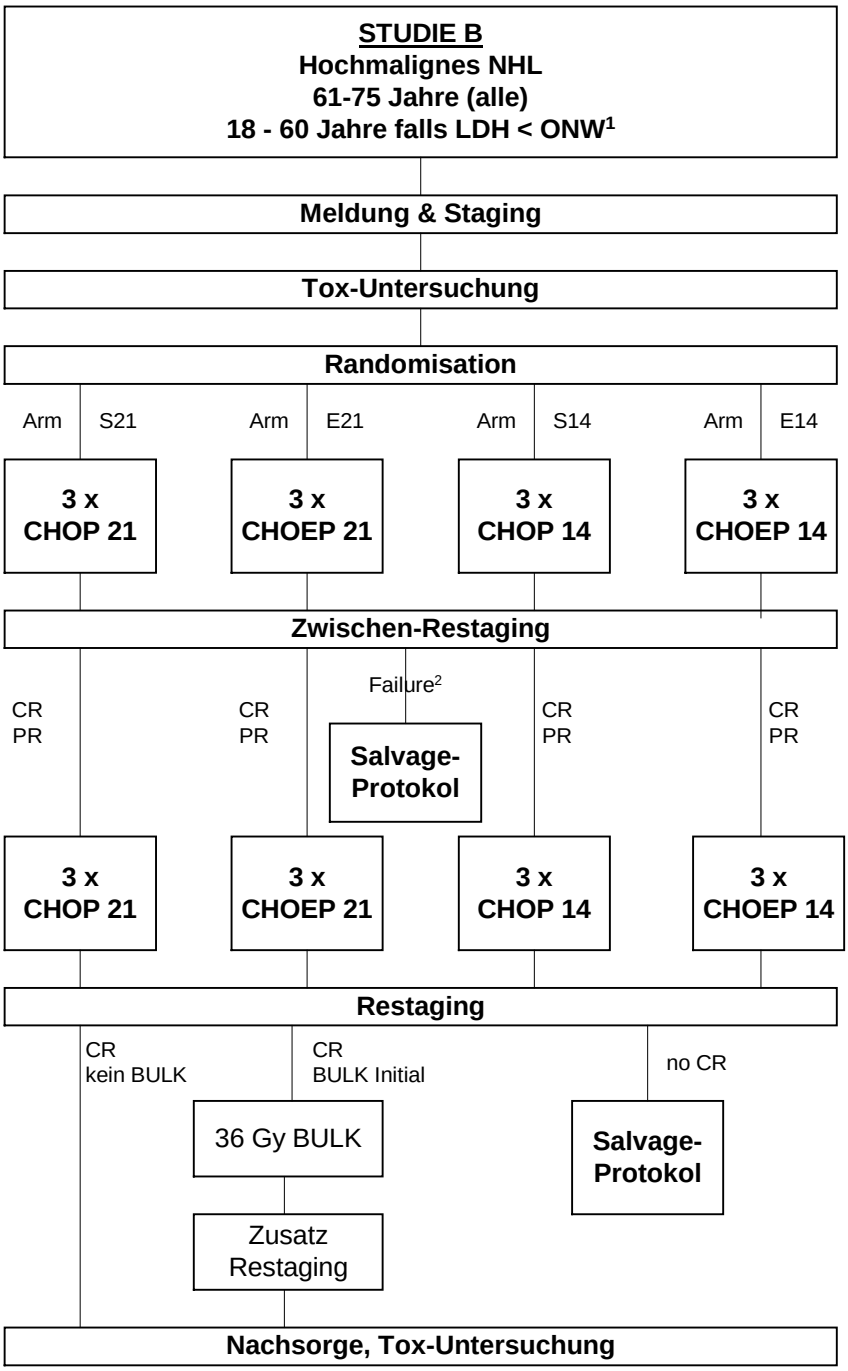
2 Projekt AgentWork

Ziel des folgenden Kapitels ist es das Projekt AGENTWORK vorzustellen und die wichtigsten Komponenten des Workflow-Management-Systems zu diskutieren.

In der Einleitung wurde bereits erwähnt, dass das Projekt AGENTWORK eine Kooperation zwischen dem Institut für Informatik und dem Institut für Medizinische Informatik, Statistik und Epidemiologie ist. Ziel von AGENTWORK ist die Automatisierung von Therapiestudien in der Hämato-Onkologie. Es wurde ebenfalls bereits erwähnt, dass es aufgrund der starken Strukturierung der Therapien möglich ist diese zu automatisieren und damit in eine Workflow-Definition zu überführen. Das Problem der Therapiestudien ist allerdings, dass eine Reihe von Ereignissen während einer Therapie eintreten können, die durch die Workflow-Definition nicht modelliert sind. Trotzdem ist es notwendig, dass auf diese Ereignisse reagiert werden muss. Oftmals sind solche Ereignisse auch nicht an eine bestimmte Situation während der Therapie gebunden und können somit jederzeit auftreten. Dieser Umstand verhindert, dass diese Ereignisse bei der Modellierung der Workflow-Definition bereits berücksichtigt werden können. Ein Beispiel für ein solches Ereignis ist zum Beispiel die allergische Reaktion eines Patienten.

Die meisten zur Zeit verfügbaren kommerziellen Workflow-Management-Systeme bieten keine Möglichkeiten auf diese Art von Situationen geeignet zu reagieren. Die Lösung des Problems besteht in der Adaptation des Workflows zur Laufzeit, um ihn an die geänderte Situation anzupassen. Bei den meisten Lösungsansätzen wird allerdings davon ausgegangen, dass die Änderungen von einem menschlichen Experten *manuell* vorgenommen werden. Für den Experten bedeutet dies, dass er sowohl Kontrollfluss, als auch Datenfluss des Workflows anpassen muss. Dies kann fehleranfällig sein und der Experte benötigt Kenntnisse im Bereich Workflow-Management und dem speziellen Anwendungsbereich, wie zum Beispiel der Medizin. Im Gegensatz dazu versucht das Projekt AGENTWORK eine weitestgehend *automatische* Entscheidung über die notwendigen Änderungen an den Workflows zu treffen. Damit erhofft man sich eine deutliche Vereinfachung der Benutzung des Workflow-Systems.

Das Gesamtkonzept des Projektes AGENTWORK wird in [MUE00] beschrieben. Die hier vorgestellte Architektur des Workflow-Management-Systems wurde im Kontext dieser Arbeit entwickelt.



[1] ONW = Oberer Normwert
[2] Failure = Pro, NC, MR
[3] TX nach 1, 2, 5 Jahren

Abbildung 6 : Hochmalignes NHL: Behandlungsplan der Studie B nach [NHL94]

Abbildung 6 zeigt einen Behandlungsplan für die Therapie von Krebs, wie er in der Hämato-Onkologie zum Einsatz kommt. Nach der Erkennung des Krebses durchläuft der Patient eine zweistufige Chemo-Therapie, die mit einer abschließenden Untersuchung und der Nachsorge beendet wird.

Wichtigste Voraussetzung für die Realisierung ist ein Mechanismus zur Erkennung von logischen Fehlern und zur automatischen Modifikation der Workflow-Instanzen. Für die Erkennung der logischen Fehler wurde ein regelbasierter Ansatz auf Basis von *F-Logic* gewählt.

2.1 F-Logic

F-Logic ist eine Logiksprache, die im Zusammenhang mit dem Projekt Florid [MM99] am Institut für Informatik der Universität Freiburg entwickelt wurde. Das Konzept von F-Logic besteht in der Erweiterung der deduktiven Programmiersprachen um die Objektorientierung für die Datenmodellierung. Damit versucht man den sogenannten *impedance mismatch* zwischen den Programmiersprachen auf der einen Seite und der Datenrepräsentation auf der anderen Seite zu überwinden. F-Logic lässt sich dadurch deutlich einfacher in eine objektorientierte Programmiersprache einbinden.

Im Rahmen des Projektes AGENTWORK bildet F-Logic die Basis für ein globales Datenmodell, auf dem alle Komponenten des Workflow-Management-Systems aufbauen. F-Logic wird zur Definition der Aktivitäten und des Datenflusses verwendet. Außerdem wird es für eine Wissensbasis benutzt, die die logischen Fehler definiert und damit die Grundlage für die Adaptation der Workflows bildet.

Die wichtigsten Gründe für die Entscheidung zu Gunsten von F-Logic sind:

- F-Logic besitzt eine *formale Semantik*, die für eine Automatisierung der Workflow-Adaptation benötigt wird.

- F-Logic besitzt eine *objektorientierte Mächtigkeit*, die eine Definition der notwendigen Daten ermöglicht.

F-Logic bietet Möglichkeiten zur Datenmodellierung mittels Klassen und Objekten, zur Definition von Regeln und zur Abfrage von Daten. Die folgende Tabelle (Tabelle 1 : Syntax von F-Logic) zeigt die wichtigsten Sprachkonstrukte und deren Syntax.

Syntax	Beschreibung
Klassen	
X::Y	Die Klasse X ist eine Subklasse der Klasse Y.
A:X	Das Objekt A ist eine Instanz der Klasse X.
x⇒Y	Das Attribut x ist ein Objekt der Klasse Y.
x⇒⇒Y	Das Attribut x ist eine Menge von Objekten der Klasse Y.
Klassen und Fakten	
x→a	Das Attribut x hat den Wert a.
x→→{a, b}	Das Attribut x enthält eine Menge mit den Werten a und b.
Regeln	
X←Y	Wenn Y gilt, dann gilt auch X.
Queries	
?-X	Alle Fakten (Daten), welche die Bedingung X erfüllen.

Tabelle 1 : Syntax von F-Logic

Die Datenmodellierung erfolgt mittels Klassen, die Attribute und Methoden besitzen, wobei Attribute als Methoden ohne Argumente aufgefasst werden. Klassen können dabei abgeleitet werden.

Beispiel:

```
person[name⇒string;
       birthday⇒date;
       sex⇒{male, female}],

patient::person,
physician::person,

patient[diagnosis⇒string,
       doctor⇒physician],
       physician[degree⇒{trainee, assistant, senior},
       patients⇒⇒patient]
```

Das Beispiel definiert drei Klassen *person*, *patient*, *physician*, wobei *patient* und *physician* von der Klasse *person* abgeleitet sind. Die Klasse *person* beinhaltet drei Attribute *name*, *birthday* und *sex*. Der Typ des Attributs *sex* ist ein Aufzählungstyp mit den beiden Werten *male* und *female*. Die Klasse *patient* ist von *person* abgeleitet und erbt damit alle Attribute der Basisklasse und besitzt darüber hinaus noch die beiden Attribute *diagnosis* und *doctor*. Die Klasse *physician* ist ebenfalls von der Klasse *person* abgeleitet und erweitert diese, um die Attribute *degree* und *patients*. Hierbei ist *patients* eine Menge von Objekten der Klasse *patient*.

Nachdem die Klassen definiert sind, ist es möglich Objekte von diesen Klassen anzulegen und diese mit Daten zu füllen. Jedes dieser Objekte muss einen eindeutigen Namen erhalten, über den das Objekt in der Folge referenziert wird.

Beispiel:

```
john:patient[name→”Parker, John”,
             birthday→1960-05-04,
             sex→male,
             diagnosis→”Leukemia”,
             doctor→anne]

anne:physician[name→”Meier, Anne”,
               birthday→1950-02-23,
               sex→female,
               degree→senior,
               patients→→{john, bob}]
```

Im Beispiel werden zwei Objekte *john* und *anne* angelegt. Das Objekt *john* ist von der Klasse *patient* und *anne* ist ein Objekt der Klasse *physician*. Die beiden Objekte haben zueinander eine Beziehung über die Attribute *patients* und *doctor*, die jeweils auf das andere Objekt verweisen.

In der Folge werden, aus Gründen der Einfachheit, alternativ zu den Ausdrücken ":" und "→" der Ausdruck "." verwendet.

Auf Basis der Klassen und Fakten können dann auch Regeln und Anfragen definiert werden.

Beispiel:

// Die Regel sagt aus, dass alle Patienten, deren Diagnose ˝leukemia type AML˝ ist,
// von der Arztin Meier, Anne behandelt werden.

D.name[“Meier, Anne”] ← P:patient ∧ P.diagnosis[“leukemia type AML”] ∧ P.doctor[D]

// Die Anfrage liefert alle Patienten, deren Arztin Meier, Anne ist und deren Diagnose
// ˝lymphoma˝ lautet.

?-P:patient ∧ P.diagnosis[“lymphoma”] ∧ P.doctor.name[“Meier, Anne”]

2.2 Architektur des adaptiven Workflow-Management-Systems

Das Ziel des Projekts AGENTWORK, die automatische Erkennung von logischen Fehlern und die automatische Modifikation der betroffenen Workflow-Instanzen, bedingt einige Änderungen an der Architektur des Workflow-Management-Systems im Verhältnis zur allgemeinen Architektur.

Die Architektur des adaptierbaren Workflow-Management-System besteht in der Hauptsache aus drei Schichten:

- **Workflow-Definitions- und Ausführungsschicht:**

  Definition und Ausführung von Workflows

- **Agentenbasierte Schicht:**

  dynamische Workflow-Adaptation

- **CORBA Management Layer:**

  Verwaltung von Workflow-Ressourcen und Datenintegration

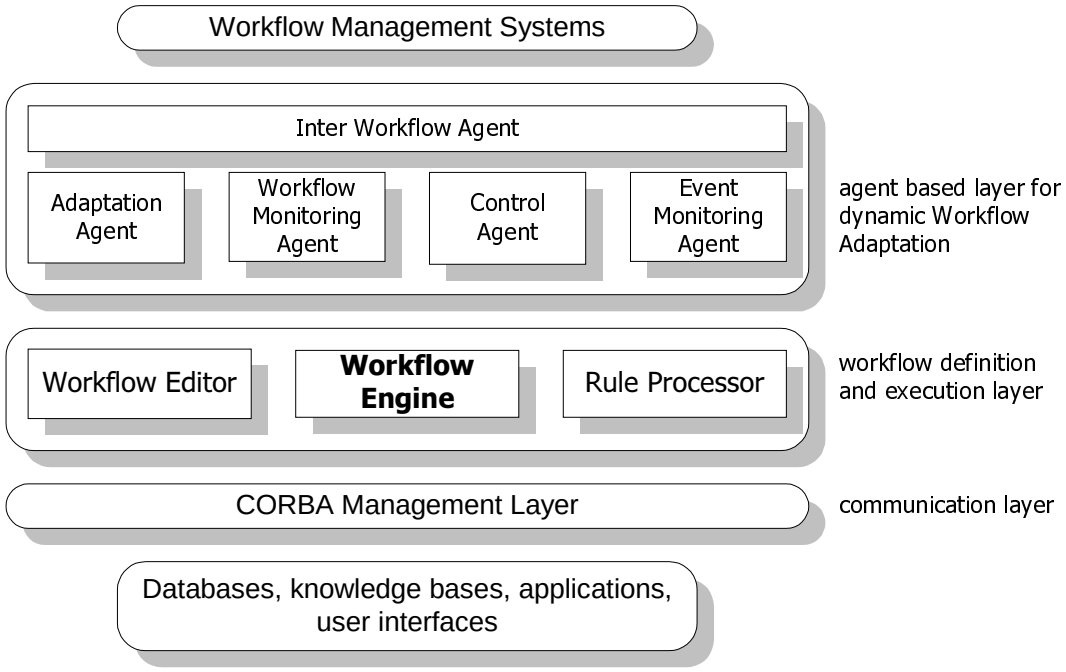


Abbildung 7 : Architektur des adaptiven Workflow-Management-Systems

2.2.1 Workflow-Definitions- und Ausführungsschicht

Die Workflow-Definitions- und Ausführungsschicht besteht aus der *Workflow Engine*, dem *Workflow Editor*, dem *Rule Processor* und der *Database Access Layer*. Die Database Access Layer ist eine Komponente, die keiner bestimmten Schicht des Workflow-Management-System zugeordnet werden kann, weil sie allen Teilen des Workflow-Management-Systems Zugriff auf die Daten der Workflow-Definitionen und Workflow-Instanzen bietet. Da die Database Access Layer im Moment nur von der Workflow Engine und dem Workflow Editor eingesetzt wird, wird die Komponente an dieser Stelle beschrieben. Dieser Teil des Workflow-Management-Systems ist Schwerpunkt der vorliegenden Diplomarbeit und deswegen soll hier nur ein kurzer Überblick über die Teile der Workflow-Definitions- und Ausführungsschicht gegeben werden. In den folgenden Kapiteln werden die Workflow Engine und die Database Access Layer im Detail beschrieben.

2.2.1.1 Workflow Editor

Der Workflow Editor ist ein graphisches Werkzeug, dessen Aufgabe in der Modellierung der Workflow-Definitionen, der Administration von Workflow-Teilnehmern und der Verwaltung von Anwendungsprogrammen besteht. Zur Sicherstellung der Korrektheit der Workflow-Definitionen bietet der Workflow Editor die Möglichkeit die Workflow-Definitionen zu verifizieren. Alle Daten, die der Workflow Editor verwaltet, werden in der Buildtime-Datenbank gesichert, über die auch die anderen Komponenten des Workflow-Management-Systems an diese Daten gelangen können.

Die Konzeption und prototypische Implementierung des Workflow Editor ist Schwerpunkt der Diplomarbeit [BOE00].

2.2.1.2 Workflow Engine

Die Funktion der Workflow Engine besteht darin, Workflow-Instanzen zu erzeugen, zu verwalten und auszuführen. Während der Ausführung einer Workflow-Instanz besteht die Aufgabe der Workflow Engine darin, den Kontrollfluss abzuarbeiten. Die hier vorgestellte Workflow Engine zeichnet sich dadurch aus, dass sie die Ausführung einer Workflow-Instanz an einer beliebigen Stelle unterbrechen und zu einem späteren Zeitpunkt fortsetzten kann.

Während der Ausführung einer Workflow-Instanz arbeitet die Workflow Engine eng mit der Database Access Layer, der CORBA Management Layer, den Workflow Clients und den Agenten der agentenbasierten Schicht zusammen. Die Database Access Layer stellt, wie bereits beschrieben, die Schnittstelle zu den Datenbanken bereit und die CORBA Management Layer wird benötigt, um Workflow Daten bereitzustellen und Anwendungsprogramme auszuführen. Die Workflow Clients sind die Schnittstelle zu den Workflow-Teilnehmern, mittels der die Benutzer über anstehende Aktivitäten informiert werden und Aktivitäten gestartet werden können. Die Agenten der agentenbasierten Schicht sind die Schnittstelle zum Adaptationssystem, mittels derer die Workflow-Instanzen unterbrochen und fortgesetzt werden können und Informationen für die Adaptation bereitgestellt werden.

2.2.1.3 Database Access Layer

Die Database Access Layer ist die Komponente des Workflow-Management-Systems, die dem Workflow Editor und der Workflow Engine eine Schnittstelle zum Datenbanksystem anbietet, in der die Workflow-Definitionen und auch die Workflow-Instanzen gesichert werden. Die Workflow-Definitionen und Workflow-Instanzen werden in zwei getrennten Datenbanken (Buildtime- und Runtime-Datenbank) gespeichert, um die Persistenz zu gewährleisten. Da für das Projekt AGENTWORK auf ein relationales Datenbanksystem (IBM DB2 Version 6.1) zurückgegriffen wurde und es einen konzeptionellen Unterschied zwischen objektorientierter Programmiersprache und relationalem Datenbanksystem gibt, wird eine Komponente benötigt, die den erwähnten *impedance mismatch* überwindet. Die Database Access Layer bietet diese Möglichkeit und ermöglicht außerdem den Zugriff auf Workflow-Definitionen und Workflow-Instanzen, ohne Informationen über das Datenbanksystem und deren Speicherstrukturen zu besitzen. Damit wird eine höhere Abstraktion zwischen Workflow-Management-System und zugrundeliegenden Speichersystem erreicht.

Normalerweise wird die CORBA Management Layer (siehe Abschnitt 2.2.3 "CORBA Management Layer") dazu verwendet, um an Daten aus einer externen Datenquelle, wie einer relationalen Datenbank, zu gelangen. Man hätte diesen Mechanismus auch dazu nutzen können, um auf die Buildtime- und Runtime-Datenbank zuzugreifen und damit an die Workflow-Definitionen und Workflow-Instanzen zu gelangen. Da die CORBA Management Layer zur Zeit noch nicht implementiert ist, muss für die Workflow Engine und den Workflow Editor eine solche Komponente entwickelt werden. Zudem ist die Database Access Layer speziell für den Zugriff auf die Buildtime- und Runtime-Datenbank optimiert und bietet somit einen schnelleren Zugriff auf die Datenbanken. Die CORBA Management Layer ist konzipiert, um auf beliebige Datenquellen zuzugreifen. Die Überwindung der Heterogenität bei den Datenquellen führt zwangsläufig zu einem zusätzlichen Overhead beim Zugriff auf die Datenquellen. Aus diesem Grund ist es sinnvoll, den Zugriff auf die Buildtime- und Runtime-Datenbank in der Database Access Layer zu implementieren.

Die Database Access Layer wurde im Rahmen dieser Diplomarbeit implementiert. Auf deren Architektur und die prototypische Implementierung wird im Kapitel 6 "Database Access Layer" genauer eingegangen.

2.2.1.4 Rule Processor

Der Regel-Interpreter (Rule Processor) wird für die Auswertung von Bedingungen gebraucht. Diese Bedingungen sind F-Logic Konstrukte, die auch vom F-Logic System ausgewertet werden. Zu diesem Zweck wird das F-Logic System in das Workflow-Management-System eingebunden.

2.2.2 Agentenbasierte Schicht

Die wichtigste Änderung bezüglich des allgemeinen Architekturmodells ist die agentenbasierte Schicht, deren Aufgabe die Erkennung von logischen Fehlern und die Adaptation der betroffenen Workflow-Instanzen ist.

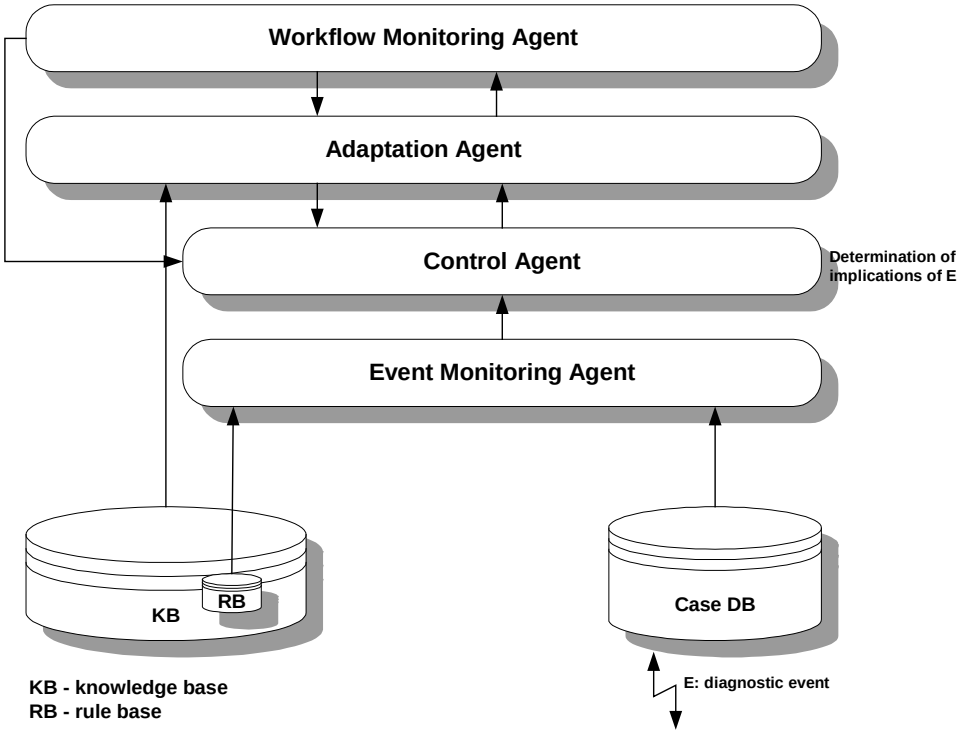


Abbildung 8 : Adaptations Agenten

Die verschiedenen Dienste dieser Schicht sind als eigenständige Agenten realisiert, die miteinander kommunizieren.

2.2.2.1 Event Monitoring Agent

Ausgangspunkt für die Adaptation von Workflow-Instanzen ist die Auslösung von Ereignissen, die einen logischen Fehler implizieren. Die Funktion des *Event Monitoring Agents* ist es, abzuprüfen, ob ein bestimmtes Ereignis einen logischen Fehler auslöst.

Ein solches Ereignis im medizinischen Anwendungsbereich wäre zum Beispiel, wenn bei einem Patienten die Anzahl der Leukozyten einen bestimmten Grenzwert überschreitet.

Der Agent besitzt zu diesem Zweck eine Regelbasis, in der für die logischen Fehler entsprechende Regeln existieren. Diese Regeln sagen aus, welche *Kontrollaktionen* auszuführen sind und für welchen Zeitraum.

Beispiel:

// Regel, die das Medikament ˝Etoposid˝ für die nächsten zwei Tage absetzt, wenn die
// Anzahl der Leukozyten unter 1500/mm³ fällt

$drop(D)@[0,2\ days] \leftarrow$
$L{:}hematological\text{-}finding[pat \rightarrow P, parameter \rightarrow leukozyte\text{-}count, unit \rightarrow \#/mm^3,$
$value \rightarrow V] \wedge V < 1500 \wedge D{:}drug\text{-}application[pat \rightarrow P', drug\text{-}name \rightarrow \text{“Etoposid“}] \wedge$
$P' == P$

Insgesamt gibt es die fünf Kontrollaktionen *add*, *drop*, *replace*, *check* und *delay*, die im Falle eines logischen Fehlers ausgelöst werden können.

Die Form dieser Kontrollaktionen sieht folgendermaßen aus:

Kontrollaktion(C, A[,B])@T

C steht für den Fall, der von dem logischen Fehler betroffen ist und bestimmt damit eindeutig die Workflow-Instanzen, die adaptiert werden müssen. *A* und *B* sind Aktivitäten, die von der Kontrollaktion betroffen sind und *T* das dazugehörige Zeitintervall, das durch das *@* Symbol vom Rest der Kontrollaktion getrennt wird. Mit Hilfe des Zeitintervalls T kann eine Kontrollaktion zeitlich begrenzt werden, um auf temporäre Bedingungen zu reagieren. Im Fall, dass die Bedingungen nicht temporär sind, kann das Zeitintervall auch unendlich sein. Dann gelten die Änderungen bis zum Ende des Workflows.

In der folgenden Tabelle werden die verschiedenen Kontrollaktionen zusammengefasst und eine kurze Beschreibung der Funktion gegeben.

Kontrollaktion	**Beschreibung**
add(C, A)@T	Für den Fall C soll innerhalb des Zeitraums T die Aktivität A eingefügt werden.
drop(C, A)@T	Für den Fall C sollen innerhalb des Zeitraums T alle Aktivitäten A entfernt werden.
replace(C, A, B)@T	Für den Fall C sollen innerhalb des Zeitraums T alle Aktivitäten A durch die Aktivität B ersetzt werden.
check(C, A)@T	Für den Fall C soll innerhalb des Zeitraums T jedes Mal beim erreichen der Aktivität A der Benutzer befragt werden, ob die Aktivität noch adäquat ist.
delay(C, A, t)@T	Für den Fall C soll innerhalb des Zeitraums T die Ausführung aller Aktivitäten A um die Zeitspanne t verzögert werden.

Tabelle 2 : Kontroll-Aktionen

Wenn ein Event ausgelöst wird, dann informiert die CORBA Management Layer den Event Monitoring Agent über dieses Ereignis. Anschließend wird das Ereignis gegen die Regelbasis des Agenten abgeprüft und getestet, ob eine der Regel ausgelöst wird. In diesem Fall wird der Control Agent aufgerufen, um die betroffenen Workflow-Instanzen anzuhalten. Anschließend kann der Adaptation Agent die Workflow-Instanzen adaptieren, die von dieser Regel betroffen sind.

2.2.2.2 Control Agent

Wenn durch den Event Monitoring Agent erkannt worden ist, dass ein logischer Fehler vorliegt, wird der *Control Agent* gestartet. Der Control Agent erhält die erforderlichen Kontrollaktionen, anhand derer er entscheidet, welche Workflow-Instanzen zu unterbrechen sind.

Nachdem die Workflow-Instanzen identifiziert sind, wird die Workflow Engine vom Control Agent aufgefordert diese anzuhalten und der Adaptation Agent wird gestartet und mit den erforderlichen Kontrollaktionen versorgt, um die notwendigen Modifikationen an den Workflow-Instanzen vorzunehmen. Wenn an einer Workflow-Instanz alle Änderungen vorgenommen wurden, wird der Control Agent informiert. Der benachrichtigt wiederum die Workflow Engine, dass die Ausführung der Workflow-Instanz fortgesetzt werden kann.

2.2.2.3 Adaptation Agent

Im Fall, dass während der Ausführung eines Workflows ein logischer Fehler eingetreten ist, wird der *Adaptation Agent* gestartet. Während des Aufrufs des Agenten werden die Kontrollaktionen übergeben, die der logische Fehler ausgelöst hat. Ziel des Adaptation Agent ist es, die betroffenen Workflow-Instanzen entsprechend den Kontrollaktionen so umzubauen, dass die Workflow-Instanzen den veränderten Bedingungen angepasst werden.

Für die Adaptation eines Workflows gibt es zwei mögliche Strategien, eine *reaktive* und eine *prädikative* Strategie.

Die Idee der reaktiven Strategie ist es, jede Änderungen erst dann auszuführen, wenn eine davon betroffene Aktivität zur Ausführung ansteht. Falls während der Ausführung einer Workflow-Instanz ein logischer Fehler auftritt, der die Workflow-Instanz betrifft, dann wird die Workflow-Instanz nicht sofort unterbrochen, sondern nur die weitere Ausführung der Workflow-Instanz überwacht. Erst wenn der Kontrollfluss der betroffenen Workflow-Instanz eine Aktivität erreicht, die von einer der Kontrollaktionen berührt wird, wird der Control Agent und der Adaptation Agent informiert. Anschließend wird die Workflow-Instanz vom Control Agent unterbrochen und der Adaptation Agent kann die Änderungen der Kontrollaktion an der Aktivität vornehmen. Wenn alle Änderungen an der Aktivität durchgeführt wurden, lässt der Control Agent die Workflow-Instanz fortfahren. Jedes Mal, wenn eine Aktivität vor der Ausführung ansteht, die von einer der Kontrollaktionen betroffen ist, wird dies wiederholt. Diese Strategie bietet sich besonders dann an, wenn eine Kontrollaktion statt eines Zeitintervalls eine *wertbasierte Bedingung*[4] enthält oder die Abschätzung der Region des Workflows, die

[4] Form: x > Wert

während des Zeitintervalls durchlaufen wird, nur sehr ungenau ist. In diesen Fällen ist es besser für jede betroffene Aktivität zu entscheiden, ob eine Adaptation notwendig ist oder nicht. Der Nachteil dieser Strategie ist, dass durch die späte Adaptation Vorlaufzeiten für bestimmte Aktivitäten nicht gewährleistet werden können und somit die rechtzeitige Ausführung dieser Aktivitäten verhindert wird. Außerdem muss die Workflow-Instanz gegebenenfalls mehrfach unterbrochen werden, worunter die Performance des Workflow-Management-Systems leidet.

Die prädikative Strategie versucht hingegen alle Änderungen sofort auszuführen und damit im Idealfall die Workflow-Instanz nur ein einziges Mal zu unterbrechen. Wenn während der Ausführung einer Workflow-Instanz ein logischer Fehler auftritt, dann wird die Workflow-Instanz vom Control Agent sofort unterbrochen. Der Adaptation Agent versucht dann als erstes die Region des Workflows zu bestimmen, die innerhalb des angegebenen Zeitintervalls, voraussichtlich durchlaufen wird. Diese Region wird auch als *Adaptationsregion* bezeichnet. Innerhalb dieser Region werden alle Aktivitäten bezüglich der Kontrollaktionen untersucht und die notwendigen Adaptationen durchgeführt. Abschließend wird der Workflow Monitoring Agent informiert und damit beauftragt, die weitere Ausführung der Workflow-Instanz zu überwachen. Für den Fall, dass die Ausführung der Workflow-Instanz langsamer oder schneller erfolgt, als durch den Adaptation Agent berechnet, wird die Workflow-Instanz erneut unterbrochen und die Adaptation an die geänderte Adaptationsregion angepasst. Dies kann bedeuten, dass Änderungen wieder rückgängig gemacht werden müssen oder zusätzliche Änderungen erforderlich sind. Der Vorteil dieser Strategie ist, dass im Idealfall die Workflow-Instanz nur ein einziges Mal unterbrochen werden muss und somit die Ausführung der Workflow-Instanz nur minimal beeinträchtigt wird. Dies funktioniert allerdings nur dann, wenn die Abschätzung der Adaptationsregion sehr genau ist.

Der Adaptation Agent unterstützt beide Strategien der Adaptation. Wenn statt eines Zeitintervalls eine wertbasierte Bedingung gegeben ist, dann wird die reaktive Strategie verfolgt, andernfalls wird immer zuerst die prädikative Strategie benutzt. Wenn sich allerdings während der Verarbeitung der Workflow-Instanz abzeichnet, dass die Abschätzungen der Adaptationsregion zu ungenau waren, dann ist es auch möglich die Strategie zu ändern und auch ohne wertbasierte Bedingung die reaktive Strategie einzusetzen.

Im Moment existiert eine prototypische Implementierung des Adaptation Agent, die untersucht, ob die vorgeschlagenen Algorithmen zur Bestimmung der Adaptations-

region und zur Modifikation der Workflow-Instanzen funktionieren. Für eine detailliertere Diskussion dieses Agenten wird auf [GRE00] und [MUE00] verwiesen.

2.2.2.4 Workflow Monitoring Agent

Die Aufgabe des Workflow Monitoring Agent besteht darin, die Ausführung Workflow-Instanzen zu überwachen, die von logischen Fehlern betroffen sind. Im Fall der prädikativen Adaptations-Strategie ist es die Aufgabe des Workflow Monitoring Agents festzustellen, ob die Abschätzung der Adaptationsregion richtig war. Falls die Abschätzung nicht korrekt war, dann muss der Workflow Monitoring Agent den Adaptation Agent und den Control Agent darüber informieren, dass die Ausführung der Workflow-Instanz schneller oder langsamer als berechnet war. Der Control Agent stoppt die Workflow-Instanz und der Adaptation Agent passt die Workflow-Instanz an die geänderte Situation an, die dadurch entstanden ist, dass die Ausführung des Workflows schneller oder langsamer war.

Die Überwachung der Workflow-Instanzen kann vom Workflow Monitoring Agent auf zwei verschiedene Methoden erfolgen, durch eine *aktive* oder eine *passive Überwachung*.

Die aktive Überwachung bedeutet, dass der Workflow Monitoring Agent vor der Ausführung jeder Aktivität von der Workflow Engine informiert wird. Zusammen mit den Informationen über die Adaptationsregion kann der Workflow Monitoring Agent dann entscheiden, ob die Berechung der Adaptationsregion korrekt war oder nicht.

Die passive Überwachung beruht darauf, dass bei der Adaptation spezielle Knoten in die Workflow-Definition eingefügt werden, die die Adaptationsregion bestimmen und als Checkpoints funktionieren. Wenn die Workflow Engine bei der Verarbeitung des Kontrollflusses an einen solchen Knoten kommt, dann wird der Workflow Monitoring Agent benachrichtigt. Der Workflow Monitoring Agent kann nun beurteilen, ob die Berechnung der Adaptationsregion richtig war oder nicht.

2.2.2.5 Inter Workflow Agent

Bei der Adaptation einer Workflow-Instanz ist zu beachten, dass verschiedene Workflow-Instanzen Abhängigkeiten untereinander haben [MR00]. Dies bedeutet,

dass Änderungen in einer Workflow-Instanz zu Änderungen in den abhängigen Workflow-Instanzen führen können. Für diese Abhängigkeitsbeziehungen zwischen Workflow-Instanzen ist der *Inter Workflow Agent* konzipiert. Die Kommunikation zwischen den verschiedenen Workflow-Instanzen erfolgt über sogenannte Kommunikationsknoten, die Daten zwischen den Workflow-Instanzen austauschen. Die Kommunikation zwischen den verschiedenen Workflow-Instanzen ist dabei nicht zwangsweise auf ein Workflow-Management-System begrenzt. Es ist durchaus vorstellbar, dass die Workflow-Instanzen auf verschiedenen Workflow-Management-Systemen laufen.

2.2.3 CORBA Management Layer

Die unterste Schicht in der Architektur des adaptiven Workflow-Management-Systems ist die sogenannte *CORBA Management Layer*. Die wichtigste Funktion der CORBA Management Layer ist die Integration aller Daten, die vom Workflow-Management-System zur Definition und Ausführung von Workflows benötigt werden. Dazu verwaltet die CORBA Management Layer alle entsprechenden Ressourcen. Die Ressourcen sind einerseits die Anwendungsprogramme und andererseits die Daten des Datenflusses. Die verwalteten Ressourcen können verteilt in einem Netzwerk existieren, so dass es notwendig ist, eine Middleware einzusetzen. Im Fall der CORBA Management Layer wurde *CORBA* (Common Object Request Broker Architecture) als ein Standard gewählt. CORBA hat den Vorteil, dass es von einer unabhängigen Organisation, der Object Management Group [OMG99], entwickelt wird und von einer großen Anzahl von Unternehmen unterstützt wird. Im Gegensatz zu den proprietären Lösungen, wie *DCOM* oder *MQSeries*, macht man sich so nicht von einem einzelnen Hersteller und dessen Entwicklungen abhängig.

Durch den Einsatz von CORBA ergibt sich, dass alle Ressourcen, wie Anwendungsprogramme und Daten, CORBA-Objekte sind. Die Definition der Daten und die Implementierung der Anwendungsprogramme verbirgt sich in den jeweiligen CORBA-Klassen. Jeder CORBA-Klasse ist eine F-Logic-Klasse zugeordnet, die diese Ressource auf den oberen Schichten des Workflow-Management-Systems repräsentiert. Dadurch können die oberen Schichten des Workflow-Management-Systems auf einem einheitlichen Datenmodell arbeiten und können somit von den variierenden Implementierungen der Ressourcen abstrahieren. Die Abbildung zwischen F-Logic-Klassen und CORBA-Klassen wird von der CORBA Management Layer übernommen.

Neben der Datenintegration hat die CORBA Management Layer eine weitere wichtige Funktion, nämlich die Auslösung und Weiterleitung von Ereignissen. Diese Ereignisse bilden die Grundlage für die Erkennung von logischen Fehlern und ist entscheidend für die dynamische Adaptation der Workflows.

Gegenwärtig existiert die CORBA Management Layer nur als Konzept und eine Implementierung steht zur Zeit noch aus.

2.3 Beispiel-Workflow

In diesem Abschnitt wird ein Beispiel-Workflow vorgestellt, der in der Folge zur Erläuterung bestimmter Konzepte verwendet werden soll. Der Workflow ist in Hinblick auf den medizinischen Kontext des Projektes AGENTWORK aus dem onkologischen Anwendungsbereich gewählt. Der Workflow soll zum Verständnis des Kontrollflusses dienen und die Verarbeitung von Workflows verdeutlichen.

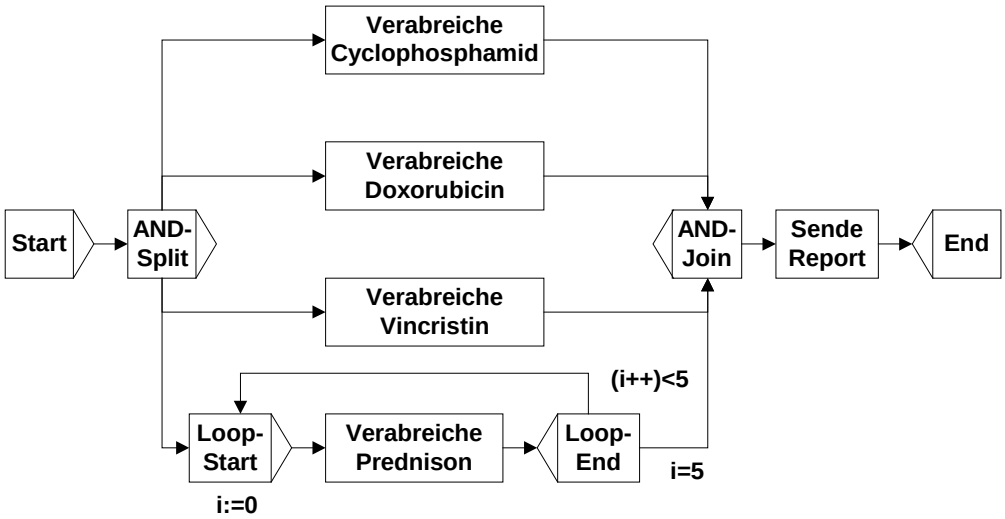


Abbildung 9 : Beispiel Workflow (CHOP 14)

Der dargestellte Workflow zeigt eine Chemo-Therapie (CHOP 14), die bei der Behandlung von Krebs zum Einsatz kommt. Nach dem Start verzweigt der Workflow in vier parallele Pfade. Jeder der Pfade beinhaltet die Verabreichung eines bestimmten Medikaments. Die oberen drei Pfade der Verzweigung verabreichen das jeweilige Medikament nur ein Mal. Der unterste Pfad der Verzweigung verabreicht das Medika-

ment hingegen sechs Mal. Dies geschieht innerhalb einer Schleife, die sechs Mal durchlaufen wird. Die Pfade der Verzweigung, die bereits zum Ende gekommen sind, warten auf das Ende des letzten Pfades. Nachdem alle Pfade der Verzweigung beendet wurden ist auch die Verzweigung beendet und der Workflow kann fortgesetzt werden. Zum Abschluss der Chemo-Therapie wird noch ein abschließender Report erzeugt, bevor der Workflow schließlich auch zum Ende kommt.

3 Workflow Buildtime

Die Aufgabe der Buildtime-Komponente des Workflow-Management-Systems besteht in der Definition, dem Test und der Verwaltung aller workflow-relevanten Informationen.

Die Informationen werden hierbei in drei Bereiche gegliedert:

1. Aktivitäten, die in Workflows verwendet werden
2. Informationen über Organisationsstrukturen
3. Workflow-Definitionen (Kontrollfluss und Datenfluss)

Der Workflow Editor stellt eine graphische Oberfläche zur Verfügung, um diese Informationen zu erfassen und zu verwalten. Zur persistenten Speicherung alle Daten werden diese in der Buildtime-Datenbank gesichert. Die Buildtime-Datenbank stellt sogleich die Schnittstelle zur Runtime-Komponente des Workflow-Management-Systems dar. Für weitergehende Informationen zum Aufbau und der Funktionsweise des Workflow Editors siehe [RBO00].

3.1 Buildtime-Modell

Das Buildtime-Modell stellt die Basis des gesamten Workflow-Management-Systems dar. Das Modell beschreibt, wie ein Workflow definiert ist und welche Konstrukte des Kontroll- und Datenflusses zur Modellierung eines Workflows existieren. Damit wird die Funktionsweise des gesamten Workflow-Management-Systems festgelegt. Aus diesem Grunde wurde das Buildtime-Modell in Zusammenarbeit mit den Verfassern der Diplomarbeiten [GRE00, BOE00] und der Leitung des Projekts AGENTWORK [MUE00] spezifiziert.

In diesem Abschnitt soll das Modell aus Sicht des Modellierers eines Workflows beschrieben werden. Auf die Umsetzung des Buildtime-Modells in ein UML-Modell und die praktische Implementierung in Form von C++ Klassen wird im Kapitel 7 "Implementierung" genauer eingegangen.

3.1.1 Aktivitäts-Definitionen und Anwendungsprogramme

Eine *Aktivitäts-Definition* ist die Beschreibung eines einzelnen Arbeitsschrittes. Die Aktivitäts-Definition wird immer im Zusammenhang mit einem *Aktivitätsknoten*[5] innerhalb einer Workflow-Definition verwendet, wobei es möglich ist, dass die Aktivitäts-Definition von verschiedenen Aktivitätsknoten in unterschiedlichen Workflow-Definitionen verwendet wird.

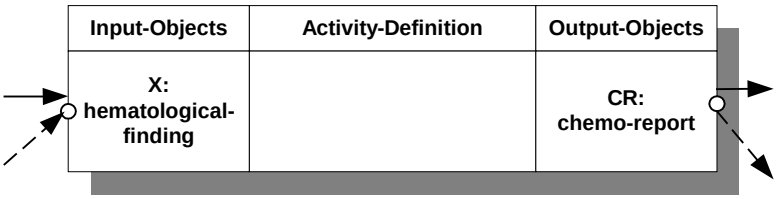


Abbildung 10 : Aktivitäts-Definition

Alle Aktivitäts-Definitionen haben gemein, dass sie über Input-Objekte und Output-Objekte verfügen, die angeben, welche Objekte von der Aktivität benötigt werden und welche Objekte von der Aktivität zurückgegeben werden. Die Input- und Output-Objekte werden durch *Objekt-Spezifikationen*[6] definiert.

So spezifiziert das Input-Objekt der Aktivitäts-Definition **X: hematological-finding**, dass ein Objekt **X** von der Klasse hematological-finding als Input erwartet wird.

Die Aktivitäts-Definitionen werden nach der Art der Aktivität in *einfache* und *komplexe Aktivitäts-Definitionen* unterschieden.

3.1.1.1 Einfache Aktivitäts-Definitionen

Eine *einfache Aktivitäts-Definition* besteht aus einer einzelnen Aktivität, die manuell oder automatisch ausgeführt werden kann.

[5] Aktivitätsknoten werden im Abschnitt 3.1.3.1 "Kontrollfluss" beschrieben

[6] Objekt-Spezifikationen werden im Abschnitt 3.1.3.2 "Datenfluss" erörtert

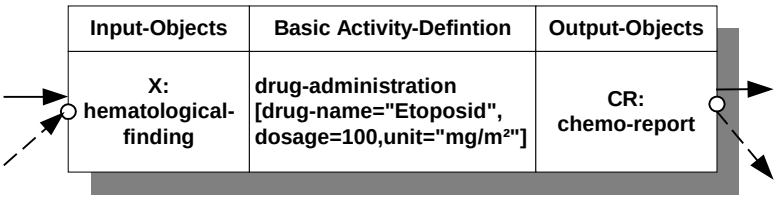


Abbildung 11 : Einfache Aktivitäts-Definition

Jeder einfachen Aktivitäts-Definition wird eine Objekt-Spezifikation zugeordnet, die die Aktivität beschreibt. Durch die F-Logic Modellierung der Objekt-Spezifikation ist der Adaptation Agent in der Lage, die Aufgabe der Aktivität festzustellen und die Aktivität entsprechend den Regeln zu modifizieren.

Beispiel:

```
// Die Aktivitäts-Definition beschreibt eine Medikamentengabe, wobei dem Patienten das
// Medikament "Etoposid" in der Dosierung 100 mg/m² verabreicht wird.

drug-administration[drug-name="Etoposid", dosage=100, unit="mg/m²"]
```

Jeder einfachen Aktivitäts-Definition ist eine Liste von Anwendungsprogramme zugeordnet, die zur Abarbeitung der Aktivität benötigt werden. Bei der Ausführung der Aktivitäts-Definition unterscheidet man zwischen *manuellen* und *automatischen Aktivitäten*. Die Art der Aktivitäts-Definition ist davon abhängig, ob die Aktivität eine Interaktion mit dem Nutzer braucht. Wenn keine Nutzerinteraktion nötig ist, dann kann die Aktivität automatisch ausgeführt werden. Andernfalls wird die Aktivität manuell ausgeführt, wobei ein Nutzer für die Ausführung der Aktivität verantwortlich ist.

Außerdem muss die einfache Aktivitäts-Definition den Datenfluss von und zu den Anwendungsprogramme definieren. Dazu werden die Input-Objekte der Aktivitäts-Definition auf die Input-Objekte der Anwendungsprogramme und die Output-Objekte der Anwendungsprogramme auf die Output-Objekte der Aktivitäts-Definition gemappt. Die Abbildung zwischen Aktivitäts-Definition und Anwendungsprogramm erfolgt durch Komponenten-Zuweisungen[7].

[7] Komponenten-Zuweisungen werden im Abschnitt 3.1.3.2 "Datenfluss" erörtert

3.1.1.2 Komplexe Aktivitäts-Definitionen

Eine *komplexe Aktivitäts-Definition* beinhaltet einen *Subworkflow*, der bei Erreichen des zugehörigen Aktivitätsknotens ausgeführt wird. Ein Subworkflow selbst ist eine Workflow-Definition, die unter Umständen auch eigenständig ausgeführt werden kann. Der Subworkflow einer komplexen Aktivitäts-Definition kann seinerseits wieder aus komplexen Aktivitäts-Definitionen bestehen, so dass eine Verschachtelung der Aktivitäts-Definitionen möglich ist.

Der Subworkflow kennt keine Input- und Output-Objekte, wie die Aktivitäts-Definition. Aus diesem Grunde werden Input- und Output-Objekte der Aktivitäts-Definition auf globale Objekte der Workflow-Definition gemappt. Über die globalen Objekte können die Aktivitäten des Subworkflows auf die Input-Objekte zugreifen und die Output-Objekte mit Daten füllen.

Die Verarbeitung der komplexen Aktivitäts-Definition wird als eine Aktivität aufgefasst, das heißt wenn der Subworkflow vollständig abgearbeitet ist, ist die Aktivität und damit auch der Aktivitätsknoten beendet.

3.1.1.3 Anwendungsprogramme

Das Workflow-Management-System benötigt *Anwendungsprogramme*, um einen bestimmten Geschäftsprozess zu realisieren. Dies bedeutet, dass zu jeder einfachen Aktivität eine Reihe von Anwendungsprogrammen assoziiert sind, die ausgeführt werden, wenn die Aktivität gestartet wird. Dabei unterscheidet man zwischen *automatischen* und *manuellen Aktivitäten*.

- Automatische Aktivitäten bedeuten, dass die assoziierten Anwendungsprogramme ohne die Interaktion mit einem Nutzer ausgeführt werden. Der verantwortliche Nutzer wird nur dann informiert, wenn ein Anwendungsprogramm nicht korrekt beendet wurde.

- Manuelle Aktivitäten hingegen bedeuten, dass keine Anwendungsprogramme assoziiert sind oder die assoziierten Anwendungsprogramme eine Nutzerinteraktion benötigen. Der Nutzer erhält in jedem Fall einen Eintrag in seine

Worklist. Wenn keine Anwendungsprogramme zu einer Aktivität assoziiert sind, dann muss der Nutzer die Aktivität von Hand ausführen und bestätigt das Ende der Aktivität, indem er den Eintrag in der Worklist auswählt. Im anderen Fall werden die Anwendungsprogramme gestartet, wenn der Nutzer den Eintrag in der Worklist auswählt, und sind dann zu Ende, wenn alle Anwendungsprogramme beendet wurden.

Die Anwendungsprogramme selbst können dabei noch nach ihren Interaktionsmöglichkeiten in verschiedene Klassen eingeordnet werden.

Typ	**Beschreibung**
(1)	Anwendungsprogramme, die nach dem Start nicht mehr vom Workflow-Management-System beeinflusst werden können, das heißt weder der Status des Anwendungsprogramms ist abrufbar, noch kann das Anwendungsprogramm an einem 2-Phasen-Commit Protokoll teilnehmen.
(2)	Anwendungsprogramme, die nach ihrem Ende eine Rückmeldung über den Status des Anwendungsprogramms liefern, aber nicht in der Lage sind an einem 2-Phasen-Commit Protokoll teilzunehmen.
(3)	Anwendungsprogramme, die über eine API zu jeder Zeit der Ausführung über den Status der Verarbeitung Auskunft geben können, aber nicht an einem 2-Phasen-Commit Protokoll teilnehmen.
(4)	Anwendungsprogramme, die ein 2-Phasen-Commit Protokoll unterstützen und somit zu jeder Zeit zurückgefahren werden können, wenn ein Workflow zurückgesetzt werden muss.

Tabelle 3 : Anwendungsprogrammstypen

Die Anwendungsprogramme können auf verschiedenen Rechnern und Plattformen gespeichert und ausgeführt werden. Die rechner- und plattformübergreifende Kommunikation zwischen Workflow-Management-System und Anwendungsprogrammen erfordert den Einsatz einer Middleware, in diesem Fall CORBA. Aus diesem Grund

werden die Anwendungsprogramme über eine CORBA-Schnittstelle angesteuert. Es wird davon ausgegangen, dass alle Anwendungsprogramme diese Schnittstelle unterstützen. Die Anwendungsprogramme, die diese Schnittstelle nicht selbst implementieren, müssen diese CORBA-Schnittstelle über den Umweg eines Wrappers bereitstellen. Der Wrapper implementiert dann die CORBA-Schnittstelle und steuert das Anwendungsprogramm. Der Wrapper verbirgt die wahre Implementierung des Anwendungsprogramms vor der Middleware, so dass die Middleware eine einheitliche Schnittstelle zu allen Anwendungsprogrammen besitzt.

3.1.2 Organisationsmodell

Ein Workflow-Management-System kommt üblicherweise in Organisationen zum Einsatz, die traditionell eine hierarchische Struktur haben. Damit ein Workflow auf die Weise ausgeführt werden kann, wie der zugrundeliegende Geschäftsprozess, ist es notwendig diese hierarchischen Strukturen im Workflow-Management-System umzusetzen.

Für das Organisationsmodell wurde aus diesem Grunde ein rollenbasierter Ansatz gewählt. Das Modell sieht so aus, dass es eine Reihe von Rollen gibt, wie zum Beispiel Chefarzt oder Krankenschwester. Jeder Benutzer des Workflow-Management-Systems wird als Person verwaltet, die eine oder mehrere Rollen bekleidet. Außerdem besteht die Möglichkeit einer Rolle einen Stellvertreter zuzuordnen, wie zum Beispiel Assistenzarzt als Stellvertreter des Stationsarztes. Damit wird es möglich, die hierarchische Struktur einer Organisation im Workflow-Management-System zu modellieren.

Für die Ausführung eines Workflows bedeutet dieses Modell, dass jede Aktivität einer Rolle zugeordnet ist, die für deren Ausführung verantwortlich ist. Im Falle einer manuellen Aktivität muss die Aktivität von einer Person ausgeführt werden, die die verantwortliche Rolle bekleidet. Bei einer automatischen Aktivität hingegen wird erst dann eine Person der verantwortlichen Rolle informiert, wenn es zu einem Fehler bei der Ausführung der Aktivität gekommen ist.

Damit der Benutzer über die anstehenden Aktivitäten informiert werden kann, die er ausführen soll, muss er sich am Workflow-Management-System anmelden. Dadurch ist das Workflow-Management-System in der Lage festzustellen, wann und wo ein Nutzer arbeitet. Auf Basis dieser Informationen kann das System entscheiden,

welcher Nutzer eine anstehende Aktivität ausführen soll. Anschließend bekommt der Nutzer einen Eintrag in seine Worklist. Wenn der Nutzer dann diesen Eintrag in der Worklist auswählt, werden die notwendigen Anwendungsprogramme an seinem Arbeitsplatz gestartet, so dass er die Aktivitäten abarbeiten kann.

3.1.3 Workflow-Definition

Die Workflow-Definition setzt sich aus zwei Teilen zusammen, dem Kontrollfluss und dem Datenfluss. Der Kontrollfluss definiert die Abfolge der Arbeitsschritte innerhalb des Workflows, während der Datenfluss den Transport von Informationen von und zu den Arbeitsschritten beschreibt.

Eine Workflow-Definition kann sowohl als eigenständiger Workflow, als auch im Rahmen einer komplexen Aktivitäts-Definition als Subworkflow eines anderes Workflows ausgeführt werden. Auf diese Weise lassen sich Workflows ineinander verschachteln. Außerdem wird es möglich häufig verwendete Abläufe wiederzuverwenden, indem man sie als Subworkflows einbindet.

3.1.3.1 Kontrollfluss

Der Kontrollfluss kennt drei Arten von Knoten, *Kontrollflussknoten*, *Aktivitätsknoten* und *Kommunikationsknoten*. Kontrollflussknoten tragen nicht zur Ausführung des Workflows bei, sondern dienen nur zur Steuerung des Kontrollflusses. Kontrollflussknoten besitzen dabei selbst keinen Datenfluss, das heißt sie haben keine Input- und Output-Objekte. Zu den Kontrollflussknoten gehören die Start- und End-Knoten, die Split- und Join-Knoten und die LoopStart- und LoopEnd-Knoten.

START- UND END-KNOTEN

Jeder Workflow besitzt genau einen *Start-* und einen *End-Knoten*.

Zur Runtime wird die Verarbeitung des Workflows im Start-Knoten begonnen und wenn der End-Knoten erreicht ist, wird der Workflow beendet.

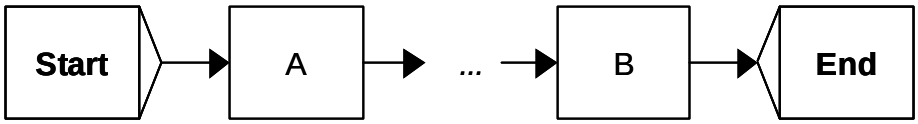


Abbildung 12 : Start- und End-Knoten Grafik von [BOE00]

LOOPSTART- UND LOOPEND-KNOTEN

LoopStart- und *LoopEnd-Knoten* werden verwendet, wenn eine Schleife in den Workflow eingefügt werden soll. Dabei umschließen die beiden Knoten die Aktivitäten, die wiederholt ausgeführt werden sollen. Wichtig ist dabei, dass die beiden Knoten immer ein Paar bilden und nicht einzeln auftreten. Der LoopEnd-Knoten besitzt zwei ausgehende Transitionen, von denen die eine zurück zum LoopStart-Knoten und die andere zum nächsten Knoten im Kontrollfluss geht. Die Transition, die zurück zum LoopStart-Knoten führt, besitzt eine wertbasierte Bedingung, die entscheidet, ob die Schleife ein nächstes mal ausgeführt wird oder nicht. Ist diese Bedingung erfüllt, so wird die Schleife ein weiteres mal durchlaufen. Sobald die Bedingung nicht mehr erfüllt ist, wird die Schleife verlassen und der Kontrollfluss fährt mit dem nächsten Knoten fort.

Eine Besonderheit ist der Datenfluss innerhalb der Schleife. Beim ersten Durchlauf der Schleife werden Datenfluss-Kanten von außerhalb der Schleife, aber dafür keine rückläufigen Datenfluss-Kanten berücksichtigt. Bei jedem weiteren Durchlauf der Schleife werden alle Datenfluss-Kanten von außerhalb ignoriert, dafür aber rückläufige Datenflusskanten verfolgt.

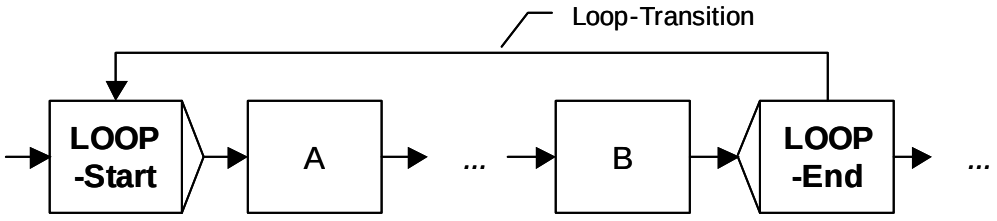


Abbildung 13 : LoopStart- und LoopEnd-Knoten Grafik von [BOE00]

SPLIT- UND JOIN-KNOTEN

Für Verzweigungen innerhalb des Kontrollflusses eines Workflows werden die *Split-* und *Join-Knoten* benutzt. Auch diese Knoten können nur als Paar verwendet werden, so dass zu jedem Split-Knoten immer genau ein Join-Knoten gehört. Es gibt verschiedene Arten von Split- bzw. Join-Knoten, so das es eine ganze Anzahl von Kombinationen zwischen Split- und Join-Knoten gibt. Diese sollen hier aber nicht diskutiert werden. Im folgenden werden dafür die einzelnen Knoten mit ihren Eigenschaften vorgestellt.

Knoten	**Beschreibung**
OR-Split	Dieser Knoten hat zwei oder mehr ausgehende Transitionen, von denen alle Transitionen, außer einer, eine Bedingung zugeordnet bekommen. Jede Transition wird nur dann aktiviert, wenn die Bedingung erfüllt ist. Die Transition ohne Bedingung, die sogenannte *Default-Transition*, wird dann aktiviert, wenn keine der anderen Transitionen aktiviert wurde. Damit ist gewährleistet, dass immer ein Kontrollpfad aktiviert wird.
AND-Split	Dieser Knoten hat zwei oder mehr ausgehende Transitionen. Hier ist es allerdings so, dass keine der Transitionen eine Bedingung erhält, so dass immer alle Transitionen aktiviert werden und damit mehrere parallele Kontrollpfade abgearbeitet werden.
ALL-Join	Dieser Knoten kann sowohl in Verbindung mit einem OR-Split, als auch mit einem AND-Split auftreten. Hierbei hat der Knoten mehr als zwei eingehende Transitionen. Der Knoten wird dabei aber erst dann aktiviert, wenn alle eingehenden Kanten aktiviert wurden. Bei der Verwendung des ALL-Join in Verbindung mit dem OR-Split werden nur die eingehenden Transitionen berücksichtigt, die vorher beim OR-Split aktiviert wurden, alle anderen Transitionen werden ignoriert.

ONE-Join-Cancel	Dieser Knoten kann auch wieder in Verbindung mit einem OR-Split oder AND-Split auftreten. Auch hat der Knoten wieder mehr als zwei eingehende Transitionen. Der Knoten wird allerdings bereits dann aktiviert, wenn eine der eingehenden Transitionen aktiviert wurde. Alle anderen Kontrollpfade, die parallel ausgeführt werden, werden gleichzeitig abgebrochen.
ONE-Join-Complete	Dieser Knoten verhält sich wie der ONE-Join-Cancel, allerdings werden die parallel abgearbeiteten Pfade nicht abgebrochen, sondern im Hintergrund noch bis zum erreichen des Join-Knoten abgearbeitet. Danach werden auch diese Kontrollpfade beendet.

Tabelle 4 : Split- und Join-Knoten

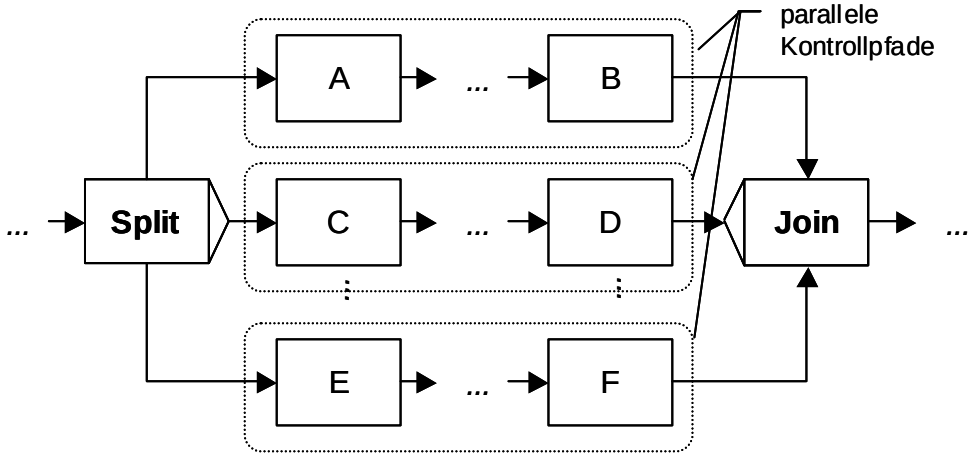


Abbildung 14 : Split- und Join-Knoten Grafik von [BOE00]

AKTIVITÄTSKNOTEN

Ein *Aktivitätsknoten* hat die Aufgabe eine Aktivität zu starten. Dem Aktivitätsknoten wird immer genau eine Aktivitäts-Definition zugeordnet, die entweder eine einfache Aktivitäts-Definition oder komplexe Aktivitäts-Definition darstellt. Im Fall einer einfachen Aktivitäts-Definition werden die Anwendungsprogramme, die der Aktivitäts-

Definition zugeordnet sind, gestartet. Bei einer komplexen Aktivitäts-Defintion hingegen wird ein kompletter Subworkflow als eine Aktivität gestartet.

Zusätzlich zu der Aktivität wird jedem Aktivitätsknoten eine Rolle zugeordnet. Diese Rolle ist dann zur Laufzeit für die Abarbeitung der Aktivität zuständig. Dies bedeutet, dass eine der zu der Rolle assoziierten Personen für die Aktivität verantwortlich ist.

KOMMUNIKATIONSKNOTEN

Ein *Kommunikationsknoten* hat die Aufgabe, die Verarbeitung des Workflows mit einer anderen Workflow-Instanz zu koordinieren. Die verschiedenen Workflow-Instanzen müssen dabei nicht zwangsläufig auf einem Workflow-Management-System laufen.

Die Kommunikation zu dem anderen Workflow-Instanzen wird mit Hilfe des Inter Workflow Agent realisiert. Der Agent übernimmt dabei vollständig die Koordination zwischen den Instanzen.

Es gibt drei Arten von Kommunikationsknoten:

Knoten	**Beschreibung**
Comm-In	Dieser Knoten wartet auf eine eingehende Kommunikation und der Kontrollfluss wird nicht fortgefahren, bis diese Kommunikation zwischen den Workflow-Instanzen geschehen ist.
Comm-Out	Dieser Knoten stellt eine Kommunikation zu einer anderen Workflow Instanz her und fährt im Kontrollfluss fort, sobald die Kommunikation zwischen den Workflow-Instanzen erfolgreich war.
Comm-Exc	Dieser Knoten wird dazu benutzt, um eine Aktivität in einer anderen Workflow-Instanz auszuführen. Der Knoten kann erst dann mit dem Kontrollfluss fortfahren, wenn die Aktivität in der anderen Workflow-Instanz zum Ende gekommen ist.

Tabelle 5 : Kommunikationsknoten

Damit ein Fehler in der Kommunikation zwischen den Workflow-Instanzen nicht dazu führt, dass der Workflow stehen bleibt, ist jedem Kommunikationsknoten eine Dauer zugeordnet. Nach Verstreichen der angegebenen Dauer wird der Verantwortliche für die Workflow-Instanz darüber informiert, dass die Kommunikation zu der anderen Workflow-Instanz nicht in der vorgesehenen Zeit stattgefunden hat. Damit ist es möglich den Workflow fortzufahren, auch wenn die Kommunikation zwischen den Workflow-Instanzen nicht zustande gekommen ist.

TRANSITIONEN

Transitionen haben die Aufgabe die Verbindung zwischen den Knoten herzustellen. Die Transition ist immer eine gerichtete Kante, wodurch die Reihenfolge, in der die Knoten durchlaufen werden, eindeutig festgelegt wird.

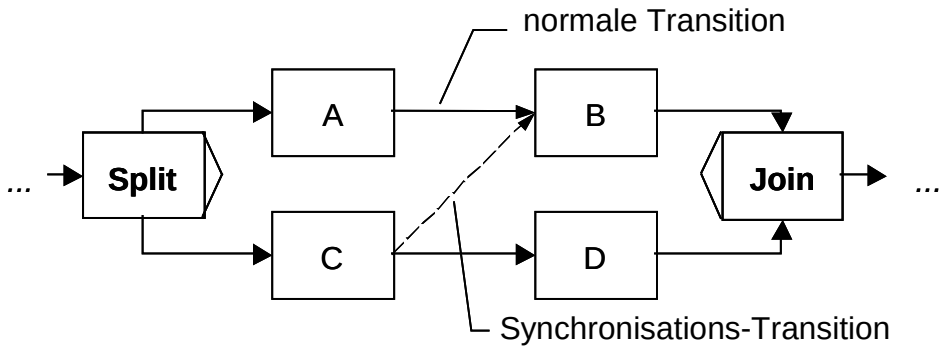


Abbildung 15 : Transitionen Grafik von [BOE00]

Das Workflow-Management-System unterstützt zwei Arten von Transitionen:

- **Normale Transitionen**

  Normale Transitionen definieren einen Kontrollpfad, das heißt eine sequentielle Folge von Knoten.

Eine normale Transition kann zwei Arten von Bedingungen beinhalten:

wertbasierte Bedingungen	Eine wertbasierte Bedingung bezieht sich auf den Wert eines Objektes des Datenflusses. Diese Art von Bedingung kann, ohne temporale Bedingung, nur für die ausgehenden Transitionen eines OR-Split-Knotens oder eines LoopEnd-Knotens verwendet werden, ansonsten muss zusätzlich eine temporale Bedingung definiert sein.
temporale Bedingungen	Eine temporale Bedingung bezieht sich auf die zeitliche Abfolge und definiert eine minimale bzw. maximale Verzögerung zwischen zwei Aktivitäten.

Eine Transition, der eine Bedingung beigeordnet ist, wird dann aktiviert, wenn die Bedingung erfüllt ist.

Neben den reinen wertbasierten bzw. temporalen Bedingungen existieren auch Bedingungen, die aus wertbasierten und temporalen Bedingungen zusammengesetzt sind. Transitionen mit einer solchen Bedingung werden dann aktiviert, wenn die minimale Wartezeit überschritten ist und zusätzlich die wertbasierte Bedingung erfüllt ist. Wenn die maximale Wartezeit erreicht ist und die wertbasierte Bedingung immer noch nicht erfüllt ist, wird eine Fehlermeldung ausgelöst.

Möchte man einer normalen Transition eine wertbasierte Bedingung zuordnen, so muss immer eine temporale Bedingung damit verbunden werden. Auf diese Weise wird verhindert, dass der Workflow an dieser Transition stehen bleibt, wenn die Bedingung nie erfüllt wird. Spätestens nach Ablauf der maximalen Wartezeit wird eine Fehlermeldung ausgelöst und der Workflow kann fortgefahren werden. Indem man die minimale Wartezeit auf *Null* setzt kann man erreichen, dass die Bedingung sich wie eine wertbasierte Bedingung verhält. Diese Einschränkung dient nur zur Sicherheit bei der Verarbeitung der Workflows.

- **Synchronisations-Transition**

  Die Verwendung von *Synchronisations-Transitionen* ist nur innerhalb paralleler Kontrollpfade einer *Split-Join-Region* möglich. Dabei übernimmt die Synchronisations-Transition die Aufgabe zwei parallele Knoten miteinander zu koordinieren. Auf diese Weise wird die Reihenfolge festgelegt, in der die parallelen Knoten abgearbeitet werden.

  Einer Synchronisations-Transition kann, im Gegensatz zu einer normalen Transition, keinerlei Bedingung zugeordnet werden. Dies ist damit zu begründen, dass die Synchronisations-Transition zur reinen Synchronisation zweier Knoten verwendet wird.

  Wichtig ist, dass weder der Quellknoten, noch der Zielknoten der Synchronisations-Transition innerhalb einer Schleife liegt. Dies würde keinen Sinn machen. Im Falle, dass der Quellknoten innerhalb der Schleife liegt, würde die Synchronisationskante schon beim ersten Durchlauf aktiviert und jeder weitere Durchlauf hätte keine Auswirkung auf die Abarbeitung des parallelen Kontrollpfades. Dieser könnte bereits nach dem ersten Durchlauf fortgefahren. Im anderen Fall, dass der Zielknoten in der Schleife liegt, würde die Synchronisationskante beim erreichen des Quellknoten aktiviert und die Schleife könnte abgearbeitet werden, als ob es die Synchronisationskante nicht gäbe.

3.1.3.2 Datenfluss

Die Modellierung des Datenflusses erfolgt vollständig objektorientiert und wird durch die Verwendung von F-Logic Klassen realisiert. Um die Verifikation des Datenflusses zur Buildtime zu erleichtern werden die F-Logic Konstrukte strukturiert abgelegt. Zur Darstellung der F-Logic Objekte werden *Objekt-Spezifikationen* und *Objekt-Pfade* verwendet.

Der Datenfluss einer Workflow-Definition stellt einen Graphen dar, in dem die Objekt-Spezifikationen die Knoten sind. Die Kanten in diesem Graphen sind die *Komponenten-Zuweisungen*. Nach der Art der Quelle und des Ziels einer Komponenten-Zuweisung unterscheidet man zwischen *internen* und *externen Datenfluss*.

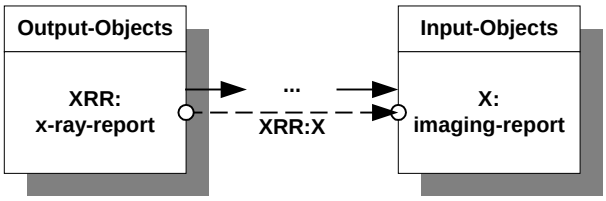


Abbildung 16 : Datenfluss

Zur Laufzeit werden die F-Logic Objekte an die CORBA Management Layer übergeben, die dann die physischen Daten für die Anwendungsprogramme bereitstellt.

Objekt-Spezifikationen

Objekt-Spezifikationen werden zur Beschreibung von Datenfluss und Aktivitäts-Definitionen verwendet.

Die Funktion der Objekt-Spezifikationen besteht darin, F-Logic Objekte durch *Namen* und *Klasse* zu definieren. Der Name des Objekts ist frei wählbar, hingegen muss die Klasse einer der bekannten F-Logic Klassen entsprechen.

Ein Sonderfall sind die *konstanten Objekt-Spezifikationen*, mit denen es möglich ist eine feste Zeichenkette oder Zahl zu definieren. Diese Art von Objekt-Spezifikation besitzt, zusätzlich zum Namen und der Klasse, einen Wert. Wenn bei der Definition kein Wert definiert wurde, dann muss bei der Initialisierung der Workflow-Instanz dieser Wert gesetzt werden. Dies geschieht durch die Interaktion mit dem Benutzer.

Beide Arten von Objekt-Spezifikationen werden auch für die Definition von globalen Objekten der Workflow-Definition verwendet. Während der Initialisierung der Workflow-Instanz müssen diese Objekt-Spezifikationen mit Daten gefüllt werden. Im Fall der globalen konstanten Objekt-Spezifikationen werden dazu die gegenebenen Werte verwendet bzw. die Werte durch Interaktion mit dem Benutzer beschafft. Für die anderen globalen Objekt-Spezifikationen müssen die Daten durch externe Datenzugriffe initialisiert werden.

Objekt-Pfade

Ein besonderer Aspekt des Datenflusses ist, dass der Strom an Daten zwischen zwei Aktivitäten nicht eins zu eins erfolgen muss. Das bedeutet, dass nicht zwangsläufig die Output-Objekte der Quellaktivität mit den Input-Objekten der Zielaktivität übereinstimmen. Vielmehr ist es möglich, einzelne Objekte oder Teile von Objekten aus der Quellaktivität zu extrahieren und Objekte in der Zielaktivität zusammenzufügen.

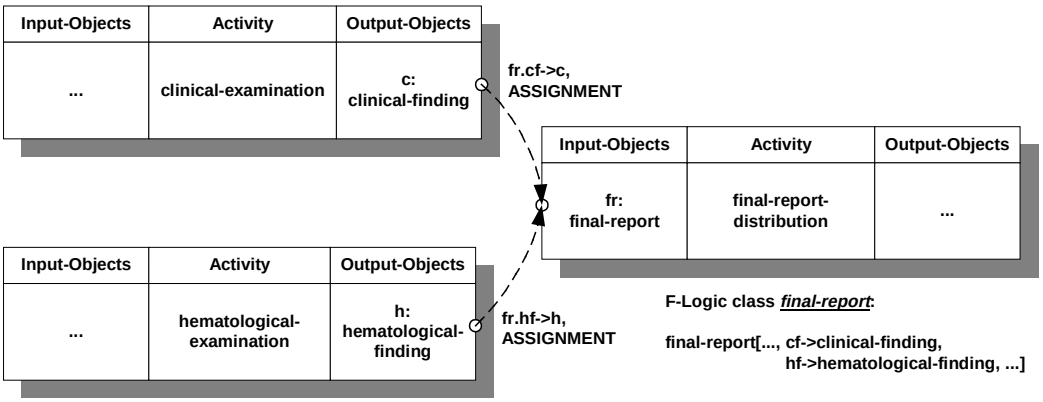


Tabelle 6 : Zusammengesetzter Datenfluss

Dies wird erreicht, indem man mit Hilfe von Objekt-Pfaden auf einzelne Komponenten eines Objektes (Attribute, Methoden oder Unterobjekte) verweist. Für den externen Datenfluss werden die Bedingungen, die für den Zugriff auf die externen Daten gebraucht werden, in den Objekt-Pfaden abgelegt.

Ein Objekt-Pfad ist dabei immer mit genau einer Objekt-Spezifikation verbunden, die das Objekt ist, auf dessen Komponenten der Objekt-Pfad zugreift. Ein Objekt-Pfad kann auch verschachtelt sein, das heißt er kann auf Komponenten der Unterobjekte zugreifen.

Beispiel:

```
// F-Logic Konstrukt: Name des Arztes von Patienten john
john:patient.doctor.name

// Objekt-Spezifikation
Name: john
Klasse: patient

// Objekt-Pfad
doctor.name
```

Wenn ein Objekt-Pfad zu einer Objekt-Spezifikation existiert, der leer ist, so verweist die Objekt-Spezifikation auf das Objekt als Ganzes. Dies geschieht dann, wenn der Datenfluss zwischen zwei Aktivitäten eins zu eins ist und das Output-Objekt der Quellaktivität gleich dem Input-Objekt der Zielaktivität ist.

Komponenten-Zuweisungen

Komponenten-Zuweisungen sind die Kanten im Datenfluss. Sie verbinden das Output-Objekt der Quellaktivität mit dem Input-Objekt der Zielaktivität. Darum besitzt eine Komponenten-Zuweisung immer zwei Objekt-Pfade, den Quellpfad und den Zielpfad.

Für die verschiedenen Arten des Datenflusses zwischen zwei Aktivitäten gibt es folgende Arten von Komponenten-Zuweisungen:

Typ	**Beschreibung**
assignment	Der Zielpfad erhält eine Referenz auf den Quellpfad. Damit greifen beide Pfade auf das gleiche Objekt zu.
shallow copy	Das Objekt des Quellpfades wird in das Objekt des Zielpfades kopiert, allerdings werden für Unterobjekte nur die Referenzen kopiert und nicht die Objekte selbst.
deep copy	Das Objekt des Quellpfades wird mitsamt aller Unterobjekte in das Objekt des Zielpfades kopiert. Damit greifen beide Pfade auf unabhängige Objekte zu.

Tabelle 7 : Arten von Komponenten-Zuweisungen

Externer Datenfluss

Ein weiterer wichtiger Aspekt des Datenflusses ist der externe Datenfluss. Außer dem Datenfluss zwischen zwei Aktivitäten gibt es auch den Datenfluss von und zu externen Datenquellen.

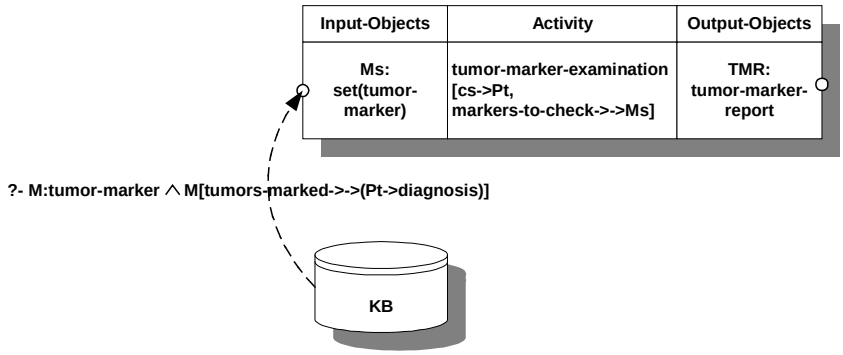


Abbildung 17 : Externer Datenfluss

Der externe Datenfluss teilt sich in zwei Bereiche:

- Der Datenfluss von einer externen Datenquelle zu einer Aktivität, das heißt, es werden Daten aus der externen Datenquelle abgerufen und einer Aktivität zur Verfügung gestellt. Die Realisierung dieser Art von externen Datenflusses erfolgt durch *Retrieval Queries*, die mittels F-Logic Anfragen auf der externen Datenquelle formulieren. Die CORBA Management Layer übersetzt diese Anfragen anschließend in eine Anfrage der entsprechende Datenquelle.

- Der Datenfluss von einer Aktivität zu einer externen Datenquelle, das heißt, es werden Daten der Aktivität in der externen Datenquelle eingefügt, geändert oder gelöscht. Der Datenfluss zur externen Datenquelle wird mit Hilfe von *Manipulation Queries* realisiert, die die entsprechenden Objekte in die Datenquelle einfügen, ändern oder löschen. Auch hier ist es Aufgabe der CORBA Management Layer die passende Aktion auf der Datenquelle auszuführen.

Externe Datenquellen können hierbei ganz unterschiedliche Quellen sein, zum Beispiel Wissensbasen, relationale oder objektorientierte Datenbanksysteme. Die Art der Quelle ist nicht festgelegt, die CORBA Management Layer muss nur die Datenquelle und die Zuordnung von F-Logic Klassen zu den Datenquellen kennen. Durch die Verwendung von F-Logic zur Modellierung des Datenflusses wird erreicht, dass die Definition des externen Datenzugriffes unabhängig von der Datenquelle ist und somit von der Heterogenität der Datenquellen abstrahiert wird. Auf den höheren Schichten kann somit auf einem einheitlichen Datenmodell gearbeitet werden.

4 Aufgaben der Workflow Runtime

Nachdem die Buildtime-Komponente das Organisationsmodell, die Aktivitäten und die Workflow-Definitionen definiert hat, ist es die Aufgabe der Runtime-Komponente die so automatisierten Geschäftsprozesse auszuführen.

Die Ausführung der Geschäftsprozesse lässt sich in die folgenden Teilaufgaben zerlegen:

- Erzeugung von Workflow-Instanzen aus den Workflow-Definitionen
- Verwaltung der erzeugten Workflow-Instanzen
- Ausführung der Workflow-Instanzen
- Interaktion mit Workflow-Teilnehmern, die zur Ausführung der Aktivitäten benötigt werden
- Ausführung der Anwendungsprogramme, die mit Aktivitäten assoziiert sind
- Fehlerbehandlung

Die aufgeführten Aufgaben werden in der Regel von den Teilen der Runtime-Komponente umgesetzt. Die Realisierung der Aufgaben werden im Kapitel 5 "Workflow Runtime" bei der Diskussion der einzelnen Teile der Workflow-Runtime besprochen. Die Fehlerbehandlung hingegen lässt sich nicht auf die Runtime-Komponente begrenzen und erfordert deswegen die Kooperation des gesamten Workflow-Management-Systems. Aus diesem Grund wird die Fehlerbehandlung im Vorfeld der Diskussion zur Runtime-Komponente erörtert.

4.1 Fehlerarten

In der bisherigen Diskussion ist erkennbar geworden, dass die Ausführung eines Workflows ein langlebiger Prozess ist, der unter Umständen Stunden, Tage oder Wochen dauern kann. Während dieser Zeit sind eine große Anzahl von Ressourcen (Hardware und Software) in die Ausführung involviert. Aufgrund der Vielzahl von Res-

sourcen und der Dauer der Ausführung ist es leicht einsichtig, dass während der Ausführung eines Workflows eine ganze Reihe von Fehlern eintreten können. Die Behandlung dieser Fehler ist für die Ausführung der Workflows und das Workflow-Management-System von entscheidender Rolle.

Aufgrund der Vielfalt an Fehlerarten ist es wichtig, zuerst einmal die verschiedenen Fehlerarten zu identifizieren. In der Hauptsache lassen sich vier verschiedenen Fehlerarten unterscheiden, (1) Gerätefehler, (2) Systemfehler, (3) Transaktionsfehler und (4) logische Fehler.

1. **Gerätefehler**

   Gerätefehler werden in der Regel durch den Ausfall oder der Fehlfunktion von Hardwarekomponenten verursacht, die für die Ausführung der Workflows benötigt werden. Im schlimmsten Fall kann ein solcher Fehler dazu führen, dass die momentan auszuführenden Workflows nicht mehr fortgesetzt werden können und dies zu einer Unterbrechung der Workflows führt.

   Ziel der Recovery muss es sein, die so unterbrochenen Workflows fortzusetzen und doch noch erfolgreich zu beenden. Dabei versucht man die Aktivitäten der Workflows, die bis zu diesem Zeitpunkt bereits erfolgreich beendet wurden, nicht rückgängig zu machen. Es werden nur die Aktivitäten zurückgefahren bzw. kompensiert, die von dem Fehler betroffen gewesen sind. Anschließend wird die Ausführung der Workflows an dieser Stelle fortgesetzt.

   In den meisten Fällen kann diese Art von Fehler dadurch gelöst werden, dass ein Datenbanksystem verwendet wird. Das Workflow-Management-System speichert dazu den Status aller Aktivitäten persistent in einer Datenbank, so dass auch nach einem möglichen Neustart des System diese Daten zur Verfügung stehen. Nach einem Fehlerfall kann für jede Aktivität anhand des Status festgestellt werden, ob die sie bereits erfolgreich beendet war oder ob sie zum Zeitpunkt des Fehlers gerade ausgeführt wurde. Die unterbrochenen Aktivitäten müssen dann zurückgefahren oder kompensiert werden. Anschließend können die Workflows fortgesetzt werden.

   Gerätefehler sollen hierbei aber nicht Schwerpunkt der Diskussion sein und werden deswegen nur erwähnt.

2. Systemfehler

Systemfehler betreffen das Workflow-Management-System und bedeuten, dass eine der Softwarekomponenten des Workflow-Management-Systems einen Fehler verursacht hat. Im Normalfall wird davon ausgegangen, dass alle Komponenten des Systems fehlerfrei arbeiten. Dies schließt aber nicht aus, dass es durch äußere Umstände trotzdem zu Fehlersituationen kommen kann. Alle Teile des Workflow-Management-Systems sind darum aufgeordert entsprechende Maßnahmen zur Fehlerbehandlung zu ergreifen, so dass das Workflow-Management-System auch im Fehlerfall seine Arbeit fortsetzen kann. Wenn sich dies nicht gewährleisten lässt, dann verhält sich diese Art von Fehler ähnlich wie die Gerätefehler und es lassen sich die gleichen Maßnahmen zur Recovery verwenden.

Auf eine detailliertere Diskussion der Systemfehler wird an dieser Stelle allerdings verzichtet.

3. Transaktionsfehler

Die dritte Art von Fehlern sind die Transaktionsfehler. Bei der Ausführung eines Workflows kann man den Workflow entweder als eine Transaktion oder eine Reihe von Transaktionen auffassen, die den Workflow von einem konsistenten Zustand in den nächsten konsistenten Zustand überführen. Wenn nun während der Ausführung eines Workflows ein Fehler eintritt, zum Beispiel durch eine gescheiterte Aktivität oder einem fehlerhaften Datenfluss, dann müssen diese Fehler vom Workflow-Management-System behandelt werden. Aufgrund der Unterschiede der Transaktionen in Datenbanksystemen und in Workflows wurden eigene Konzepte für die Fehlerbehandlung in Workflows entwickelt.

Die Behandlung von Transaktionsfehlern wird zum momentanen Zeitpunkt vom Workflow-Management-System noch nicht unterstützt, da dies eine Unterstützung durch die CORBA Management Layer voraussetzen würde, um Daten und Anwendungsprogramme transaktionsgeschützt auszuführen. Trotzdem werden ausgewählte Konzepte für Workflow-Transaktionen im folgenden Abschnitt 4.2 "Konzepte der Fehlerbehandlung" genauer dargestellt und dahingehend untersucht, inwieweit diese Konzepte für die Unterbrechung der Workflows bei logischen Fehlern genutzt werden können.

4. **Logische Fehler**

   Die aus Sicht des Projektes AGENTWORK wichtigste Fehlerart sind die logischen Fehler, wobei die Erkennung und Behandlung dieser Fehler zentrales Anliegen des Workflow-Management-Systems ist. Diese Art von Fehlern unterscheidet sich in sofern von den Transaktionsfehlern, dass bei diesen nur die technischen Fehler während der Ausführung des Workflows behandelt werden. Wie bereits in früheren Kapiteln erwähnt worden ist, können während der Ausführung eines Workflows Ereignisse eintreten, die eine Modifikation des Kontroll- und Datenflusses des Workflows verlangen. Diese Ereignisse lassen sich aber nicht als technische Fehler identifizieren und müssen deshalb separat behandelt werden. Die ausführliche Diskussion zur Behandlung von logischen Fehlern erfolgt im Rahmen der Workflow Engine und findet sich im nächsten Kapitel (siehe 5.1.4 "Logische Fehler und Unterbrechungen").

4.2 Konzepte der Fehlerbehandlung

Ein Workflow besteht aus einer Anzahl von Aktivitäten, die während der Laufzeit des Workflows ausgeführt werden. Der Workflow kann dabei als eine Transaktion oder eine Reihe von Transaktionen aufgefasst werden, die den Workflow von einem konsistenten Zustand in den nächsten konsistenten Zustand überführen. Wenn während der Ausführung einer Aktivität ein Fehler auftritt, wird ein Mechanismus zur adäquaten Behandlung der Fehler benötigt, der flexibel auf Fehler reagiert und die korrekte und zuverlässige Ausführung des Workflows gewährleistet.

Workflow-Transaktionen unterscheiden sich allerdings in einigen Punkten von den Transaktionen, wie man sie von den Datenbanksystemen kennt.

Die wichtigsten Unterschiede zwischen Datenbank-Transaktionen und Workflow-Transaktionen sind:

- Workflow-Transaktionen sind langlebige Prozesse, die Stunden bis Wochen dauern können. Datenbank-Transaktionen haben hingegen eine sehr kurze Ausführungsdauer, die im Bereich von Millisekunden bis Minuten liegt.

- Im Fehlerfall sollen Workflow-Transaktionen nicht wie Datenbank-Transaktionen vollständig zurückgesetzt werden (*Backward Recovery*), sondern nur die Aktivitäten zurückgesetzt werden, die von einem Fehler betroffen sind, um anschließend den Workflow an dieser Stelle fortzufahren (*Forward Recovery*). Damit wird versucht den Arbeitsverlust im Fehlerfall auf ein Minimum zu reduzieren, da sonst bei langlebigen Workflows ein sehr großer Arbeitsverlust zu erwarten ist.

- Workflow-Transaktionen enthalten Aktivitäten, die sich nicht alle zurücksetzen lassen, sondern teilweise nur kompensiert werden können. Bei Datenbankoperationen lässt sich hingegen eine Rücksetzbarkeit aller Datenbankoperationen garantieren.

- Workflow-Transaktionen können Aktivitäten enthalten, die eine Beziehung zueinander besitzen. Wenn eine solche Aktivität von einem Fehler betroffen ist, so müssen auch alle abhängigen Aktivitäten rückgängig gemacht werden oder kompensiert werden. Bei Datenbank-Transaktionen wird dagegen von einer Atomarität der Transaktion ausgegangen, so dass es keine Abhängigkeiten zwischen Transaktionen gibt.

Wenn zum Beispiel bei der Behandlung eines Patienten ein Fehler eintritt, so ist es nicht wünschenswert alle Aktivitäten des Workflows, sprich die ganze Behandlung, rückgängig zu machen. Ziel der Recovery sollte es eher sein nur die Aktivitäten rückgängig zu machen, die von einem Fehler betroffen sind. Abgesehen davon ist es in manchen Fällen auch gar nicht möglich, eine Aktivität rückgängig zu machen. Die Verabreichung eines Medikaments ist zum Beispiel eine Aktivität, die nicht mehr rückgängig gemacht werden kann. Vielmehr wird versucht die entsprechende Aktivität zu kompensieren, um die Auswirkungen der Aktivität ungeschehen zu machen. Im Fall der Medikamentengabe kann dies bedeuten, dass ein anderes Medikament verabreicht werden muss, um die erste Medikamentengabe zu kompensieren.

Das für Datenbank-Transaktionen gültige *ACID-Paradigma*[8] kann dadurch so nicht auf Workflow-Transaktionen angewendet werden. Dies hat zu einer Abschwächung des ACID-Paradigmas für Workflow-Transaktionen geführt. Speziell die Atomarität und Isolation des ACID-Paradigmas ist für Workflows so nicht mehr praktikabel. In der Zwischenzeit wurden einige Konzepte für Workflow-Transaktionen entwickelt, die das Problem der Fehlerbehandlung in Workflows versuchen zu lösen. Zu den wichtigsten Konzepten für Workflow-Transaktionen gehören *Sagas*, *ConTracts*, *Atomaritäts-Sphären* und *Kompensations-Sphären*.

Im folgenden sollen ausgewählte Konzepte vorgestellt werden, die sich speziell mit der Forward Recovery beschäftigen. Die Ideen, die diesen Konzepten zugrunde liegen, lassen sich auch für die Unterbrechung der Workflows verwenden, wie sie für die Behandlung von logischen Fehlern benötigt wird (siehe 5.1.4 "Logische Fehler und Unterbrechungen").

- **ConTracts**

  Einen Ansatz für Workflow-Transaktionen stellt das an der Universität Stuttgart entwickelte ConTract-Modell [RW91, RS95] dar. Dieses Konzept beschäftigt sich besonders damit, das ACID-Paradigma um Möglichkeiten der Forward Recovery zu erweitern.

  Ein Workflow wird im ConTract-Modell als *Contract* bezeichnet, der aus einer Reihe von Aktivitäten, den sogenannten *Steps*, besteht. Jeder dieser Steps entspricht dabei in der Regel einer Transaktion, wobei das Modell auch zulässt, dass mehrere Steps zu einer Transaktion zusammengefasst werden. Für die Backward Recovery im Fehlerfall ist jedem Step ein Kompensations-Step zugeordnet, der ausgeführt wird, wenn der entsprechende Step fehlschlägt. Wenn während der Ausführung des Contracts ein Fehler eintritt, wird der fehlgeschlagene Step kompensiert und anschließend die Ausführung des Contracts fortgesetzt, wobei der fehlgeschlagene Step erneut ausgeführt wird oder ein alternativer Step gestartet werden kann.

[8] **ACID** = **A** – Atomicity, **C** – Consistency, **I** – Isolation, **D** - Durability

Jedem Step eines Contracts können zusätzlich sogenannte Invarianten zugeordnet werden, die anwendungsspezifische Prädikate zur Beschreibung von Zustandsbedingungen sind. Diese Invarianten können als Eingangs- bzw. Ausgangsbedingung verwendet werden, um Bedingungen zu spezifizieren, die ein Step vor seiner Ausführung bzw. nach seiner Beendigung erfüllen muss. Im ConTract-Modell werden die Invarianten hauptsächlich dafür verwendet, um den Zugriff mehrerer Steps auf ein Objekt zu synchronisieren. Die Invarianten lassen sich aber auch dazu verwenden, um Bedingungen zu spezifizieren, die die Durchführung von Kompensationen garantieren.

Zur Behandlung von Systemfehlern wird außerdem für jeden Contract ein persistenter *Kontext* verwaltet, der die Zustände von globalen Variablen, Zwischenergebnissen, usw. enthält und der im Fehlerfall zur Recovery verwendet wird. Am Ende des Contracts wird dieser Kontext dann freigegeben.

- **Atomaritäts-Sphären**

  Einer der interessantesten Ansätze ist das Konzept der Atomaritäts-Sphären. Die Idee ist, mehrere Aktivitäten, die eine Beziehung zueinander haben, in einer Atomaritäts-Sphäre zusammenzufassen. Die Aktivitäten innerhalb der Atomaritäts-Sphäre werden dabei entweder alle erfolgreich ausgeführt oder alle abgebrochen. Wenn alle Aktivitäten erfolgreich beendet wurden, dann wird auch die Atomaritäts-Sphäre erfolgreich beendet, andernfalls werden alle Aktivitäten abgebrochen und die Atomaritäts-Sphäre wird auch abgebrochen. Wenn es zum Abbruch der Atomaritäts-Sphäre kommt, dann wird die Atomaritäts-Sphäre erneut gestartet und versucht die Atomaritäts-Sphäre doch noch erfolgreich zu beenden. Eine Besonderheit ist, dass nur die Aktivitäten in die Transaktion einbezogen werden, die vom Kontrollfluss auch erreicht werden. Das heißt, Aktivitäten, die nicht ausgeführt werden, werden so behandelt, als ob sie nicht zu der Atomaritäts-Sphäre gehören und haben somit auch keinen Einfluss auf den Erfolg oder Misserfolg der Atomaritäts-Sphäre. Voraussetzung für die Verwendung von Atomaritäts-Sphären ist allerdings, dass die Implementierungen der Aktivitäten eine Transaktionsunterstützung bieten.

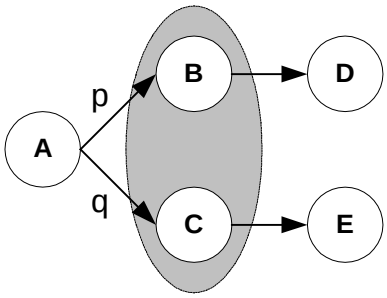


Abbildung 18 : Atomaritäts-Sphäre nach [LR99]

Das Beispiel einer Atomaritäts-Sphäre zeigt zwei Aktivitäten (B und C), die zu einer Atomaritäts-Sphäre zusammengefasst wurden. Wenn beide Aktivitäten gestartet werden, müssen auch beide erfolgreich zum Ende kommen oder beide werden abgebrochen. Anders sieht das aus, wenn eine der Bedingungen (p und q) nicht erfüllt ist und damit nur eine der beiden Aktivitäten ausgeführt wird. In dieser Situation hat die Aktivität, die nicht ausgeführt wird, keinen Einfluss auf die andere Aktivität und nur die auszuführende Aktivität nimmt an der Transaktion teil. Auf jeden Fall wird die Ausführung des restlichen Workflows ganz normal fortgesetzt, wenn die Atomaritäts-Sphäre schließlich erfolgreich beendet wurde.

- **Kompensations-Sphären**

  Ein weiterer wichtige Ansatz sind die sogenannten Kompensations-Sphären. Die Idee hinter dem Konzept der Kompensations-Sphäre ist, dass mehrere Aktivitäten, die eine Beziehung zueinander haben, in einer Kompensations-Sphäre zusammengefasst werden. Vorteil dieses Konzepts ist es, dass die Implementierungen der Aktivitäten nicht zwangsläufig eine Transaktionsunterstützung bieten müssen, dafür aber eine Möglichkeit der Kompensation. Wenn während der Ausführung einer Aktivität, innerhalb der Kompensations-Sphäre, ein Fehler auftritt, dann muss nicht nur diese Aktivität wiederholt werden, sondern alle Aktivitäten, die bis dahin ausgeführt wurden. Vorher müssen aber erst einmal alle Aktivitäten, die bis dahin ausgeführt wurden, in umgekehrter Reihenfolge kompensiert werden. Anschließend kann die Kompensations-Sphäre erneut gestartet werden, um die Aktivitäten doch noch komplett auszuführen.

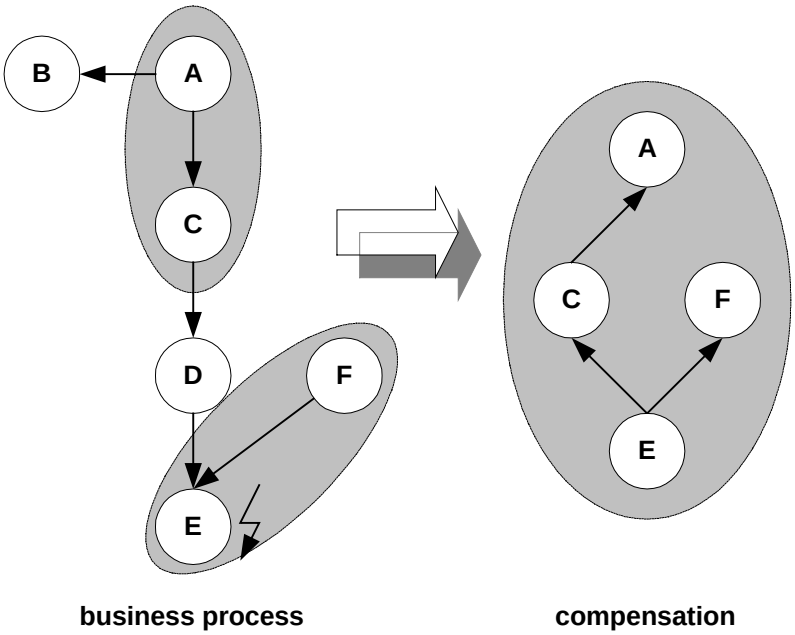


Abbildung 19 : Kompensations-Sphäre nach [LR99]

Das Beispiel einer Kompensations-Sphäre zeigt einen Workflow, in dem vier Aktivitäten (B, C, E und F) zu einer Kompensations-Sphäre zusammengefasst worden sind. Bei der Ausführung dieser Kompensations-Sphäre ist einer der Aktivität (E) ein Fehler aufgetreten. In dieser Situation wird die Ausführung des Workflows unterbrochen. Anschließend werden alle bis dahin ausgeführten Aktivitäten, innerhalb der Kompensations-Sphäre, durch die dazugehörigen Kompensationsfunktionen in umgekehrter Reihenfolge korrigiert.

5 Workflow Runtime

Die Runtime-Komponente besteht in der Hauptsache aus den drei Teilen, Workflow Engine, Worklist und Workflow Clients. Mittelpunkt der folgenden Diskussion liegt dabei auf der Workflow Engine, die den Schwerpunkt der vorliegenden Arbeit bildet.

5.1 Workflow Engine

Wie bereits, bei der Beschreibung der allgemeinen Architektur eines Workflow-Management-Systems, erwähnt, bildet die Workflow Engine den zentralen Teil des Workflow-Management-Systems. Die Workflow Engine ist für die Ausführung der Workflow-Instanzen verantwortlich und interagiert dabei mit den Workflow-Teilnehmern und Anwendungsprogrammen.

Die Aufgabe der Workflow Engine lässt sich in vier Teilaufgaben aufschlüsseln:

- Erzeugung, Verwaltung, Ausführung und Unterbrechung der Workflow-Instanzen
- Abarbeitung des Kontrollflusses
- Abarbeitung des Datenflusses
- Steuerung der Anwendungsprogramme (Start, Stop, Rollback)

Die hier vorgestellt Workflow Engine zeichnet sich speziell dadurch aus, dass eine Workflow-Instanz jederzeit zur Adaptation unterbrochen werden kann und anschließend an der unterbrochenen Stelle fortgefahren werden kann.

Eine weitere Besonderheit bilden der Datenfluss und die Steuerung der Anwendungsprogramme. Die Verteilung von Daten und Anwendungsprogrammen führt dazu, dass sie von der Middleware verwaltet und gesteuert werden. Die CORBA Management Layer stellt die Schnittstelle zur Middleware dar und bietet Möglichkeiten zur Verwaltung und Steuerung von Daten und Anwendungsprogrammen. Das Ergebnis ist die enge Zusammenarbeit zwischen Workflow Engine und CORBA Management Layer, um die Aufgaben der Workflow Engine auszuführen.

5.1.1 Architektur der Workflow Engine

Die Architektur der Workflow Engine wird in der folgenden Abbildung (Abbildung 20) dargestellt. Die Abbildung zeigt die verschiedenen Teile der Workflow Engine und die Schnittstellen zu den wichtigsten anderen Komponenten des Workflow-Management-Systems. Die Funktion der einzelnen Teile der Workflow Engine werden in den folgenden Anschnitten besprochen.

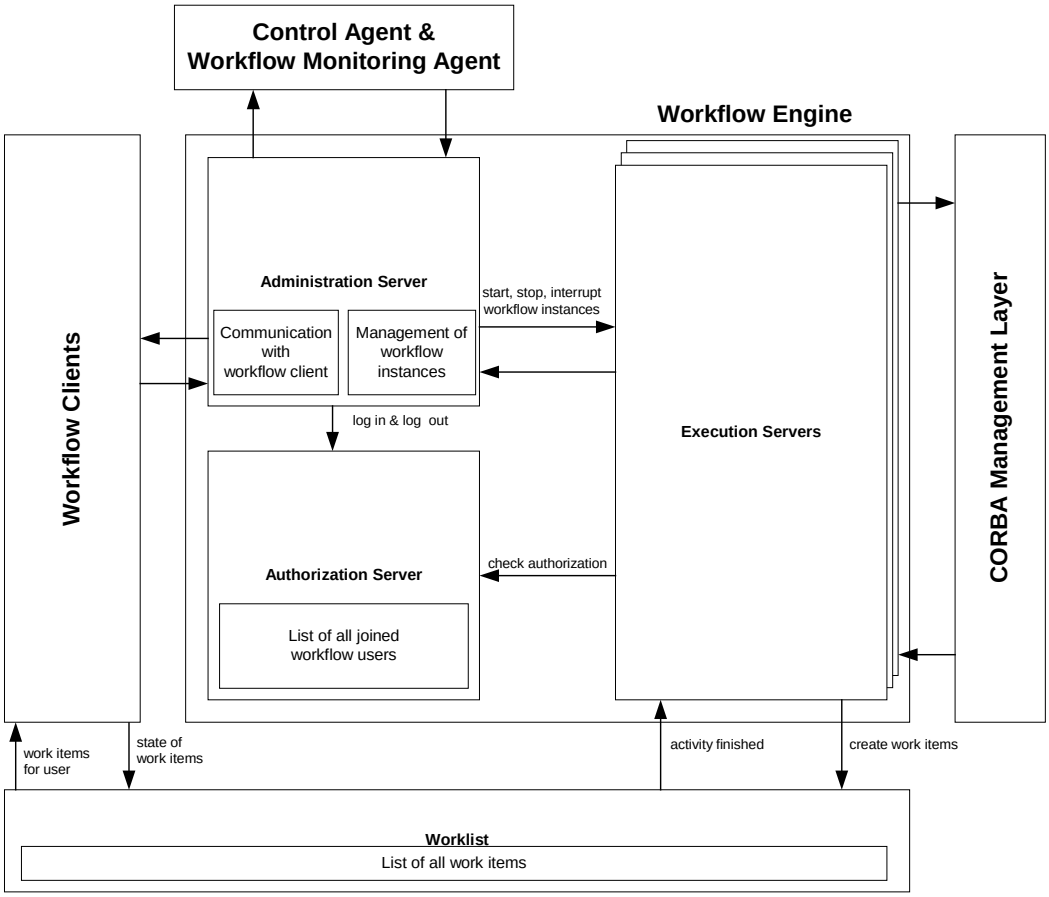


Abbildung 20 : Architektur der Workflow Engine auf Basis von [LR99]

5.1.1.1 Administration Server

Der *Administration Server* ist der Koordinator der Workflow Engine. Aufgabe des Administration Servers ist es, Workflow-Instanzen zu erzeugen, zu verwalten und zu steuern. Außerdem stellt er die Schnittstelle zu den Workflow Clients und den Agenten des Workflow-Management-Systems dar. Intern interagiert der Administration

Server mit dem *Authorization Server* und dem *Execution Server*. Das Interface zum Authorization Server bezieht sich auf das Anmelden bzw. Abmelden der Benutzer. Die Steuerung der Workflow-Instanzen erfolgt über die Schnittstelle zu den Execution Servern, die für die Ausführung der Workflow-Instanzen verantwortlich sind. Die Steuerung beinhaltet dabei das Starten, Stoppen und Unterbrechen der Workflow-Instanzen.

5.1.1.2 Authorization Server

Der *Authorization Server* hat die Aufgabe, die Benutzer des Workflow-Management-Systems zu verwalten. Er überprüft das Login und Passwort, wenn sich ein Benutzer anmeldet und authorisiert jede Aktion des Nutzers, bis dieser sich am System abmeldet.

5.1.1.3 Execution Server

Funktion der Execution Server ist die Ausführung der Workflow-Instanzen. Die Workflow Engine beinhaltet dazu mehrere Instanzen des Execution Servers, um jede Workflow-Instanz von einem eigenen Execution Server ausführen zu lassen. Diese Vorgehensweise ist notwendig, um mehrere Workflow-Instanzen parallel ausführen zu können. Ansonsten wäre es nur möglich immer eine Workflow-Instanz zu einer bestimmten Zeit auszuführen, was für ein Workflow-Management-System nicht akzeptabel sein kann.

Außerdem wird jeder parallele Pfad in einer Workflow-Instanz in einer separaten Instanz des Execution Servers abgearbeitet. Das heißt, wenn im Kontrollfluss eine Verzweigung erreicht wird, wird für jeden parallelen Pfad der Verzweigung ein eigener Execution Server gestartet, der diesen Pfad abläuft. Wenn das Ende des Pfades erreicht ist, wird der Execution Server beendet und zur aufrufenden Instanz des Execution Servers gewechselt, die für die weitere Verarbeitung des Kontrollflusses verantwortlich ist. Durch diesen Mechanismus lässt sich die Parallelität innerhalb des Workflows auch bei der Ausführung verwirklichen. Außerdem ergibt sich eine hierarchische Ordnung der Execution Server, wobei einer Instanz des Execution Servers eine Reihe weiterer Instanzen untergeordnet sein können.

Für die Realisierung des Datenflusses und den Start der Anwendungsprogramme kooperiert der Execution Server mit der CORBA Management Layer, die diese Funktionalitäten über eine Schnittstelle zur Verfügung stellt.

Die andere Schnittstelle des Execution Servers betrifft die Worklist. Wenn bei der Abarbeitung des Kontrollflusses eine manuelle Aktivität erreicht wird, so erzeugt er einen Eintrag in der Worklist, um die Ausführung der Aktivität durch den Benutzer zu veranlassen. Nach der Beendigung der Aktivität wird der Execution Server benachrichtigt und kann mit weiteren Ausführung der Workflow-Instanz fortfahren.

5.1.2 Erzeugung und Verwaltung der Workflow-Instanzen

Bevor ein konkreter Workflow ausgeführt werden kann, muss aus der Workflow-Definition eine Workflow-Instanz erzeugt werden. Die Workflow-Definition dient in diesem Zusammenhang als Vorlage für die konkreten Workflow-Instanzen.

Der Unterschied zwischen der Workflow-Definition und der Workflow-Instanz besteht darin, dass die Workflow-Instanz den Kontroll- und Datenfluss der Workflow-Definition um die konkreten Daten und die Statusinformationen für Knoten, Kanten und Anwendungsprogramme erweitert. Das bedeutet, dass jeder konkrete Workflow eine eigene Umgebung hat, in der alle Aktionen während der Ausführung des Workflows stattfinden. Der aber wohl wichtigste Grund, warum die Informationen der Workflow-Definition für jede Instanz kopiert werden ist die Adaptation der Workflow-Instanzen zur Laufzeit. Wenn nämlich Änderungen an einer Workflow-Instanz vorgenommen werden, so haben diese keine Auswirkungen auf andere Workflow-Instanzen, sondern sind nur auf die betroffene Workflow-Instanz begrenzt.

Die Erzeugung einer Workflow-Instanz verläuft in drei Phasen (Abbildung 21):

1. **Erzeugung einer eineindeutigen ID für die Workflow-Instanz**

   Damit die verschiedenen Workflow-Instanzen voneinander unterschieden werden und von der Workflow Engine verwaltet werden können, muss jede Workflow-Instanz eine eineindeutige Identifikation (kurz ID) bekommen. Diese Identifikation setzt sich aus drei Teilen zusammen, dem Namen des Erzeugers,

dem Namen der Workflow-Definition und dem Erzeugungsdatum der Workflow-Instanz. Im Anschluss wird diese Identifikation zur Adressierung der Workflow-Instanz benutzt.

2. Kopieren der Workflow-Definition in die Workflow-Instanz

Nachdem eine eindeutige Identifikation generiert wurde, werden die Informationen der Workflow-Definition in die Workflow-Instanz kopiert. Dies bedeutet, dass die Daten aus der Buildtime-Datenbank in die Runtime-Datenbank kopiert werden. Damit ist die Erzeugung der Workflow-Instanz fast abgeschlossen, was noch fehlt sind die Statusinformationen.

3. Setzen der Statusinformationen für die Workflow-Instanz

Im letzten Schritt werden die notwendigen Statusinformationen für die Workflow-Instanz, die Knoten, die Transitionen und die Anwendungsprogramme gesetzt. Nachdem dies geschehen ist, ist die Workflow-Instanz bereit ausgeführt zu werden. Dies geschieht allerdings nicht automatisch, sondern muss explizit vom Benutzer angefordert werden.

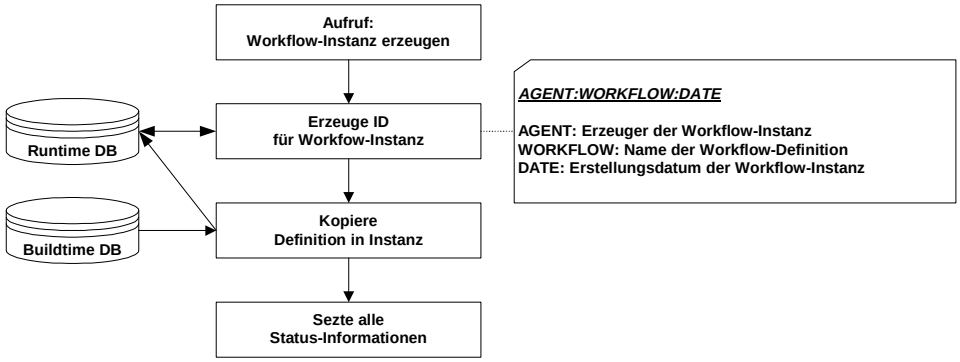


Abbildung 21 : Erzeugen einer Workflow-Instanz

5.1.3 Verarbeitung der Workflow-Instanzen

Nachdem die Workflow-Instanz von einem Benutzer erzeugt worden ist, befindet sie sich in der Runtime-Datenbank und kann ausgeführt werden. Der Benutzer kann nun

die Ausführung des Workflows anfordern. Wenn dies geschieht, startet der Administration Server eine neue Instanz des Execution Servers, der die Workflow-Instanz verarbeitet.

In der ersten Phase, nach dem Start der Workflow-Instanz, wird damit begonnen die globalen Objekte zu erzeugen. Dazu werden die Objekt-Spezifikationen der globalen Objekte an die CORBA Management Layer übergeben, die die dazugehörigen Objekte erzeugt. Anschließend werden die globalen Objekte mit den Daten aus externen Datenquellen gefüllt oder die Daten über ein User Interface vom Benutzer abgefragt. Mit diesem Schritt ist die Initialisierung der Workflow-Instanz abgeschlossen und die eigentliche Ausführung des Workflows kann beginnen.

Um die Ausführung beginnen zu können muss zunächst einmal der Start-Knoten gesucht werden. Die Ausführung des Workflows funktioniert so, dass sich ausgehend vom Start-Knoten immer von einem Knoten zum nächsten bewegt wird und damit der Kontrollfluss-Graph durchlaufen wird. Dies wird solange fortgesetzt, bis schließlich der End-Knoten erreicht wird und der Workflow damit beendet ist.

Wenn beim Durchlaufen des Kontrollfluss-Graphen ein Knoten erreicht wird, muss dieser verarbeitet werden. Im ersten Schritt wird geprüft, ob der Knoten auch wirklich ausgeführt werden kann. Dazu muss man wissen, dass ein Knoten nicht automatisch ausgeführt werden darf, wenn er beim Durchlaufen des Kontrollfluss-Graphen erreicht wird. Vielmehr ist es so, dass ein Knoten mehr als eine eingehende Kante haben kann und der Knoten erst dann ausgeführt wird, wenn alle eingehenden Kanten aktiviert wurden.

Nachdem sichergestellt ist, dass der Knoten ausgeführt werden darf, wird im zweiten Schritt entschieden, wie der Knoten zu verarbeiten ist. Dazu unterscheidet man zwischen Kontrollflussknoten auf der einen Seite und Aktivitätsknoten auf der anderen Seite.

Falls es sich bei dem Knoten um einen Aktivitätsknoten handelt, wird weiterhin unterschieden, ob dem Knoten eine einfache oder komplexe Aktivitäts-Definition zugrunde liegt. Dazu wird die Aktivitäts-Definition aufgesucht, die dem Aktivitätsknoten zugeordnet ist.

Sofern es sich bei der Aktivitäts-Definition um eine komplexe handelt, heißt dies, dass sich hinter dem Aktivitäts-Knoten ein Subworkflow verbirgt. Der Subworkflow wird dann als eigenständiger Workflow ausgeführt, was heißt, dass für diesen Subworkflow eine eigene Instanz eines Execution Servers gestartet wird, der diesen Subworkflow

bearbeitet. Nachdem der Subworkflow beendet ist, wird der Execution Server beendet, und zur Ausführung des übergeordneten Workflows zurückgekehrt. In diesem Moment ist die Verarbeitung dieses Aktivitätsknotens beendet und es kann der nächste Knoten verarbeitet werden.

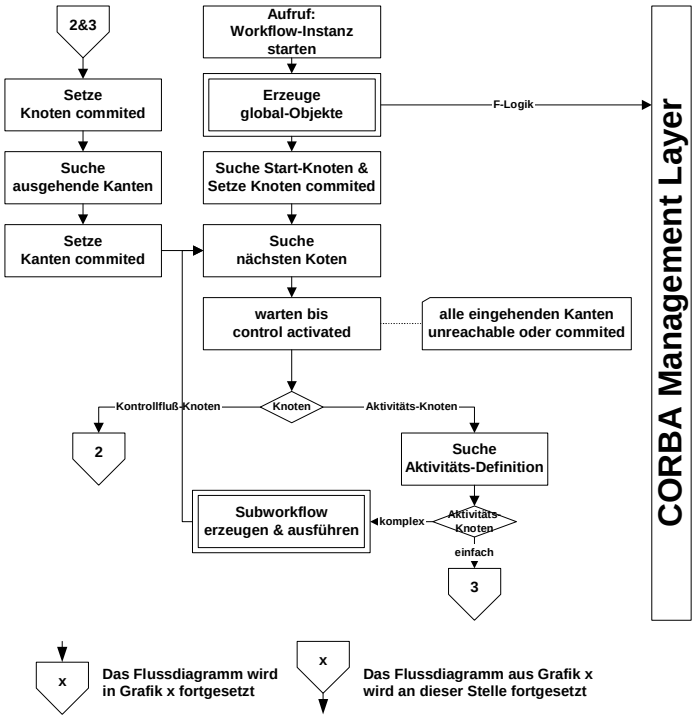


Abbildung 22 : Verarbeiten einer Workflow-Instanz (1)

Wenn dem Aktivitätsknoten im anderen Fall eine einfache Aktivitäts-Definition zugeordnet ist, muss die zugrundeliegende Aktivität ausgeführt werden.

Nachdem die Aktivitäts-Definition des Aktivitätsknoten herausgesucht worden ist, wird versucht den Datenfluss für die Aktivitäts-Definition bereitzustellen. Hierbei wird zwischen internen und externen Datenfluss unterschieden.

Falls Input-Objekte als externer Datenfluss modelliert sind, dann werden die Objekt-Spezifikationen an die CORBA Management Layer übergeben, die die dazugehörigen Objekte erzeugt und mit den Daten aus der externen Datenquelle füllt.

Falls Input-Objekte als interner Datenfluss modelliert sind, dann muss unterschieden werden, wie auf die Daten zugegriffen werden soll. Innerhalb des internen Daten-

flusses werden drei Arten von Abbildungen zwischen Quellobjekten und Zielobjekten unterschieden, nämlich assignment, shallow copy und deep copy.

Im Falle von Assignment müssen nun nur die Objekt-Mappings ausgewertet werden, um die Referenzen auf die benötigten Objekte zu erhalten. In den beiden anderen Fällen werden zuerst die Objekt-Spezifikationen an die CORBA-Management-Layer übergeben, um neue Objekte zu erstellen. Anschließend werden die Objekt-Mappings ausgewertet, um die erstellten Objekte mit Daten zu füllen.

Wenn alle Input-Objekte zur Verfügung stehen, kann die Aktivität gestartet werden. Zur Ausführung der Aktivität werden nun die Anwendungsprogramme gesucht, die mit der Aktivitäts-Definition assoziiert sind. Anschließend werden die Input-Objekte der Aktivitäts-Definition auf die Input-Objekte der Anwendungsprogramme abgebildet. Hinterher werden dann die Anwendungsprogramme gestartet. Wenn anschließend alle Anwendungsprogramme beendet sind, werden die Output-Objekte von den Anwendungsprogrammen empfangen und auf die Output-Objekte der Aktivitäts-Definition abgebildet. Damit ist der Aktivitätsknoten beendet, und es kann mit dem nächsten Knoten fortgefahren werden.

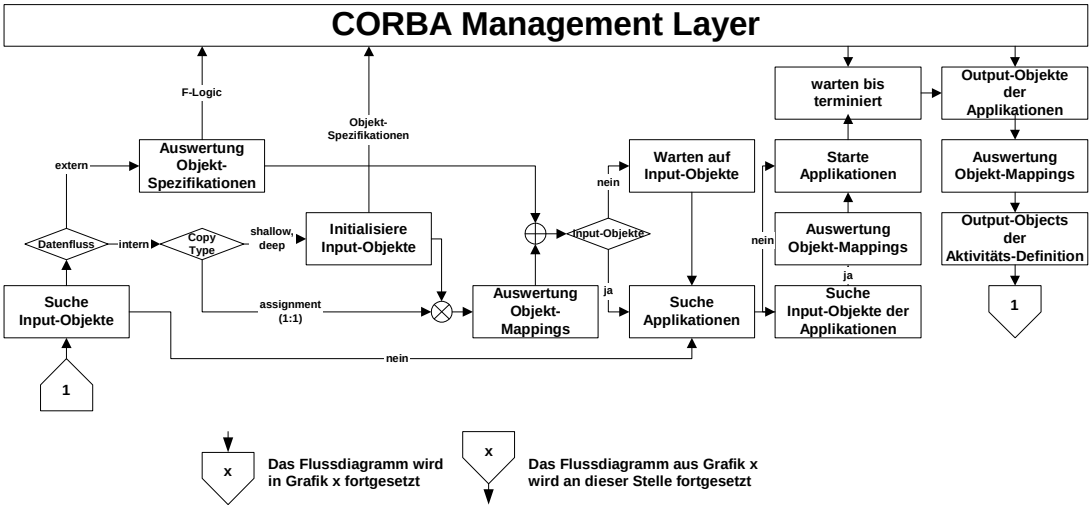


Abbildung 23 : Verarbeiten einer Workflow-Instanz (2)

Die zweite Art von Knoten, die verarbeitet werden muss, sind Kontrollflussknoten. Wenn beim Durchlaufen des Kontrollfluss-Graphen auf ein Kontrollflussknoten gestoßen wird, wird zuerst untersucht, um was für einen Kontrollflussknoten es sich dabei

handelt. Es wird hierbei zwischen End-Knoten, Join-Knoten und Split-Knoten/Loop-Knoten unterschieden.

Der erste Fall sind End-Knoten. Ein End-Knoten signalisiert, dass der Workflow zum Ende gekommen ist und die Ausführung der Workflow-Instanz beendet werden muss. Die Aufgabe besteht nun darin, die Statusinformationen der Workflow-Instanz in der Runtime-Datenbank zu sichern und die Instanz des Execution Servers zu beenden.

Der zweite Fall sind Split- und Loop-Knoten. Wenn es sich bei dem gefundenen Knoten um einen OR-Split oder AND-Split handelt, dann werden in der ersten Phase die ausgehenden Kanten des Split-Knotens gesucht und für jeden der parallelen Pfade eine eigene Instanz des Execution Servers gestartet, der den Pfad verarbeitet. Bevor die Ausführung der parallelen Pfade gestartet wird, werden die Bedingungen an den ausgehenden Kanten des Split-Knoten ausgewertet. Die Instanzen des Execution Servers, deren Bedingungen nicht erfüllt sind, werden sofort wieder beendet. Anschließend ist die Verarbeitung des Split-Knoten beendet und die Ausführung wird mit der Verarbeitung der parallelen Pfade fortgesetzt. Anders sieht das bei LoopStart- und LoopEnd-Knoten aus. Bei einem LoopStart-Knoten wird nur die ausgehende Kante gesucht und anschließend sofort mit der Ausführung des nächsten Knoten fortgefahren. Bei einem LoopEnd-Knoten werden zuerst die ausgehenden Kanten gesucht und ferner die Bedingungen an den ausgehenden Kanten ausgewertet, um zu entscheiden, welche der beiden Kanten benutzt werden soll. Im einen Fall wird die Schleife erneut durchlaufen und im anderen Fall die Schleife verlassen. Nach der Entscheidung, welche Kante benutzt wird, wird mit der Ausführung des nächsten Knoten fortgefahren.

Der dritte Fall sind Join-Knoten. In dieser Situation hat einer der parallelen Pfade das Ende des Pfades erreicht und muss nun überprüfen, ob er der erste Pfad ist, der den Join-Knoten erreicht hat. Wenn bereits ein anderer der parallelen Pfade den Join-Knoten erreicht hat, kann der Pfad einfach beendet werden. Dies bedeutet, dass die Statusinformationen des Pfades gesichert werden müssen und anschließend die Instanz des Execution Servers beendet wird. Hingegen, wenn der Pfad der erste ist, der den Join-Knoten erreicht hat, dann wird zur übergeordneten Instanz des Execution Servers gewechselt, um die Koordination der parallelen Pfade zu übernehmen. Wenn der Join-Knoten ein ONE-Join-Cancel ist, dann können, nach dem Erreichen des Join-Knotens durch den ersten Pfad, alle Pfade beendet werden, auch wenn diese noch nicht alle vollständig abgearbeitet wurden. Anschließend wird mit der Ausführung des nächsten Knotens fortgefahren. Wenn der Join-Knoten ein ONE-Join-Complete ist,

wird der Pfad beendet, der den Join-Knoten erreicht hat. Die anderen Pfade werden aber weiterhin ausgeführt und erst dann beendet, wenn sie den Join-Knoten erreichen. In der Zwischenzeit wird allerdings bereits mit der Verarbeitung des weiteren Kontrollflusses fortgefahren. Wenn hingegen der Join-Knoten ein ALL-Join ist, dann wird darauf gewartet, dass alle parallelen Pfade zum Ende gekommen sind. Erst dann werden alle Pfade beendet und mit der Ausführung des nächsten Knotens fortgefahren.

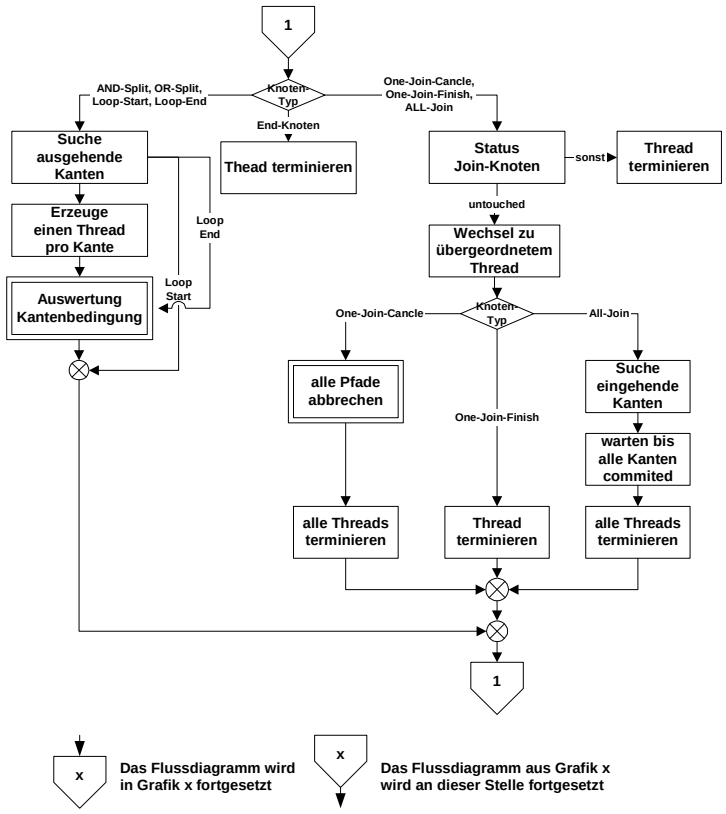


Abbildung 24 : Verarbeiten einer Workflow-Instanz (3)

5.1.3.1 Lebenszyklus der Workflow-Instanzen

Während der Verarbeitung eines Workflows durchläuft die Workflow-Instanz eine Reihe von Zuständen, die Auskunft über den Status des Workflows geben. Die folgende Abbildung zeigt alle Zustände und Übergänge, die die Workflow-Instanz erreichen kann.

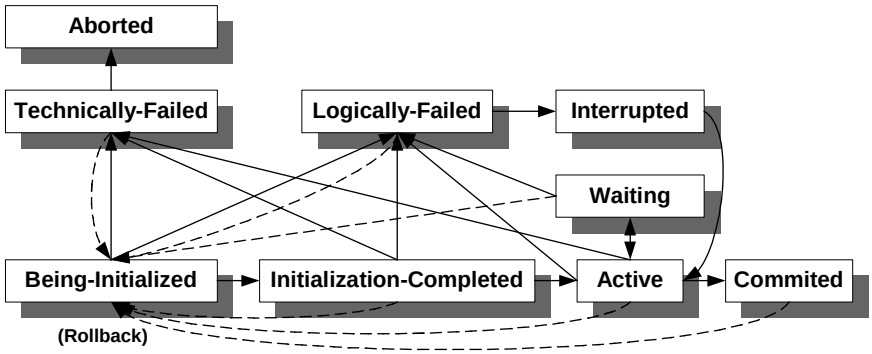


Abbildung 25 : Lebenszyklus der Workflows

Der initiale Status einer Workflow-Instanz ist immer *Being-Initialized*, der angenommen wird, wenn ein Benutzer aus einer Workflow-Definition eine Workflow-Instanz erzeugen lässt. Dieser Status wird beibehalten, solange die Workflow-Definition in die Workflow-Instanz kopiert wird, die Statusinformationen für Knoten, Kanten und Anwendungsprogramme gesetzt und die globalen Daten bereitgestellt werden.

Wenn während dieser Initialisierung ein Fehler auftritt, weil zum Beispiel globale Daten nicht verfügbar sind oder die Verbindung zur Datenbank unterbrochen ist, dann wird der Zustand der Workflow-Instanz auf den Status *Technically-Failed* gesetzt. In diesem Fall wird der Benutzer über die gescheiterte Initialisierung der Workflow-Instanz informiert. Anschließend kann der Benutzer entscheiden, ob der Workflow zurückgesetzt oder beendet werden soll. Im ersten Fall wird versucht die Initialisierung der Workflow-Instanz erneut durchzuführen, um den Workflow dennoch auszuführen. Im zweiten Fall wird die Workflow-Instanz in den Zustand *Aborted* überführt. Der Status *Aborted* ist ein finaler Zustand, in dem alle Aktionen, die bis zu diesem Augenblick ausgeführt wurden, rückgängig gemacht werden und die Workflow-Instanz anschließend gelöscht wird.

Im Normalfall wird die Initialisierung aber erfolgreich sein, dann wird der Status der Workflow-Instanz auf *Initialization-Completed* gesetzt. Dieser Zustand besagt, dass die Workflow-Definition erfolgreich in die Workflow-Instanz kopiert wurde, für alle Knoten, Kanten und Anwendungsprogramme der initiale Status gesetzt wurde und alle globalen Daten zur Verfügung stehen. In diesem Moment ist die Workflow-Instanz bereit, ausgeführt zu werden.

Üblicherweise wird nach der erfolgreichen Initialisierung die Workflow-Instanz vom Benutzer gestartet und die Instanz wechselt in den Status *Active*. Dieser Zustand sagt aus, dass die Workflow-Instanz in diesem Moment ausgeführt wird.

Die Workflow-Instanz wechselt in den Zustand *Waiting*, wenn der Benutzer die Ausführung des Workflows stoppt. Dies bedeutet, dass der Kontrollfluss nicht weiter abgearbeitet wird, keine Einträge in der Worklist mehr erzeugt werden und keine weiteren Aktivitäten gestartet werden. In dieser Situation wird darauf gewartet, dass alle laufenden Aktivitäten zum Ende kommen. Anschließend wird die Workflow-Instanz gesichert und dem Workflow alle Ressourcen entzogen. Die Workflow-Instanz bleibt in diesem Zustand, bis der Benutzer den Workflow explizit fortfahren lässt oder bis zum Ablauf einer bestimmten Zeitspanne.

Neben der Suspendierung einer Workflow-Instanz ist es immer auch möglich die Workflow-Instanz abzubrechen, weil zum Beispiel der Workflow nicht mehr notwendig ist. In dieser Situation wird die Ausführung des Workflows unterbrochen und die Workflow-Instanz in den Zustand *Technically-Failed* überführt. Anschließend werden alle Aktivitäten, die während der Ausführung des Workflows ausgeführt wurden, rückgängig gemacht und die Workflow-Instanz gelöscht, indem die Workflow-Instanz in den Zustand *Aborted* wechselt.

Falls während der Ausführung kein Fehler aufgetreten ist und der Kontrollfluss des Workflows den End-Knoten erreicht, dann ist die Workflow-Instanz beendet und wird in den Zustand *Commited* versetzt. Außerdem werden alle Informationen der Workflow-Instanz gesichert und dem Workflow alle Ressourcen entzogen. Dadurch, dass alle Informationen des Workflows gespeichert werden, kann im nachhinein immer nachvollzogen werden, was während der Ausführung des Workflows geschehen ist. Speziell im medizinischen Anwendungsbereich ist es wichtig, dass alle Aktionen, die während der Ausführung des Workflows durchgeführt wurden, protokolliert werden, um eine Dokumentation der Behandlung sicherzustellen.

Bis jetzt wurde nur diskutiert, dass der Workflow erfolgreich ausgeführt oder abgebrochen wird. Natürlich ist es jederzeit möglich, dass während der Lebensdauer eines Workflows ein *logischer* oder *technischer Fehler* auftritt.

Im Falle eines *technischen Fehlers* wird die Ausführung der Workflow-Instanz unterbrochen und in den Zustand *Technically-Failed* versetzt. Im Anschluss wird der Benutzer informiert, der entscheidet, ob der Workflow zurückgesetzt oder abgebrochen werden soll. Beim Abbruch des Workflows wird der Zustand der Workflow-Instanz auf

Aborted gesetzt. Wenn der Workflow zurückgesetzt werden soll, wird der Zustand der Workflow-Instanz auf *Being-Initialized* gesetzt und die Ausführung des Workflows erneut gestartet.

Bei einem *logischen Fehler* wird ebenfalls die Ausführung der Workflow-Instanz unterbrochen, allerdings wird der Zustand auf *Logically-Failed* gesetzt. Im Anschluss wechselt die Workflow-Instanz in den Zustand *Interrupted* und der Control Agent wird informiert, dass der Workflow adaptiert werden kann. Im Zustand *Interrupted* bleibt die Workflow-Instanz, bis die Adaptation des Workflows abgeschlossen ist und der Workflow fortgefahren werden soll. Sobald die Workflow-Instanz wieder ausgeführt wird, wechselt der Zustand nach *Active*.

5.1.3.2 Lebenszyklus der Knoten

Während der Ausführung eines Workflows durchläuft nicht nur der gesamte Workflow verschiedene Zustände, sondern natürlich auch die in dem Workflow enthaltenen Knoten, Transitionen und Anwendungsprogramme. In der folgenden Abbildung werden die verschiedenen Zustände der Knoten und die Unterschiede zwischen Kontroll- bzw. Kommunikationsknoten auf der einen Seite und Aktivitätsknoten auf der anderen Seite gezeigt.

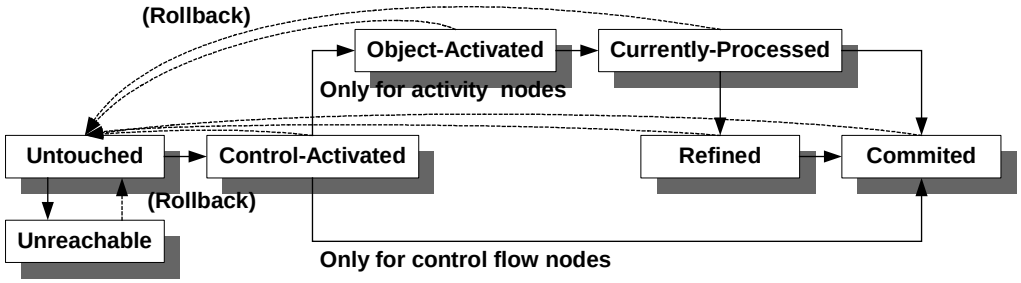


Abbildung 26 : Lebenszyklus der Knoten

Wenn eine Workflow-Instanz erzeugt wird, dann werden alle Knoten des Workflows am Anfang auf den Status *Untouched* gesetzt. Dies bedeutet, dass die Knoten im Kontrollfluss noch nicht erreicht sind und auch noch nicht verarbeitet wurden.

Wenn der Workflow gestartet ist und der Kontrollfluss durchlaufen wird, dann gibt es Knoten, die vom Kontrollfluss nicht mehr erreicht werden können. Dies sind zum Beispiel alle Knoten, die sich in einem Pfad einer Split-Join-Region[9] befinden, der nicht durchlaufen wird. Alle Knoten, die im Kontrollfluss nicht mehr erreicht werden können, werden auf *Unreachable* gesetzt.

Andernfalls, wenn der Knoten beim durchlaufen des Kontrollflusses erreicht wird und alle eingehenden Kanten aktiviert sind, wird der Status des Knotens auf *Control-Activated* gesetzt. Jetzt wird unterschieden, ob es sich bei dem aktuellen Knoten um einen Kontroll- bzw. Kommunikationsknoten oder um einen Aktivitätsknoten handelt.

Für den Fall, dass es sich um einen Kontroll- bzw. Kommunikationsknoten handelt, wird der Status des Knoten anschließend sofort auf *Commited* gesetzt. Damit ist der Knoten abgearbeitet und der Kontrollfluss kann fortfahren.

Andernfalls, wenn der Knoten ein Aktivitätsknoten ist, wird der Datenfluss überprüft. Dies bedeutet, dass geprüft wird, ob alle eingehenden Datenflusskanten die notwendigen Daten bereitstellen. Wenn alle Daten zur Verfügung stehen, wird der Knoten in auf *Object-Activated* gesetzt.

Nachdem der Kontrollfluss und der Datenfluss des Knotens aktiviert sind, kann die Aktivität gestartet werden. Es wurde bereits erwähnt, dass es zwei Arten von Aktivitäten gibt, einfache und komplexen Aktivitäten. Wenn es sich um eine einfache Aktivität handelt, dann geht der Knoten in den Zustand *Currently-Processed* über und die Anwendungsprogramme der Aktivität werden gestartet. Bei einer komplexen Aktivität geht der Status zwar zuerst auch in den Zustand *Currently-Processed* über. Wenn aber der dazugehörige Subworkflow gestartet wird, dann geht der Status in den Zustand *Refined* über. Dieser Zustand wird beibehalten, solange der Subworkflow ausgeführt wird.

Wenn die Aktivität, die dem Aktivitätsknoten zugrundegelegen hat, beendet ist, geht auch der Aktivitätsknoten in den Zustand *Commited* über.

Analog zum Workflow kann auch jeder einzelne Knoten zurückgesetzt werden. Dann wird der Zustand des Knotens auf *Untouched* gesetzt.

[9] Die Split-Join-Region ist der Bereich des Kontrollflusses zwischen einem Split-Knoten auf der einen Seite und dem dazugehörigen Join-Knoten auf der anderen Seite.

5.1.3.3 Lebenszyklus der Transitionen

Die Zustände, die eine Transition während der Lebensdauer durchlaufen, werden in der folgenden Abbildung dargestellt.

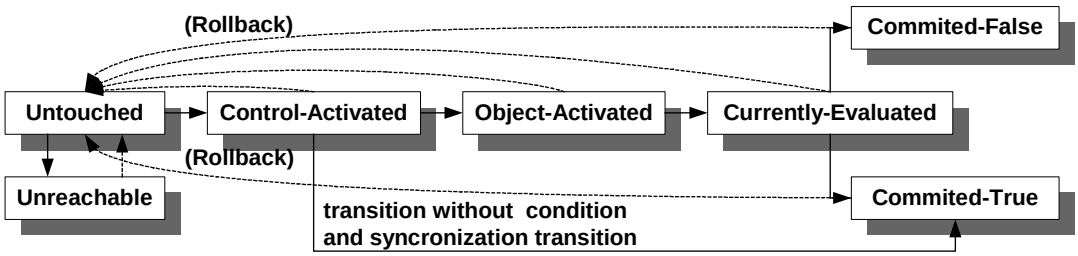


Abbildung 27 : Lebenszyklus der Transitionen

Wenn die Workflow-Instanz erzeugt wird, dann werden die Zustände aller Transitionen des Workflows auf *Untouched* gesetzt. Dies signalisiert, dass der Kontrollfluss die Transition noch nicht erreicht hat.

Vorausgesetzt, dass eine Transition vom Kontrollfluss nicht mehr erreicht werden kann, wird der Zustand der Transition aus *Unreachable* gesetzt. Das passiert zum Beispiel, wenn sich die Transition in einem Pfad einer Split-Join-Region befindet, der nicht verarbeitet wird. Eine gesonderte Stellung nehmen die Synchronisations-Transitionen ein, die anders als die normalen Transitionen auf *Commited-True* gesetzt werden, wenn sie im Kontrollfluss nicht mehr erreicht werden können. Dies hängt damit zusammen, dass es wenig Sinn macht zwei Aktivitäten miteinander zu synchronisieren, wenn eine der beiden Aktivitäten gar nicht ausgeführt wird.

Andernfalls wird bei Erreichen der Transition durch den Kontrollfluss der Zustand der Transition auf *Control-Activated* gesetzt. Der Folgezustand von *Control-Activated* ist davon abhängig, um was für eine Transition es sich handelt.

Eine normale Transition ohne eine Bedingung bzw. eine Synchronisations-Transition wird bereits aktiviert, wenn der Kontrollfluss die Transition erreicht. Dies bedeutet, dass die Transition nach Erreichen des Zustands *Control-Activated* sofort in den Zustand *Commited-True* übergeht, der gleichzeitig der Endzustand ist.

Eine normale Transition mit Bedingungen muss, bevor die Transition aktiviert werden kann, die Bedingungen auswerten. Das heißt, dass zuerst die erforderlichen Daten für die Auswertung der Bedingungen zur Verfügung stehen. Wenn die Daten bereitstehen, geht die Transition in den Zustand *Object-Activated* über. Anschließend werden die Bedingungen ausgewertet und die Transition befindet sich im Zustand *Currently-Evaluated*. In Bezug auf das Ergebnis der Bedingungen wird schließlich der Endzustand *Commited-True* oder *Commited-False* erreicht.

Wenn die Transition durch ein Rollback zurückgesetzt wird, dann wird der Zustand auf *Untouched* gesetzt.

5.1.3.4 Lebenszyklus der Anwendungsprogramme

Für die Ausführung und Unterbrechung von Workflows ist es wichtig zu wissen, in welchem Zustand sich die Anwendungsprogramme befinden, die im Rahmen des Workflows gerade ausgeführt werden. Die Art der Informationen, die dem Workflow-Management-System zur Verfügung stehen, hängt von der Art der Anwendungsprogramme ab. Bei der Vorstellung des Buildtime-Modells wurde bereits auf die vier verschiedenen Arten von Anwendungsprogrammen eingegangen. In der folgenden Abbildung sind die Zustände und Übergänge zwischen den Zuständen dargestellt, die ein Anwendungsprogramm annehmen kann.

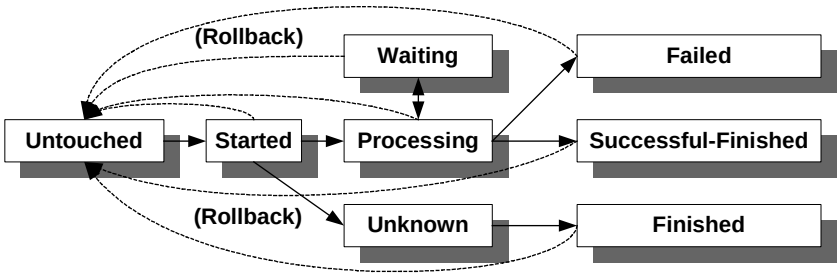


Abbildung 28 : Lebenszyklus der Anwendungsprogramme

Bei der Erzeugung einer Workflow-Instanz werden alle Anwendungsprogramme am Anfang auf *Untouched* gesetzt. Bis zum Start bleibt das Anwendungsprogramm in diesem Zustand.

Wenn bei der Verarbeitung des Kontrollflusses ein Aktivitätsknoten erreicht wird, dann wird die dazugehörige Aktivität gestartet. Im Falle einer einfachen Aktivität werden die zu der Aktivität assoziierten Anwendungsprogramme gestartet. Sobald ein Anwendungsprogramm gestartet wird, wechselt der Status des Anwendungsprogramms zu *Started*.

Nachdem ein Anwendungsprogramm gestartet ist, unterscheiden sich die Folgezustände in Abhängigkeit vom Typ des Anwendungsprogramms. Anwendungsprogramme, die während der Ausführung keinen Status zurückgeben können (Anwendungsprogramme vom Typ 1 & 2), gehen anschließend in den Zustand *Unknown* über. Alle anderen Anwendungsprogramme (Anwendungsprogramme vom Typ 3 & 4) gehen in den Folgezustand *Processing* über. Falls ein solches Anwendungsprogramm suspendiert wird, wird dies registriert und der Status des Anwendungsprogramms wird auf *Waiting* gesetzt. Im Gegenzug wird natürlich auch festgestellt, wenn das Anwendungsprogramm weiter ausgeführt wird und der Zustand wechselt nach *Processing* zurück.

Wenn das Anwendungsprogramm zum Ende kommt, erreicht es eine von drei Endzuständen. Auch hier ist es so, dass der Endzustand davon abhängt, von welchem Typ das Anwendungsprogramm ist. Im einfachsten Fall (Anwendungsprogramm vom Typ 1) kann vom Workflow-Management-System nur festgestellt werden, dass ein Anwendungsprogramm zum Ende gekommen ist. Diese Art von Anwendungsprogramm erreicht nur den Endzustand *Finished*. Alle anderen Arten von Anwendungsprogramme bieten weitergehende Informationen über das Ergebnis des Anwendungsprogramms (erfolgreiches Ende oder nicht). Mögliche Endzustände dieser Anwendungsprogramme sind *Successful-Finished* und *Failed*.

Falls es dazu kommt, dass der Workflow bzw. der Aktivitätsknoten zurückgefahren werden soll, dann müssen auch die in der Ausführung befindlichen Anwendungsprogramme zurückgesetzt werden. Dies ist allerdings nur für Anwendungsprogramme vom Typ 4 automatisch möglich, da nur diese Art von Anwendungsprogramm ein 2-Phasen-Commit Protokoll unterstützt. Alle anderen Arten von Anwendungsprogrammen unterstützen dieses Protokoll nicht und müssen im Zweifelsfall vom Benutzer

bzw. Administrator des Workflow-Management-Systems manuell zurückgesetzt werden.

5.1.4 Logische Fehler und Unterbrechungen

Der Schwerpunkt des Projekts AGENTWORK liegt aber in der Behandlung sogenannter logischer Fehler, weshalb die Behandlung dieser Fehlern im Blickpunkt der anschließenden Betrachtung liegen soll.

Bei der Vorstellung des Projekts AGENTWORK wurde bereits darauf eingegangen, dass die Erkennung von logischen Fehlern und die Adaptation der Workflow-Instanzen durch die agentenbasierte Schicht des Workflow-Management-Systems geleistet wird. Dem Control Agent kommt dabei die Aufgabe zu, die logischen Fehler zu erkennen und der Adaptation Agent baut die betroffenen Workflow-Instanzen um.

Die Workflow Engine hat in diesem Zusammenhang die Aufgabe, die Workflow-Instanzen zu unterbrechen, die von einem logischen Fehlern betroffen sind.

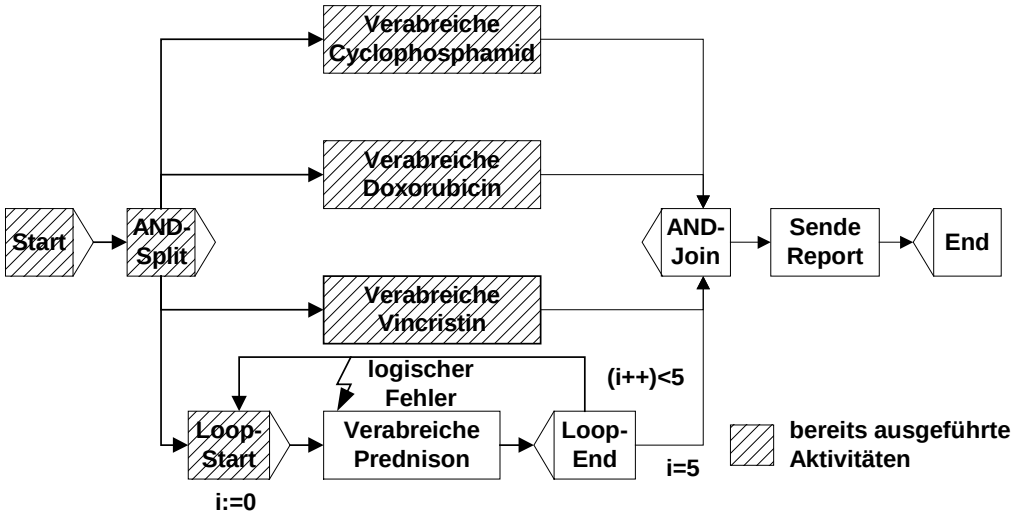


Abbildung 29 : Unterbrechung einer Workflow-Instanz

Das Unterbrechen einer Workflow-Instanz bedeutet, dass keine Aktivität mehr zur Ausführung ansteht und sich keine Aktivität in der Ausführung befindet.

Für das Beispiel (Abbildung 29) würde dies bedeuten, dass die Gabe aller noch nicht verabreichten Medikamente unterbleiben muss.

Die Workflow Engine muss zwei Schritte ausführen, um eine Workflow-Instanz zu unterbrechen:

- Die Ausführung aller anstehenden Aktivitäten, die im Zustand control-activated bzw. object-activated sind, muss unterbunden werden, indem keine neuen Work Items in die Worklist eingefügt werden.
- Die Ausführung aller Aktivitäten, die im Zustand currently-processed sind, muss unterbrochen werden.

Der erste Schritt zur Unterbrechung einer Workflow-Instanz stellt kein Problem dar, da die Aktivitäten noch nicht ausgeführt wurden. Dem gegenüber ist die Unterbrechung der Aktivitäten, die bereits ausgeführt werden, ein Problem.

Wenn eine Aktivität kein Anwendungsprogramm zugeordnet ist, dann muss der verantwortliche Benutzer informiert werden, um die Ausführung der Aktivität zu verhindern. Andernfalls ist eine Aktivität mit einer Reihe von Anwendungsprogrammen verknüpft, die in einer heterogenen und verteilten Umgebung ausgeführt werden. Im Rahmen des Projekts AGENTWORK werden vier verschiedene Arten von Anwendungsprogrammen unterschieden, die ganz unterschiedliche Möglichkeiten der Steuerung bieten. Bei der Unterbrechung der Aktivität müssen die Anwendungsprogramme demzufolge ebenfalls unterbrochen werden. Die Unterbrechung eines Anwendungsprogramms ist davon abhängig, welche Möglichkeiten der Steuerung die Anwendung hat. Wenn die Anwendung kein Rollback oder Kompensation unterstützt, muss der verantwortliche Benutzer bzw. Administrator informiert werden, um die Anwendung manuell zu unterbrechen und die Aktionen der Anwendung rückgängig zu machen. Alle Anwendungsprogramme, die ein Rollback unterstützen oder eine Kompensationsfunktion zur Verfügung stellen, können automatisch von der Workflow Engine unterbrochen werden. Dabei ist die Möglichkeit der Kompensation minimale Vorraussetzung für die automatische Unterbrechung einer Aktivität. In vielen Fällen, wie zum Beispiel bei der Gabe eines Medikaments, gibt es auch keine andere Möglichkeit, als die Kompensation. Erst wenn alle Anwendungsprogramme einer Aktivität unterbrochen sind, ist auch die Aktivität unterbrochen.

Ein weiteres Problem stellen die Daten dar, die von den Anwendungsprogrammen während der Ausführung erzeugt oder manipuliert wurden. Dem Workflow-Management-System müssen nur die Daten bekannt sein, die für die Ausführung der Workflow-Instanz benötigt werden. Alle anderen Daten müssen die Anwendungsprogramme dem System nicht bekannt machen. Dies hat aber zur Folge, dass auch nur die Anwendungsprogramme Änderungen auf den Daten rückgängig machen können. Dies wird problematisch, wenn ein Anwendungsprogramm weder Transaktionen unterstützt, noch eine Kompensation.

Für die Unterbrechung der Aktivitäten ergeben sich damit zwei verschiedene Ansätze:

- Ideal wäre, wenn nur die Aktivitäten unterbrochen würden, die von dem logischen Fehler betroffen sind und alle anderen aktiven Aktivitäten bis zum Ende ausgeführt würden. Dies würde das Problem der Unterbrechung von Anwendungsprogrammen auf ein Minimum reduzieren. Der Nachteil dieses Ansatz wäre allerdings, dass die Adaptation bis zum Ende aller Aktivitäten warten müsste und dies im Zweifel ein langwieriger Prozess sein kann. Dies kann bei akuten Ereignissen, die eine sofortige Reaktion erfordern, keine adäquate Lösung sein. Speziell, da die Workflow Engine keine genauen Aussagen über die Dauer der Ausführung einzelner Aktivitäten machen kann, dies ist problematisch.

  Für das Beispiel würde das bedeuten, dass nur die Aktivität "Verabreiche Prednison" unterbrochen wird und alle anderen Aktivitäten bis zum Ende ausgeführt werden. Leider kann die Workflow Engine keine Aussagen über die Dauer der anderen aktiven Aktivitäten treffen, so dass bei einem dringenden Ereignis die Unterbrechung der Workflow-Instanz zu lange dauern kann. Wenn der Patient zum Beispiel allergisch auf das Medikament Prednison reagiert, dann muss sofort gehandelt werden und kann nicht gewartet werden, bis alle anderen Medikamente verabreicht wurden.

- Ein anderer Ansatz ist, dass alle aktiven Aktivitäten sofort unterbrochen werden, unabhängig davon, ob sie vom logischen Fehler betroffen sind oder nicht. Dies impliziert, dass auch alle aktiven Anwendungsprogramme der Workflow-Instanz sofort beendet werden müssen. Alle Anwendungsprogramme, die sich automatisch zurückfahren lassen, müssen vom Workflow-Management-System

unterbrochen werden und für alle anderen Anwendungsprogramme muss der Administrator benachrichtigt werden, damit dieser die Anwendungsprogramme beendet. Der Vorteil dieser Variante ist, dass die Unterbrechung sehr schnell erfolgt und somit schnell auf akute Ereignisse reagiert werden kann. Demgegenüber steht der Nachteil, dass dadurch Aktivitäten wiederholt ausgeführt werden müssen, die nicht von diesem Ereignis betroffen waren.

Am Beispiel würde dieses Verfahren bedeuten, dass alle Aktivitäten unterbrochen werden und der Patient kein Medikament verabreicht bekommt, bis die Änderungen am Workflow vorgenommen sind. Der Nachteil diese Verfahrens ist, dass auch die Gabe der Medikamente abgesetzt wird, die nicht von dem logischen Fehler betroffen sind. Im schlimmsten Fall könnte dies heißen, dass der Patient lebenswichtige Medikamente nicht mehr erhält, weil deren Verabreichung unterbrochen wurde.

Im derzeitigen Stadium des Projekts AGENTWORK wird für die Unterbrechung der Workflow-Instanzen der zweite Ansatz verfolgt. In der Zukunft wäre es wünschenswert, wenn die beiden Ansätze kombiniert werden könnten. Ziel müsste es sein, nur die aktiven Aktivitäten zu unterbrechen, die von dem logischen Fehler betroffen sind oder deren Ausführungsdauer zu lang ist. Dazu müsste aber eine Entscheidung über die Dringlichkeit des Ereignisses getroffen werden, um eine maximale Wartezeit festzulegen, bis die Workflow-Instanz unterbrochen ist.

5.2 Worklist

Die *Worklist* enthält eine Liste der *Work Items* für die Benutzer des Workflow-Management-Systems. Ein *Work Item* ist in diesem Kontext die Repräsentation eines Arbeitsschrittes innerhalb eines Workflows, der von einem bestimmten Benutzer des Workflow-Management-Systems ausgeführt werden muss. Die Worklist stellt damit die Schnittstelle zwischen Worklist Handler[10] auf der einen Seite und Workflow Engine auf der anderen Seite.

[10] Der Worklist-Handler ist in den meisten Implementierungen von Workflow-Management-Systemen Teil des Workflow-Clients.

Im einfachsten Fall funktioniert die Worklist so, dass die Workflow Engine für jeden Arbeitsschritt, der von einem Benutzer ausgeführt werden muss, die erforderlichen Work Items in die Liste einfügt und der Worklist Handler die Work Items aus der Liste abruft, um sie dem Benutzer zur Ausführung der Arbeitsschritte anzuzeigen.

Durch die Integration des Worklist Handlers in den Workflow Client ergibt sich, dass der Worklist Handler verteilt arbeitet. Dies hat zur Folge, dass es verschiedene Möglichkeiten für die Implementierung der Worklist gibt, um diese zu unterstützen. In der Folge sollen vier verschiedene Ansätze (Abbildung 30) erläutert werden [HOL97].

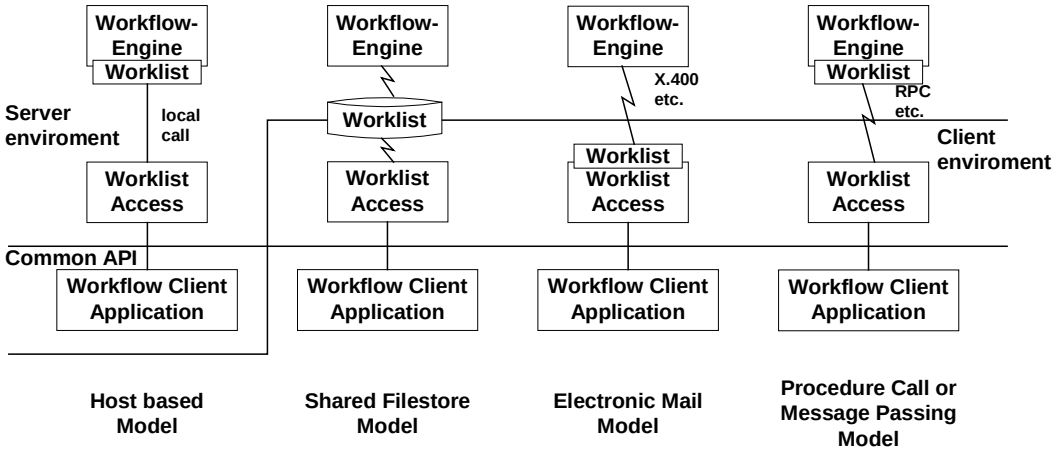


Abbildung 30 : Verschiedene Ansätze für die Worklist nach [HOL97]

- **Host based Model**

  Das erste Modell geht von einem Host-basierten Ansatz aus, wobei Workflow Engine und Workflow Client auf dem Host laufen und das User Interface per Terminal oder Remote Login zum Benutzer exportiert wird. In diesem Fall befinden sich sowohl Worklist, als auch Worklist Handler auf dem gleichen Rechner und die Kommunikation zwischen den Komponenten kann lokal erfolgen.

- **Shared Filestore Model**

  Das zweite Modell geht erstmalig von einer verteilten Umgebung aus, in der Workflow Engine und Workflow Client nicht auf dem gleichen Rechner ausgeführt werden. Die Kommunikation zwischen Worklist und Worklist Handler soll

allerdings nicht direkt erfolgen, sondern über eine gemeinsame Datenquelle[11]. Diese gemeinsame Datenquelle liegt zwischen Worklist und Worklist Handler und es kann von beiden Seiten darauf zugegriffen werden.

- **Electronic Mail Model**

  Das dritte Modell geht davon aus, dass die Verteilung von Work Items an die Benutzer analog zum Versenden von elektronischer Post (E-Mail) ist. Aus diesem Grunde werden die Mechanismen der elektronischen Post zur Kommunikation zwischen Worklist und Worklist Handler verwendet. Bei diesem Ansatz ist es allerdings erforderlich, die Work Items an alle Clients zu senden, die dann entscheiden, welche Work Items lokal ausgeführt werden müssen. Dies hat zur Folge, dass in diesem Szenario die Worklist üblicherweise auf Seiten des Clients angesiedelt ist.

- **Procedure Call oder Message Passing Model**

  Das letzte Modell geht von einer Client/Server-Kommunikation zwischen Worklist und Worklist Handler aus. Die Art der Kommunikation, ob Sockets oder Remote Procedure Call, ist hierbei der konkreten Implementierung überlassen. Der Vorteil dieses Ansatzes ist es, dass die Worklist auch als eigenständige Komponente implementiert werden kann. Dies bietet sich speziell dann an, wenn mehrere Workflow Engines eingesetzt werden sollen, die eine gemeinsame Worklist benutzen.

Im Rahmen des Projektes AGENTWORK wurde sich für das letzte Modell entschieden, weil es die größtmögliche Freiheit bezüglich der Implementierung bietet. Im momentanen Stadium des Projektes AGENTWORK wird davon ausgegangen, dass nur eine einzelne Workflow Engine einzusetzt wird. Es ist aber zu einem späteren Zeitpunkt durchaus denkbar mehrere Workflow Engines zu koppeln, um die Performance und die Skalierbarkeit des Workflow-Management-Systems zu verbessern. In dieser Situation ließe sich mit dem Message-Passing Modell auf einfache Weise die Worklist als

[11] Die Art der Datenquelle ist nicht festgelegt, d.h. es kann sich dabei um eine Datei, Datenbank oder etwas ähnlichen handeln.

eigenständige Komponente realisieren, die von allen Workflow Engines gemeinsam genutzt werden könnte.

5.3 Workflow Clients

Der Workflow Client ist, neben dem Workflow Editor, das zweite graphische Werkzeug, über das der Benutzer mit dem Workflow-Management-System interagiert.

Der Benutzer nutzt den Workflow Client dazu, um zwei Arten von Aufgaben durchzuführen:

- Administrative Aufgaben, wie das Erzeugen, Ausführen oder Abbrechen von Workflow-Instanzen
- Benutzerspezifische Aufgabe, wie die Ausführung von Aktivitäten

Um diese Aufgaben realisieren zu können, muss der Workflow Client sehr eng mit der Workflow Engine und der CORBA Management Layer zusammenarbeiten. Die CORBA Management Layer wird benötigt, um Anwendungsprogramme auszuführen und die notwendigen Daten für die Anwendungsprogramme bereitzustellen. Die Workflow Engine wird gebraucht, um einerseits die administrativen Aufgaben nachzukommen und andererseits, um die Work Items aus der Worklist abzurufen. Dabei repräsentieren die Work Items die Aktionen, die der Benutzer ausführen muss.

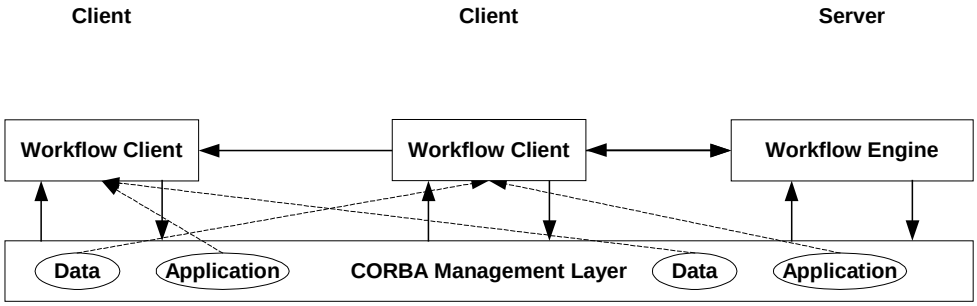


Abbildung 31 : Workflow Clients

5.3.1 Worklist Handler

Der Worklist Handler ist eine Softwarekomponente, die Teil des Workflow Clients ist. Aufgabe des Worklist Handlers ist die Kooperation mit der Worklist auf Seiten des Servers. Einerseits ruft der Worklist Handler alle Work Items aus der Worklist ab, die der Benutzer ausführen soll und andererseits informiert er die Worklist über den Status der ausgeführten Work Items.

5.3.2 User Interface

Das User Interface ist die graphische Oberfläche des Workflow Client. Über diese Schnittstelle arbeitet der Benutzer mit dem Workflow-Management-System. Bevor der Nutzer mit dem System arbeiten kann, muss er sich anmelden. Mittels der Anmeldung am System wird der Benutzer einem Rechner zugeordnet, so dass der Benutzer die Aufgaben erhält, die für ihn bestimmt sind. Die Work Items, die der Worklist Handler von der Worklist abruft, werden in der Form eines E-Mail Clients sortiert aufgelistet. Der Benutzer kann dann durch Anklicken dieser Einträge der Liste die entsprechende Aufgabe ausführen und die notwendigen Anwendungsprogramme starten. Nachdem die Aufgabe erledigt ist, wird sie aus der Liste entfernt und wird vom Workflow-Management-System als beendet angesehen. Am Ende der Arbeit meldet sich der Benutzer am System ab, um zu signalisieren, dass er keine weiteren Aufgaben ausführt. Das Workflow-Management-System ordnet dann die Aufgaben anderen Benutzern zu, die diese ausführen können.

6 Database Access Layer

Dieses Kapitel diskutiert die Probleme, die bei der Entwicklung einer objektorientierten Anwendung entstehen, wenn diese eine relationale oder objektrelationale Datenbank zur Speicherung der Daten verwendet. Das hauptsächliche Problem besteht darin, dass die Konzepte der objektorientierten Programmiersprache und der relationalen Datenbank voneinander differieren. Ein möglicher Ansatz zur Lösung dieses Problems stellt eine sogenannte *Database Access Layer* dar.

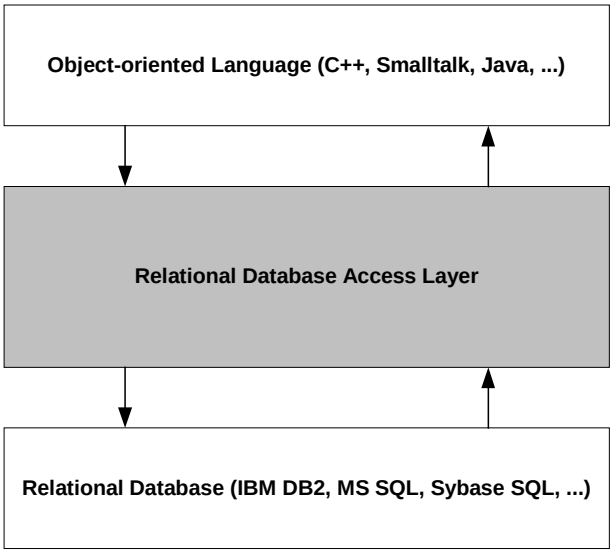


Abbildung 32 : Database Access Layer

Die Database Access Layer bildet dabei das Bindeglied zwischen objektorientiertem Paradigma und dem Konzept der relationalen Datenbank. Für das Anwendungsprogramm bedeutet dies, dass die Database Access Layer die Persistenz und Konsistenz der Objekte garantiert und die datenbankspezifischen Speicherungsstrukturen verbirgt.

Im Rahmen des Projektes AGENTWORK wird die Database Access Layer dazu verwendet, um Workflow Editor und Workflow Engine eine Schnittstelle zur Buildtime- und Runtime-Datenbank zur Verfügung zu stellen. Damit soll das Erzeugen, Ändern und Löschen von Workflow-Definitionen und Workflow-Instanzen ermöglicht werden. Dadurch, dass der Quellcode zur Datenbank-Programmierung aus den Komponenten des

Workflow-Management-Systems extrahiert wurde, ist es relativ einfach zu einem späteren Zeitpunkt auch anderen Komponenten des System den Zugriff auf die Datenbanken zu ermöglichen. Speziell für Teile der agentenbasierten Schicht ist dies denkbar, um ihnen Zugang zu den Workflow-Instanzen zu ermöglichen.

In [TOP98] findet sich auch ein Beispiel für den Einsatz einer Database Access Layer im Rahmen eines kommerziellen Projektes. Dies zeigt, dass das Konzept der Database Access Layer als Bindeglied zwischen Anwendungsprogramm und relationalem Datenbanksystem allgemein anerkannt ist.

Im weiteren Verlauf des Kapitels wird die Architektur der Database Access Layer detaillierter beschrieben und auf Probleme und Lösungen bei der Realisierung dieser Komponente eingegangen.

6.1 Architektur der Database Access Layer

Der Aufbau der Database Access Layer lässt sich in der folgenden dreischichtigen Architektur darstellen (Abbildung 33).

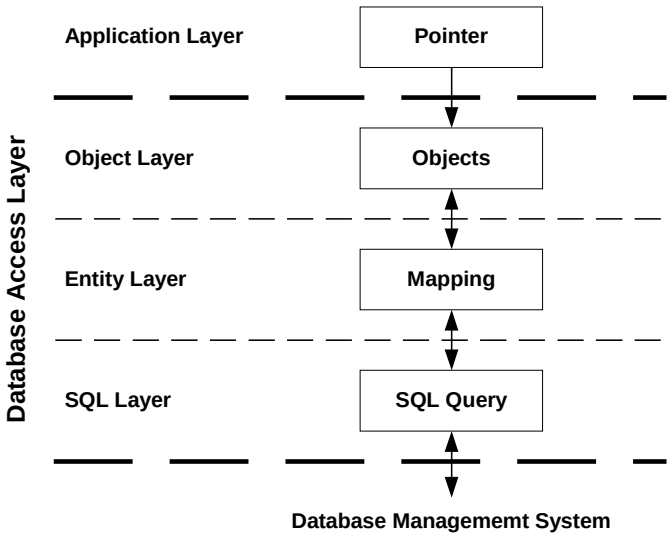


Abbildung 33 : Architektur der Database Access Layer

- **SQL Layer**

  Die SQL Layer ist die unterste Schicht der Database Access Layer und bildet die Schnittstelle zum zugrundeliegenden Datenbanksystem. Die Funktion dieser Schicht besteht darin, alle Aktionen auf der Datenbank auszuführen. Die Schicht ist in der Lage Datensätze zu lesen, einzufügen, zu ändern und zu löschen. Die dazu notwendigen SQL-Statements werden von dieser Schicht erzeugt.

- **Entity Layer**

  Die Entity Schicht ist die zentrale Schicht der Database Access Layer und ihr fällt die wichtigste Aufgabe zu, nämlich die Abbildung der Objekt auf die dazugehörigen Relationen. Die Aufgabe ist dadurch so anspruchsvoll, dass zwischen verschiedenen Objekten Beziehungen bestehen, die auf Ebene der Relationen ebenfalls abzubilden sind (siehe 6.4 Komplexe Objekte).

- **Object Layer**

  Die oberste Schicht der Database Access Layer bildet die Object Layer, welche die Schnittstelle zur darrüberliegenden Anwendung bildet. Die Funktion dieser Schicht besteht darin, der Anwendung die benötigten Daten in Form von Objekten zur Verfügung zu stellen. Die Anwendung sieht von der darrunterliegenden Datenbank und den Datenstrukturen nichts. Die Anwendung kann vollständig auf Objekten arbeiten. Dies bedeutet, dass die Anwendung neue Objekte anlegen kann, den Inhalt von Objekten ändern kann und auch Objekte löschen kann. Die Objekt Layer muss die Objekte verwalten (siehe 6.3 Objekt-Management), die Persistenz der Daten gewährleisten (siehe 6.2 Persistenz von Objekten) und die dazu notwendigen Aktionen auf der Datenbank anstoßen.

Eine wichtige Voraussetzung für die Arbeit der Database Access Layer ist, dass alle Objekte, die während der Ausführung der darrüberliegenden Anwendung verwendet werden, von der Database Access Layer erzeugt und verwaltet werden, um die Persistenz der Objekte zugesichert werden kann. Wenn zum Beispiel die Workflow

Engine eine neue Workflow-Instanz erzeugt, dann muss die Database Access Layer die notwendigen Objekte erzeugen und verwalten, um später die Daten der Workflow-Instanz in der Runtime-Datenbank zu sichern.

Die Objekte, die die Database Access Layer verwaltet, werden solange nicht in die Datenbank gesichert, bis die darrüberliegende Anwendung dies explizit anfordert. Aus diesem Grunde ist es wichtig, dass die Anwendung die Sicherung der Daten anordnet, wenn ein konsistenter Zustand erreicht wird.

6.2 Persistenz von Objekten

Eine der wichtigsten Zusicherungen der Database Access Layer an die darrüberliegenden Anwendung ist, dass alle Objekte persistent sind. Diese Zusicherung bedeutet, dass nach einem Sicherungspunkt alle Daten der zu sichernden Objekte in der Datenbank gespeichert sind. Dies lässt sich auf zwei Arten erreichen.

Die erste Möglichkeit ist, jedem Objekt einen eindeutigen Status zuzuordnen. Der Objekt-Status gibt an, ob die Daten des Objekts bereits in der Datenbank gespeichert sind bzw. ob sich die Daten des Objekts seit der letzten Speicherung geändert haben. Bei Erreichen eines Sicherungspunktes wird dann anhand des Objekt-Status entschieden, wie mit den Daten des Objektes verfahren werden muss.

Jedes Objekt befindet in einem der folgenden Zustände:

- **active**

  Ein Objekt hat den Status *active*, wenn die Daten des Objekts aus der Datenbank ausgelesen wurden und seitdem nicht modifiziert wurden. Beim Erreichen eines Sicherungspunktes brauchen die Daten des Objekts nicht gesichert werden, da sie sich nicht geändert haben.

  Nach Erreichen eines Sicherungspunktes werden alle aktiven Objekte in den Status *active* versetzt, da sie die gleichen Daten wie die Datenbank enthalten. Nicht mehr benötigte Objekte werden hingegen aus dem Speicher entfernt, da deren Daten auch aus der Datenbank gelöscht wurden.

- **modified**

  Ein Objekt hat den Status *modified*, wenn die Daten des Objekts aus der Datenbank gelesen wurden und in der Zwischenzeit geändert wurden. Beim Erreichen eines Sicherungspunktes müssen die zugehörigen Daten in der Datenbank aktualisiert (UPDATE) werden.

- **deleted**

  Ein Objekt, das den Status *deleted* hat, wird nicht mehr benötigt. Die Daten dieses Objekts sind aber noch in der Datenbank und müssen beim Erreichen des nächsten Sicherungspunkt aus der Datenbank gelöscht (DELETE) werden.

- **new**

  Ein Objekt mit dem Status *new* ist neu erzeugt worden und die Daten dieses Objekts sind noch nicht in der Datenbank gespeichert. Im nächsten Sicherungspunkt in die Datenbank eingebracht (INSERT) werden.

- **new_deleted**

  Ein Objekt, mit dem Status *new_deleted* wurde neu erzeugt und wird in der Zwischenzeit aber nicht mehr benötigt. Ein solches Objekt hat keinen Einfluss auf die Datenbank und kann deswegen sofort aus dem Speicher entfernt werden.

Die folgende Abbildung (Abbildung 34) zeigt die möglichen Übergänge zwischen den verschiedenen Zuständen der Objekte. Die Abbildung zeigt außerdem, dass ein Objekt entweder neu erzeugt wird (*new*) oder aus Daten der Datenbank erstellt wird (*read*).

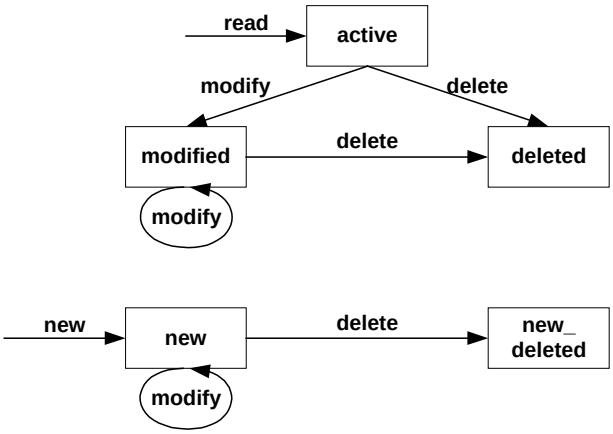


Abbildung 34 : Objekt-Status Diagramm

Der Vorteil dieses Verfahrens besteht darin, dass die Belastung des Datenbanksystems auf ein Minimum reduziert wird. Es werden immer nur die Daten in der Datenbank geändert, wo sich der Status des dazugehörigen Objekts geändert hat.

Die Nachteile dieses Verfahrens sind, dass jedes Objekt zusätzliche Informationen über den Status mitführen muss und nicht mehr benötigte Objekte erst bei Erreichen des nächsten Sicherungspunktes aus dem Speicher gelöscht werden können. Bei einer kleinen Anzahl von Objekten sind die Nachteile vertretbar und es empfiehlt sich die Verwendung dieses Verfahren. Wenn die Anzahl der Objekte größer wird und sich die Zeit zwischen den Sicherungspunkten verlängert, dann wird der Speicher-Overhead immer größer.

Die andere Möglichkeit benutzt eine andere Herangehensweise. Die Idee ist, alle Daten der zu sichernden Objekte bei jedem Sicherungspunkt neu in der Datenbank zu speichern. Das heißt, dass alle alten Daten der Objekte gelöscht werden und anschließend die neuen Daten in die Datenbank eingefügt werden. Alle abhängigen Objekte, die Beziehung zu den Objekten haben, müssen dabei mit gesichert werden, um die Beziehungen zwischen den Objekten zu erhalten.

Der Vorteil dieses Verfahrens ist, dass es keinen zusätzlichen Speicher-Overhead durch Statusinformationen und nicht mehr benötigten Objekten gibt.

Der Nachteil dieses Verfahrens ist aber, dass an einem Sicherungspunkt sehr viele Daten in die Datenbank eingebracht werden müssen. Dies kann zu einer hohen Belastung des Datenbanksystems führen.

6.3 Objekt-Management

Eine Aufgabe der Database Access Layer ist die Erzeugung und Verwaltung aller Objekte, deren Daten in der Datenbank gespeichert werden. Die Database Access Layer muss dazu garantieren, dass ein Objekt, dessen Daten in der Datenbank gespeichert sind, sich nur ein einziges mal im Speicher befindet. Ansonsten wäre es möglich, dass zwei Varianten eines Objektes existieren, die unabhängig voneinander modifiziert werden könnten. Das Ergebnis wäre ein inkonsistenter Zustand der Daten in der Datenbank, wenn die Objekte gesichert würden. Die Database Access Layer muss dafür sorgen, dass das nicht passiert.

Wenn die darrüberliegende Anwendung ein neues Objekt erzeugen möchte, wird von der Database Access Layer zuerst überprüft, ob das Objekt bereits im Speicher existiert. Wenn das Objekt bereits existiert, so bekommt die Anwendung einen Zeiger auf das Objekt. Im anderen Fall wird das Objekt erzeugt und mit den Daten aus der Datenbank gefüllt. Die darrüberliegende Anwendung erhält wiederum einen Zeiger auf das erzeugte Objekt. Diese Objektverwaltung ist eine Art von Cache-Mechanismus, bei dem nur dann Daten aus der Datenbank gelesen werden, wenn diese nicht bereits im Speicher sind.

Wenn die Workflow Engine zum Beispiel eine neue Workflow-Instanz erzeugen möchte, dann wird von der Database Access Layer zuerst geprüft, ob die Instanz mit dem angegebenen Namen nicht bereits im Speicher existiert. Wenn die Instanz bereits vorhanden ist, dann erhält die Workflow Engine einen Zeiger auf das Objekt. Ansonsten wird ein neues Objekt für die Workflow-Instanz angelegt und überprüft, ob eine Workflow-Instanz mit diesem Namen in der Datenbank gespeichert ist. Im einen Fall wird das Objekts mit den Daten aus der Datenbank gefüllt. Im anderen Fall wird eine komplett neue Workflow-Instanz erzeugt. Auf diese Weise lässt sich verhindern, dass jemals zwei Workflow-Instanz mit dem gleichen Namen im Speicher oder der Datenbank existieren. Analoges gilt natürlich auch für die Workflow-Definitionen.

6.4 Komplexe Objekte

Ein wichtiges Problem der Database Access Layer sind die Referenzen zwischen Objekten. Objekte, die Referenzen zu anderen Objekten besitzen, werden hier auch als *komplexe Objekte* bezeichnet.

Wenn man davon ausgeht, dass jede Klasse einer Tabelle der Datenbank entspricht und jedes Attribut einer Klasse einer Spalte der dazugehörigen Tabelle entspricht, dann bedeutet dies, dass die Beziehungen zwischen den Klassen auch zwischen den Tabellen der Datenbank existieren.

Für die Database Access Layer impliziert dies, dass sie diese Beziehungen unterstützen muss. Das folgende Beispiel zeigt eine solche Beziehung.

Beispiel:

```
Anwendung:
class point{
        X : integer;
        Y : integer;
};

class node{
        Type : NodeType;
        ...
        Position : Point;
};

Datenbank:
create table point(
        ID int,
        X int,
        Y int,
        primary key(ID)
)

create table node(
        Type char(10),
        ...
        Position int,
        foreign key (Position) references (point)
)
```

Auf Seiten der Anwendung gibt es zwei Klassen *point* und *node*, von denen die Klasse *node* eine Beziehung zur Klasse *point* hat. Diese Beziehung spiegelt sich dann im Datenbank-Schema in Form eines Fremdschlüssels wieder. Hier gibt es zwei Tabellen *point* und *node*. Ein Attribut der Tabelle *node* ist dabei ein Fremd-

schlüssel auf die Tabelle *point*. Damit wird die Beziehung zwischen den Klassen auch auf Ebene der Tabellen umgesetzt.

Wenn man vom Datenbank-Schema ausgeht, so gibt es drei Kardinalitäten, die es gilt zu unterscheiden:

- **1:1 Beziehungen**

  Eine 1:1 Beziehung bedeutet, dass ein Datensatz der einen Tabelle genau einem Datensatz der anderen Tabelle zugeordnet ist. Wenn man dies auf Objekte überträgt, so bedeutet dies, dass ein Objekt Unterobjekt eines anderen ist.

- **1:n Beziehungen**

  Eine 1:n Beziehung sagt aus, dass n Datensätze der einen Tabelle auf einen Datensatz der anderen Tabelle verweisen. Auf Objekte angewandt heißt dies, dass n verschiedene Objekte eine Referenz auf ein und dasselbe Objekt besitzen.

- **n:m Beziehungen**

  Eine n:m Beziehung besagt, dass n Datensätze der einen Tabelle auf m Datensätze der anderen Tabelle verweisen. Dabei ist es möglich, dass zwei Datensätze auf den selben Datensatz verweisen. Wenn man versucht dies auf Objekte zu übertragen, dann bedeutet dies, dass ein Objekt eine Reihe von Referenzen zu anderen Objekten hat, zu denen auch andere Objekte Referenzen haben können.

Die Database Access Layer muss diese Beziehungen und die Abbildung zwischen Klassen und Tabellen kennen, um diese Umsetzung vornehmen zu können.

7 Implementierung

In den letzten Kapiteln wurde das Konzept des Workflow-Management-Systems für dynamische Adaptation und speziell der Workflow Engine vorgestellt. Im nachstehenden Kapitel wird die Umsetzung dieses Konzepts in Form einer prototypischen Implementierung vorgestellt. Im ersten Teil wird das UML-Modell dargestellt, dass die Grundlage für die Implementierung ist. Auf Basis dieses UML-Modells wurden die Buildtime- und Runtime-Datenbank modelliert, um die notwendigen Daten der Workflow-Definitionen und Workflow-Instanzen persistent zu speichern. Anschließend werden die Implementierungen der Database Access Layer und der Workflow Engine exemplarisch vorgestellt.

7.1 UML-Modell

Dem Workflow-Management-System liegt ein gemeinsames Klassenmodell zugrunde. Dieses Modell beruht auf dem in Kapitel 3 vorgestellten Buildtime-Modell zur Modellierung einer Workflow-Definition. Für den Workflow Editor wird dieses Modell um die graphischen Informationen der Workflow-Definition erweitert. Dem gegenüber wird für die Workflow Engine dieses Modell um die notwendigen Statusinformationen für die Workflow-Instanzen erweitert.

Das UML-Modell definiert die Klassen für die drei Bereiche des Buildtime-Modells:

- Organisationsstrukturen
- Aktivitäten und Anwendungsprogramme
- Workflow-Definition mit Kontrollfluss und Datenfluss

Das UML-Modell wurde in Zusammenarbeit mit den anderen Beteiligten[12] des Projektes AGENTWORK erarbeitet.

[12] Robert Müller [MUE00], Ulrike Greiner [GRE00], Rainer Böhme [BOE00]

Die folgende Tabelle (Tabelle 8 : UML-Klassen) zeigt alle Klassen des UML-Modells mit einer kurzen Beschreibung der Klassen:

UML-Klasse	**Beschreibung**
Hilfsklassen	
Duration	Die Klasse definiert die Zeitspannen, die für Anwendungsprogramme, Transitionen, Kommunikationsknoten verwendet werden
Organisationsklassen	
Role	Die Klasse definiert die Rollen
Person	Die Klasse definiert die Workflow-Teilnehmer
Aktivitäts-Klassen	
Activity-Definition	Abstrakte Basisklasse aller Aktivitäts-Definitionen
Basic-Activity-Definition	Die Klasse definiert die einfachen Aktivitäts-Definitionen
Complex-Activity-Definition	Die Klasse definiert die komplexen Aktivitäts-Definitionen
Application	Die Klasse definiert die Anwendungsprogramme, die für eine einfache Aktivitäts-Definition benötigt werden
Kontrollfluss-Klassen	
Workflow-Definition	Die Klasse definiert die Workflow-Definitionen
Workflow-Node	Abstrakte Basisklasse aller Knoten
Comm-Node	Die Klasse definiert die Kommunikationsknoten
Control-Node	Die Klasse definiert die Kontrollflussknoten
Loop-Start-Node	Die Klasse ist von Kontrollflussknoten abgeleitet und definiert einen LoopStart-Knoten
Activity-Node	Die Klasse definiert die Aktivitätsknoten
Workflow-Transition	Die Klasse definiert die Transitionen
Datenfluss-Klassen	
Object-Specification	Die Klasse definiert die Objekt-Spezifikationen für die

	Beschreibung der Aktivitäts-Definitionen, Input- und Output-Objekten
Constant-Specification	Die Klasse ist von Objekt-Spezifikationen abgeleitet und definiert die konstanten Objekt-Spezifikationen
Manipulation-Query	Die Klasse definiert die Manipulation Queries für den externen Datenfluss zu einer externen Datenquelle
Retrieval-Query	Die Klasse definiert die Retrieval Queries für den externen Datenfluss von einer externen Datenquelle
Object-Path	Die Klasse definiert die Objekt-Pfade, die Teile der Objekte definieren
Method	Die Klasse definiert die Methoden, die Zugriff auf Teile der Objekte definieren
Component-Mapping	Die Klasse definiert die Komponenten-Abbildungen, die die Abbildungen von Output-Objekten auf Input-Objekte modellieren
Component-Constraint	Die Klasse definiert die Komponenten-Bedingungen, die einfache Bedingungen auf Objekten gestatten

Tabelle 8 : UML-Klassen

Das komplette UML-Modell mit allen Beziehungen zwischen den Klassen findet sich im Anhang (siehe Anhang A).

7.2 Datenbanken

In den vorangegangenen Kapiteln wurde bereits darauf eingegangen, dass die Daten des Workflow-Management-Systems persistent gespeichert werden müssen. Dies betrifft einerseits die Workflow-Definitionen und andererseits die Workflow-Instanzen. Die Daten werden in zwei voneinander getrennten Datenbanken gespeichert, der Buildtime-Datenbank und der Runtime-Datenbank.

Workflow-Definitionen und Workflow-Instanzen ist gemein, dass sie im Kern auf dem vorher vorgestellten UML-Modell basieren und dieses um die entsprechenden Daten für Buildtime bzw. Runtime erweitern. Das Ergebnis ist, dass beide Datenbanken eine sehr ähnliche Datenbankstruktur besitzen.

Im folgenden (Tabelle 9 : Tabellen der Datenbanken) werden die Tabellen aufgelistet, die in beiden Datenbanken existieren, und ihre Beziehung zu den UML-Klassen gezeigt.

Tabelle	**UML-Klassen**	**Beschreibung**
Duration	Duration	
Person	Person	
Role	Role	
PersonRole		Abbildung der n:m Beziehung zwischen Role und Person
Workflow	Workflow-Definition	
Activity	Activity-Definition, Basic-Activity-Definition, Complex-Activity-Defintion	
Node	Workflow-Node, Comm-Node, Control-Node, Loop-Start-Node, Activity-Node	
Application	Application	
Transition	Workflow-Transition	
RetrievalQuery	Retrival-Query	
ManipulationQuery	Manipulation-Query	
ObjectSpec	Object-Specification	
Method	Method	
ObjectPath	Object-Path	

ConditonValues		Abbildung der Beziehung zwischen Object-Specification, Object-Path und Workflow-Transition bzw. Object-Specification, Object-Path und Retrieval-Query
Parameters		Abbildung der n:m Beziehung zwischen Object-Path und Method
CompConstraint	Component-Constraint	
CompMapping	Component-Mapping	

Tabelle 9 : Tabellen der Datenbanken

Als Datenbanksystem, in dem die Datenbanken implementiert werden, wurde IBM DB2 Version 6.1 gewählt. Dieses Datenbankverwaltungssystem zählt zu den relationalen Datenbanksystemen.

Bei der Wahl des Datenbanksystems wurde bewusst auf die Verwendung eines objektorientierten Datenbanksystems verzichtet, weil die Zukunft einiger Systeme wie O2 unsicher erschien. Die Verwendung eines objektorientierten Datenbanksystems hätte zwar die Speicherung der Workflow-Definitionen und Workflow-Instanzen deutlich erleichtert, aber es wäre fraglich gewesen, ob in Zukunft diese Systeme weiterentwickelt werden bzw. eine Unterstützung dieser Systeme in Zukunft existieren wird.

7.2.1 Buildtime-Datenbank

Die Buildtime-Datenbank wird dazu verwendet, um die Workflow-Definitionen des Workflow Editors zu speichern. Die Workflow Engine greift auf diese Workflow-Definitionen zu, wenn sie aus einer Workflow-Definition eine Workflow-Instanz erzeugen soll.

Die Buildtime-Datenbank speichert sowohl die eigentliche Workflow-Definition, als auch die Positionen der graphischen Elemente der Workflow-Definition. Zu diesen graphischen Elementen zählen die Knoten, die Kanten des Kontrollflusses und die

Kanten des Datenflusses, mittels derer die Workflow-Definition graphisch modelliert werden kann.

Ein Knoten innerhalb der Workflow-Definition benötigt genau einen Punkt, der über x- und y-Koordinate ausgedrückt werden kann.

Die Kanten des Kontroll- und Datenflusses der Workflow-Definition dagegen bestehen aus mehreren Punkten, die zusammen die Kante ergeben. Die Punkte der Kanten werden jeweils in einer eigenen Tabelle gespeichert und besitzen eine Referenz auf die Kante zu der sie gehören. Die Reihenfolge der Punkte auf der Kante wird dadurch bestimmt, dass die Punkte nummeriert werden.

In der folgenden Tabellen (Tabelle 10 : Buildtime-Datenbank) werden die entsprechenden Tabellen der Buildtime-Datenbank mit den dazugehörigen Attribute aufgelistet und kurz beschrieben.

Tabelle	**Attribut**	**Beschreibung**
Node	xPosition	x-Koordinate des Knotens
	yPosition	y-Koordinate des Knotens
TransitionPosition	ID	Fremdschlüssel auf die Tabelle Transition, in der die Workflow-Transitionen gespeichert werden
	Number	Reihenfolge der Punkte auf der Kante
	xPosition	x-Koordinate des Punktes
	yPosition	y-Koordinate des Punktes
MappingPosition	Source/Target	Fremdschlüssel auf die Tabelle CompMapping, in der die Component-Mappings gespeichert werden
	Number	Reihenfolge der Punkte auf der Kante
	xPosition	x-Koordinate des Punktes
	yPosition	y-Koordinate des Punktes

Tabelle 10 : Buildtime-Datenbank

Das komplette ER-Modell der Buildtime-Datenbank findet sich im Anhang B.

7.2.2 Runtime-Datenbank

In der Runtime-Datenbank werden im Gegensatz zur Buildtime-Datenbank die Workflow-Instanzen gespeichert, die von der Workflow Engine ausgeführt werden.

Die Runtime-Datenbank muss dadurch die Workflow-Definition um die entsprechenden Statusinformationen der Workflow-Instanz erweitern. Das bedeutet, dass für den Workflow, die Knoten, die Transitionen und die Applikationen die nötigen Statusinformationen gespeichert werden müssen. Zusätzlich zu den Statusinformationen werden in der Datenbank auch Informationen über den zeitlichen Ablauf der Workflow-Instanz gespeichert. Für Aktivitätsknoten wird außerdem auch noch die verantwortliche Person und Rolle gespeichert, die im Falle einer manuellen Aktivität dafür verantwortlich ist.

In der folgenden Tabellen (Tabelle 11 : Runtime-Datenbank) werden die entsprechenden Tabellen der Runtime-Datenbank mit den dazugehörigen Attribute aufgelistet und kurz beschrieben.

Tabelle	**Attribut**	**Beschreibung**
Workflow	Status	Status der Workflow-Instanz
	StartTime	Beginn der Ausführung der Workflow-Instanz
	FinishTime	Ende der Ausführung der Workflow-Instanz
Node	Status	Status des Knotens
	StartTime	Beginn der Ausführung des Knotens
	FinishTime	Ende der Ausführung des Knotens
	Role_	Rolle, die für die Ausführung des Knotens verantwortlich ist
	Person_	Person, die für die Ausführung des Knotens verantwortlich ist
Transition	Status	Status der Transition
	StartTime	Beginn der Transition
	FinishTime	Ende der Transition

ApplicationStatus	ApplicationID	Fremdschlüssel auf die Tabelle Application, in der die Applications gespeichert werden
	Status	Status des Applikation
	StartTime	Beginn der Ausführung der Applikation
	FinishTime	Ende der Ausführung der Applikation
ObjectSpec	Time	Zeit, wenn das dazugehörige Objekt

Tabelle 11 : Runtime-Datenbank

Das komplette ER-Modell der Runtime-Datenbank findet sich im Anhang B.

7.3 Database Access Layer

Die Database Access Layer ist das Bindeglied zwischen den beiden Datenbanken und den Komponenten des Workflow-Management-Systems. Die Database Access Layer hat dabei die Aufgabe, den Komponenten des Workflow-Management-Systems das Arbeiten auf Workflow-Definitionen und Workflow-Instanzen zu ermöglichen, ohne dass die Komponenten Informationen über das zugrundeliegende Datenbanksystem und die Struktur der Datenbanken besitzen.

Die Implementierung der Database Access Layer erfolgt mit Microsoft Visual C++ 6.0 und wird als Dynamic Link Libary (DLL) realisiert, so dass die Database Access Layer in alle Komponenten des Workflow-Management-Systems eingebunden werden kann.

Die Database Access Layer implementiert das Klassenmodell (Abbildung 35 : Klassen-Schema), das auf dem in Kapitel 6.1 vorgestellten UML-Modell beruht. Aufgrund der fehlenden CORBA Management Layer beschränkt sich das Klassen-Modell auf die Implementierung des Kontrollflusses und verzichtet dabei auf die Implementierung des Datenflusses. Durch das Klassenmodell ist die Database Access Layer in der Lage, Workflow-Definitionen bzw. Workflow-Instanzen als Objekte zu erzeugen und diese mit den Daten aus den beiden Datenbanken zu füllen. Die Komponenten des Workflow-Management-Systems, wie Workflow Engine und Workflow Editor, können anschließend auf diesen Objekten arbeiten.

Die drei Schichten[13] der Database Access Layer ziehen sich durch alle Klassen, die den Zugriff auf die Datenbanken und die wechselseitige Abbildung der Daten auf Objekte in ihrer Implementierung verbergen.

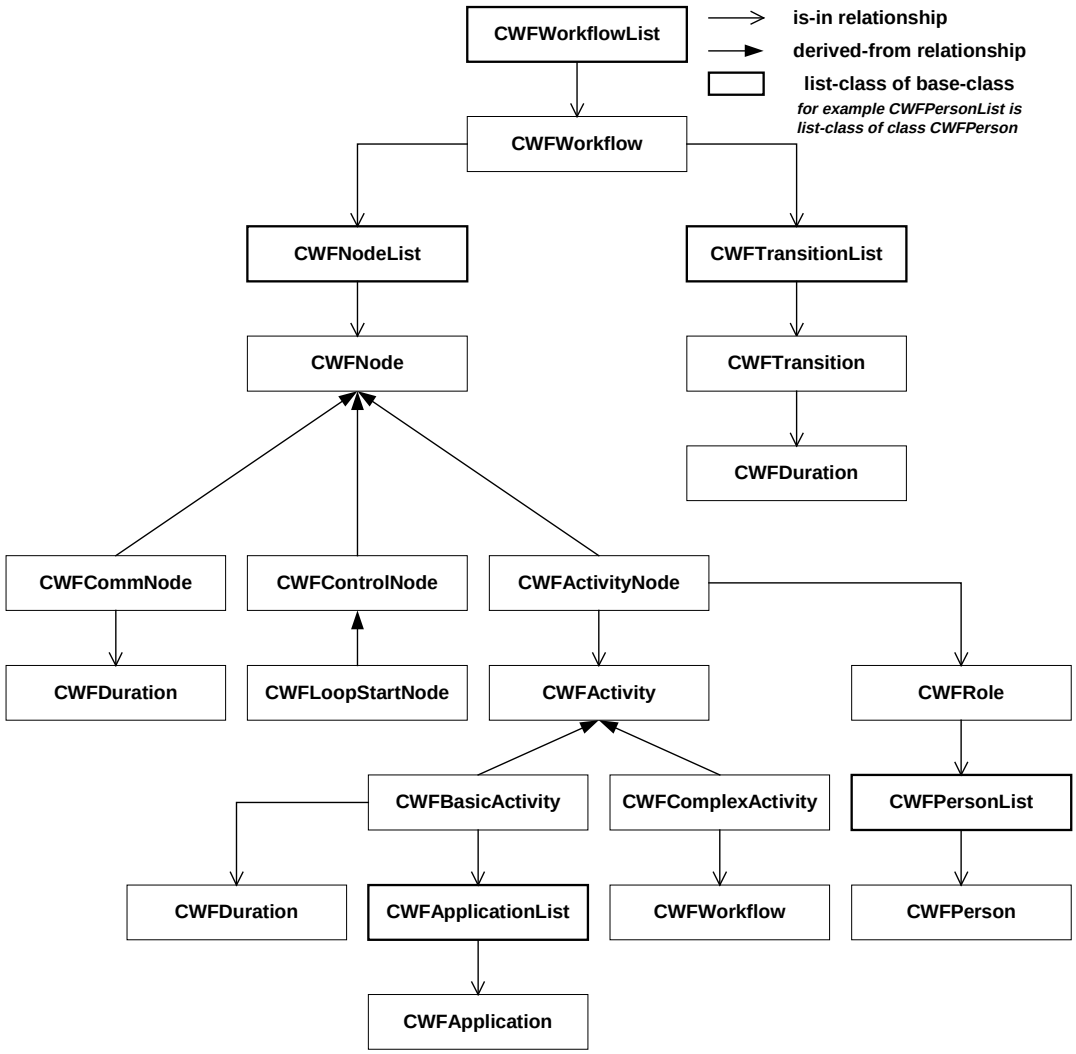


Abbildung 35 : Klassen-Schema

Die Objekte des Klassenmodells werden nach globalen und lokalen Objekten unterschieden. Globale Objekte sind Objekte, die in mehr als einem Workflow verwendet werden. Zu diesen Objekten gehören zum Beispiel CWFRole, CWFPerson und CWFActivity. Rollen, Personen und Aktivitäts-Definitionen werden unabhängig von einem Workflow definiert und können in allen Workflows verwendet werden. Dem

[13] SQL-Layer, Entity-Layer und Object-Layer

gegenüber stehen die lokalen Objekte, wie CWFNode oder CWFTransition. Diese Objekte werden nur im Kontext eines bestimmten Workflows definiert und können kein zweites mal verwendet werden. Dieser Unterschied zwischen globalen und lokalen Daten wirkt sich auf die Speicherung der Daten in den Objekten aus. Während Änderungen an lokalen Objekten erst beim Erreichen des nächsten Sicherungspunktes in die Datenbanken eingebracht werden, werden Änderungen an den globalen Objekten sofort in die Datenbanken geschrieben. Damit wird verhindert, dass Änderungen in den globalen Objekten zu inkonsistenten Zuständen in den Workflows führen.

Beim Erreichen eines Sicherungspunktes wird die Workflow-Definition bzw. Workflow-Instanz in die Datenbank geschrieben. Dabei wird der Workflow komplett neu in die Datenbank eingebracht. Das bedeutet, dass eine alte Version des Workflows vorher gelöscht wird. Dieses Verfahren wurde gewählt, weil damit der Speicher-Overhaed für die Statusinformationen gespart wird und sich die Beziehungen bei komplexen Objekten besser gewährleisten lassen. Das andere Verfahren würde einen deutlichen Mehraufwand verlangen, um die Beziehungen zwischen den Objekten bzw. Daten zu wahren.

7.4 Workflow Engine

Die Implementierung der Workflow Engine setzt auf der Implementierung der Database Access Layer auf. Die Implementierung der Workflow Engine erfolgt mittels Microsoft Visual C++ 6.0, das auch für die Database Access Layer bereits verwendet wird. Wie auch bei der Database Access Layer wird bei der prototypischen Umsetzung der Workflow Engine auf die Realisierung des Datenflusses verzichtet und nur der Kontrollfluss modelliert.

Durch das Fehlen der CORBA Management Layer ist die Funktionsweise der Workflow Engine nur begrenzt möglich, da die Anwendungsprogramme und Daten des Workflow-Management-Systems von der CORBA Management Layer verwaltet werden sollen.

Die Workflow Engine wird auf den Objekten der Workflow-Instanzen, die von der Database Access Layer bereitgestellt werden, den Kontrollfluss ablaufen und für die Aktivitäten Einträge in der Worklist erzeugen.

Im Rahmen der prototypischen Implementierung wird die Worklist als Teil der Workflow Engine realisiert, um die Erzeugung der Work Items zu kontrollieren. Die Worklist wird aber als separate Klasse realisiert, so dass es im weiteren Verlauf des Projektes AGENTWORK möglich ist, die Worklist als eigenständige Komponente zu realisieren. Speziell dann, wenn zu einem späteren Zeitpunkt der Einsatz mehrere Workflow Engines geplant ist, wird es nötig die Worklist aus der Workflow Engine zu extrahieren.

Die wichtigste Aufgabe, die mit der prototypischen Implementierung der Workflow Engine erreicht werden soll, ist der Test der Unterbrechungsmechanismen für Workflow-Instanzen. Die Unterbrechungsmechanismen lassen sich allerdings nicht alleine von der Workflow Engine realisieren, sondern erfordern die enge Zusammenarbeit mit der CORBA Management Layer, um die Ausführung der Aktivitäten und Anwendungsprogramme zu unterbrechen. Speziell die Unterbrechung der Anwendungsprogramme erfolgt innerhalb der CORBA Management Layer, die diese Anwendungsprogramme verwaltet und steuert. Soweit die Realisierung der Unterbrechungsmechanismen in der Workflow Engine möglich ist, erfolgt dies im Rahmen der Implementierung.

8 Zusammenfassung

8.1 Diskussion der Ergebnisse

Aufgabenstellung der vorliegenden Arbeit war die Konzeption und die prototypische Implementierung einer Workflow Engine für dynamische Adaptation. Dabei ließ sich das Ziel der Arbeit in zwei Komplexe gliedern, die Konzeption der Workflow Engine und anschließend die prototypische Realisierung der erarbeiteten Konzepte. Der Schwerpunkt lag hierbei auf der konzeptionellen Arbeit.

In der ersten Phase wurden die Anforderungen an das Workflow-Management-System für dynamische Adaptationen erarbeitet und diese Anforderungen in ein Buildtime-Modell umgesetzt. Das Buildtime-Modell definiert, welche Möglichkeiten der Workflow-Modellierung es gibt. Damit wird die Funktionsweise des gesamten Workflow-Management-Systems festgelegt. Das Modell wurde hinterher in ein UML-Modell überführt, welches die Klassen für Workflow-Definitionen und Workflow-Instanzen definiert.

Anschließend wurden auf Basis des UML-Modells die beiden ER-Modelle für die Buildtime- und Runtime-Datenbank modelliert. Die Buildtime-Datenbank wird zur Speicherung der Workflow-Definitionen verwendet und erweitert das UML-Modell um die graphischen Aspekte der Workflow-Definition. Die Runtime-Datenbank wird hingegen zur Speicherung der Workflow-Instanzen benutzt und erweitert das UML-Modell um die Statusinformationen der Workflow-Instanz.

Anschließend wurden die Architektur der Workflow Engine und die Algorithmen zur Erzeugung, Verarbeitung und Unterbrechung der Workflow-Instanzen entworfen.

In der zweiten Phase wurde die prototypische Implementierung der Workflow Engine realisiert. Dazu mussten drei Teile realisiert werden, die Buildtime- und Runtime-Datenbank, die Database Access Layer und die Workflow Engine.

Die Implementierungen der Database Access Layer und der Workflow Engine beschränkte sich dabei auf den Kontrollfluss und klammert den Datenfluss aus. Ursache dafür war das Fehlen der CORBA Management Layer, die zur Realisierung des Datenflusses benötigt wird. Auch die Unterbrechung konnte nur in soweit umgesetzt werden, wie dies im Rahmen der Workflow Engine möglich war. Die konkrete Unterbre-

chung der Anwendungsprogramme eines Workflows war zu diesem Zeitpunkt noch nicht möglich, da dies von der CORBA Management Layer durchgeführt werden muss.

Abschließend lässt sich aber trotzdem sagen, dass das Ziel der Diplomarbeit erreicht wurde.

8.2 Ausblick

Mit der Implementierung der Workflow Engine wurde eine der ersten Komponenten des Workflow-Management-Systems umgesetzt.

Aus Sicht der Workflow Engine ist der nächste Schritt die Implementierung der CORBA Management Layer, die die Anbindung des Workflow-Management-Systems an die Middleware darstellt. Erst wenn diese Schicht existiert, ist es möglich den Datenfluss zu realisieren und Anwendungsprogramme zu starten.

Für die prototypische Implementierung der CORBA Management Layer könnte die Implementierung des externen Datenflusses insofern vereinfacht werden, dass auf die Unterstützung von heterogenen Datenquellen verzichtet wird. In einer ersten Phase wäre es vorstellbar sich auf eine externe Datenquelle zu beschränken, um die Komplexität der CORBA Management Layer deutlich zu reduzieren. Dies würde es ermöglichen, dass das Workflow-Management-System bereits zu einem frühen Zeitpunkt im konkreten Anwendungsbereich eingesetzt und getestet werden könnte.

Im medizinischen Einsatzbereich des Workflow-Management-Systems geht der Trend zu einer zentralen Erfassung und Verwaltung der medizinischen Daten. Als Beispiel dient das Krankenhaus-Informations-Systeme (Medical Control Center) der Firma Meierhofer [MEI00], das eine zentrale Patienten Datenbank besitzt. Im Fall der Firma Meierhofer ist die zentrale Datenbank Oracle 8.0 oder Microsoft SQL Server.

Die nächsten Schritte bei der Entwicklung des Workflow-Management-Systems werden die Entwicklung eines Workflow Clients, die Implementierung der restlichen Agenten der Agentenbasierten Schicht und der Aufbau einer Wissensbasis für den Adaptations-Agent sein.

Wenn alle Komponenten des Workflow-Management-Systems existieren, müssen diese in das Gesamtsystem integriert werden und die entsprechenden Schnittstellen zwischen den Teilen implementiert werden. Wenn dies geschehen ist, kann auch das

gesamte Workflow-Management-System im medizinischen Anwendungsbereich getestet und optimiert werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[BOE00] Böhme, R.: *Konzeption und prototypische Implementierung eines Workflow Editors*, Diplomarbeit, Institut für Informatik, Universität Leipzig, 2000

[CHR+99] Cichocki, A.; Helal, A.; Rusinkiewicz, M.; Woelk, D.: *Workflow and Process Automation - Concepts and Technology*, Kluwer Academic Publishers, 1999

[EL96] Eder, J.; Liebhart, W.: *Workflow Recovery*, IEEE Computer Society Press, 124-134, June 1996, Institut für Informatik, Universität Klagenfurt, 1996

[GRE00] Greiner, U.: *Ein wissensbasierter Agent zur ereignisorientierten Adaptation von Workflows*, Diplomarbeit, Institut für Informatik, Universität Leipzig, 2000

[HOL97] Hollingsworth, D.: *Workflow Management Coalition – The Workflow Reference Model*, Workflow Management Coalition, 1997

[HR99] Härder, T.; Rahm, E.: *Datenbanksysteme Konzeption und Techniken der Implementierung*, Springer, 1999

[IBMCA98] *IBM MQSeries Workflow Concept and Architecture Version 3. 1*, IBM Cooperation, 1998

[IBMBT98] *IBM MQSeries Workflow Getting Started with Buildtime Version 3. 1*, IBM Cooperation, 1998

[IBMRT98] *IBM MQSeries Workflow Getting Started with Runtime Version 3. 1*, IBM Cooperation, 1998

[IBMDB99] *IBM DB2 SQL Reference,* IBM Cooperation, 1999

[KRU96] Kruglinski, D.: *Inside Visual C++ Version 4.0*, Microsoft Press, 1996

[KEL98] Keller, W.: *Object-Relational Access Layers - A Roadmap, Missing Links and more Patterns*, EA Generali, 1998, http://ourworld.compuserve.com/hompages/WofgangWKeller/

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

[LAW97] Lawrence, P.: *WfMC Workflow Handbook 1997*, John Wiley & Sons, 1997

[LR99] Leymann, F.; Roller, D.: *Production Workflow - Concepts and Techniques*, Prentice Hall, 1999

[MRW+98] Müller, R.; Heller, B.; Löffler, M.; Rahm, E.; Winter, A.: *AgentWork: A Knowledge- based Workflow System for Distributed Cancer Therapy*, Proceedings of the German Medical Informatics Conference (GMDS98), 63-66, 1998

[MM99] May, W.; Marrón, P.: *Florid*, Institut für Informatik, Universität Freiburg, 1999

Literaturverzeichnis

[MR99] Müller, R.; Rahm, E.: *Rule-Based Dynamic Modification of Workflows in a Medical Domain*, Proceedings of the German Database Conference (BTW 99), 429-448, 1999, http://dol.uni-leipzig.de/pub/1999-14

[MEI00] Meierhofer EDV Beratung: *http://www.meierhofer.de*, 2000

[MUE00] Müller, R.: *Event-Oriented Dynamic Adaptation of Workflows – Model, Architecture and Implementation*, Dissertation, Institut für Informatik, Universität Leipzig, 2000

[MR00] Müller, R.; Rahm, E.: *Dealing with Logical Failures for Collaborating Workflows,* Proceedings of Fifth International Conference on Cooperative Information Systems (CoopIS), Eilat, Israel, 2000

[OMG99] *The Common Object Request Broker: Architecture and Specification*, Object Management Group, Inc. (OMG), 1999

[RW91] Reuter, A.; Wächter, H.: *The ConTract Model*, Institut für Informatik, Universität Stuttgart, 1991

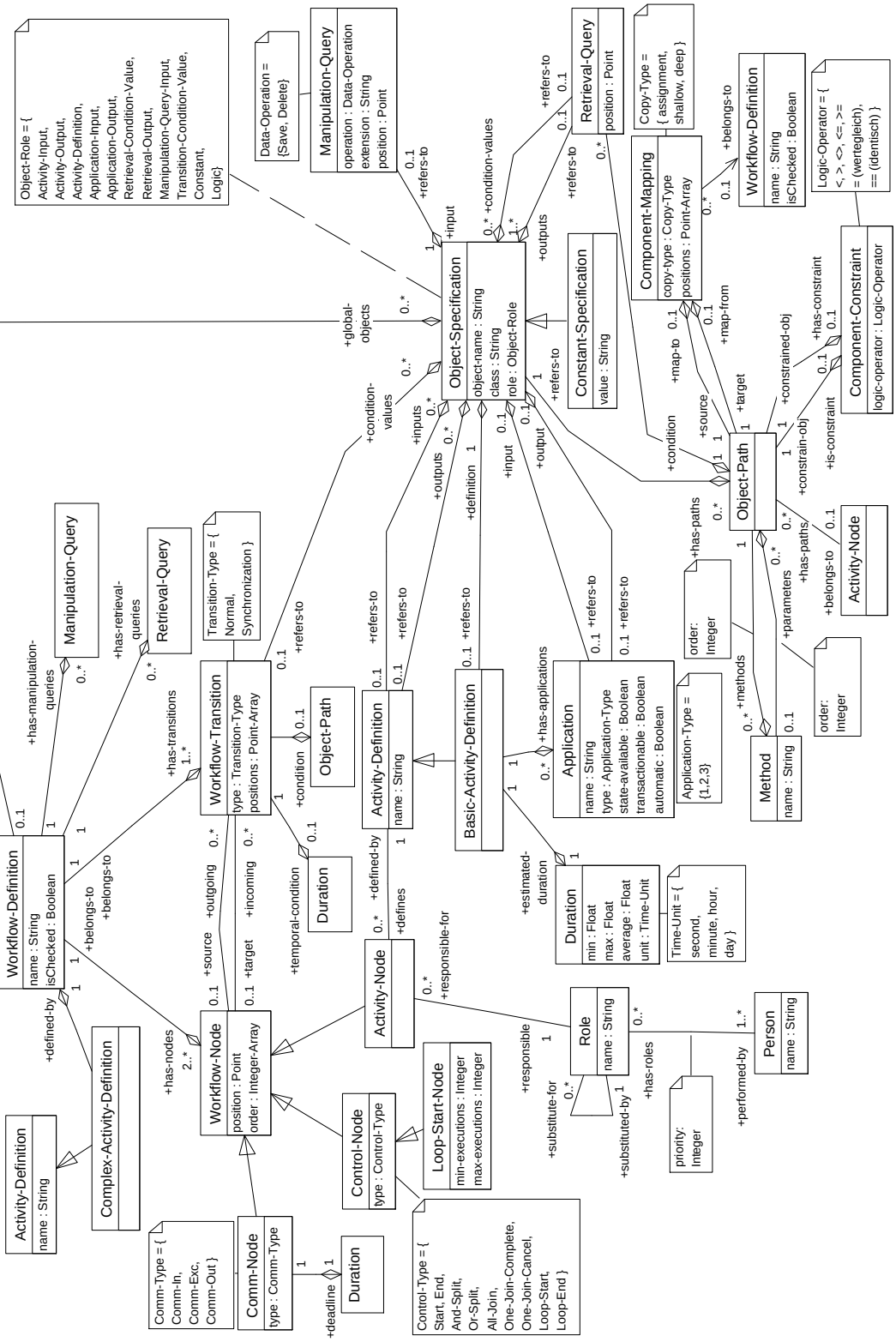
[RS95] Reuter, A.; Schwenkreis, F.: *ConTracts – A Low Level Mechanism for Building General-Purpose Workflow-Management-System*, Institut für Informatik, Universität Stuttgart, 1995

[TOT97] Toth, V.: *Visual C++ 5 – Das Kompendium*, Markt & Technik, 1997

[TOP98] Tophoven, B.: *Building an Object Relational Database Access Layer*, sd&m, 1998, http://www.sdm.de/e/www/index.htm

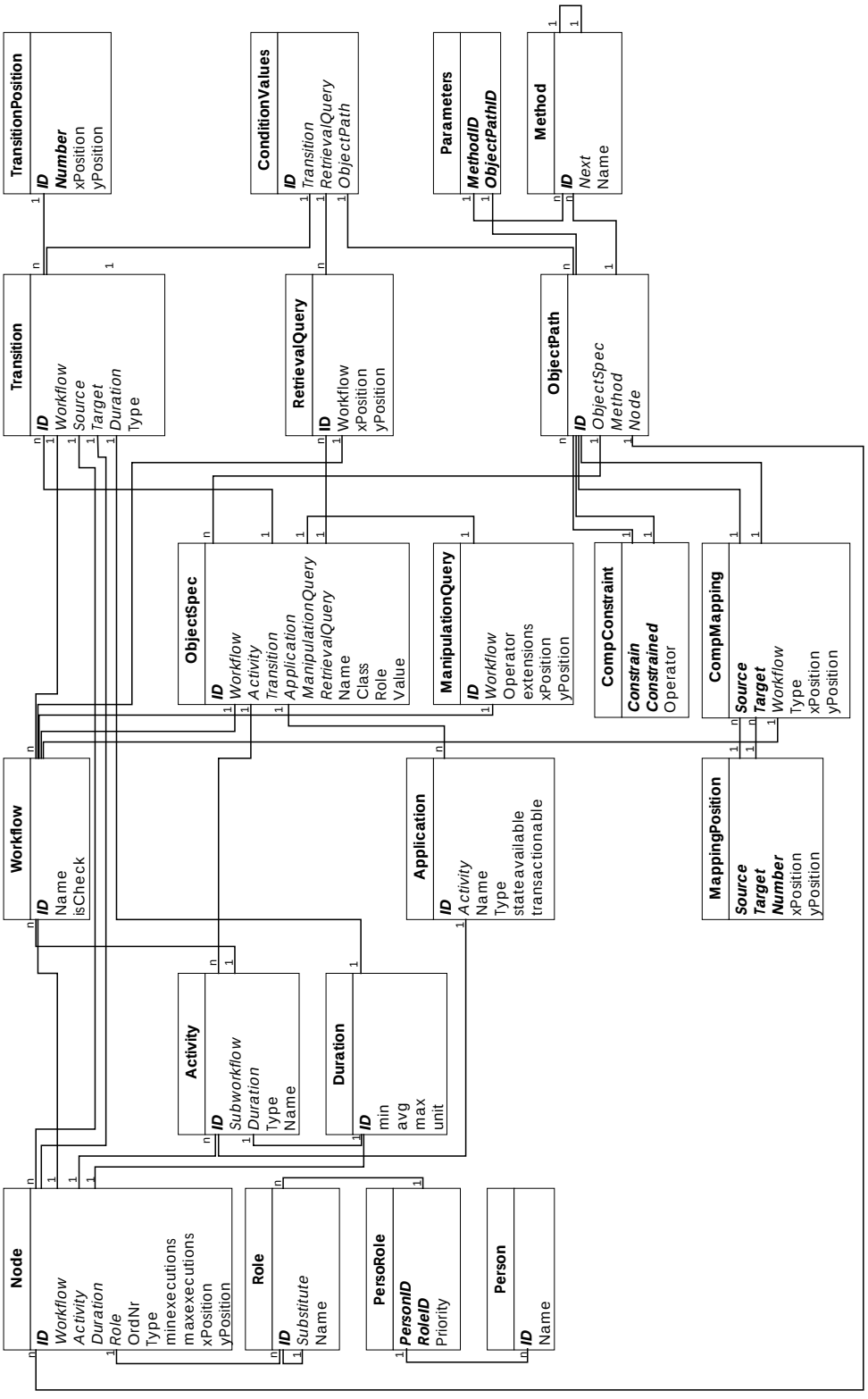
[WMC99] *Workflow Management Coalition – Glossary & Terminologie*, Workflow Management Coalition, 1999

Anhang A - UML Modell

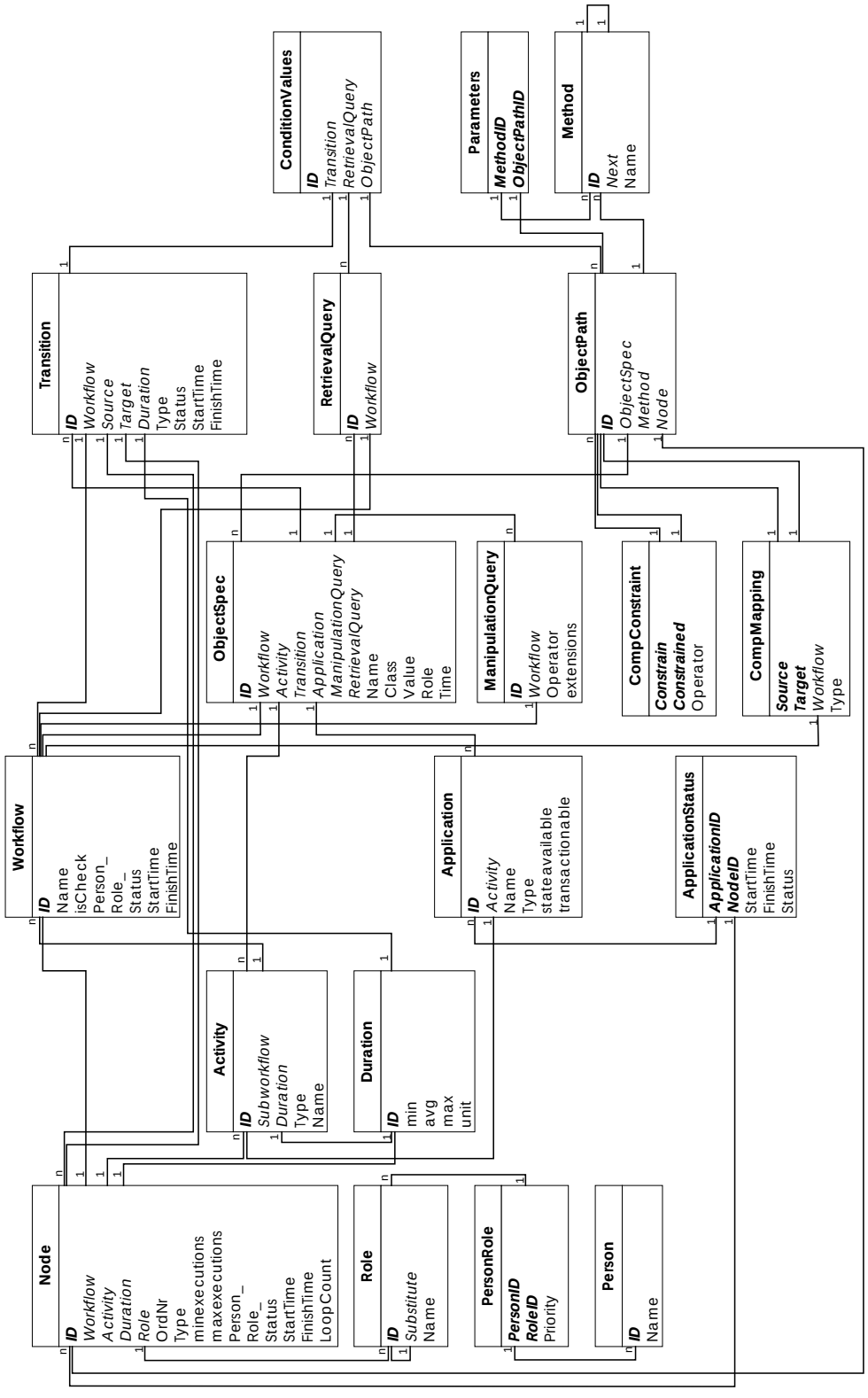


UML-Modell des Workflow-Management-Systems

Anhang B - Datenbankschemata

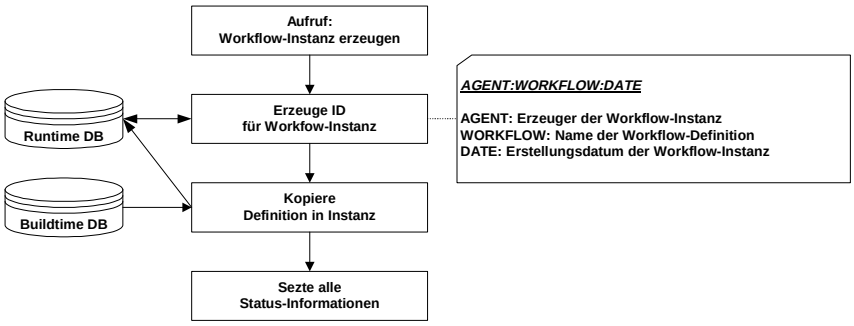


Entity-Relationschip-Modell der Buildtime Datenbank



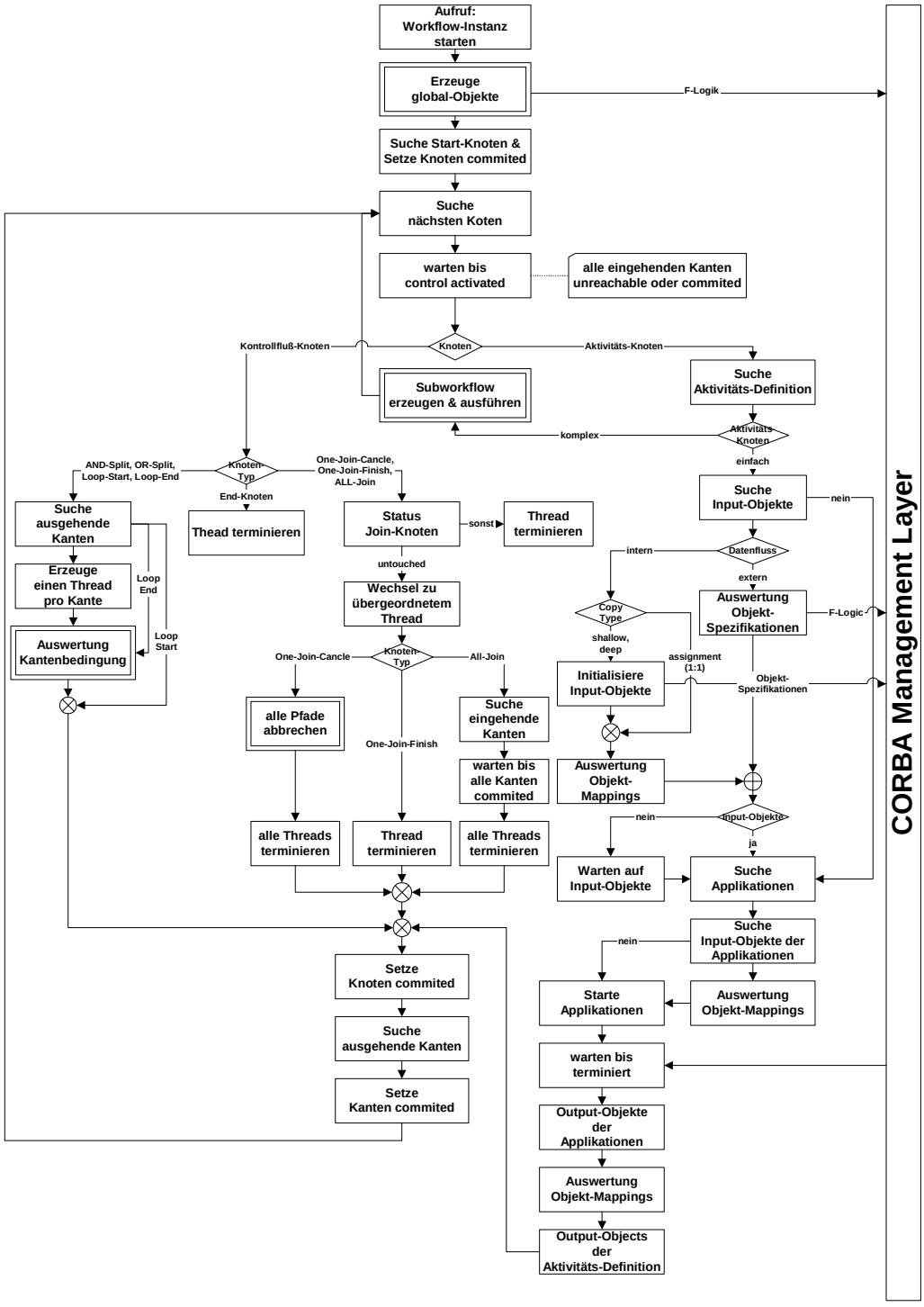
Entity-Relationship-Modell der Runtime Datenbank

Anhang C - Flussdiagramme



Flussdiagramm – Workflow-Instanz erzeugen

Anhang C - Flussdiagramme



Flussdiagramm – Workflow-Instanz ausführen

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Alexander Dietzsch Leipzig, 10. Juli 2000