Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Diplomarbeit

Transaktionsunterstützung für verteilt ausgeführte mobile Prozesse

Alexander Holbreich

1holbrei@informatik.uni-hamburg.de
Studiengang Wirtschaftsinformatik
Matr.-Nr. 5407404
Fachsemester 10

Erstgutachter: Professor Dr. W. Lamersdorf
Zweitgutachter: Professor Dr. W. Menzel

Inhaltsverzeichnis

1 Einleitung

Laut dem europäischen Statistikamt (EUROSTAT) wuchs die Zahl der Mobilfunkteilnehmer in Europa in den Jahren von 1995 bis 2003 im Durchschnitt jährlich um 42%. Waren im Jahre 1995 nur 5 der 100 Europäer Mobilfunkteilnehmer, so sind es 2003 schon 80 aus 100 [Lum05]. Die mobile Kommunikation hat sich in unserem Leben fest etabliert und erobert nach und nach neue Anwendungsfelder. Mobile Dienste und Anwendungen beschränken sich nicht mehr auf reines Telefonieren, sondern werden immer umfangreicher und komplexer. Mit steigendem Wachstum der Gerätezahlen nimmt auch die Dichte der angebotenen Dienste zu.

Es kann angenommen werden, dass der Bedarf an mobiler Middleware, welche diesen Herausforderungen begegnen kann, in naher Zukunft weiter zunehmen wird. So wurden bereits viele Ansätze zur Unterstützung mobiler Anwendungen entwickelt. Dabei hat sich herausgestellt, dass mobile Systeme im Gegenteil zu klassischen Verteilten Systemen i.d.R. weniger Verteilungstransparenz aufweisen, dafür aber mehr Ortsbezug sowie Bewusstsein über ihre Mobilität (Awareness) besitzen. Die meisten mobilen Systeme unterstützen diese Konzepte bereits, doch sind diese oft nur für wenige monolithische Anwendungen, welche kurzzeitige Aufgaben erfüllen, konzipiert [Kun05].

Um der Vision des Ubiquitos Computing, die eine nahtlose und unsichtbare Integration der Rechner ins tägliche Leben der Benutzer in der Zukunft beschreibt, näher zu kommen und dem Bedarf immer komplexer werdenden Aufgaben gerecht zu werden, sollten mobile Middleware Systeme auch mehr leisten können. Wünschenswert ist dabei die Möglichkeit der Integration von komplexen Aufgaben. Weiterhin ist sogar die Verwendung vorab unbekannter Dienste, welche anhand bestimmter Kriterien dynamisch ausgewählt werden könnten, gewollt. Zudem sollten auch die langlebigen Aufgaben besser unterstützt werden, was die meisten mobilen Systeme zurzeit noch nicht bieten können [Wei93, KZL07].

Komplexe Aufgaben können dabei als eine Verkettung von einfachen Aufgaben zu einem Prozess modelliert werden. Im weiteren Verlauf dieser Arbeit werden solche Prozesse genauer definiert und als Mobile Prozesse bezeichnet (vgl. 2.4.4). Mobile Prozesse sind mit anderen, nicht mobilen (Business-) Prozessen bzw. Workflows verwandt, aber in der Regel durch ihre Anpassung an die Besonderheiten mobiler Umgebung (ressourcenschwache Geräte, unzuverlässige Verbindungen usw. (vgl. 2.2.1), eher leichtgewichtig [KLZ06].

Mobile Prozesse werden auf mobilen Middleware Systemen ausgeführt, deren Hauptaufgabe darin besteht, Mobile Prozesse sicher und im Sinne des Benutzers

auszuführen. Dabei werden einzelne Teilaufgaben mit Hilfe von in der Umgebung vorhandenen Diensten durchgeführt. Kennzeichnend für eine mobile Middleware ist, dass sie auf mobilen Geräten in einer mobilen Umgebung ausgeführt wird. Die Hardware und Systemvoraussetzungen, auf denen mobile Middleware-Plattformen aufsetzten, sind damit i.d.R. in ihrer Leistung relativ stark eingeschränkt und können sich schnell zu einem Engpass des Systems entwickeln [KLZ06].

Daher sollten solche Systeme möglichst sparsam und effizient mit den zur Verfügung stehenden Ressourcen umgehen können. Insbesondere sollten häufige Ausfälle von Netzwerkknoten und Verbindungen berücksichtigt werden. In diesem Zusammenhang gewinnt die Migrationsfähigkeit der Prozesse umso mehr an Bedeutung. Ein Prozess ist dann migrationsfähig, wenn er seinen Ausführungsort während seiner Abarbeitung wechseln kann, so dass die Kontrolle der Prozessausführung auf ein anderes Gerät übergeht. Genau genommen ist es die Fähigkeit der mobilen Middleware einen Prozess migrationsfähig machen zu können [Kun05a, KLZ06].

Weiterhin können verteilt ausgeführte Mobile Prozesse als an die mobile Umgebung angepasste Workflows angesehen werden (vgl. 2.4.4), wobei viele Konzepte, die für Workflows bereits erforscht und entwickelt wurden, im Kontext der verteilt ausgeführten Mobilen Prozesse noch nicht betrachtet worden sind. Dazu gehört unter Anderem eine effiziente Transaktionsunterstützung. Jedoch sollten auch Mobile Prozesse transaktional geschützt ablaufen können, so dass unerwünschte Nebenwirkungen und Konsistenzverletzungen eingeschränkt werden.

Dabei sind die Transaktionskonzepte der gängigen Workflow-Systeme nicht ohne Anpassungen auf Mobile Prozesse übertragbar. Denn die meisten Workflow-Systeme, und insbesondere die in der Praxis am häufigsten etablierten, wurden für gänzlich andere Umgebungen entworfen und implementiert. Vergleichbar dimensionierte Hardware- und Software-Voraussetzungen sind in einer mobilen Umgebung nicht gegeben. Aber auch die konzeptionellen Unterschiede und Anpassungen der Mobilen Prozesse, welche im weiteren Verlauf genauer beschrieben werden, machen die direkte Anwendung bestehender Konzepte oder gar Infrastrukturen besonders problematisch. Nichtsdestotrotz können einige allgemeine Konzepte durch adäquate Anpassungen verwendet werden.

Dieses zu untersuchen und dabei einen Weg zu finden, benutzerzentrierte, verteilt ausgeführte Mobile Prozesse mit einer mobilitäts- und migrationsadäquaten, effizienten und flexiblen Transaktionsunterstützung anreichern zu können, ist der Gegenstand dieser Arbeit.

1.1 Aufgabenstellung, Zielsetzung

Das primäre Ziel dieser Arbeit besteht in der Entwicklung eines für verteilt ausgeführte Mobile Prozesse geeigneten Transaktionskonzepts. Es werden dabei ein Transaktionskonzept und ein Ablauf- bzw. Koordinationsmechanismus gesucht, welche die Besonderheiten einer verteilten mobilen Umgebung sowie der Mobilen Prozesse berücksichtigen. Zudem sollen konkrete Transaktionsmodelle adaptiert oder falls es nicht möglich ist, auch neu erarbeitet werden.

Die entwickelte Transaktionsunterstützung soll die Ausführung der Mobilen Prozesse sichern. Dazu sollen vorerst wünschenswerte Eigenschaften und Kriterien des Transaktionsschutzes für Mobile Prozesse ausgearbeitet und anschließend in dem Lösungsvorschlag umgesetzt werden. Dabei sollte die Transaktionsmodellierung nach Möglichkeit nicht in den Kontrollfluss eingreifen, sondern auf einer parallelen Ebene stattfinden. Außerdem wird die größtmögliche Flexibilität der Transaktionsmodellierung angestrebt.

Die Umsetzung umfasst den Entwurf der Softwarekomponenten, welche die konzeptionellen Ergebnisse prototypisch implementieren sollen. Der Prototyp soll dabei in die bestehende Middleware-Architektur des Forschungsprojekts DEMAC (*Distributed Environment for Mobility-Aware Computing*[1]) integriert werden.

1.2 Vorgehensweise

Nach der Einleitung wird der Themenbereich der Diplomarbeit vorgestellt. Es werden die Grundzüge der Verteilen Systeme, sowie die Besonderheiten von Mobile Computing und mobilen Middleware beschrieben. Anschließend werden allgemeine Grundlagen der Prozesse vorgestellt, wobei ein besonderes Augenmerk auf Fähigkeiten und Eigenschaften der verteilt ausgeführten Mobilen Prozesse gerichtet wird. Außerdem wird der Begriff der Dienste in Bezug auf Mobile Prozesse definiert.

Ferner werden die notwendigen Grundlagen der Transaktionen dargestellt. Erweiterungen klassischer Transaktionen sowie Kompensationsbasierte Transaktionen werden erläutert und ausführlich behandelt. Danach erfolgt die Darstellung der Transaktionskontrolle in Workflows und Mobilen Transaktionen. Weiterhin werden Anforderungen an eine Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse formuliert. Die zuvor beschriebenen Transaktionskonzepte, werden anschließend anhand dieser Anforderungen bewertet.

[1] Vgl. http://vsis-www.informatik.uni-hamburg.de/projects/demac/

Die Thematik der Transaktionsunterstützung setzt den Aufbau der Arbeit fort. Hierbei werden in Kapitel 4 die Herausforderungen einer Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse systematisiert. Auf dieser Grundlage werden anschließend eigene Konzepte zur Transaktionsunterstützung entwickelt. Dabei werden Modelle der Sequenziellen und Join Transaktionen sowie die Flexible Synchronisierte Transaktion vorgestellt, erklärt und anschließend bewertet.

Im letzten Abschnitt werden Entwürfe der Softwarekomponente der DEMAC Architektur vorgestellt, sowie die Erweiterungen der in DEMAC eingesetzten Prozessbeschreibungssprache DPDL erläutert. Dabei werden der Zweck sowie die Funktionsweise der einzelnen Komponenten erläutert.

Zum Schluss werden die Ergebnisse zusammengefasst sowie ein Überblick der Erweiterungs- und Optimierungsmöglichkeiten gegeben.

2 Grundlagen Mobiler Prozesse

In diesem Abschnitt werden wichtige Aspekte verteilt ausgeführter Mobiler Prozesse erläutert. Zudem werden einige Grundlagen Verteilter Systeme, des Mobile Computings sowie von Prozessen und Workflows dargestellt. Da Verteilte Systeme das Fundament der meisten in dieser Arbeit behandelten Themen bilden, werden diese im Nachfolgenden definiert und erklärt. Danach werden die Begriffe Mobile Computing und mobile Middleware eingeführt und erläutert. Zum Ende des Kapitels werden Prozesse und Workflows behandelt.

2.1 Verteilte Systeme

In der Literatur sind mehrere und teilweise recht unterschiedliche Definitionen Verteilter Systemen zu finden, welche dabei die unterschiedlichen Eigenschaften der verteilten Systeme betonen. In dieser Arbeit wird eine Definition von Tanenbaum verwendet, welche in Sinne diese Arbeit nur die grundlegenden Aspekte der Verteilten Systeme benennt:

> *A distributed System is a collection of independent computers that appears to its users as a single coherent system* [TaSt02, Suite 2].

Dabei sind zwei Aspekte besonders hervorgehoben. Erstens, die Hardwarekomponenten sind autonom. Zweitens, die Software eines Verteilten Systems, ist so konzipiert, dass die Verteilung des Systems vor dem Systembenutzer versteckt wird [TaSt02].

Verteilte Systeme basieren in der Regel auf mehrschichtigen Architekturen. Softwareschichten die hauptsächlich die Eigenschaften eines Verteilten Systems realisieren sind auch als Middleware bekannt. Somit ist es auch eine Aufgabe der Middleware, den Benutzern einen einfachen Zugang zu entfernten Ressourcen zu ermöglichen und diese dabei kontrolliert zwischen den Benutzern aufzuteilen. Als Ressource kann dabei kontextabhängig praktisch jede Hard- und Softwarekomponente auftreten. Beispiele dafür sind Drucker, Speichermedien, Daten, Dateien, Webseiten, Netzwerke und auch Dienste. In der vorliegenden Arbeit sind Dienste von besonderem Interesse, da diese als Basiselemente den Mobilen Prozessen dienen. Deshalb werden im weiteren Verlauf Dienste ausführlich betrachtet (vgl. 2.5) [TaSt02].

Die Verteilung der Systeme resultiert in einer höheren Komplexität der Verteilten Anwendungen. Deswegen ist es sinnvoll immer wiederkehrende Verteilungsaspekte nicht in jeder Applikation neu zu entwickeln, sondern diese nur einmal in der Middleware eines

Verteilten Systems bereit zu stellen. Damit abstrahieren die Anwendungen von der Existenz der Verteilung, was zur Reduzierung ihrer Komplexität führt [TaSt02, Ben04].

Wenn ein Verteiltes System sich dem Benutzer und Applikationen bezüglich einer Eigenschaft als ein Ein-Computer-System präsentieren kann, wird es bezüglich dieser Eigenschaft als *transparent* bezeichnet. In der nachfolgenden Tabelle 1 sind Transparenzarten nach dem *Open Distributed Processing Reference Model* [ISO95] aufgelistet:

Tabelle 1: Verschiedene Transparenzarten (nach [ISO95])

Transparenzart	**Beschreibung**
Access	Verbergen der Unterschiede in der Daten-Repräsentation und in der Art, wie auf Ressourcen zugegriffen wird.
Location	Verbergen des Orts der Ressource.
Migration	Verbergen des Umstandes, dass eine Ressource ihren Aufenthaltsort ändern kann.
Relocation	Verbergen des Umstandes, dass eine Ressource ihren Ort wechseln kann, während sie noch genutzt wird.
Replication	Verbergen der Information darüber ob die „Originalressource“ oder ihre Kopie verwendet wird.
Concurrency	Verbergen der Tatsache, dass eine Ressource von mehreren Benutzern geteilt wird.
Failure	Verbergen von Ausfällen und von Wiederherstellmaßnahmen einer Ressource.
Persistence	Verbergen der Information darüber, ob eine (Software) Ressource im transistenten oder im persistenten Speicher ist.

Durch Transparenz in einem Verteilten System, wird den Anwendungen eine vereinfachte Sicht einer sonst i.d.R. komplexen verteilten Umgebung geboten. Dennoch gibt es Situationen in denen die Durchsetzung aller hier beschriebenen Transparenzarten nicht angestrebt werden sollte. Dieses ist beispielsweise in Middleware-Plattformen für mobile Systeme der Fall, welche sich stark in Hardware, Kommunikation und Art der Nutzung unterscheiden. Die Besonderheiten einer mobilen Umgebung und ihre Bedeutung für die mobile Middleware werden im nächsten Abschnitt betrachtet [Mül02].

2.2 Mobile Computing

Unter dem Begriff Mobile Computing wird die Datenverarbeitung auf mobilen Recheneinheiten verstanden. Dabei steht die dazugehörige Kommunikation zwischen den Benutzern, mobilen Geräten, Anwendungen und Diensten im Vordergrund. Unter Mobile Computing werden also Systeme und Anwendungen verstanden, die einem Benutzer einen Zugriff auf verschiedenste Dienste mittels mobiler Rechengeräte ermöglichen. Dabei werden Objekte allgemein als mobil bezeichnet, wenn sie dazu geeignet sind, ohne großen Aufwand ihren Standort wechseln zu können. Beispielhaft besitzen Laptops, Personal Digital Assistants (PDAs), Mobiltelefone und Smartphones sowie in Gegenstände eingebettete Computer diese Eigenschaft [Rot02, ZhNi06].

Mobilität ist eine der wesentlichen Prämissen in Mobile Computing. Dabei unterscheidet Pandya [Pan99] drei Arten von Mobilität:

- **Endgerätemobilität (Terminal Mobility):** Es handelt sich dabei um die Mobilität der Endgeräte, während die Beziehung zwischen dem Gerät und dem Benutzer statisch ist. Endgerätemobilität wird mit einer drahtlosen Verbindung assoziiert, was dazu führt, dass ein Benutzer das Endgerät bei sich tragen muss. Das System kennt somit die Identität des Geräts.
- **Benutzermobilität (Personal Mobility):** Diese Art der Mobilität basiert auf einer dynamischen Zuordnung zwischen dem Endgerät und dem Benutzer. Das bedeutet, dass ein Benutzer auf Dienste von verschiedenen Endgeräten zugreifen kann. Ein System ist also in der Lage, einen Benutzer unabhängig von seinem Aufenthaltsort und benutztem Endgerät identifizieren zu können.
- **Dienstmobilität (Service Mobility):** Unter Dienstmobilität wird die technische Fähigkeit des Netzwerks verstanden, einen bestimmten Dienst, an dem vom Benutzer erwünschten Gerät bzw. Ort, bereit zu stellen.

Die Benutzer von Computersystemen der heutigen Zeit wollen sowohl beruflich als auch privat mobil sein. So werden Computer z.B. in der Form von Mobiltelefonen oder Notebooks immer öfter mitgenommen. Dabei ermöglicht die Miniaturisierung der Endgeräte, die Entwicklungen auf dem Gebiet der drahtlosen Netzwerke und eine weltweite Vernetzung, die Entwicklung solcher Dienste, welche vor einigen Jahren noch undenkbar waren [Rot05].

Diese Entwicklungen lassen sich durch vergleichbare in der Elektronikbranche vorhersagen. Laut Mooreschen Gesetz verdoppelt sich durch den technischen Fortschritt etwa alle 24 Monate die Komplexität von integrierten Schaltkreisen. Der gleiche Trend ist auch bei der Leistung der Elektronikbauteile zu beobachten. Mit der immer

fortschreitenden technischen Entwicklung werden auch die mobilen Technologien immer reifer und bieten immer größere Leistungssteigerungen bei sinkenden Kosten [ZhNi06].
Im Zusammenhang mit Mobile Computing wird oft der verwandte Begriff *Ubiquitos Computing* verwendet. Dabei beschreibt Ubiquitos Computing eine Vision der Zukunft in der (mobile) Geräte in das tägliche Leben der Benutzer nahtlos integriert sind. Dabei agieren diese Geräte im Hintergrund, kommunizieren unter einander, und stehen dabei dem Benutzer jederzeit und überall zur Verfügung ohne selbst viel Aufmerksamkeit zu erzeugen. Dabei ist vorstellbar, dass solche Gegenstände wie Armbanduhren, Ringe, Kugelschreiber u.ä., Rechnerfunktionen übernehmen werden [Wei91, Wei93, Rot05].

Auch *Personal Computing* gehört zum Bereich des Mobile Computing. Dabei wird betont, dass mobile Geräte sehr viel mehr zur Verwaltung und Verarbeitung persönlicher Daten wie Telefonnummer, persönliche Termine, Notizen usw. verwendet werden. Auch Kommunikationsverbindungen mobiler Geräte werden im Vergleich zu stationären Systemen viel häufiger zur Übertragung von persönlichen Daten genutzt [Pan99].

2.2.1 Mobile Geräte und ihre Eigenschaften

Mobile Geräte unterscheiden sich von stationären Geräten in vielfacher Hinsicht. An dieser Stelle werden aber nur die grundlegenden Unterschiede behandelt, die für das Thema dieser Arbeit von Bedeutung sind. Forman und Zahorjan nennen Kommunikation, Mobilität und Portabilität als essentielle Eigenschaften von mobilen Geräten, auf denen die meisten anderen Besonderheiten und Einschränkungen der mobilen Geräte beruhen [FoZa94].

In der Regel erfordern mobile Geräte zur Kommunikation eine drahtlose Verbindung. Es handelt sich dabei z.B. um Mobilfunktechnologien wie GSM (*Global System for Mobile Communitcation*) oder UMTS (*Universal Mobile Telecommunication Systems*), lokale Funknetze wie WLAN (*Wireless Local Area Network*) oder Bluetooth sowie Technologien der *Infrared Data Association* (IrDA). Im Vergleich zu stationären Verbindungen hat eine auf Radiowellen basierte Kommunikation mehr Hindernisse bei der Signalausbreitung. Dies ist auf die Interaktion von Radiowellen mit der Umgebung zurückzuführen. Daraus ergibt sich, dass die drahtlose Kommunikation durch relativ schmale Bandbreiten, hohe Fehlerraten und häufige Verbindungsabbrüche gekennzeichnet ist [FoZa94].

Die Portabilität und Handlichkeit und damit natürlich die Größe mobiler Geräte ist ein zentraler Aspekt, welcher immer wieder über die Benutzerakzeptanz und den Markterfolg entscheidet. Um mobilen Geräten Portabilität zu verleihen, sind die Hersteller bestrebt diese Geräte im „Armbanduhrformat“ zu bauen. Jedoch kann die kleine Größe der Geräte oft nur mit Einschränkungen in den technischen Eigenschaften der Geräte erzielt werden.

Tabelle 2: Typische Eigenschaften mobiler Geräte

	Nokia 6230i[2]	**Nokia N80**[3]	**Poket LOOX C550**[4]	**ThinkPad T60**[5]
Typ	Mobiltelefon	Smartphone	PDA	Notebook
Akkulaufzeit bei dauerhafter Auslastung	Max 5. Stunden	Max. 3 Stunden Gesprächszeit	Max. 3 Stunden	Max. 6,8 Stunden
Gewicht	99g	134g	160g	2,09Kg
Rechenleistung	Ca. 161,9MHz[6]	220MHz[7]	520MHz	1.83 GHz (dual core)
Speicherkapazität	2048KB RAM[8] 32Mb intern, Speicherkarte	16384Kb RAM[9] 40Mb intern, Speicherkarte	64Mb RAM 128 ROM	1Gb DDR2 RAM 80GB HDD
Kommunikation	GSM, Bluetooth, IrDa	GSM, UMTS, Bluetooth, IrDa	WLAN, Bluetooth, IrDa	LAN, WLAN, IrDa, Modem
Betriebssystem	Proprietär	Symbian OS v9.1[10]	Microsoft Windows Mobile 5.0 Premium	Windows XP
Programmierbarkeit	MIDP 2.0 CLDC 1.1	Java ME, MIDP 2.0, CLDC 1.1	Java ME, Personal Profile, C++	beliebig

[2] Quelle: Datenblatt Nokia 6230i, http://www.nokia.de, 11.2006

[3] Quelle: Datenblatt Nokia N80, http://www.nokia.de, 11.2006

[4] Quelle: Datenblatt Poket LOOX C550, http://www.fujitsu-siemens.de, 11.2006

[5] Quelle: Datenblatt ThinkPad T60 2007FUG, http://www.lenovo.com, 11.2006

[6] Quelle: Testbericht http://www.club-java.com/TastePhone/J2ME/MIDP_Java_telephone.jsp?m=208&brand=Nokia&model=6230i

[7] Quelle: Datenblatt Nokia N80, http://www.areamobile.de/datenblatt/Nokia/N80/673_1.html, 11.2006

[8] Siehe 6.

[9] Quelle: Testbericht, http://www.club-java.com/TastePhone/J2ME/MIDP_Java_telephone.jsp?m=403&brand=Nokia&model=N70-1, 11.2006

[10] Quelle Datenblatt Nokia N80, http://www.symbian.com/phones/nokia_n80.html, 11.2006

Dieses wird besonders bei der Stromversorgung beobachtet. Ein mobiles Gerät bezieht seine Energie von einer eingebauten Batterie. Der technische Fortschritt in der Batterieentwicklung findet dabei nicht so dynamisch, wie bei den integrierten Schaltkreisen, statt. Energie für einen längeren Zeitraum zu speichern erfordert heute die Verwendung von relativ großen Batterien. Die Hersteller von mobilen Geräten sind jedoch bestrebt nur relativ kleine Batterien zu verwenden, um die hohe Portabilität der Geräte zu erhalten. Aus diesen Gründen haben mobile Geräte im allgemeinen Fall eine relativ schwache Stromversorgung, welche durch eine wiederaufladbare Batterie gewährleistet ist [FoZa94].

Je kleiner jedoch die Batteriekapazität ist, desto öfter muss das Gerät wieder aufgeladen werden. Möglicherweise kann dies nicht an jedem Ort oder unterbrechungsfrei geschehen. Bei der Ausführung von längeren Aufgaben kann dieser Umstand zu Schwierigkeiten führen und sollte in der Middleware bzw. in den Anwendungen berücksichtigt werden (vlg. 2.3.1).

Angepasst an die Größenverhältnisse und die schwachen Stromquellen sind i.d.R. auch die CPU-Leistung und das Speichervolumen der mobilen Geräte begrenzt. Im Allgemeinen verhält sich die Leistung proportional zur Gerätgröße. Zum Vergleich sind in Tabelle 2 technische Daten verschiedener mobiler Geräte aufgelistet. Diese geben die gegenwärtig typischen Charakteristika einiger gängiger Gerätetypen wieder [FoZa94].

Portable Geräte sind durch äußere Einflüsse im besonderen Maße gefährdet, denn sie werden in verschiedenen Umweltsituationen eingesetzt. Dabei können sie Schaden nehmen, verloren gehen oder einfach gestohlen werden [FoZa94].

Die Mobilität der Geräte verschärft die oben beschriebenen Kommunikationsprobleme. Mobile Geräte können physisch aus der Reichweite der Funkverbindung gebracht werden. Folglich hat ein mobiles Gerät oftmals keine ortsbezogene, feste Adresse und kann zwischen verschiedenen Netzwerken und Kontexten transportiert werden. Daraus resultieren einige mobilitätsspezifische Probleme, wie z.B. Adressierungsprobleme oder Herausforderungen, welche durch einen ständigen Kontextwechsel (vgl. 2.3.1) (auch migrierende Lokalität genannt) entstehen. [FoZa94].

2.3 Middleware für Mobile Computing

Existierende Middleware-Technologien wie Transaktions-, Nachrichten-, und Objektorientierte Middleware-Plattformen wurden mit dem Ziel konzipiert einen hohen Transparenzgrad zu erreichen (vgl. 2.1). Diese Technologien ermöglichen es die Verteilung der Systeme für Benutzer und Entwickler weitgehend unsichtbar zu machen. Zudem werden diese Middleware-Technologien für stationäre verteilte Systeme mit

Festnetzwerken entwickelt und besitzen somit zum Teil andere Vorraussetzungen [CEM02, Emm00].

Deshalb sind diese Middleware-Systeme nicht ohne weiteres für eine mobile Umgebung geeignet. Dieses ist zunächst darin begründet, dass die Interaktionsroutinen wie Remote Procedure Calls (*RPC*) [BiNe84] oder Objekt-Aufrufe (*Object Requests*) eine konstante und relativ zuverlässige Verbindung zwischen den beteiligten Komponenten voraussetzen. Diese ist aber in einer mobilen Umgebung im Regelfall nicht gegeben (vgl. 2.2.1). Weiterhin basieren die meisten objektorientierten Middleware Systeme wie CORBA [Pop98] auf einer synchronen Kommunikation. Das bedeutet, dass kommunizierende Komponenten simultan erreichbar sein müssen. In einer mobilen Umgebung ist jedoch häufiger der Fall anzutreffen, dass die beteiligten Knoten nicht zur selben Zeit betriebsbereit sind. Dieser Umstand erschwert die Anwendung synchroner Kommunikationsmechanismen. Weiterhin unterstellen traditionelle Verteilte Systeme eine stationäre Ausführungsumgebung mit festen Orten für Knoten, bekannten Diensten usw., was jedoch einen Kontrast zu einem dynamischen Szenario einer mobilen Umgebung darstellt. [CEM02].

Dies zeigt, dass die Anwendbarkeit von traditionellen Middleware-Systemen in einer mobilen Umgebung nicht ohne Anpassungen möglich ist. Die Besonderheiten einer mobilen Umgebung müssen in einer mobilen Middleware berücksichtigt werden. Diese Konzepte werden im Folgenden genauer dargestellt. In Abbildung 1 sind dazu die wichtigsten Eigenschaften von mobilen und klassischen Middleware Systemen gegenüber gestellt.

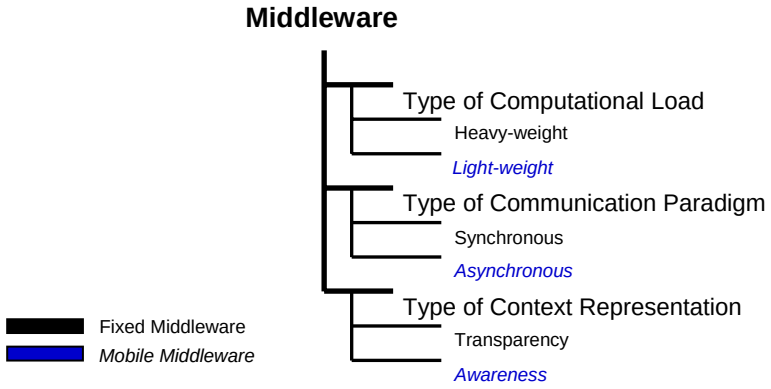

Abbildung 1: Eigenschaften von Middleware-Systemen (nach [CEM02])

Die Abbildung 1 zeigt, dass mobile Middleware-Systemen aufgrund der Geräteeigenschaften eher leichtgewichtig sind. Der Grund dafür sind die begrenzten Ressourcen

mobiler Geräte. Weiterhin erfordern nicht-permanente Verbindungen die Fokussierung auf asynchrone Kommunikationsverfahren, um nicht von einer in vielen Fällen nicht realisierbaren simultanen Verbindung von Client und Server abhängig zu sein. Schließlich unterscheiden sich mobile Middleware-Plattformen durch ihre Awareness – die Fähigkeit Umgebungsinformation wahrzunehmen [CEM02].

2.3.1 Awareness

Stationäre Computersysteme werden an einem bestimmten Ort aufgestellt und benutzt. Ortswechsel sind bei solchen Systemen unüblich und finden entweder gar nicht oder selten statt. Entsprechend geringe Bedeutung besitzen die Informationen über die Umgebung solcher Systeme, da der Ort, Umfeld, Kontext usw. sich kaum ändern. Wo die Umgebungsparameter einen Einfluss auf stationäre Computersysteme haben, können sie meist bei der Installation berücksichtigt werden. Mobile Systeme hingegen können ihren Einsatzort öfters wechseln. Entsprechend hohe Relevanz besitzen demgemäß verschiedene Umgebungsparameter, welche sich im Zuge der Bewegung des Geräts ändern können und deren Werte einen Einfluss auf die Nutzung des Gerätes nehmen können [Mül02].

Der Einbezug sich ändernder Umgebungsbedingungen wird als *Awareness* bezeichnet. Wobei die eigentliche Reaktion auf sich ändernde Bedingungen als *Adaptation* bezeichnet wird. Awareness und Adaptation zusammen, werden als *Adaptivity* bezeichnet [SNKP94, Mül02].

Relevante Parameter einer Umgebung sind z.B. nach Brown [Bro96]: *„der Ort, Identitäten von Personen in der Umgebung des Benutzers, Tageszeit, Jahreszeit, Temperatur, usw.“*, welche zusammen als *Kontext* zu verstehen sind. Allerdings sind diese Parameter nur Beispiele eines Kontextes, denn ein Kontext lässt sich über eine einfache Auflistung von relevanten Umgebungsparametern nicht präzise definieren. Es ist nicht klar, ob die nicht genannten Umgebungsparametern einer solchen Definition zum Kontext gehören oder nicht [DeAb00].

Diese Defizite lassen sich durch eine abstrakte Definition des Kontexts weitgehend beseitigen. Deshalb schlagen Dey und Abowd folgende Definition vor:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.* [DeAb00, Seite 3]

Diese Definition verdeutlicht, dass ein Kontext situations- und anwendungsabhängig ist. Da der Kontext alle relevanten Daten der Umgebung einer mobilen Anwendung darstellt, lässt sich die Awareness in mobilen Systemen also allgemeiner als die Wahrnehmung des Kontextes (*Context-Awareness)* definieren. So schlagen Dey und Abowd dazu folgende Definition vor:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task* [DeAb00, Seite 6].

2.4 Prozesse

Ein Prozess besteht aus verknüpften Tätigkeiten (*Tasks*) und einer Menge von Bedingungen, welche die Ausführungsreihenfolge dieser Tätigkeiten bestimmen. Unter einer Tätigkeit wird dabei ein logischer Arbeitsschritt verstanden, welcher als ein Ganzes von einer Ressource ausgetragen wird. Eine Ressource ist dabei eine Person oder Maschine oder eine Gruppe von Personen und Maschinen. Dies bedeutet jedoch nicht, dass beim Ausführen einer Aufgabe (*Task*) keine anderen Ressourcen involviert werden können. Wichtig ist lediglich, dass eine Ressource für die Ausführung verantwortlich ist [AaHe04].

Prozesse sind in allen möglichen Bereichen anzutreffen, in diesem Abschnitt werden zunächst Geschäftsprozesse vorgestellt anhand derer die grundlegenden Eigenschaften von Prozessen erläutert werden. Prozesskontrollstrukturen stehen dabei im Vordergrund. Weiterhin wird die Abgrenzung zu Workflows erläutert, sowie die Besonderheiten von Mobilen Prozessen dargestellt.

2.4.1 Geschäftsprozesse

Geschäftsprozesse (*Business Processes*) ist heute wohl die bekannteste und am besten untersuchte Prozessart. Dabei haben Geschäftsprozesse vor allem eine betriebswirtschaftliche Bedeutung. Hammer und Champy definieren einen Geschäftsprozess wie folgt:

> *We define business process as a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer* [HaCh04, Seite 38].

Bei der Analyse von Geschäftsprozessen, betrachtet man die gesamte Kette der einzelnen Tätigkeiten, die letztendlich der Kundenzufriedenheit dienen. Dabei muss ein Geschäfts-

prozess nicht automatisiert sein, es geht lediglich um vorhandene oder geplante Vorgänge in der Realität [HaCh04].

Geschäftsprozesse werden kontinuierlich verbessert, an Veränderungen adaptiert oder gar neu gestaltet (*Business Process Reengineering*). Diese Aufgaben können durch IT Systeme, wie z.B. *Business Process Modeling Tools* (*BPMT*) weitgehend unterstützt werden. Weiterhin besteht ein Bedarf an Automatisierung von Geschäftsprozessen, um die Effizienz, Geschwindigkeit und Wirtschaftlichkeit der Betriebe zu steigern. Dabei werden automatisierte Prozesse oder Prozessteile als *Workflows* (vgl. 2.4.3) bezeichnet und von *Workflow Management Systemen (WFMS)* ausgeführt [GeTs98, HaCh04].

In beiden Fällen müssen Prozesse verstanden und erfasst werden. Die Prozesserfassung resultiert in einem Prozessmodell bzw. einer Prozessbeschreibung. Dabei ist eine Prozessbeschreibung im Grunde eine Abstraktion eines real existierenden Prozesses. Die Beschreibung kann mit Hilfe einer Prozessbeschreibungssprache bewerkstelligt werden, welche an sich ein Metamodell für die Beschreibung von Prozessmodellen ist. Beispiele für Prozessbeschreibungssprachen sind XPDL [NoMa02, Aal03], als eine technologieunabhängige Definitionssprache, oder BPEL4WS [ACD+03], welche als Vervollständigung des Web Service Protokollstapels angesehen werden kann [GeTs98].

Unabhängig von der Sprache muss in einer Prozessbeschreibung festgelegt werden, wie ein Prozess ausgeführt werden soll, in welcher Reihenfolge die Aktivitäten ausgeführt werden und wie die Weitergabe von Daten erfolgen soll. Außerdem werden bei Geschäftsprozessen Information über Prozessteilnehmer benötigt. Zur Modellierung dieser Informationen dient die, durch das Metamodell der jeweiligen Sprache verfügbare, Sammlung an speziellen Konstrukten und Funktionen. Einige, für diese Arbeit relevante, Konstrukte werden im nächsten Abschnitt genauer vorgestellt [AaHe04].

2.4.2 Kontrollfluss

Die Prozessbeschreibung enthält unter anderem die Definitionen von Teilschritten bzw. Aktivitäten und den dazugehörigen Bedingungskontrollstrukturen. Dabei sind einige Kontrollflusselemente wie Sequenz, Split und Join für viele Prozessbeschreibungssprachen grundlegend. Sie haben auch im Rahmen dieser Arbeit eine besondere Bedeutung, denn durch Kontrollflusselemente wird auch die Transaktionsmodellierung bzw. Transaktionsausführung beeinflusst. Im Folgenden werden diese Elemente detailliert dargestellt.

Kontrollflusskomponenten

Jede Prozessbeschreibung hat zwei Grundelemente zur Verfügung: *Aktivitäten* und *Kontrollflusskonnektoren*. Im Folgenden werden diese Elemente einzeln beschrieben.

Aktivität: Eine *Aktivität (Activity)* stellt eine Abstraktion der Teilaufgabe (Arbeitspakets) innerhalb eines Prozesses dar. Die Definition einer Aktivität beinhaltet die Spezifikation der Tätigkeit, die Ein- und Ausgangsdaten, sowie Methoden zur Feststellung der Vollständigkeit der durchgeführten Arbeit. Dabei kann die Durchführung einer Aktivität automatisch durch eine technische Anlage oder eine Softwarekomponente organisiert werden. Sie kann aber auch teilweise oder ganz von Menschen erledigt werden [LeRo99].

Kontrollflusskonnektoren: Eine Prozessdefinition beschreibt wie die einzelnen Aktivitäten untereinander verknüpft sind und in welcher Reihenfolge sie ausgeführt werden. In einer Graphstruktur basiert diese Beschreibung auf *Kontrollflusskonnektoren (Control Connectors)*, welche die Aktivitäten untereinander verbinden. Eine gerichtete Verbindung, die von der Aktivität *A* zur Aktivität *B* verläuft wie in der Abbildung 2 zu sehen ist, beschreibt, dass *B* ein potenzieller Nachfolger von *A* ist. Eine Aktivität kann mehrere Vorgänger und Nachfolger besitzen [LeRo99].

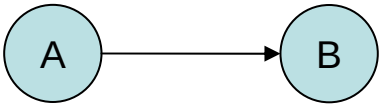

Abbildung 2: Kontrollflusskonnektor

B ist korrekterweise nur ein potenzieller Nachfolger, weil die Kontrollflusskonnektoren mit Bedingungen verknüpft werden können, welche eine Reihenfolgebeziehung zwischen den Aktivitäten außer Kraft setzten können. In der Abbildung 3 ist eine Erweiterung der Ausgangsdarstellung (siehe Abbildung 2) zu sehen, die mit einer *Transitionsbedingung* angereichert wurde.

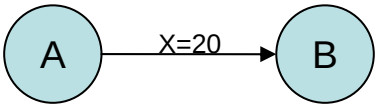

Abbildung 3: Darstellung einer Transitionsbedingung

Eine Transitionsbedingung ist eine Beschreibung der Umstände unter welchen der Übergang von einer Aktivität zur Nächsten möglich ist. Zur Laufzeit müssen dabei alle definierten Umstände gegeben sein, sonst wird die Transitionsbedingung zu „false“ evaluiert und die nachfolgende Aktivität *B* wird nicht ausgeführt [LeRo99].

Weiterhin ist es möglich *Aktivierungs-* und *Exit-Bedingungen* für eine Aktivität festzulegen. Aktivierungsbedingungen steuern nach bestimmten Vorgaben die Aktivierung einer Aktivität. So ist es beispielsweise möglich eine Aktivität erst zu einem bestimmten Zeitpunkt zu starten. Wobei Exit-Bedingungen erst unmittelbar nach der Abarbeitung einer Aktivität evaluiert werden. Exit-Bedingungen können beispielsweise zur Kontrolle der Ergebnisse einer Aktivität eingesetzt werden [Lero99].

Kontrollflussstrukturen

Die Verwendung von Kontrollflusskonnektoren darf nicht beliebig geschehen, Leymann und Roller [LeRo99] definieren zwei Restriktionen an Kontrollflusskonnektoren:

1. Zwei gegebene Aktivitäten können maximal mit einem Kontrollflusskonnektor verbunden werden. So sind zwei Aktivitäten entweder untereinander gar nicht oder mit nur einem Kontrollflusskonnektor verbunden.
2. Der von allen Kontrollflusskonnektoren gebildete Graph ist azyklisch. Daher sind Zyklen im Graph nicht erlaubt.

Diese Restriktionen wurden auferlegt, um das Verständnis der Prozessmodelle zu erleichtern, welche mit diesen Konstrukten modelliert werden. Dabei wurde die Azyklität bereits in Zusammenhang mit der Strukturierten Programmierung unter anderem zur Verbesserung der Programmwartbarkeit vorgeschlagen. Weiterhin können durch diese Einschränkungen unerwünschte Nebeneffekte vermieden werden. Die Kontrollflussstrukturen wie Sequenz, Join und Split erfüllen diese Restriktionen und sind dabei für die meisten Prozessbeschreibungssprachen grundlegend. Nachfolgend werden diese Kontrollflussstrukturen detailliert dargestellt [Fai85, LeRo99].

Sequenz: Eine einfach gerichtete Verbindung zwischen mindestens zwei Knoten ist eine Sequenz. Auch eine sequenzielle Ausführungsreihenfolge in Prozessen wird in dieser Arbeit als Sequenz bezeichnet. Eine Sequenz kann beliebig viele Knoten enthalten, dabei sind alle Knoten in einer Reihe mit einander verbunden und bilden keine Verzweigungen. In der Abbildung 2 ist die einfachste Variante einer Sequenz dargestellt. In der Abbildung 4 ist eine Sequenz mit 4 Aktivitäten zu sehen.

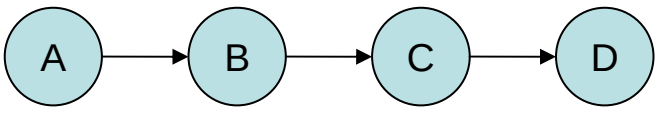

Abbildung 4: Kontrollflussstruktur Sequenz

Split: Ein Split besteht aus mindestens drei Aktivitäten, wobei die Ausgangsaktivität mehr als einen ausgehenden Pfeil besitzt. Abbildung 5 verdeutlicht dieses.

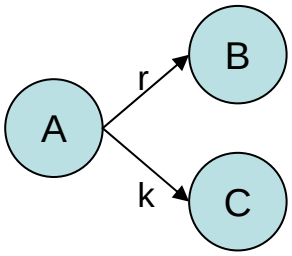

Abbildung 5: Kontrollflussstruktur Split

Nach einem Split können einzelne Prozessstränge nebenläufig ablaufen. Dabei hängt es von den Transitions- und Aktivierungsbedingungen ab. Denn es kann passieren, dass auf einem der Stränge die Prozessausführung nicht weitergeht, weil eine von diesen Bedingungen zu „false“ evaluiert wurde. Die Bezeichnung Split ist auch auf die Aktivität anwendbar. In der Abbildung 5 fungiert Aktivität *A* als *Split-Aktivität* [LeRo99].

Ein Split kann eine Split-Bedingung aufweisen, welche definiert auf welchen Strängen die Prozessausführung weitergeht. Bei einem „AND“ wird die Ausführung auf allen Pfaden fortgesetzt, bei einem „XOR“ ist die Ausführung nur auf einem Prozesspfad zugelassen. Falls eine Splitbedingung in einem Meta-Modell nicht explizit zur Verfügung steht, kann sie dennoch logisch gebildet werden. Wenn beispielsweise, die von der Aktivität *A* ausgehenden Konnektoren *r* und *k* (siehe Abbildung 5), folgende Transitionsbedingungen besitzen: *{a>=0}* für *r* und *{a<0}* für *k*, so kann bei jedem *a* jeweils nur eine der Transitionsbedingungen zur „true“ evaluiert werden.

Join: Eine Aktivität wird nur dann als eine *Join-Aktivität* bezeichnet, wenn sie mehr als einen eingehenden Kontrollflusskonnektor besitzt. In der Abbildung 6 ist Aktivität *C* eine Join-Aktivität [LeRo99].

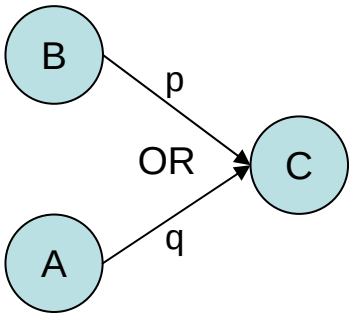

Abbildung 6: Kontrollflussstruktur Join

Joins haben dabei eine Synchronisationsfunktion, denn sie führen die zuvor nebenläufig ablaufenden Prozessstränge wieder zusammen. Für eine bessere Kontrolle der Synchronisation können die so genannten *Join-Bedingungen* formuliert werden. Dabei ist eine Join-Bedingung ein boolescher Ausdruck über die einkommenden Transitionsbedingungen einer Join-Aktivität. Die Auswertung der Join-Bedingung findet aber erst nach der Evaluierung der Transitionsbedingungen aller einkommender Kontrollflusskonnektoren statt. In der Abbildung 6 ist ein Fall gezeigt, in welchem zuerst die Transitionsbedingungen *p* und *q* ausgewertet werden müssen bevor die Join-Bedingung *OR* evaluiert werden kann [LeRo99].

Das Definieren des Kontrollflusses hat Ähnlichkeiten mit dem Programmieren. Mit der Unterstützung von Split und Join unterstützt ein Metamodell auch die Nebenläufigkeit. Die Modellierung von Nebenläufigkeit bringt eine gewisse Komplexität mit sich. Beispielsweise kann eine Join-Aktivität in einem Deadlock verharren, wenn z.B. die Transitionsbedingung bei einem der einkommenden Konnektoren zu „false“ evaluiert wurde, wobei die Join-Bedingung ein „AND“ darstellt. In diesem Fall ist eine Deadlock-Eliminierung oder Prävention notwendig [LeRo99, CES71].

2.4.3 Workflows

Workflows sind automatisierte Geschäftsprozesse. Prozessautomatisierung beinhaltet die Koordination, Kontrolle und Kommunikation der Aktivitäten mit der Unterstützung eines Computersystems. Die Aktivitäten selber können dabei entweder auch automatisiert sein oder durch Menschen abgearbeitet werden [GeTs98].

Leymann und Roller gehen in [LeRo99] auf die Beziehungen zwischen Geschäftsprozessen und Workflows explizit ein. Dieser Zusammenhang ist in der Abbildung 7 dargestellt.

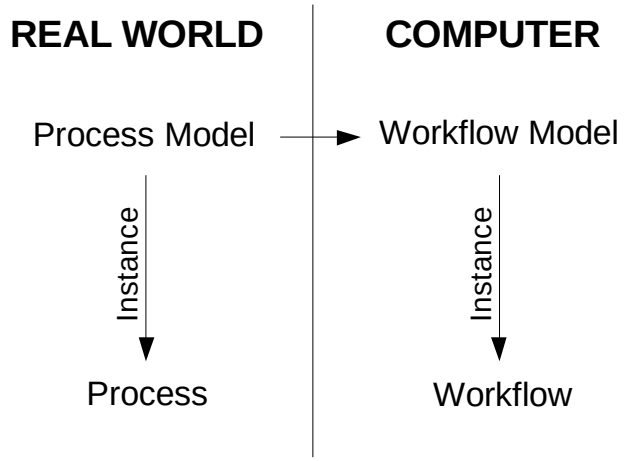

Abbildung 7: Prozesse und Workflows (nach [LeRo99])

Ein Prozessmodell beschreibt die Struktur eines Geschäftsprozesses in der realen Welt. Es beschreibt also alle möglichen Pfade in einem Geschäftsprozess. Dabei ist das Prozessmodell eine Schablone, welche zur Erzeugung von konkreten Prozessinstanzen (*Instanzierung)* genutzt werden kann. Die Geschäftsprozesse müssen jedoch nicht auf Computersystemen ablaufen. Nur die Prozesse oder Prozessteile, die auf einem Computersystem ablaufen können, werden als *Workflowmodell (Workflow model)* bezeichnet. Hierbei kann es oft vorkommen, dass ein Workflowmodell nur einen kleinen Ausschnitt des Prozessmodells darstellt. Ein Workflowmodell ist dabei nur eine Schablone für Workflowinstanzen, sowie ein Prozessmodell es für seine Prozessinstanzen ist [LeRo99].

Klassische Workflows werden von einem *Workflow Management System (WfMS)* verwaltet. Dabei besteht ein WfMS aus Komponenten zur Interpretation der Workflowbeschreibung, sowie Erzeugung und Ausführung der Workflowinstanzen. Bei der Ausführung von Workflows werden die Interaktionen mit den Teilnehmern sowie Applikationen von dem WfMS kontrolliert [WfMC99].

Der Begriff des transaktionalen Workflows (*Transactional Workflow)* verdeutlicht die Bedeutung, die Transaktionen in Workflows einnehmen können. Dabei handelt es sich um eine koordinierte Ausführung von den in Beziehung stehender Aktivitäten, welche dabei auf heterogene, autonome und verteile Systeme, unter Ausnutzung von transaktionalen Eigenschaften, zugreifen. Workflows sind dann transaktional, wenn ihre Workflowmodelle und WfMS Transaktionen abbilden und absichern [ShRu93, RuSh95].

2.4.4 Mobile Prozesse

Während in der Wirtschaft eine Prozessorientierung und Prozessautomatisierung durch Workflows mittlerweile weitgehend akzeptiert ist, ist der Einsatz von Prozessen im mobilen Umfeld stark unterentwickelt. Der Austausch und die verteilte Ausführung von Prozessen werden von keinem der bereits bestehenden Ansätze ausreichend unterstützt. Dennoch kann eine prozessorientierte Betrachtung eine bessere Modellierung und Handhabung von komplexen oder sich relativ oft verändernden Aufgaben ermöglichen. Weiterhin können durch Prozesse lang andauernde Aufgaben in einer mobilen Umgebung unterstützt werden [Kun05].

Kunze, Zaplata und Lamersdorf schlagen folgende Definition der *Mobilen Prozesse* vor:

> *A mobile process is a sequence of (remote) services which may last over a longer period of time and span several devices during its execution. The results of the process are the effects the initiator expects from it* [KZL06, Seite 2].

Dabei werden mobile Prozesse als eine Art mobile Workflows betrachtet, denn die Trennung von den Prozessen in der realen Welt und Prozessen in einem Computersystem wird in der Definition nicht explizit vorgenommen. Im Unterschied zu betrieblichen Geschäftsprozessen sind Mobile Prozesse eher benutzerzentriert. Dies bedeutet, dass sie im Auftrag eines Benutzers gestartet und ausgeführt werden. Dabei basiert die Applikationslogik in traditionellen mobilen Systemen in der Regel auf explizit definierten lokalen und entfernten Diensten. Im Gegenzug dazu, ermöglichen Mobile Prozesse eine partielle Verlagerung der Applikationslogik in die Middleware, wobei Applikationen nur die benötigten Daten sammeln, um eine Prozessbeschreibung generieren oder vervollständigen zu können [KZL06].

Ferner kann die Ausführung Mobiler Prozesse sehr lange andauern (Stunden, Tage oder Wochen). In dieser Zeit können in einer mobilen Umgebung dramatische Veränderungen stattfinden, daher ist in diesem Fall die Strategie der späten Bindung erforderlich, jedoch nicht immer ausreichend. Daher wird bei der Ausführung der Mobilen Prozesse eine opportunistische Strategie angewendet die besagt, dass solange eine Prozessausführungskomponente eines Geräts auf die für die Ausführung benötigten Dienste zugreifen kann, ist sie auch für die Prozessausführung verantwortlich. Alternativ, z.B. im Fehlerfall oder, wenn eine passende Dienstinstanz vom aktuellen Gerät nicht gefunden werden kann, kann die Prozessausführung versuchen andere Geräte zu finden, welche in der Lage sind, Mobile Prozesse auszuführen. Danach kann der aktuelle Prozess mit seinem Ausführungsstatus auf eines dieser Geräte übertragen und weiter ausgeführt werden. Diese Transferfähigkeit wird in dieser Arbeit auch als *Migrationsfähigkeit* der Mobilen Prozesse bezeichnet. Da Mobile Prozesse während der Ausführung mehrere Geräte umspannen können, werden sie auch synonym als verteilt ausgeführte Mobile Prozesse bezeichnet [KLZ06].

Der Einbezug von Diensten ist ein weiterer Aspekt von Mobilen Prozessen. Jede Aktivität im Prozess wird zur Laufzeit durch eine Dienstinstanz ausgeführt. Entscheidend ist, dass Dienste sehr abstrakt beschrieben werden. Genau genommen werden im Prozess nur Dienstklassen beschrieben. Erst zur Laufzeit und erst, wenn eine Dienstinstanz benötigt wird, wird eine der Dienstklasse entsprechende Dienstinstanz in der Umgebung gesucht. Dabei können verschiedene nichtfunktionale Aspekte wie z.B. Verbindungsart, oder Kosten einbezogen werden [Kun05].

2.5 Dienste

Service-Oriented Computing ist ein Paradigma, welches die Dienste (*Services*) als fundamentale Elemente für die Anwendungs- und Systementwicklung vorsieht und nutzt.

Dienstorientierung ist somit eng mit der *Service Oriented Architecture* (*SOA*) verwandt [Pap03].

Dabei wird SOA im Referenzmodell der Service Oriented Architecture von OASIS [MLMB+06] wie folgt definiert:

> *Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains* [MLMB+06, Seite 8].

Im Unterschied zur objektorientierten Programmierung, wo der Fokus auf der Bündelung von Daten mit Operationen liegt, fokussiert sich SOA auf eine Aufgabe bzw. Geschäftsfunktion. Organisationsbereiche oder Domänen haben verschiedene Geschäftsfähigkeiten *(Capabilities)*, welche durch entsprechende Dienste bereitgestellt werden können [MLMB+06].

Dienste *(Services)* sind also Mechanismen, die den Zugang zu Fähigkeiten (*Capabilities*) ermöglichen. Der Zugang findet hierbei durch veröffentliche Schnittstelle des Dienstes statt, wobei die Spezifizierten Bedingungen *(Constraints and Policies)* eingehalten werden. Dabei werden die Implementierungsdetails eines Dienstes nicht offengelegt. Die Implementierung eines Dienstes muss nicht automatisiert sein – er kann komplett oder teilweise durch menschliche Aktivitäten aufgebaut sein [MLMB+06, All06].

Zwei weitere grundlegende Eigenschaften der SOA sind eine lose Kopplung der Dienste und die Verwendung von offenen Standards. Lose Kopplung bedeutet, dass Dienste erst bei Bedarf dynamisch gesucht und eingebunden werden. Das Einbinden der Dienste findet zur Laufzeit statt, und muss zum Zeitpunkt der Programmübersetzung nicht bekannt sein. Der Dienst ist dabei durch seine Schnittstelle beschrieben. In einer Schnittstelle befindet sich somit die gesamte benötigte Information die das technische Einbinden und Nutzen eines Dienstes ermöglicht. Genau dieser Umstand macht eine konsequente Verwendung von Standards nötig. Nur standardkonforme Komponenten können automatisch gefunden und eingebunden werden. Zurzeit stellen Web Services den verbreiteten Standard in diesem Zusammenhang dar [DJMZ05].

Dabei haben die von Mobilen Prozessen verwendeten Dienste mit den Diensten einer SOA viel gemeinsam. Auch diese sind durch ihre Schnittstellen beschrieben und werden zur Laufzeit dynamisch gesucht und unter vereinbarten Bedingungen ausgeführt [KLZ06].

Im Fokus dieser Arbeit liegt die Gegebenheit, dass Dienste bzw. Dienstklassen, lediglich durch ihre Schnittstellen beschrieben und verwendbar sind. So das keine Annahmen über ihre Implementierung gemacht werden können. So kann im Unterschied zu Datenbankoperationen ein Dienst komplexe Handlungen (z.B. mit Benutzerinteraktionen) abbilden, was zu langen Ausführungszeiten führen kann. Jedoch ist die Semantik der Dienste sowie Aspekte der Dienstbereitstellung oder gar Architektur der Dienste nicht das Thema dieser Arbeit.

3 Transaktionskonzepte

Im ersten Teil dieses Kapitels werden grundlegende transaktionale Konzepte erläutert. Diese werden anschließend als Basis für weitergehende Diskussionen genutzt. Dabei wird als Erstes der Begriff der Transaktion definiert. Danach wird das Konzept der Kontrollsphären erläutert, sowie die Aktionseigenschaften einer Transaktion beschrieben. Weiterhin werden die Grundlagen der Transaktionsverteilung erklärt.

Auf dieser Grundlage aufbauend wird danach ein Überblick über einige wichtige und bewährte Transaktionskonzepte gegeben, wobei in dieser Arbeit zwei Grundtypen von Transaktionen unterschieden werden: die *klassischen* und die *kompensationsbasierten*. Es werden allgemeine Transaktionsansätze der beiden Typen vorgestellt. Zudem werden auch spezielle Transaktionskonzepte für Workflow- und Mobile Systeme vorgestellt. Zum Schluss werden die Anforderungen an die Transaktionsunterstützung explizit definiert und damit die vorgestellten Transaktionskonzepte bewertet.

3.1 Transaktionen

Abstrakt gesehen ist eine *Transaktion* eine Menge von logisch zusammengehörenden Operationen bzw. Aktionen, die unter bestimmten Garantien ausgeführt werden, wobei mehrere Arten der Operationen unterschieden werden (vgl. 3.1.2). Die Grundlagen des Transaktionskonzepts wurden bereits in den siebziger Jahren entwickelt, dabei handelt es sich um Transaktionen, die heute als klassisch, traditionell oder auch als ACID-Transaktionen bezeichnet werden. Ein Anstoß zur Entwicklung des Transaktionskonzeptes hat sich aus dem Konzept der Kontrollsphären (vgl. 3.1.1) ergeben, das einen ersten Versuch darstellt die zusammenhängenden Fragen der Ausführungskontrolle, Recovery und Nebenläufigkeit systematisch zu untersuchen [GrRe93].

Klassische Transaktionen sind auf dem Gebiet der Datenbank Systeme (DBS) entstanden und beruhen auf den so genannten ACID Eigenschaften. ACID steht dabei für *Atomarität (Atomicity)*, *Konsistenz (Consistency)*, *Isolation (Isolation)* und *Dauerhaftigkeit (Durability)*. Im Folgenden werden diese Eigenschaften näher erläutert [HaRe83, GrRe93]:

- *Atomarität*: Entweder werden alle Aktionen innerhalb einer Transaktion ausgeführt oder keine davon. Es gibt also kein Zwischenergebnis, so dass eine Transaktion eine unteilbare Aktion darstellt.
- *Konsistenz*: Eine Transaktion darf nur ein konsistentes Ergebnis produzieren. Dabei ist ein Ergebnis dann konsistent, wenn es alle definierten Integritätsbedingungen erfüllt.

- *Isolation*: Transaktionen können nebenläufig ausgeführt werden. Dabei dürfen sie sich gegenseitig nicht beeinflussen. Im Idealfall ist das Verhalten von nebenläufigen Transaktionen serialisierbar. Serialisierbarkeit bedeutet, dass ein Programm, welches unter Transaktionsschutz abläuft, sich exakt so verhält, als ob es in einem Einbenutzer-Modus ablaufen würde. Dies wird i.d.R. durch Sperrmechanismen erreicht. Beispielsweise können die von einer Transaktion gebrauchten Objekte bis zu ihrem Ende gesperrt werden, so dass konkurrierende Transaktionen auf gesperrte Objekte nicht zugreifen können.
- *Dauerhaftigkeit*: Die Änderungen einer erfolgreich ausgeführten Transaktion werden dauerhaft. Sie werden zur abgebildeten Realität und können nur durch andere Transaktionen mit einem Gegenalgorithmus rückgängig gemacht werden.

Ein Transaktionsmodell, welches die ACID Eigenschaften strikt erfüllt, wird als *Flache Transaktion (Flat transaction)* bezeichnet. Es wurde für kurze (Buchungs-) Operationen im Bereich der Bankwirtschaft entwickelt und ist heute eine verbreitete Transaktionsart, die von nahezu jedem Datenbanksystem unterstützt wird. Alle Aktionen in einer Flachen Transaktion befinden sich zwischen ihrem Anfang, also dem Befehl BEGIN WORK, und ihrem Ende (COMMIT WORK). Innerhalb dieser Grenzen werden entweder alle Operationen erfolgreich ausgeführt oder alle zurückgesetzt [GrRe93].

ACID-Transaktionen haben sich vor allem in Datenbanksystemen mit relativ kurz andauernden Transaktionen gut bewährt, weswegen dieses erfolgreiche Konzept durch Anpassung auch auf andere Bereiche angewandt wurde. Die Erweiterungen von ACID Transaktionen werden in Kapitel 4 dargestellt.

3.1.1 Kontrollsphären

Viele Transaktionskonzepte wurden bereits Anfang der siebziger Jahre am Konzept der *Kontrollsphären* (*Spheres of control: SoC*) untersucht. Mit Hilfe von Kontrollsphären hat man versucht die Konsistenzprobleme anwendungsunabhängig in Griff zu bekommen, die bei Programmausführungen in verteilten Mehrbenutzersystemen entstehen können. Dabei besagt das Grundkonzept der Kontrollsphären, dass jede Art von Ausführungskontrolle zunächst ihre logischen Grenzen (Kontrollsphäre) kennzeichnen soll. Wenden Transaktionen dieses Konzept an, so sind die Grenzen der Transaktionen i.d.R. eindeutig definiert. [GrRe93, Dav78].

Kontrollsphären sind als ein allgemeines Konzept zu verstehen und können verschiedenen Zwecken dienen, unter anderem der Transaktions-, Versions- und Zugriffskontrolle. Wobei eine Kontrollsphäre, aus Sicht der Außenwelt, eine Atomare Operation bildet. Diese kann ebenfalls aus Kontrollsphären aufgebaut sein. Eine Kontrollsphäre verwaltet die Freigabe

von Änderungen, die durch Operationen innerhalb der Sphäre gemacht werden und führt das Monitoring und das Aufzeichnen von Funktionsabhängigkeiten durch. Im Fehlerfall kann somit die Ausführungshistorie analysiert werden [GrRe93, Dav78].

Zur Verdeutlichung werden die beschriebenen Zusammenhänge am Beispiel der Prozesskontrolle (*process control*) erläutert. Es handelt sich dabei nicht um in 2.4 vorgestellte Prozesse, sondern um allgemeine Ablaufe auf einem Computersystem, also z.B. um Prozesse eines Betriebsystems. Die Prozesskontrolle stellt sicher, dass Operationen auf jeder Ebene einer Abstraktionshierarhie durch Operationen einer darunterliegenden Ebene definiert werden. Somit sind den Operationen einer Ebene nur die Operationen einer darunterliegenden Schicht sichtbar. Dadurch werden zwei Eigenschaften der Prozesskontrolle möglich: *Prozessatomarität* und *Prozessfreigabe* [Dav78].

Die Prozessatomarität gestattet es, eine Operation als atomar auf einer Ebene der Abstraktionshierarhie zu betrachten auch, wenn ihre Implementierung aus einer Menge von parallelen und seriellen Operationen auf der nächsttieferen Ebene zusammengesetzt ist. Abbildung 8 veranschaulicht die Hierarchie eines atomaren Prozesses. Dabei wird die Funktion *A1* als ein Ganzes von außerhalb gesehen. Für *A1* sind die Funktionen *B1* und *B2* sichtbar, nicht aber *C1* und *C2*. *C1* und *C2* sind parallele Prozesse (Funktionen), die nur für *B2* sichtbar sind [Dav78].

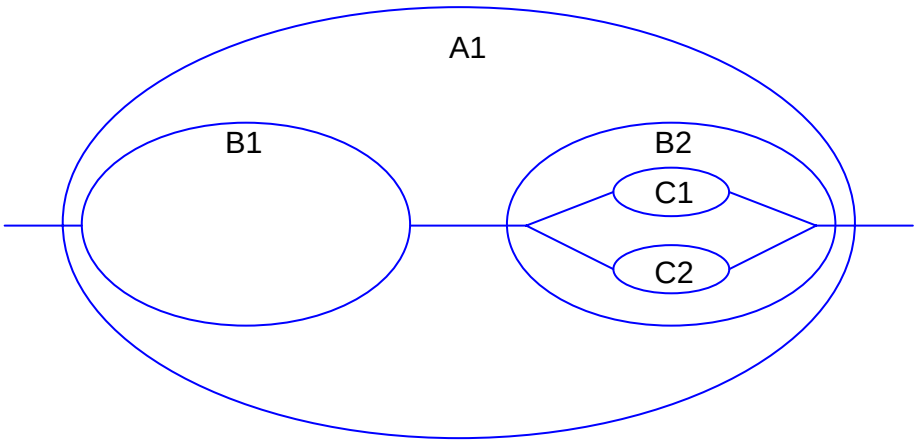

Abbildung 8: Hierarchien von Kontrollbereichen: Prozessatomarität (nach [Dav78])

Prozessfreigabe bedeutet, dass Änderungen einer Funktion, solange diese ausgeführt wird, lokal bleiben. Somit werden die Ergebnisse erst nach einem erfolgreichen Abschluss der Funktion freigegeben. Solange die Ergebnisse nicht freigegeben sind, kann das System ein einseitiges Rücksetzen durchführen. Einseitig bedeutet in diesem Fall, dass beim Rücksetzen nicht nach der Erlaubnis der Sphärenteilnehmer gefragt wird [Dav78].

3.1.2 Aktionsarten

Einige Transaktionskonzepte setzen bestimmte Eigenschaften der beteiligten Operationen voraus, deshalb werden diese Eigenschaften in diesem Abschnitt genauer betrachtet. Der Aktionsbegriff (*Action*) ist in diesem Fall ein Synonym für den Operationsbegriff. Jedoch ist der Begriff einer Aktion eher abstrakt, so dass darunter alle und nicht nur die computergestützten Operationen gemeint sind. Dabei unterscheiden Gray und Reuter zwischen drei Arten von Aktionen [GrRe93]:

- *Ungeschützte Aktionen (unprotected actions):* Diesen Aktionen fehlen alle ACID Eigenschaften außer der Konsistenz.
- *Geschützte Aktionen (protected actions):* Aktionen dieses Typs externalisieren ihre Ergebnisse nicht, bevor sie ganz fertig sind. Das bedeutet, dass, falls während der Ausführung ein Fehler auftritt, die Änderungen auf den „Vorher-Zustand" zurückgesetzt werden können. Wenn aber das Aktionsende erreicht wurde, ist die unilaterale Rücksetzung nicht mehr möglich. Geschütze Aktionen sind auch von den anderen Aktionen isoliert und besitzen damit die ACID Eigenschaften.
- *Reale Aktionen (real actions):* Diese Aktionsart betrifft die Änderungen bzw. Aktionen in der realen, physischen Welt. In diesem Sinne sind sie nur schwer oder gar nicht umkehrbar. Das Bohren einer Öffnung in ein Metallstück oder die Bargeldausgabe an einem Geldautomat sind Beispiele hierfür. Reale Aktionen können konsistent und isoliert sein. Wenn sie einmal ausgeführt wurden, bleiben sie definitiv dauerhaft. Es ist also besonders schwer diese Aktionen in einer höherwertigen atomaren Aktion abzubilden. Das folgt aus ihrer Irreversibilität. Wenn für eine Aktion die Option „Zurück" nicht existiert, kann auch eine höherwertige Aktion die solche Aktionen einschließt, nicht (atomar) zurückgesetzt werden.

Ungeschützte Aktionen können im Unterschied zu realen Aktionen umgekehrt werden. Damit ist es möglich sie in höherwertige Operationen einzuschließen, welche als Ganzes die ACID-Eigenschaften garantieren. Die Implementierung eines solchen höherwertigen Mechanismus sollte die Umkehrmechanismen von nichtgeschützten Aktionen beherrschen. Dieses geschieht beispielsweise in einem Datenbanksystem: während auf den unteren Ebenen eines DBMS einige ungeschützte Lese/Schreib-Operationen stattfinden, werden sie auf Benutzerebene zu Transaktionen zusammengefasst. Weiterhin können Aktivitäten in Workflows, als Aktionen aufgefasst werden, da sie bei der Ausführung der Workflows auch an Transaktionen teilnehmen können. Sie können ACID Eigenschaften garantieren, oder Aktionen der physischen Welt abbilden [GrRe93].

3.2 Erweiterungen Klassischer Transaktionen

In 3.1 wurden die Grundlagen der klassischen Transaktionen vorgestellt. Das Grundmodell einer klassischen Transaktion ist die flache Transaktion, welche in der Praxis eine breite Anwendung fand. Jedoch sind ACID Transaktionen in ihrer Ursprungsform für viele moderne Anwendungen und Prozesse ungeeignet, da es sich dabei oftmals um lang andauernde Anwendungen handelt. Bei der Ausführung von lang andauernden Transaktionen erweisen sich die ACID Eigenschaften oftmals als zu strickt, da durch länger anhaltende Sperren Performanz- und Durchsatzprobleme entstehen können. Deshalb wurden auf der Basis von ACID-Transaktionen auch weitere Transaktionsmodelle entwickelt, die im Folgenden vorgestellt werden [GrRe93].

3.2.1 Erweiterungen einer Flachen Transaktion

Aufgaben mit einer sehr großen Anzahl an Operationen, wie z.B. zahlreiche Aktualisierungen bei Jahreszinsenberechnung auf allen Konten bei einer Bank, können durch die Grundform der Flachen Transaktion nicht effizient abgebildet werden. Im Fehlerfall muss die gesamte bis dahin gemachte Arbeit abgebrochen und wiederholt werden, was in der Regel zu vermeiden gilt. Im Folgenden werden Erweiterungen des Grundmodells vorgestellt, welche lang andauernde Transaktionen besser unterstützen sollen [GrRe93].

Sicherungspunkte

Um ein teueres Rücksetzen einer lang andauernden Transaktion zu vermeiden, hat man Transaktionen durch Einführen von *Sicherungspunkten (Save Points)* in kleinere interne Einheiten aufgeteilt. Diese Sicherungspunkte können mit dem Aufruf von SAVE WORK installiert werden. Sie ermöglichen das schrittweise Ausführen bzw. Rücksetzen einer Transaktion. In einem Fehlerfall muss nicht die gesamte Transaktion rückgängig gemacht werden, sondern nur bis zum letzten Sicherungspunkt (es ist durchaus möglich auch mehrere Schritte zurück zu gehen). Ab diesem Punkt kann die Transaktion wieder anlaufen. Wie auch bei der Flachen Transaktion werden die Ergebnisse erst nach einem COMMIT nach Außen freigegeben [GrRe93].

Verkettete Transaktionen

Verkettete Transaktionen (Chained Transactions) sind eine Variation der Sicherungspunkte. Im Unterschied zu den flüchtigen Sicherungspunkten handelt es sich jedoch hierbei um einzelne aneinander geketteten Transaktionen. Statt einen Sicherungspunkt zu setzen wird die Teiltransaktion mit dem Befehl CHAIN WORK festgeschrieben und gleichzeitig eine

neue gestartet. Dieser Übergang zwischen den Transaktionen ist eine atomare Operation und die gesetzten Sperren bleiben erhalten und sind global. Die Isolation bleibt damit während der gesamten Transaktion trotzdem erhalten [GrRe93].

Zu den oben geschilderten Vorteilen dieses Modells kommen noch die Performanzvorteile hinzu. Dabei können nicht mehr benötigte Sperren bei jedem Teil-COMMIT freigegeben werden. Nachteilig ist die Lockerung der Atomaritätseigenschaft. Im Falle eines Systemabsturzes kann diese nicht vollständig garantiert werden. Eine Wiederherstellung ist dabei lediglich bis zum letzten Teil-COMMIT möglich, was in der Regel akzeptiert werden kann. [GrRe93].

3.2.2 Verteilte Transaktionen

Unter einer *verteilten Transaktion (Distributed Transaction)* wird meistens eine Flache Transaktion verstanden, welche in einer verteilten Umgebung abläuft. Während des Ablaufes müssen, abhängig von den betroffenen Daten, mehrere Netzwerkknoten eingebettet werden. Folglich hängt die Struktur einer einfachen verteilten Transaktion von der Datenverteilung im Netzwerk ab. [GrRe93].

Eine verteilte Transaktion erfüllt alle ACID-Eigenschaften der Flachen Transaktion, obwohl die Transaktionskontrolle in diesem Fall über mehrere Netzwerkknoten verteilt ist. Um die globale Konsistenz und Atomarität wahren zu können, müssen bei einem COMMIT alle beteiligten Knoten ihre Änderungen festschreiben und freigeben (externalisieren). Entsprechend sollten im Abbruchsfall alle in der Transaktion beteiligten Knoten ihre Änderungen zurücknehmen, ohne sie vorher externalisiert zu haben.

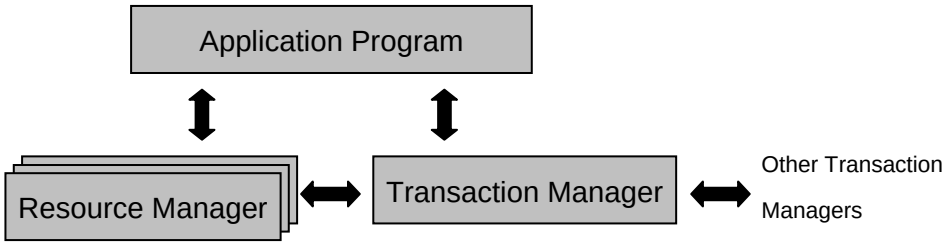

Abbildung 9: X/Open Transaction Model (nach [BeNe97])

Zur Verdeutlichung dieses Ablaufs ist in der Abbildung 9 das Transaktionsmodell des X/Open (heute Open Group[11]) Konsortiums dargestellt. In diesem Modell kann ein *Transaktionsmanager* (*Transaction Manager)* mit der Applikation, den *Resource Managern* sowie mit anderen Transaction Managern kommunizieren. Die unter der Verwaltung eines

[11] Vgl. http://www.opengroup.org/

Resource Managers stehenden Ressourcen sind beispielsweise Datenbanken, Dateien, Nachrichten und andere Objekte auf die im Ablauf einer Transaktion zugegriffen werden kann. Dabei führt ein Transaktionsmanager im Auftrag der Applikation Grundoperationen der Transaktionen (BEGIN, COMMIT, ABORT) aus [BeNe97].

Im Zuge der Ausführung einer verteilten Transaktion wird auf Ressourcen zugegriffen, welche dabei von verschiedenen Resource Managern verwaltet werden können. Beim Zugriff auf diese Resourcen registrieren sich die verantwortlichen Resource Manager beim lokalen Transaktionsmanager. Wenn eine Transaktion beendet werden soll, muss eine Entscheidung über den Transaktionsausgang an alle Resource Manager propagiert werden [BeNe97].

Dieser Vorgang läuft mit Hilfe so genannter Commit-Protokolle ab. Bei der Beschreibung von Commit-Protokollen werden die beteiligten Resource Manager oft als *Teilnehmer* bezeichnet, wobei der Transaktionsmanager die Rolle eines *Koordinators* erhält. Das bekannteste Commit-Protokoll ist das *Zwei-Phasen-Commit-Protokoll (two-phase commit protocol: 2PC)*. Wie der Name des Protokolls bereits sagt, wird eine Commit-Entscheidung in zwei Phasen getroffen und an alle Teilnehmer propagiert [BeNe97].

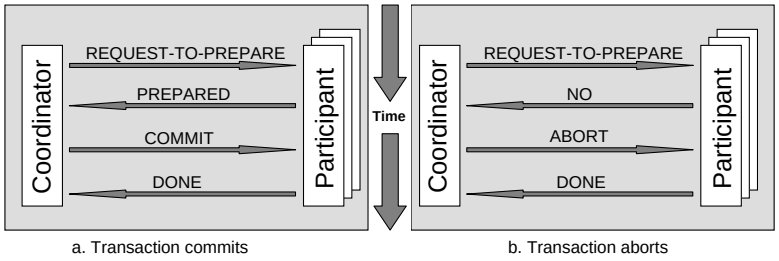

Abbildung 10: Zwei-Phasen-Commit-Protokoll (nach [BeNe97])

Der schematische Ablauf des 2PC Protokolls ist in der Abbildung 10 dargestellt, wobei links ein erfolgreicher Verlauf und rechts ein Abbruch der Transaktion zu sehen ist. In der ersten Phase schickt ein Koordinator an alle Teilnehmer eine REQUEST-TO-PREPARE-Nachricht. Darauf hin antworten die Teilnehmer mit einer PREPARED-Nachricht, wenn sie zum Festschreiben breit sind. Anderenfalls wird mit einer NO-Nachricht geantwortet. Der Koordinator wartet dabei bis alle Teilnehmer geantwortet haben. Die zweite Phase des Protokolls beginnt, wenn der Koordinator alle Nachrichten empfangen hat oder Zeitrestriktionen überschritten wurden. Wenn ein Koordinator nur PREPARED-Nachrichten von allen Transaktionsteilnehmern erhalten hat, trifft er die Entscheidung zur Ausführung des Commits. In anderen Fällen entscheidet er abzubrechen. Danach schickt der Koordinator seine Entscheidung an alle Teilnehmer. Beim Empfang einer COMMIT-Nachricht schreiben die Teilnehmer ihre lokalen Änderungen fest, beim Empfang einer

ABORT-Nachricht setzen sie diese Änderungen zurück. Danach wird mit einer bestätigenden Nachricht DONE geantwortet [BeNe97].

Weiterhin behandelt das 2PC-Protokoll auch Fehler, die z.B. durch verloren gegangene Nachrichten entstehen können. Dazu können in einigen Fällen (z.B. bei Transaktionen mit nur lesenden Zugriffen) einige Optimierungen vorgenommen werden. Beides wird jedoch im Rahmen dieser Arbeit nicht detailliert betrachtet [BeNe97].

Zur Sicherstellung der Isolation kann zusätzlich ein so genannter Lock Manager verwendet werden, welcher die Sperren aller beteiligten Ressourcen und konkurrierende Transaktionen verwaltet. Damit werden auch die Konsistenzeigenschaften erfüllt [GrRe93].

3.2.3 Geschlossen geschachtelte Transaktion

Geschlossen geschachtelte Transaktionen (Closed Nested Transactions) können als Generalisierung der Transaktion mit Sicherungspunkten aufgefasst werden. Jedoch formen sie die einzelnen Untertransaktionen zur einen Transaktionshierarchie. Wobei die Struktur einer geschachtelten Transaktion von funktionalen Aspekten bestimmt wird. Im Unterschied zu Verteilten Transaktionen (vgl. 3.1.3) können die Untertransaktionen in geschlossen geschachtelten Transaktionen unabhängig von einander ein COMMIT oder ein ROLLBACK ausführen. Dabei beschreibt die folgende formale Definition das Modell und ihr Verhalten präzise [Mos81, GrRe93]:

- Eine geschachtelte Transaktion ist eine Baumstruktur von Teiltransaktionen, wobei die Subtransaktionen wiederum geschachtelte oder Flache Transaktionen sind.
- Blätter des Transaktionsbaumes sind Flache Transaktionen. Die Pfadlänge von der Wurzel zu den Blättern, kann sich auf verschiedenen „Ästen“ unterscheiden.
- Die Transaktion an der Wurzel heißt Top-Level Transaktion, die anderen sind Untertransaktionen. Ein Vorgänger einer Transaktion heißt *Vater* (*parent)*, eine Nachfolger-Transaktion heißt *Kind (child)*.
- Eine Untertransaktion kann festschreiben oder abbrechen. Das Festschreiben einer Untertransaktion hat keinen Effekt bis seine Vatertransaktion ebenfalls sein COMMIT ausführt. Führt eine Untertransaktion ihr COMMIT aus, werden ihre Ergebnisse nur der jeweiligen Vatertransaktion sichtbar gemacht. Dieses bedeutet, dass effektiv die Untertransaktionen nur dann festschreiben, wenn die Top-Level Transaktion festschreibt.
- Wenn eine Transaktion irgendwo in dem Transaktionsbaum ein ROLLBACK ausführt, müssen auch alle Kindtransaktionen einen ROLLBACK ausführen. Somit haben die Untertransaktionen nur eine eingeschränkte Dauerhaftigkeitseigenschaft.

Da die Strukturierung des Transaktionsbaumes in der Regel funktional geschieht, können einzelne Transaktionsbaumäste meist effizient unabhängig von einander auf verschiedenen Netzwerkknoten ausgeführt werden. Somit wird ein hoher Grad an paralleler Ausführung ermöglicht. Auch die Möglichkeit des partiellen Scheiterns ist vorteilhaft beim Einsatz von Geschachtelten Transaktionen in einem verteilten Umfeld. In diesem Fall muss wegen des Ausfalls eines Knotens, der in einer Transaktion beteiligt war, eventuell nicht die ganze Transaktion abgebrochen werden. Die gescheiterte Ausführung eines Astes kann z.B. auf einem anderen Knoten wiederholt werden. Diese Eigenschaften werden mit der Anzahl von beteiligten Knoten bzw. mit der Zunahme von Ausfallwahrscheinlichkeiten der einzelnen Komponenten immer bedeutender [GrRe93].

3.3 Kompensationsbasierte Transaktionen

Kompensationsbasierte Transaktionen unterscheiden sich von klassischen Transaktionen durch ihre optimistische Strategie. Die meisten Modelle geben die gesetzten Sperren möglichst früh wieder frei, so dass auch Ergebnisse früher externalisiert werden. Zu dieser Art von Transaktionen werden alle Modelle gezählt, die auf Kompensationsmechanismen zurückgreifen. Dabei kann die offen geschachtelte Transaktion als Basis aller kompensationsbasierten Modelle aufgefasst werden.

3.3.1 Offen geschachtelte Transaktion

Offen geschachtelte Transaktionen (Open Nested Transactions) sind „anarchische“ Verwandte von geschlossen geschachtelten Transaktionen. In der offen geschachtelten Variante können die Untertransaktionen jeweils ganz unabhängig von anderen Transaktionen festschreiben oder zurücksetzen, so dass ACID Eigenschaften nicht mehr ohne weiteres gewährleistet sind. Die semantische Atomarität kann jedoch mit Hilfe von *Kompensationstransaktion (Compensating Transaction)* gewahrt werden. Dabei ist für eine Transaktion T eine Transaktion C dann kompensierend, wenn sie semantisch die Ergebnisse einer erfolgreich abgelaufenen Transaktion T wieder aufhebt. Semantische Kompensation kann dabei nicht nur eine Wiederherstellung eines vorherigen Zustandes bedeuten, sondern, wenn z.B. dieser nicht mehr wiederherstellbar ist, eine Abfolge beliebiger Aktionen sein, welche eine semantische Kompensation in Bezug auf die Funktionalität darstellt [Gra81, GrRe93].

Die offen geschachtelte Transaktion ist, durch die Abwesenheit von Sichtbarkeits- und COMMIT-Regelungen wie bei einer geschlossen geschachtelten Transaktion, sehr flexibel. Die Art der Anordnung von Subtransaktionen kann den jeweiligen Systemanforderungen angepasst werden. Diese Eigenschaft macht offen geschachtelte Transaktionen zur Basis für zahlreiche spezifische Modelle. Einige Modelle sehen dabei eine globale Kompensation vor, wobei, im Falle eines Scheiterns einer Untertransaktion, die Auswirkungen von der

gesamten globalen Transaktion kompensiert werden. Auch Mischformen sind möglich. Weiterhin ermöglichen kompensationsbasierte Lösungen einen leichteren Einbezug von nicht geschützten Aktionen.

3.3.2 Sagas

Saga ist ein einfaches Konzept, welches auf offen geschachtelten Transaktionen aufbaut. Die Schachtelung ist in diesem Fall jedoch auf zwei Stufen begrenzt. Eine *Saga* ist eine lang andauernde Transaktion, welche mit Hilfe von gewöhnlichen und kompensierenden Transaktionen modelliert wird. Dabei wird eine langlebige Transaktion in Subtransaktionen *T1, T2, …, Tn* geteilt. Diese Subtransaktionen werden sequenziell ausgeführt und externalisieren ihre Ergebnisse sofort. Zu jeder Subtransaktion gibt es dabei dazugehörende kompensierende Transaktionen *C1, C2, …, Cn*. Es sind nur zwei Abläufe von Subtransaktionen in einer Saga möglich. Im ersten Fall laufen alle Transaktion *T1* bis *Tn* erfolgreich durch und Saga erfolgreich abschließt. Der zweite Fall tritt dann auf, wenn eine der Subtransaktionen *Ti* scheitert. In diesem Fall werden die kompensierenden Transaktionen nacheinander in umgekehrter Reihenfolge gestartet, somit ergibt sich der folgende Verlauf *T1, T2, …, Ti, Ci, …, C2, C1* [GMS87].

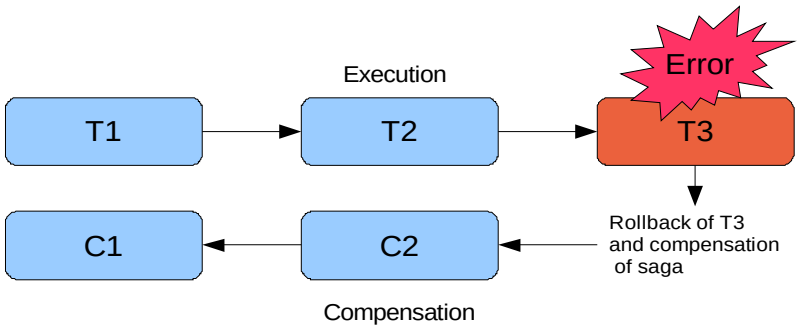

Abbildung 11: Saga-Ablauf im Fehlerfall (nach [Rie05])

Dieser Verlauf ist in der Abbildung 11 dargestellt. Die Subtransaktionen *T1, T2* werden erfolgreich durchgeführt, während *T3* scheitert. Da angenommen wird, dass T3 eine (Flache) klassische Transaktion ist, wird sie einfach auf den alten Zustand zurückgesetzt, so dass nur die kompensierende Transaktionen *C2* und *C1* gestartet werden müssen.
Eine Subtransaktion in einer Saga kann, sobald sie fertig ist, festschreiben, ohne auf andere Subtransaktionen in der Saga zu warten, die einzelnen Subtransaktionen besitzen dabei die ACID Eigenschaften. Im Falle ihres Scheiterns gibt es zwei Möglichkeiten der Fehlerbehebung: *Forward* und *Backward Recovery*.

Dabei werden unter Forward Recovery Maßnahmen des erneuten Anlaufs der gescheiterten Subtransaktion verstanden. Forward Recovery kann somit das Zurücksetzen

der gesamten Saga verhindern, was bei lang andauernden Sagas einen großen Nutzen darstellt. Dadurch wird vermieden, dass bereits erledigte Arbeit komplett zurückgesetzt werden muss. Da Forward Recovery jedoch nicht immer funktioniert, muss die Backward Recovery ausgeführt werden. Backward Recovery bedeutet das Aufrufen von Kompensationstransaktionen in der umgekehrten Reihenfolge [GMS87].

In einer Saga ist die Isolationseigenschaft aufgeweicht, da die einzelnen Subtransaktionen ihre Ergebnisse sofort externalisieren. Demzufolge können durch Kompensation nicht die Wiederherstellung aller ursprünglichen Objektzuständen garantiert werden, da die externalisierten Ergebnisse möglicherweise bereits von anderen Transaktionen weiterverwendet wurden. Unter Backward Recovery wird daher semantisch durchgeführte Kompensation, anstatt der physischen Rücksetzung eines vorherigen Zustandes, verstanden [KLS90, GMS87].

3.4 Transaktionskontrolle in Workflows

In diesem Abschnitt werden die speziell für Workflows entwickelten Transaktionskonzepte dargestellt. Zunächst werden die *Atomaren Sphären* [Ley97], welche auf klassischen Transaktionen basieren, vorgestellt. Anschließend wird die *Kompensationssphäre* [Ley95] erläutert.

3.4.1 Atomare Sphären

Einem Workflow Management System zu ermöglichen, transaktionale Programme (als Aktivitäten) in eine neue Transaktion aufzunehmen, ist die primäre Absicht von *Atomaren Sphären*. Durch Atomare Sphären werden die WfMS um generische Möglichkeiten zur Demarkation und Fehlerbehandlung der Transaktionen angereichert [LeRo99].

Leymann beschreibt eine *Atomare Sphäre* (*Atomic Sphere*) als eine Ansammlung von Aktivitäten, welche die folgenden drei Eigenschaften besitzen [Ley97]:

1. Alle Aktivitäten in der Sphäre haben einen transaktionalen Aufbau. Darunter werden die von Außen gesetzten Transaktionsgrenzen von den Aktivitätsimplementierungen verstanden. Solche Aktivitäten entsprechen den, in dieser Arbeit definierten, geschützten Aktivitäten.
2. Alle Aktivitäten einer Atomaren Sphäre, die einen Vorgänger außerhalb der Sphäre besitzen, haben stets denselben Vorgänger. Oder gar keine Aktivität hat einen Vorgänger außerhalb der Sphäre.
3. Entweder führen alle teilnehmenden Aktivitäten einer Atomaren Sphäre die Transaktion erfolgreich zu Ende oder alle setzen zurück.

Die Erste Eigenschaft ermöglicht die Implementierung von Atomaren Sphären mit *globalen Transaktionen*. Als globale Transaktion werden solche Transaktionen bezeichnet, welche nicht nur einfache Operationen, sondern mehrere Transaktionen (*Lokale Transaktionen*) umfassen, wie das beispielsweise bei Saga der Fall ist (vgl. 4.2.2). Workflow-Systeme setzen die Grenzen der globalen Transaktion über die beteiligten transaktionalen Aktivitäten, welche hinter den Aktivitäten stehen. Damit laufen diese als Subtransaktionen der globalen Transaktion ab [Ley97].

In der Abbildung 12 ist ein Prozessausschnitt mit einer Atomaren Sphäre *S* zu sehen, welche aus zwei Aktivitäten *B* und *C* besteht. Die dritte Eigenschaft der Atomaren Sphäre definiert die Atomarität der Ausführung. In diesem Fall bedeutet es, dass Aktivitäten *B* und *C* entweder beide festschreiben oder keine davon. Wenn die Sphäre scheitert, wird das Workflow Management System die Prozessnavigation durch die Atomare Sphäre wiederholen. Bei Wiederholung wird die Sphäre mit genau den gleichen Aktivitäten betreten mit denen sie das erste Mal angefangen hat. Diese Aktivitäten werden als *Start-Set* von der Atomaren Sphäre bezeichnet. Beispielsweise, wenn die Ausführung der Atomare Sphäre, in der Abbildung 12, beim ersten Mal nur mit Aktivität B beginnt, weil Transitionsbedingung *q* zu *„false“* und *p* zu *„true“* errechnet wurden, so muss auch bei allen Wiederholungen nur Aktivität *B* behandelt werden [LeRo99].

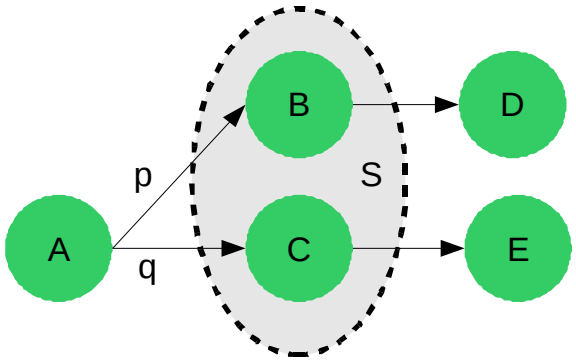

Abbildung 12: Beispiel für eine gültige Atomare Sphäre (nach [LeRo99])

Die zweite Eigenschaft bzw. Bedingung der Atomaren Sphären, wurde aus Gründen der Geschwindigkeit- und Komplexitätsminderung definiert. Denn solange die Aktivitäten innerhalb der Atomaren Sphäre ausführbar sind, wird das Workflow Management System die Globale Transaktion nicht abschließen. Konsequenterweise sollte die Atomare Sphäre möglichst schnell durchlaufen. Anderenfalls werden die Sperren der Ressourcemanager, welche von den Implementierungen der Aktivitäten benutzt werden, lange gehalten, was zum Performanz- und Durchsatzminderungen der ganzen Umgebung führt. Damit die Sphären schneller durchlaufen können, muss zumindest die zweite Bedingung eingehalten

werden, da dieses eine gleichzeitige Identifikation aller ausführbaren Mitglieder des Start-Sets gewährleistet [LeRo99].

Abbildung 13 veranschaulicht eine Atomare Sphäre *S*, welche die zweite Bedingung und die Struktur der Atomaren Sphäre nicht einhält. Die Atomic Sphere *S* hat dabei einen Start-Set *{A, B, D, G}*. Die Aktivitäten *A* und *D* haben beispielsweise verschiedene Vorgänger außerhalb der Sphäre. Es ist also möglich, dass der Vorgänger von *D* erst Stunden später, nach dem *A* gestartet wurde, beendet wird. Dieses kann zu einer unakzeptabel langen Laufzeit von *S* führen. Weiterhin, wenn *A* vollendet ist, muss das Workflow Management System eventuell auf *K* warten, um zu entscheiden, ob *B* ausgeführt werden soll, was wieder zu unakzeptablen Wartezeiten führen kann [LeRo99].

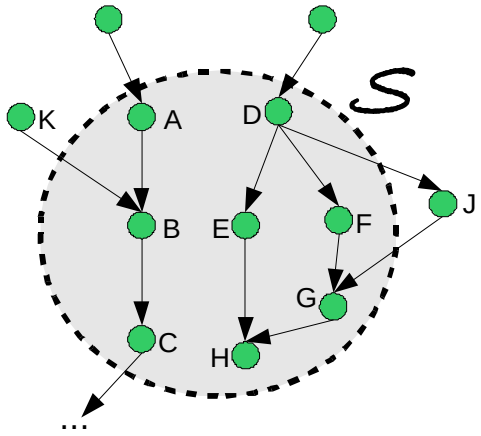

Abbildung 13: Beispiel für ungültige Atomare Sphäre (nach [LeRo99])

Die zweite Strukturbedingung muss demnach bei Atomaren Sphären strikt eingehalten werden. Zudem, gibt es weitere Regeln, wie z.B., dass in einer Atomaren Sphäre nur automatische Aktivitäten sein dürfen, welche nicht auf einen Benutzer warten müssen und i.d.R. schneller ablaufen [LeRo99].

3.4.2 Kompensationssphären

In [Ley95] wird ein weiterer transaktionaler Ansatz für Workflow Management Systeme beschrieben, es handelt sich dabei um *Kompensationssphären* (*Compensation Spheres)*. Eine Kompensationssphäre kann über beliebige Aktivitäten eines Workflows definiert werden. Es sind keine Struktureinschränkungen für eine Kompensationssphäre definiert. Weiterhin können in einer Kompensationssphäre sowohl geschützte als auch nicht geschützte Aktivitäten gemischt vorkommen. Die Aktivitäten in einer solchen Sphäre sind i.d.R. semantisch eng verwandt. Wenn einer der Schritte (Aktivität) in der Sphäre nicht korrekt ausgeführt wird, muss er korrigiert werden. Auch alle anderen Aktivitäten in der Sphäre,

die bereits ausgeführt waren, müssen dabei korrigiert werden. Die Korrektur fehlerhafter Schritte wird mit dem Aufruf von kompensierenden Aktivitäten gewährleistet. Dabei hat eine kompensierende Aktivität alle Eigenschaften einer gewöhnlicher Aktivität[Ley95].

Grundsätzlich sieht eine Kompensationssphäre zwei Arten der Kompensierung vor. Im ersten Fall wird jede Aktivität in der Kompensationssphäre als ein Paar, bestehend aus einer normalen Aktivität (Teil von durchzuführender Arbeit) und dazugehöriger kompensierender Aktivität, realisiert. Im Abbruchsfall werden alle bis zu diesem Zeitpunkt ausgeführten Aktivitäten identifiziert, anschließend die zugehörigen kompensierenden Aktivitäten in umgekehrter Reihenfolge ausgeführt. Genau genommen wird dazu von dem Workflow Management System ein neuer Kompensierungsprozess konstruiert und anschließend ausgeführt. Diese Art der Kompensation wird dabei als diskret bezeichnet [Ley95].

In der Abbildung 14 ist eine Kompensationssphäre *S* zu sehen, welche aus den Aktivitäten *A, C, E, F* besteht. Dabei erkennt man auf der Abbildung, dass die Sphäre nicht unbedingt zusammenhängend sein muss. Weiterhin stellt die Abbildung dar, was passiert, wenn die Sphäre, nach dem die Aktivität *E* ausgeführt wurde, scheitert. Rechts wird ein vom Workflow Management System erzeugtes Prozessmodell *P(S)* gezeigt, welche auf dem aktuellen Status von *S* entstanden ist. Da als letztes die Aktivität *E* bereits ausgeführt wurde, muss diese auch als erste kompensiert werden, nachfolgend werden kompensierende Aktivitäten von *C, F* und schließlich von *A* aufgerufen.

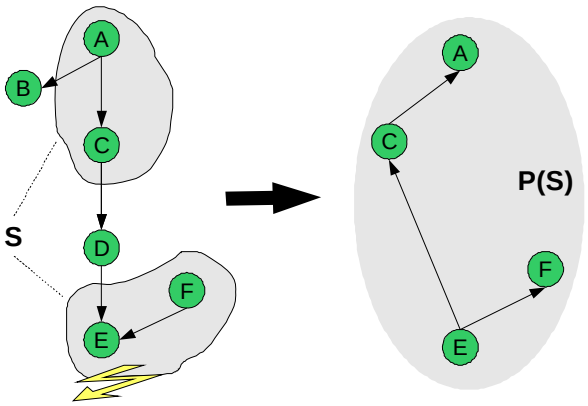

Abbildung 14: Abbruch einer Compensation Sphere (nach [LeRo99])

Dabei sieht die alternative Art der Kompensation vor, dass eine kompensierende Aktivität für die ganze Sphäre definiert wird. Dabei kann diese Aktivität einen ganzen Prozess beinhalten. Im Fehlerfall wird diese Aktivität unabhängig vom Zustand der Sphäre ausgeführt. Ein solches Modellieren der Kompensation wird als global bezeichnet [Ley95].

Die Funktionalität der Kompensationssphären kann mit etwas mehr Aufwand, durch Hinzunahme von zusätzlichen Kontrollflusskonnektoren und Transitionsbedingungen, in der Regel auch ohne die Kompensationssphären modelliert werden. Dieses Vorgehen führt jedoch zu einer Reihe von Nachteilen. Erstens, weist ein solches Prozessmodell vergleichbar höhere Komplexität auf, da alle Alternativen und Ausnahmen explizit definiert werden müssen. Die Behandlung von Ausnahmen wird aber oft nicht als ein Aspekt der Prozessmodellierung, sondern als ein eher technischer Aspekt der IT verstanden [LeRo99].

Letztendlich ermöglicht der Einsatz von Kompensationssphären eine erhöhte Flexibilität der Prozessmodellierung. Beispielsweise, kann eine längst abgeschlossene Kompensationssphäre auch zum späteren Zeitpunkt noch zurückgesetzt werden, wenn die Information über einen Fehler in der Sphäre erst später bekannt wird [LeRo99].

3.5 Mobile Transaktionen

Unter Mobilen Transaktionen werden Transaktionen für mobile Informationssysteme verstanden. Mobile Informationssysteme unterscheiden sich von statischen durch den Einsatz mobiler Geräte. Dabei wird eine mobilfunkbasierte Systemarchitektur vorausgesetzt, welche dabei folgende Ebenen unterschiedet [HTK05]:

- **Stationäre Rechner:** Stationäre Rechner sind ständig an das Festnetz angebunden.
- **Basisstationen:** Es handelt sich um dedizierte stationäre Rechner über die mobile Rechner drahtlos mit dem Festnetz verbunden werden können.
- **Mobile Rechner:** Sind nicht ständig mit dem Festnetz verbunden. Sie können jederzeit entkoppelt sein und sich jederzeit drahtlos mit dem Festnetz verbinden. Ansonsten gelten die in 2.2.1 definierten Eigenschaften.

Es wird auch angenommen, dass stationäre sowie mobile Rechner entsprechende stationäre und mobile Datenbanken unterhalten. Mobile Transaktionen sichern dabei die Operationen auf diesen Datenbanken ab [HTK05]. Dabei basieren mobile Transaktionsmodelle in der Regel auf offen geschachtelten Transaktionen, wie z.B. die *Kangaroo-Transaktion* [DHB97]. Ihr Fokus liegt dabei auf der Unterstützung einer transparenten Migration von einer Mobilfunkzelle zur Nächsten, der Fortsetzung der Transaktionsausführung nach einer temporären Entkopplung der mobiler Rechner sowie der Verteilung von Arbeitsschritten zwischen dem mobilen Rechner und der Basisstation [HTK05].

3.6 Anforderungsdefinition

In dieser Arbeit wird für verteilt ausgeführte Mobile Prozesse (vgl. 2.4.4) eine Transaktionsunterstützung erarbeitet. Dazu werden in diesem Abschnitt Systemanforderungen festgelegt, die ein zu entwickelndes Transaktionssystem für Mobile Prozesse erfüllen sollten. Als Erstes werden Systemanforderungen beschrieben, die sich aus den Besonderheiten mobiler Umgebung ergeben. Danach werden Anforderungen definiert, welche für die Einhaltung von speziellen Eigenschaften Mobiler Prozesse notwendig sind. Anschließend werden Anforderungen an die Transaktionen und Transaktionsmodelle aufgestellt.

3.6.1 Anforderungen aus den Eigenschaften Mobiler Systeme

Mobile Prozesse werden primär auf Mobilen Systemen eingesetzt. Deswegen lassen sich, aus den im Abschnitt 2.1.1 beschriebenen Eigenschaften der Mobilen Systeme, folgende Anforderungen ableiten:

M1 Toleranz der Knotenausfälle: Mobile Prozesse werden in einer Mobilen Umgebung ausgeführt. Dabei können die beteiligten Geräte für eine längere Zeit ausfallen oder unerreichbar bleiben. Das betrifft alle beteiligten Geräte, jene, die Prozesse ausführen und alle anderen, die z.B. nur Dienste anbieten oder Prozessinstanzen nur weiterleiten. Das zu entwickelnde Transaktionssystem sollte jedoch solche Knotenausfälle nach Möglichkeit tolerieren können.

M2 Geringer Kommunikationsaufwand: Drahtlose Netze haben in der Regel schmale Bandbreiten, außerdem ist ihre Nutzung im Vergleich zu Festnetzverbindungen teuerer. Deswegen sollte ein möglicher Kommunikationsoverhead bei der Ausführung von Transaktionen möglichst klein gehalten werden.

Außerdem gelten auch für eine Transaktionsunterstützung, die bereits an Mobile Prozesse formuliert wurden, Anforderungen bezüglich eines schonenden Umgangs mit Ressourcen mobiler Geräte. Dazu zählen Speicher, Rechenleistung und Energiequellen usw. (vgl. [Zap05]).

3.6.2 Anforderungen aus Mobilen Prozessen

Auch die spezifischen Eigenschaften von Mobilen Prozessen müssen bei der Transaktionsunterstützung berücksichtigt werden. Daraus resultieren für eine Transaktionsunterstützung folgende Anforderungen:

P1 Abstraktion von der Netzwerkstruktur: Die Transaktionsunterstützung soll von keiner bestimmten Netzwerktopologie bzw. Netzwerkarchitektur oder gar Verbindungstechnologie abhängig sein. Die, während der Ausführung Mobiler Prozesse gebildeten Netzwerke, haben dabei eher einen Ad-hoc-Charakter und können mit verschiedenen Kommunikationstechnologien realisiert werden, was i.d.R. dynamisch und situationsabhängig geschieht. Die Prozessausführungsschicht abstrahiert weitgehend von technologiespezifischem Wissen, so dass eine Transaktionsunterstützung für ein solches System dieses dementsprechend auch können sollte [Kunz05a].

In der Literatur, die sich mit Transaktionen im mobilen Umfeld beschäftigt, wird jedoch meistens von anderen Prämissen ausgegangen. Es handelt sich i.d.R. um mobilfunkadäquate Systemarchitektur, welche nicht nur mobile Geräte sondern auch einen vergleichbar zuverlässiges Backbone-Festnetz einbezieht. Mobile Geräte in verschiedenen Funkzellen werden teilweise auch über das Backbone-Netz verbunden. Von derartigen Annahmen wird in dieser Arbeit abstrahiert [ImBa94, DuKu99].

P2 Unterstützung von Migration: Mobile Prozesse können während der Ausführung ihren Ausführungsort wechseln, so dass die Kontrolle über die Prozessausführung von einem Gerät auf ein anderes übergeht. Dieses führt dazu, dass Migrationen innerhalb von Transaktionen stattfinden können, was zu einer erschwerten Durchführung der Transaktionssicherung führen kann. Das zu entwickelnde Transaktionssystem sollte jedoch die Migrationseigenschaften der Mobilen Prozesse nach Möglichkeit unterstützen und diese nicht außer Kraft setzen [Kun05a].

P3 Effiziente Unterstützung langer Transaktionen: Mobile Prozesse können unter Umständen lange andauern, besonders dann, wenn Menschen in die Abarbeitung integriert sind oder öfters Verbindungsabbrüche vorhanden sind. Auch unter diesen Bedingungen sollte die transaktionale Ausführung gewährleistet sein [Kun05a].

3.6.3 Allgemeine Anforderungen an Transaktionszusicherungen

Zur Aufstellung von allgemeinen Anforderungen an die Transaktionsunterstützung können einige Erkenntnisse aus Workflows-Systemen übernommen werden, denn verteilt ausgeführte Mobile Prozesse sind mit Workflows verwandt (vgl. 2.4.4). Es wurden bereits einige Transaktionskonzepte für Workflows entwickelt und erprobt, obwohl auch dort noch kein einheitlicher Anforderungskatalog existiert – nicht zuletzt, weil es sehr viele verschiedene Workflow-Arten in verschiedenen Entwicklungsstadien gibt. Dennoch können allgemeine Erkenntnisse durchaus angewendet werden.

T1 Gewährleistung von Atomarität: Atomarität sollte unterstützt werden. Dabei bedeutet (klassische) Atomarität, dass ein atomares Arbeitspaket entweder vollständig oder gar nicht ausgeführt wird. Die kleinste Granularität der Atomarität besitzen die Aktivitäten – sie werden in dieser Arbeit immer als atomar angesehen.

Da Transaktionen in Mobilen Prozessen im allgemeinen Fall sehr lange dauern können, kann nicht gefordert werden, dass Dienste, welche hinter den Aktivitäten stehen, ihre Ergebnisse bis zum Transaktionsende nicht freigeben sollen. Auf der Prozessebene sollte daher zumindest, ermöglicht durch Kompensationsverfahren, semantische Atomarität gewährleistet werden.

Nur in speziellen Anwendungsfällen mit sehr kurzen Durchlaufzeiten könnte die Durchsetzung der Nicht-Externalisierung der Ergebnisse von Nutzen sein und sollte nach Möglichkeit berücksichtigt werden.

T2 Sicherstellung der Konsistenz: Da Mobile Prozesse auf Diensten basieren, sollten die Dienstaufrufe auch in Fehlerfällen semantisch konsistent bleiben. Ein Aspekt der Konsistenz ist die Kontrolle der Korrektheit der Ausführungsfolgen der Aktivitäten. Weiterhin wird angenommen, dass im Falle einer erfolgreichen Durchführung einer Aktivität, auch ein konsistenter Zustand hinterlassen wird. Wenn jedoch die Kompensationsaktivitäten im Ablauf der Recovery Prozesse ausgeführt werden müssen, sollte eine Transaktionsunterstützung nach Möglichkeit eine erfolgreiche Durchführung dieser Kompensationsaktivitäten durchsetzen können.

T3 Unterstützung verschiedener Dienstarten: In verteilt ausgeführten Mobilen Prozessen stehen hinter abstrakten Aktivitäten konkrete Dienste. Dabei werden drei Arten von Aktionen einer Transaktion (vgl. 3.1.2), die auch entsprechend auf Dienste übertragbar sind, unterschieden. Damit werden *ungeschützte*, *geschützte* und *reale Dienste* unterschieden. So können die bei der Transaktionsdurchführung möglicherweise benötigten Meta-Informationen über die Dienste durch die Bestimmung der Dienstart gewonnen werden.

Aus Sicht eines Dienstbenutzers ist ein Dienst durch seine Schnittstelle beschrieben (vgl. 2.5). Da bei Mobilen Prozessen keine Einschränkungen in der Benutzung von Dienstklassen definiert sind, muss die Information über die Dienstart aus ihrer Schnittstelle gewonnen werden können.

Eine Dienstschnittstelle beschreibt Geschäftsoperationen, die den so genannten *vertikalen Protokollen* folgen und damit funktionale Aspekte modellieren. Orthogonal dazu gibt es *horizontale Protokolle,* welche die nichtfunktionalen, technischen Aspekte der Dienstbenutzung – wie Zuverlässigkeit, Sicherheit und transaktionaler Schutz – ermöglichen. Horizon-

tale Protokolle können auf verschiedene Dienste, unabhängig von ihren Aufgaben, angewendet werden. Weiterhin sind beide Protokollarten in der Schnittstelle eines Dienstes sichtbar [ACKM04].

Das zu entwickelnde Transaktionskonzept soll nach Möglichkeit alle Dienstarten unterstützen können. Dabei soll die Komplexität der Dienste möglichst gering gehalten werden, bzw. unverändert bleiben.

T4 Mächtigkeit der Transaktionsdefinitionen: Die Definition von Transaktionen sollte möglichst flexibel und mächtig sein. Als flexibelste Variante ist die Unterstützung von *flexiblen Transaktionsbereichen* wünschenswert.

Im Folgenden auch als *flexible Transaktion* bezeichnet, handelt es sich dabei um eine, vom Prozessfluss unabhängige, Kennzeichnung der Transaktionsgrenzen. Jede Aktivität eines Prozesses, kann in eine flexible Transaktion aufgenommen werden. Dabei sind keine Regeln zur Struktur der Transaktionsgrenzen einer flexiblen Transaktion definiert, das heißt, Transaktionsgrenzen sind in keiner Weise von der Prozessstruktur abhängig. Dieser Umstand kann insbesondere dazu führen, dass eine solche Transaktion, Aktivitäten einschließt, welche im Prozessablauf nicht benachbart sind. Dies ist in der Abbildung 15 zu sehen. Die flexible Transaktion *T1* umspannt in diesem Beispiel nicht benachbarte Aktivitäten *A, B, C.*

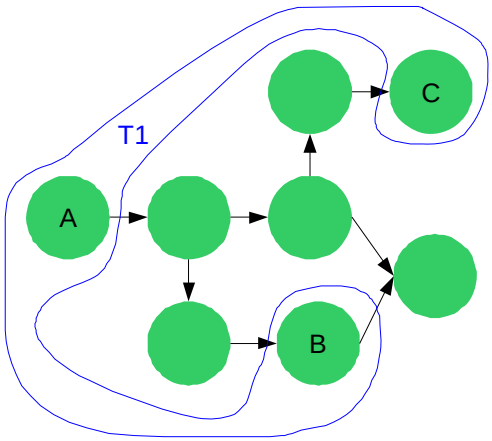

Abbildung 15: Flexible Transaktionsbereiche

Wenn jedoch die Abbildung von flexiblen Transaktionsbreichen nicht möglich ist, sollte eine Transaktionsunterstützung in ihrer Flexibilität möglichst nah an die flexiblen Transaktionsbereiche kommen. Eine Transaktionsdemarkation sollte jedoch mindestens die Kontrollflussstrukturen Sequenz, Split sowie Join und, wenn möglich, deren Kombinationen umfassen.

Aspekte der Isolation und Dauerhaftigkeit

Isolation ist eine wichtige Eigenschaft moderner Datenbanksysteme (*DBS*), wird jedoch von den meisten gegenwärtigen WfMS nicht unterstützt. Dieses liegt an den unterschiedlichen Scheduling-Mechanismen (*Concurrency Control*) der beiden Systeme. In Datenbanksystemen wird die Isolation durch einen so genannten *Scheduler* gewährleistet. Dabei bestehen Transaktionen aus atomaren Lese-/Schreiboperationen, die vom Scheduler protokollkonform ausgeführt werden. In Workflow Systemen muss die Isolationssicherung nicht nur über Lese-/Schreiboperationen, sondern auch über externe Programmaufrufe gewährleistet werden. Doch gerade externe Systeme sind dem WfMS i.d.R. nicht sichtbar und behalten so ihre Autonomie. Daher externalisieren beteiligten Ressourcen ihre Ergebnisse sobald sie ihre Funktion abgeschlossen haben und weichen damit die Isolation auf. Daher können bei einer Isolationskontrolle in Workflows meist nicht alle Aktivitäten kontrolliert werden, so dass man die Forderung nach einer globalen Isolation bei solchen Systemen in der Regel aus Machbarkeitsgründen fallen lässt [ElDu98].

Vergleichbare Verhältnisse sind auch bei verteilt ausgeführten Mobilen Prozessen anzutreffen. Mobile Prozesse greifen während ihrer Ausführung auf Dienste zu, dabei bleiben die Dienste in der Art und Weise der Diensterbringung autonom. Daher ist es für die Prozessverwaltung nicht ohne weiteres möglich die globale Isolation der Transaktionen auf Prozessebene zu gewährleisten. Die Forderung nach globaler Isolation wird daher in dieser Arbeit von Anfang an nicht gestellt.

Dabei kann eine lokale Isolation durchaus gegeben sein. Dienste können ihre Aufrufe so verwalten, dass diese während der Abarbeitung voneinander isoliert bleiben. Lokale Isolation ist jedoch nicht das Thema dieser Arbeit, da sie im Ermessen von konkreten Dienstbetreibern geschehen wird und sich dabei auf ein breites Feld von Erkenntnissen aus der Datenbanktheorie stützten kann.

Dauerhaftigkeit von Transaktionen sollte vorausgesetzt werden. Die Ergebnisse einer erfolgreich abgelaufenen Aktivität sollten dauerhaft bleiben. Allerdings geht es dabei um einen internen Zustand der einzelnen Dienste, welcher wie auch die lokale Isolation im Ermessen des Betreibers liegt. Dabei können keine allgemeinen Forderungen an die physische Verfügbarkeit der Daten aufgestellt werden, da Mobile Prozesse in sehr heterogenen Hardwareumgebungen ablaufen können. Außerdem können diese Forderungen zwischen verschiedenen Diensttypen variieren. Daher werden auch bezüglich der Dauerhaftigkeit in dieser Arbeit keine Anforderungen gestellt.

3.7 Bewertung der Transaktionskonzepte

In vorherigen Abschnitten wurden verschiedene Transaktionskonzepte und Modelle vorgestellt. An dieser Stelle sollen diese bewertet und anhand der zuvor formulierten Anforderungen auf ihre Eignung für den Einsatz in mobilen Systemen überprüft werden.

Wie bereits erwähnt, basieren klassische Transaktionen auf Sperrverfahren. Bei lang andauernden Transaktionen sind die sperrenbasierten Isolationsverfahren jedoch nicht effizient, da Ressourcen während des Transaktionsablaufs gesperrt bleiben. Ausgefallene Netzwerkknoten blockieren Commit-Protokolle, so dass Sperren noch länger gehalten werden. Außerdem erfordern Commit-Protokolle die Kommunikation zwischen allen Teilnehmern der Transaktion, so dass dadurch ein nicht zu unterschätzendes Nachrichtenaufkommen entsteht. Je länger die Sperren in einer Transaktion gehalten werden, desto problematischer ist ihr Einsatz in einer verteilten mobilen Umgebung. Aus diesen Gründen erfüllen die klassischen Transaktion und ihre Erweiterungen weder M1 (Toleranz der Knotenausfälle) noch die M2 (geringer Kommunikationsaufwand) Anforderungen. Dazu gehört auch die Atomare Sphäre, weil sie auf klassischen Verteilten Transaktionen basiert.

Die kompensationsbasierte Varianten der offenen geschachtelten Transaktionen wie Sagas, Kompensationssphären sowie auch Mobile Transaktionen werden den Anforderungen M1 und M2 gerecht.

Die Forderung nach der Unabhängigkeit von der Netzwerkstruktur (P1) erfüllt eine Saga am besten, da sie keine Anforderungen an diese stellen. Dagegen schneiden die Mobilen Systeme am schlechtesten ab. Diese Transaktionen erfordern eine in 3.5 beschriebene Architektur. Diese Architekturvoraussetzung ist der Hauptgrund für die Uneignung der Mobilen Transaktionen für die Transaktionssicherung der Mobilen Prozesse. Jedoch erfüllen auch Workflow Transaktionen diese Anforderung nicht vollständig. Die Transaktionsverwaltung in Workflows wird zentral durchgeführt (vgl. 2.4.3), so dass alle nebenläufigen Prozesspfade auf dem gleichen Gerät liegen. In Mobilen Prozessen können jedoch die einzelnen Ausführungspfade des Prozesses auf verschiedenen Geräten gleichzeitig ausgeführt werden. Auch klassische Verteilte Transaktionen werden der Anforderung P1 nur bedingt gerecht. Die klassischen Verteilten Transaktionen können unabhängig von ihren Ausführungsumgebungen und Netzwerkstrukturen eingesetzt werden, jedoch erfordert der Einsatz der Commit-Protokolle zumindest einen relativ zuverlässigen Koordinator (vgl. 3.2.2).

Außer der Mobilen Transaktion unterstützen keine der betrachteten Transaktionskonzepte die Migration (Anforderung P2). Auch bei Mobilen Transaktionen handelt es sich nur um eine eingeschränkte Migration, da, im Zuge der Bewegung der beteiligten mobilen Geräte

zwischen den Funkzellen, die Transaktionsverwaltung nur von einer Basisstation zu einer anderen migrieren kann.

Die Atomarität (T1) wird von allen Transaktionskonzepten unterstützt. Bei der Unterstützung der Konsistenz (T2) liegen die Unterschiede wieder zwischen den kompensationsbasierten und traditionellen Transaktionen. Weitgehende Unterstützung dieser Eigenschaften wird von klassischen Transaktionen und der Atomaren Sphäre gewährleistet. Bei Kompensationsbasierten Verfahren kann eine semantische Atomarität nur durch eine erfolgreiche Durchführung von Kompensationshandlugen erfüllt werden. Dies kann in der Regel jedoch selbst von WfMS nicht garantiert werden, somit muss davon ausgegangen werden, dass Kompensationsbasierte Modelle nur eingeschränkt die Konsistenzeigenschaft (T2) erfüllen können.

Während kompensationsbasierte Transaktionsmodelle bei geschützten, sowie nicht geschützten Aktionen und Diensten zum Einsatz kommen können, können traditionelle Transaktionen nur geschützte Aktionen einbeziehen. Deswegen wird die Anforderung T3 durch klassische Transaktionen und Atomare Sphäre nicht unterstützt.

Die Anforderung nach möglichst flexiblen Möglichkeiten zur Transaktionsdefinition (T4) ist nur auf die Workflow Transaktionen anwendbar. Dabei erfüllt eine Kompensationssphäre diese Anforderung ganz und die Atomare Sphäre nur teilweise, da in der Atomaren Sphäre Einschränkungen an die Strukturen definiert sind (vgl. 3.4.1).

Tabelle 3: Ergebnisse der Analyse von Transaktionsmodellen

	M1	M2	P1	P2	P3	T1	T2	T3	T4
Klassische (mit Erweiterungen)	-	-	+/-	-	-	+	+	-	n.a.
Atomare Sphären	-	-	+/-	-	-	+	+	-	+/-
Sagas	+	+	+	-	+	+	+/-	+	n.a.
Kompensationssphären	+	+	+/-	-	+	+	+/-	+	+
Mobile Transaktionen	+	+	-	+/-	+	+	+/-	+	n.a.

\+ Wird unterstützt +/- Teilweise Unterstützung

\- Wird nicht unterstützt n.a. Nicht anwendbar

In der Tabelle 3 sind die Analyseergebnisse zusammenfassend dargestellt. Die Untersuchung hat gezeigt, dass keine der betrachteten Transaktionsmodelle für die Transaktionsunterstützung verteilt ausgeführter Mobiler Prozesse geeignet ist. In erster Linie liegt es an der fehlenden Unterstützung der Migrationseigenschaft. Mobile Transaktionen sehen zwar einige Migrationsaspekte der Transaktionsverwatung vor, scheiden

jedoch wegen ihrer Architekturvoraussetzungen (vgl. 3.5) von der weiteren Betrachtung aus.

Auch klassische Transaktionsmodelle sowie die Atomaren Sphären können nicht als Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse betrachtet werden, da sie die Anforderungen M1 (Toleranz der Knotenausfälle) und M2 (Toleranz der Knotenausfälle) nicht erfüllen und auch bei lang andauernden Transaktionen nicht eingesetzt werden können.

Auch Sagas und Kompensationssphären können die verteilt ausgeführte Mobile Prozesse transaktional nicht sichern, da sie vor allen die Migrationseigenschaft nicht erfüllen. Allerdings können diese Modelle als Basis für die Entwicklung von migrationsfähigen Transaktionsmodellen genutzt werden.

4 Transaktionsunterstützung

In diesem Kapitel werden mögliche Systemarchitekturlösungen für eine effiziente und flexible Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse aufgezeigt und bewertet. Zunächst werden die grundsätzlichen Architektureigenschaften einer Transaktionsunterstützung für verteilte Mobile Prozesse erläutert. Danach werden anhand dieser Prinzipien die entwickelten Transaktionskonzepte vorgestellt. Abschließend werden die vorgestellten Konzepte anhand der in 3.6 formulierten Kriterien bewertet.

4.1 Komplexität

Im Abschnitt 3.6 wurden die Anforderungen an die zu entwickelnde Transaktionsunterstützung ausarbeitet. Angestrebt wird dabei eine größtmögliche Flexibilität des Systems. Insbesondere ist eine Unterstützung von allen Dienstarten und Prozessgraphstrukturen wünschenswert. Die Eigenschaften des Mobile Computings sowie der verteilt ausgeführten Mobilen Prozessen werden jedoch von, den in Kapitel 3 betrachteten, Transaktionsmodellen nicht berücksichtigt. Dabei erhöhen gerade diese, im Vergleich zu stationären Systemen, die Komplexität einer Transaktionsunterstützung. Zusammenfassend handelt es sich um folgende Umstände der Komplexitätserhöhung:

- **Mobile Umgebung:** Die Mobile Umgebung erfordert zusätzliche Maßnahmen zur Behebung der verschiedenen Netzwerkfehlern. Dazu sind Vorkehrungen bzw. Modellierungs- und Entwicklungsentscheidungen nötig, die die relativ leistungsarme Hardware mobiler Geräte berücksichtigen.
- **Migrationsfähigkeit Mobiler Prozesse:** Die Migration führt dazu, dass die in einem Prozess modellierten Transaktionen über mehrere Geräte verteilt sein können. In Bezug auf den Kontrollfluss, kann sowohl sequenzielle als auch parallele Verteilung der Transaktion auftreten. Wenn also ein Prozess Verzweigungen in seinem Verlaufsgraph aufweist, die innerhalb einer Transaktion liegen, können während des Transaktionsablaufs mehrere parallele Ausführungspfade entstehen. Dieses liegt daran, dass nach einem Split auch die Migration der Pfade voneinander unabhängig stattfinden kann. Dieser Umstand führt zu einer komplexen Situation, weil die einzelnen parallelen Ausführungspfade keine Informationen übereinander haben (weder über den aktuellen Status noch den aktuellen Ausführungsort).
- **Abstrakte Beschreibung der Dienste:** Eine weitere Komplexitätssteigerung erfährt das System durch die abstrakte Beschreibung der Dienste. Aktivitäten eines Prozesses werden durch Dienste realisiert, die jedoch bei der Prozessmodellierung nur abstrakt als Dienstklasse beschrieben werden. Erst zur Ausführungszeit werden die konkreten Dienstinstanzen ermittelt. Dabei ist es die Aufgabe einer Transaktion

bestimmte Zusicherungen für die Zustandsübergänge des Systems zu erfüllen i.d.R. geht es dabei um die Konsistenzerhaltung. Hierbei wird der Systemzustand über die konkreten Dienstinstanzen bzw. deren Status definiert. Das kann dazu führen, dass die konkreten Dienstinstanzen während der gesamten Transaktionsausführung bekannt sein müssen damit die Transaktionsunterstützung bei Bedarf darauf zurückgreifen kann.

Allerdings kommen die genannten Faktoren nicht immer gleichzeitig vor. In einigen Fällen, z.B. bei einfachen Prozessesverläufen, kann die Komplexität weit geringer sein, da man von einigen Komplexitätsfaktoren abstrahieren kann. Dieser Umstand bietet eine gute Basis für eine systematische Lösungsentwicklung, wobei die Komplexität schrittweise angegangen werden kann.

4.2 Vergleich von Transaktionsansätzen

Um der Anforderung T4 (Mächtigkeit der Transaktionsdefinition) gerecht zu werden, sollte die Transaktionsdefinition möglichst frei geschehen, so dass im Idealfall sogar die flexiblen Transaktionsbereiche unterstützt werden (vgl. 3.6.3). Da verteilt ausgeführte Mobile Prozesse während ihrer Ausführung zwischen Geräten migrieren können, bedeutet dies, dass eine Transaktion in der Regel mehrere (mobile) Geräte umspannt. Folglich handelt es sich dabei um eine Art der verteilten Transaktionsausführung.

In Kapitel 4 wurden verteilte Transaktionsmodelle analysiert, es hat sich dabei herausgestellt, dass kompensationsbasierte Modelle durch ihre optimistische Vorgehensweise für eine mobile Umgebung besser geeignet sind als klassische Modelle. In erster Linie liegt es daran, dass klassische Transaktionsmodelle auf der Verwendung von Sperren basieren, was bei der Ausführung von klassischen Transaktionen dazu führt, dass alle beteiligten Geräte bzw. Dienste ihre Ressourcen bis zur einen Entscheidung über den Transaktionsausgang blockieren müssen.

Bei einem kompensationsbasierten Ansatz werden keine Sperren verwendet. Die Ausführung in der Transaktion unterscheidet sich in einfachen Fällen kaum von einer gewöhnlichen Prozessausführung. Erst beim Auftreten von Fehlersituationen müssen Kompensationsmaßnahmen getroffen werden. Bereits diese Gegenüberstellung der beiden Paradigmen zeigt, dass klassische Transaktionsmodelle für eine allgemeine Verwendung in verteilt ausgeführten mobilen Prozessmodellen weniger geeignet sind.

Ein weiterer wichtiger Aspekt ist, dass Teilnehmer einer Transaktion die gleiche Entscheidung über den Transaktionsausgang treffen müssen. Zur Propagierung einer Entscheidung an alle Teilnehmer wird in Verteilten Transaktionen ein Commit-Protokoll

verwendet, das jedoch einen Koordinator benötigt (vgl. 3.1.3). Die Verwendung eines Commit-Protokolls setzt eine Kommunikation nach den Regeln eines Commit-Protokolls voraus. Dieses bedeutet, dass bei jeder Transaktion viele Nachrichten verschickt werden müssen. Dazu sollten die beteiligten Knoten vom Koordinator möglichst durchgängig erreichbar sein, um die Rate der gescheiterten Transaktionen gering zu halten. Auch der Koordinator sollte sehr zuverlässig und dabei möglichst immer erreichbar sein, da während des Ausfalls eines Koordinators die Teilnehmer blockiert bleiben. Dies ist bei allen Commit-Protokollen der Fall. In der Ablaufumgebung Mobiler Prozesse kann man jedoch solche durchgehend zuverlässige und erreichbare Knoten nicht voraussetzen [BeNe97].

Hingegen sind kompensationsbasierte Transaktionen in einfachen Fällen ohne nebenläufige Pfade auf keine Koordination angewiesen. Dies beruht auf den, der Kompensation zu Grunde liegenden, offen geschachtelten Transaktionen. In offen geschachtelten Transaktionen führen beteiligte Subtransaktionen sofort nachdem sie fertig sind ein COMMIT aus. Die Subtransaktionsroutinen warten also keine Entscheidung über den globalen Transaktionsausgang ab. Im Falle des Scheiterns, wird die globale Atomarität durch kompensierende Transaktionen hergestellt. Dies wird in 5.4.1 an einem konkreten Modell erläutert. In komplizierten Fällen, welche die Ausführung von nebenläufigen Pfaden in einer Transaktion zulassen (vgl. 5.3), ist man auf einen Synchronisationsmechanismus angewiesen. Die Entwicklung eines effizienten Synchronisationsmechanismus ist dabei eine zentrale Aufgabe.

Im Allgemeinen ist damit ein kompensationsbasierter Ansatz für die Unterstützung von verteilten Mobilen Prozessen besser geeignet, während der Einsatz von klassischen Konzepten eher als zweitrangig angesehen wird. Im nächsten Abschnitt werden verschiedene Transaktionskonzepte unter den Aspekten der Parallelität betrachtet.

4.3 Unterstützung der Parallelität

Nebenläufige Ausführungspfade in einem Mobilen Prozess können durch Split entstehen. Wenn dabei mindestens einer der Prozesspfade auf ein anderes Gerät migriert, entsteht sogar echte Parallelität – Prozessausführungspfade laufen auf verschieden physischen Knoten ab. Dabei kann bei Transaktionen in Mobilen Prozessen die Nebenläufigkeit relativ effizient auf drei Art und Weisen unterstützt werden:

1. Durch ein Migrationsverbot
2. Durch den Einsatz eines Koordinators bei klassischen Transaktionen
3. Durch Verwendung eines Synchronisationsmechanismus bei kompensationsbasierten Transaktionen

Jedoch ist die Einschränkung der Migrationsfähigkeit Mobiler Prozesse im allgemeinen Fall nicht zu empfehlen, da die Vorteile eines solchen Transakionsschutzes in der Regel durch die Nachteile eines Migrationsverbots überschattet werden. Das Aussetzen der Migration resultiert in einer erhöhten Wahrscheinlichkeit, dass die benötigten Dienste nicht gefunden werden und die Prozessausführung öfters abgebrochen wird. Migration wird unter anderem für eine bessere Dienstfindung eingesetzt, wenn ein Gerät den benötigten Dienst nicht finden kann, wird die Prozessausführung zu einem anderem Gerät weitergeleitet, welcher dann den Prozess weiter ausführen kann. Auch wenn der Kontext eines Mobilen Geräts sich so ändert, dass eine weitere Ausführung des Prozesses an diesem Gerät nicht mehr möglich ist, kann auf ein anderes Gerät migriert werden und so die Prozessausführung erhalten bleiben. Migration ist also eine der definierenden Eigenschaften von Mobilen Prozessen.

Auch der Einsatz von klassischen Transaktionen sollte in Mobilen Prozessen eher eine Ausnahme darstellen (vgl. 4.2). Dabei eignet sich eine kompensationsbasierte Lösung in einfachen Fällen am besten. Damit hängt die Anwendbarkeit der kompensationsbasierten Ansätze zur Behandlung von Parallelität weitgehend von der Güte des Synchronisationskonzeptes ab.

Ziel der Synchronisation zwischen den beteiligten Pfaden ist das Propagieren der Information über den Transaktionsausgang. Wenn eine Transaktion wegen eines Fehlers in einem der Prozessstränge scheitern muss, müssen alle in der Transaktion definierten Aktivitäten in allen parallel ablaufenden Strängen kompensiert werden. Dabei soll dies nur einmal geschehen.

Eine Herausforderung entsteht dadurch, dass einzelne Stränge im Normalfall migrationsbedingt über den physischen Ort und Status möglicher anderer Stränge nichts wissen, da einzelne Stränge unabhängig voneinander migrieren. Ein möglicher Lösungsvorschlag wäre, alle beteiligten Prozessstränge einer Transaktion bei jedem Migrationsschritt zu informieren. Dieser Ansatz skaliert jedoch schlecht, denn je mehr Prozessstränge beteiligt sind, desto mehr Nachrichten müssen ausgetauscht werden. Außerdem müssen dabei alle Geräte direkt oder indirekt untereinander erreichbar sein, was in einer mobilen Umgebung nicht garantiert werden kann.

Eine andere Möglichkeit besteht in der Verwendung eines gemeinsamen Synchronisationspunktes, der allen Beteiligten bekannt sein sollte. Dazu sollte dieser Synchronisationspunkt nach Möglichkeit für alle Transaktionsteilnehmer erreichbar sein. Wenn diese Vorraussetzung gegeben ist, skaliert dieser Ansatz deutlich besser. In 5.4.3 und 5.4.4 werden dazu konkrete Modelle vorgeschlagen.

4.4 Transaktionsmodelle für Mobile Prozesse

In diesem Abschnitt wird die Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse systematisch entwickelt. Als Erstes wird die Transaktionsunterstützung für einen sequenziellen Prozessverlauf dargestellt. Dabei werden zunächst ein Grundmodell und anschließend seine möglichen Erweiterungen behandelt. Danach werden Transaktionen über die Kontrollflussstruktur Join behandelt. Es wird dabei die *Join Transaktion* vorgestellt. Abschließend wird dann die Transaktionsunterstützung für DPDL-Prozesse mit *Flexiblen Synchronsierten Transaktionen* erläutert.

4.4.1 Transaktionsunterstützung für sequenzielle Abläufe

Eine relativ einfache Variante einer Transaktion im verteilten Prozess ist eine Transaktion über einen sequenziellen Prozessverlauf. Da in diesem Fall einzelne Aktivitäten nacheinander ausgeführt werden und keine Nebenläufigkeit besteht, müssen auch keine synchronisationsspezifischen Überlegungen getroffen werden, was die Komplexität diesbezüglich verringert. Weiterhin kann sogar von den Migrationsaspekten bzw. von der Hardwareebene abstrahiert werden, da die gesamte Transaktionsverwaltung, im sequenziellen Fall, zu jedem Zeitpunkt immer nur einmal vorhanden ist.

Das im Folgenden beschriebene Transaktionsmodell wird als *Sequenzielle Transaktion* bezeichnet. Zunächst wird dazu ein Beispiel aufgezeigt, das mit einer Sequenziellen Transaktion atomar ausgeführt werden kann. Anschließend wird das Modell der Sequenziellen Transaktion genau erläutert.

Angenommen, die Aktivitäten *A*, *B* und *C* sind ein sequenzieller Abschnitt eines Prozesses. Dabei könnte *A*, das Aushändigen eines Theatertickets bedeuten, *B* das Abbuchen des Ticketpreises vom Konto des Käufers und *C* die Aufnahme eines Eintrag in die Statistikdatenbank, wo z.B. die Kontonummer und der Ticketpreis des aktuellen Kaufs notiert werden. Dabei werden Aktivitäten *A, B, C* sequenziell, nacheinander ausgeführt. Es ist sinnvoll diese drei Aktivitäten, welche im Grunde genommen eine Kaufaktion abbilden, in einer Transaktion ablaufen zu lassen. Dabei werden entweder alle drei Aktivitäten erfolgreich durchlaufen oder, beim Scheitern, alle Inkonsistenzen semantisch kompensiert. Dies lässt sich gut mit einer Sequenziellen Transaktion abbilden.

Allgemein gesehen, geht dieses Modell zunächst von einem sequenziellen Abschnitt eines Prozesses aus, für den es definiert wird. Vorerst wird davon ausgegangen, dass eine Sequenzielle Transaktion zusammenhängend sein muss, und es innerhalb der Transaktion keine zusätzlichen Transitionen gibt, die in bzw. aus der Transaktion führen (außer der Start- und Endaktivitäten).

Das Transaktionsmodell an sich entspricht dabei in weiten Teilen dem Konzept der Kompensationssphäre und dem Saga-Prinzip (vgl. Abschnitte 3.3.2 und 5.4.2). Abbildung 16 stellt dieses schematisch dar. Eine Sequenzielle Transaktion *ST1* wird über einen sequenziellen Abschnitt des Prozesses definiert. Jede Aktivität in der Transaktion bildet ihre eigene (Sub-)Transaktion *T1* bis *Tn*. Dabei können geschützte sowie nicht geschützte Aktionen einbezogen werden. Der Typ des Dienstes spielt in diesem Fall keine Rolle, denn die Subtransaktionen werden, dem Saga Prinzip folgend, sofort festschreiben.

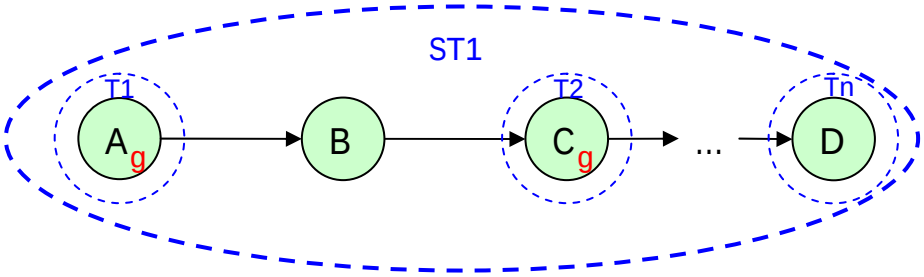

Abbildung 16: Saga über eine Sequenz

Wenn die Prozessausführung die Sequenzielle Transaktion *ST1* erreicht, wird das Transaktionssystem den Anfang der Transaktion wahrnehmen und einleiten. Dabei muss das Start-Set der Transaktion gesichert werden, welches bei eventuellen Wiederholungen der Transaktion genutzt wird (vgl. Start-Set bei Atomaren Sphären in 3.4.1). Danach kann die erste Aktivität gestartet werden. Dabei werden geschützte Aktivitäten nach einem erfolgreichen Durchlauf sofort einen COMMIT durchführen und Ergebnisdaten der Aktivität für weitere Aktivitäten im Prozess freigeben. Nicht geschützte Aktivitäten besitzen keine Transaktionssemantik und werden ganz normal ausgeführt, das System notiert dabei ihren Ausführungsstatus. Wenn schließlich *Tn* erfolgreich festschreibt, wird auch die gesamte Transaktion als erfolgreich betrachtet.

Eine Sequenzielle Transaktion kann auch früher erfolgreich abschließen, nämlich dann, wenn eine der Transitionsbedingungen innerhalb der Transaktion zu „false“ ausgewertet wurde, so dass die Prozessausführung vorzeitig beendet wird. In der Abbildung 16 dargestellte Transaktion ist es z.B. dann der Fall, wenn eine Transitionsbedingung zwischen den Aktivitäten *B* und *C* zu „false“ evaluiert. Dies stellt einen gültigen Prozessverlauf dar und damit einen gültigen Transaktionsausgang.

Allerdings können während der Ausführung auch Fehlersituationen auftreten, welche behandelt werden müssen. Dabei können zwei Hauptfehlerursachen unterschieden werden. Der erste Fehlertyp ist dann gegeben, wenn der Prozess innerhalb einer Sequenziellen Transaktion nicht mehr weiter kommen kann, weil z.B. kein passender Dienst erreicht werden kann, die Migration gerade nicht möglich ist oder eine

Überschreitung einer zeitlichen Restriktion zur Ausführung einer Aktivität vorliegt und demzufolge die Prozessausführungsumgebung den Abbruch des ganzen Prozesses ankündigt. Beim zweiten Fehlertyp handelt es sich um einen logischen Fehler während oder nach der Ausführung eines Dienstes. In einem kompensationsbasierten Modell ist die Früherkennung eines Fehlers von großer Bedeutung. Nachdem der Fehler gefunden wurde, muss er behoben werden. Es ist wünschenswert, dass bei der Ausführung lang andauernder Prozesse möglichst wenig bereits durchgeführter Arbeit wieder verloren geht. Jedoch kann im ersten Fall Forward Recovery nicht mehr eingesetzt werden, da die Prozessausführungsumgebung bereits keine Möglichkeit zum Fortfahren erkannt hat. In diesem Fall müssen die spezifizierten Backward-Recovery-Maßnahmen gestartet werden. Im Falle eines logischen Fehlers sollte zunächst Forward Recovery betrieben werden, damit die bereits erledigte Arbeit nicht einfach verfällt.

Forward Recovery wurde im Kontext der Sagas bereits eingeführt (vgl. 3.2.2) und bedeutet in Sequenziellen Transaktionen das wiederholte Ausführen gescheiterter Aktivitäten. Die wiederholte Ausführung kann mit derselben oder mit einer anderen Dienstinstanz versucht werden. Bei einem logischen Fehler muss der Dienst eventuell gesucht bzw. es muss zum Dienst hin migriert werden. Kann beispielsweise kein anderer passender Dienst gefunden werden, ist auch der Versuch, den bereits verwendeten und gescheiterten Dienst erneut aufzurufen, sinnvoll, weil das Scheitern durch unzuverlässige Verbindungen, Überlastungen oder durch andere Nebeneffekte nur kurzfristig aufgetreten sein könnte.

Wenn die Forward Recovery nicht erfolgreich war, kann die Backward Recovery gestartet werden um semantisch die durchgeführten Änderungen umkehren (vgl. 3.2.2). In Sequenziellen Transaktion handelt es sich um einen speziellen Kompensationsprozess. Der Kompensationsprozess wird bei der Prozessmodellierung mitgestaltet, dabei kann anlehnend an die Kompensationssphären die Kompensation auf zwei Art und Weisen geschehen. Es wird dabei zwischen einer *diskreten* und *globalen Kompensationsgranularität* unterschieden. Im ersten Fall gibt es zu jeder Geschäftslogik-Aktivität auch eine kompensierende Aktivität; Im zweiten Fall gibt es eine kompensierende Aktivität für die ganze Transaktion [LeRo99].

Wenn die Kompensationsgranularität diskret ist, muss der Kompensationsprozess dynamisch erzeugt werden, dieser ist in der Abbildung 17 dargestellt. Bei einem sequenziellen Verlauf ist die Erzeugungslogik relativ einfach. Es müssen lediglich kompensierende Aktivitäten zu allen, bereits erfolgreich abgeschlossenen, Aktivitäten in umgekehrter Reihenfolge aufgestellt und ausgeführt werden. Dabei wird der Kompensationsprozess wie ein gewöhnlicher verteilt ausgeführter Mobiler Prozess gestartet und in der Umgebung ausgeführt. In der Abbildung 17 ist zu sehen, dass beim

endgültigen Scheitern der Transaktion hinter der Aktivität *C*, die Transaktionsunterstützung einen kompensierenden Prozess von dem aktuellen Gerät *G* aus einleitet.

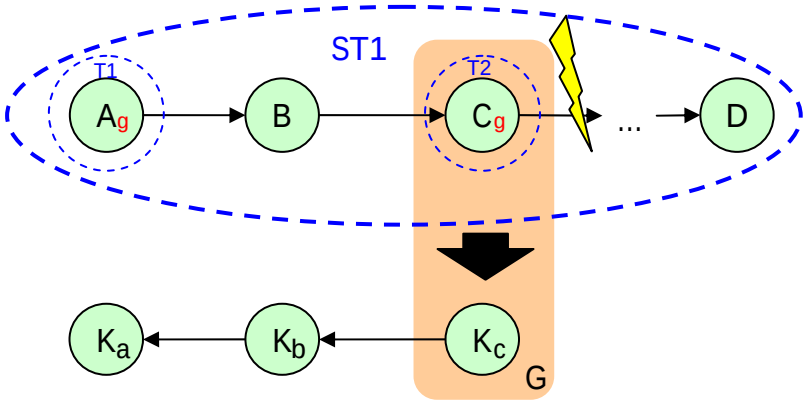

Abbildung 17: Schematische Darstellung der diskreten Kompensation

Der Kompensationsprozess unterscheidet sich nur bei der Dienstwahl von einem gewöhnlichen Prozess. Kompensation muss sinngemäß mit denjenigen konkreten Dienstinstanzen stattfinden, welche bereits bei der Ausführung verwendet wurden. Diese Information muss dem Prozess bekannt und bis zum Prozessende verfügbar sein.

An der diskreten Kompensation können ungeschützte Dienste teilnehmen. Dazu müssen aber Dienste bzw. Dienstanbieter eine Vorraussetzung erfüllen, neben der eigentlichen Geschäftsoperation in der Schnittstelle des Dienstes auch eine zugehörige Kompensationsoperation zu deklarieren. Im diskreten Fall können dann nur solche Dienste verwendet werden, welche diese Voraussetzung erfüllen.

Dennoch können hinter vielen Aktivitäten reale Diensten stehen, zu denen es möglicherweise keine direkten, sinngemäßen Kompensationsaktionshandlungen gibt. Solche Aktivitäten, ohne zugehörige Kompensationsmöglichkeiten, können somit nicht mit einer diskreten Kompensationsgranularität modelliert werden. Es ist aber denkbar, dass eine Kompensationshandlung für eine ganze Transaktion definierbar ist, ohne, dass dieses für die einzelnen Aktivitäten der Transaktion möglich ist.

Beispielhaft soll dies am Prozess „Dokumentherstellung“ gezeigt werden. Dieser Prozess besteht aus einer Sequenz an Tätigkeiten: der Informationssuche, Dokumentvorbereitung, Papiereinlagerung und dem Ausdrucken eines Dokumentes. Wenn im letzten Schritt, beim drucken Fehler entstehen, weil z.B. die Tinte im Drucker nicht für das ganze Dokument ausreicht, ist das Ergebnisdokument fehlerhaft. Dabei ist die diskrete Kompensierung der einzelnen Schritte nicht möglich bzw. nicht sinnvoll. Allerdings kann der ganze Prozess durch den Wegwurf des Ergebnisses kompensiert werden. Diese Kompensierung bedeutet

zwar keine wirkliche Wiederherstellung des vorherigen Zustandes, weil dass Papier trotzdem verbraucht wurde, in Bezug auf die Dokumentherstellung ist sie jedoch akzeptabel.

Eine solche Art und Weise der Kompensation ist dann möglich, wenn die Kompensationsgranularität als global gewählt ist. In diesem Fall besteht der Kompensationsprozess aus nur einer Aktivität, die einen ganzen Subprozess abbilden kann. Eine solche Kompensationsaktivität steht, wie bereits beschrieben, für die ganze sequenzielle Transaktion und muss die Ergebnisse bereits durchgeführter Aktivitäten semantisch umkehren, unabhängig davon nach welcher Aktivität die Transaktion gescheitert ist. Es ist nahe liegend, dass die Formulierung solcher Kompensationsaktivitäten nicht immer möglich ist. Allerdings soll diese Möglichkeit und die Entscheidungsfreiheit dem Benutzer überlassen werden.

Beispielsweise könnte die Kompensation einer solchen Transaktion darin bestehen, dass der Benutzer beim Scheitern der Transaktion lediglich informiert wird, was i.d.R. nahezu immer möglich sein wird. Der Benutzer kann dann z.B. die Kompensation manuell vornehmen. Dabei kann er möglicherweise, auf die mitgelieferten Informationen über die genauen Fehlerstellen, benutzte Dienstinstanzen oder gar Gründe des Scheiterns, zugreifen.

Schließlich, wenn die Kompensation erfolgreich verlaufen ist, kann die Transaktion noch einmal wiederholt werden. Dabei wird die Transaktion mit dem bereits vorhandenen Start-Set erneut gestartet.

4.4.2 Erweiterungen für die Sequenzielle Transaktion

Das in 4.4.1 vorgestellte Grundmodell lässt sich auf verschiedene Weisen erweitern. Im Folgenden werden wichtige Erweiterungsmöglichkeiten vorgestellt. Dabei liegt der Schwerpunk auf der Erweiterung der einfachen Sequenz durch Hinzunahme von komplexeren Kontrollflussstrukturen und zusätzlicher Funktionalität.

Erweiterung S1: Hinzunahme von nicht geschützten Pfaden

Die erste Erweiterung beschreibt, wie zusätzliche, von der Sequenziellen Transaktion nicht geschützten Prozesspfade, mit geschützten Pfaden verbunden werden können. Abbildung 18 stellt dieses schematisch dar. Zu sehen ist, dass Aktivitäten *G*, *E* und *F* sich außerhalb der Transaktion *ST2* befinden. Durch diese Aktivitäten gebildeten Prozesspfade werden als nicht geschützt bezeichnet.

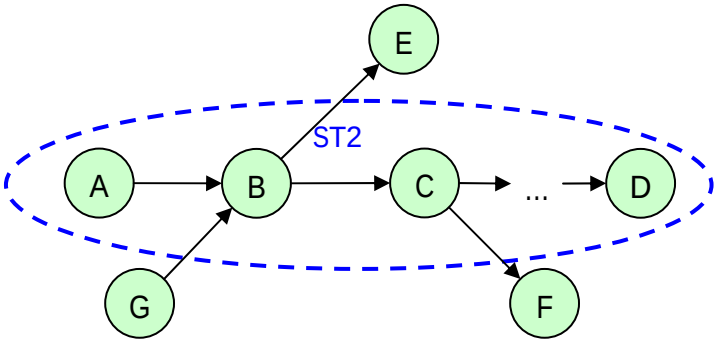

Abbildung 18: Sequenztransaktion mit nicht geschützten Konnektoren

Dabei bilden alle in die Transaktion einströmenden Pfade das Start-Set der Transaktion. Das Start-Set muss für eine eventuelle Wiederholung der Transaktion bewahrt beleiben. Im Falle einer Sequenziellen Transaktion stellt das keinen besonderen Zusatzaufwand dar, da die Aktivitäten in der Transaktion strikt nacheinander ausgeführt werden. Erst, wenn das Start-Set erheblich größer wird, kann dies zu Schwierigkeiten führen, da das Start-Set bei Prozessmigration mit übertragen werden muss. In dem in der Abbildung 18 dargestellten Fall, besteht das Start-Set aus allen Eingangsparametern der Aktivitäten *A* und *G*.

Die ausgehenden nicht geschützten Pfade, in der Abbildung 18 durch Aktivitäten E und *F* gebildet, erfordern jedoch eine gesonderte Betrachtung. Wenn die Sequenzielle Transaktion im ersten Durchlauf beispielsweise hinter der Aktivität *C* scheitert, besteht die Wahrscheinlichkeit, dass *E* und *F* zuvor gestartet wurden. Wenn die gesamte Transaktion wiederholt werden soll, kann es dazu führen, dass *E* und *F* mehr als einmal durchgeführt werden. Selbst, wenn ein wiederholtes Starten von diesen Aktivitäten ausschlossen werden kann, bleibt die Wahrscheinlichkeit, dass bei der Wiederholung andere Daten ausgewertet und im weitern Verlauf verwendet werden. Dies würde dazu führen, dass nicht geschützte Prozesspfade auf nicht mehr aktuellen Daten operieren würden, was einer „Dirty-Read-Anomalie“ gleich steht.

Die Kompensationssphären sehen dazu einen *Backout-Schutz* vor. Der Backout-Schutz kann dabei explizit angegeben werden, damit der Kontrollfluss die Transaktionsgrenzen nicht vor dem Transaktionsende verlässt. Eine Transaktion kann dabei dann als abgeschlossen betracht werden, wenn keine Aktivitäten in dieser Transaktion aktiviert werden können. Voreingestellt bleibt jedoch der Backout-Schutz nicht aktiv. Entweder werden bei der Transaktionsmodellierung solche Anomalien zugelassen, oder z.B. durch das Festlegen der Eigenschaft „Wiederholungsversuche“ auf „0“ oder durch entsprechende Erweiterung von Transaktionsgrenzen vermieden [Ley95].

Um einen Backout-Schutz auch für Sequenzielle Transaktionen der Mobilen Prozesse zu ermöglichen, müssen die angrenzenden Aktivitäten, die ausgeführt werden können,

gesperrt werden, so dass die Prozessausführung diese nicht ausführt. Beim Beenden der Transaktion sollten diese Aktivitäten dann für die Ausführung freigegeben werden. Auch im Abbruchsfall werden die gesperrten Aktivitäten freigegeben jedoch, aufgrund der Zurücksetzung der vorgelagerten Bedingungen, nicht mehr ausgeführt, so dass dadurch mögliche Inkonsistenzen vermieden werden können.

Erweiterung S2: Garantierte Sequenz in einem komplexen Graph

In einer Sequenziellen Transaktion gibt es keinen Synchronisationsaufwand für mehrere nebenläufige Pfade, da diese nicht entstehen können. Daneben gibt es komplexere Kontrollflussstrukturen, die zwar Verzweigungen aufweisen, jedoch garantieren, dass zur Ausführungszeit in der Transaktion keine nebenläufigen Pfade entstehen.

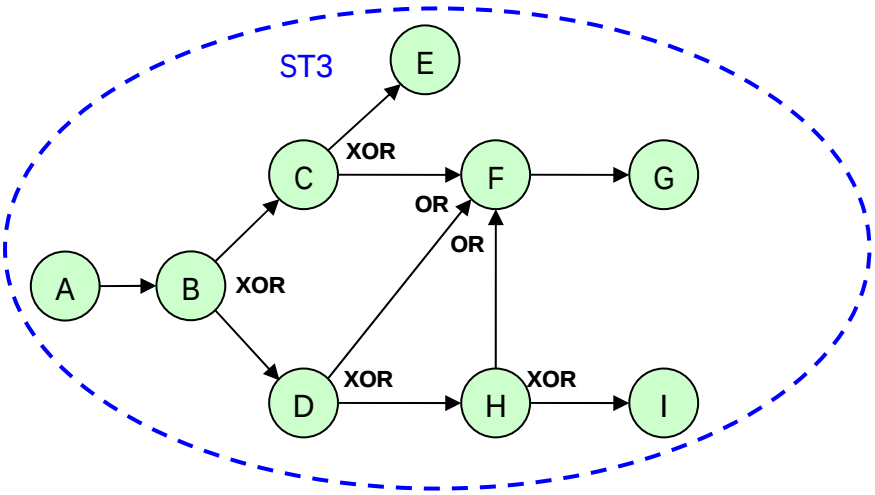

Abbildung 19: Sequentieller Verlauf im komplexen Graph

Dieser Fall ist in der Abbildung 19 zu sehen. Die Transaktion *ST3* umschließt einen Prozessteil, welcher vier Splits enthält. Dabei ist allerdings durch die *Split-Bedingung* „XOR“ immer sichergestellt, dass bei der Ausführung nur einer der Pfade weiterverfolgt werden kann. Somit kann auch diese Transaktion als eine Sequenzielle Transaktion modelliert werden.

Diese Erweiterung kann ohne Erschwernisse eingesetzt werden, wobei der Schwerpunkt der Umsetzung auf die Prozessmodellierung verlagert wird. Diese Erweiterung des Modells kann zusammen mit der Erweiterung S1 eingesetzt werden.

Erweiterung S3: Aufhebung der Verbundenheit

Bisher wurde angenommen, dass Sequenzielle Transaktion über zusammenhängende Prozessstrukturen gebildet werden kann. Diese Restriktion kann jedoch bei Bedarf gelockert werden. In diesem Fall ist eine Sequenzielle Transaktion nicht verbunden. Dieses bedeutet, dass zwischen den von der Transaktion abgesicherten Aktivitäten auch nicht geschützte Pfade vorkommen dürfen.

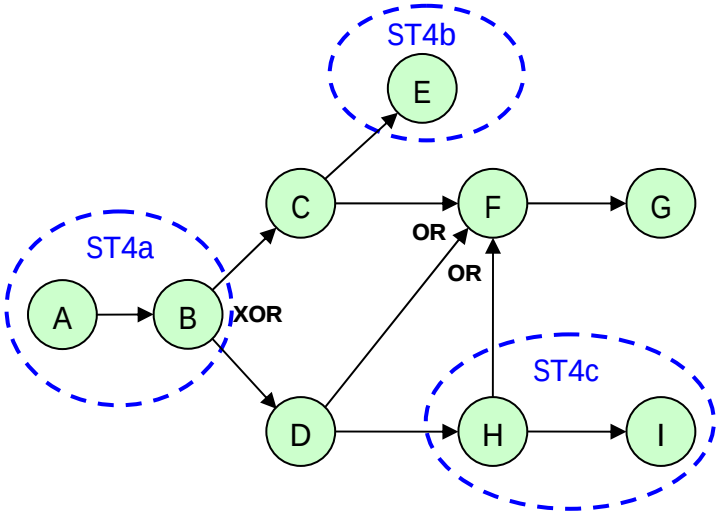

Abbildung 20: Nicht verbundene Sequenzielle Transaktion

In der Abbildung 20 ist dazu ein Beispiel dargestellt. Zu sehen ist eine Sequenzielle Transaktion *ST4*, die aus drei nicht verbundenen Bereichen besteht. Dabei reicht in diesem Fall zum Ausschluss der Nebenläufigkeit nur eine „XOR" – Bedingung. Diese garantiert, dass nachdem Abschnitt *ST4a* verlassen wird, die Prozessausführung entweder in den Abschnitt *ST4b* oder in den Abschnitt *ST4c* oder keinen davon wieder eintritt. Es werden jedoch niemals die Abschnitte *ST4b* und *ST4c* zusammen ausgeführt, was die Atomarität garantiert.

Hierbei können außerhalb der Transaktionsgrenzen beliebige Fehlersituationen entstehen, auf Grund derer, Teile der Transaktion möglicherweise nicht erreicht werden. Dieses muss bei der Transaktionsmodellierung berücksichtigt werden. So kann dabei der Backout-Schutz (vgl. Erweiterung S1) nicht verwendet werden, da sonst die Transaktion in einem Deadlock verharrt. In der Abbildung 20 kann dies verfolgt werden. Zu sehen ist, dass Bereich *ST4a* nicht verlassen werden kann, da die gesamte Transaktion nicht zu Ende ist. Die gesamte Transaktion kann wiederum nicht abschließen, weil noch nicht alle Bereiche der Transaktion ausgeführt wurden.

4.4.3 Join Transaktion

In der Sequenziellen Transaktion können keine echten parallelen Pfade abgebildet werden. Dennoch ist ein transaktionaler Schutz auch bei der Ausführung von parallelen Pfaden wünschenswert. Dies kann bei einem kompensationsbasierten Ansatz durch Verwendung eines Synchronisationspunktes erreicht werden (vgl. 6.4). Entscheidend ist dabei der Ort des Synchronisationskoordinators.

Die parallel durchgeführten Pfade synchronisieren sich in Joins. Die Join-Aktivität ist in der Prozessbeschreibung eindeutig definiert, so dass alle parallelen Pfade einen eindeutigen Standort des physischen Join-Knotens kennen. Es ist also nahe liegend auch für die Transaktionssynchronisation diesen Knoten zu verwenden. Dazu ist in der Abbildung 21 ein schematisches Modell einer *Join Transaktion* zu sehen. Eine Join Transaktion besteht aus mehreren Sequenztransaktionen, deren Pfade durch eine Join Aktivität vereinigt werden.

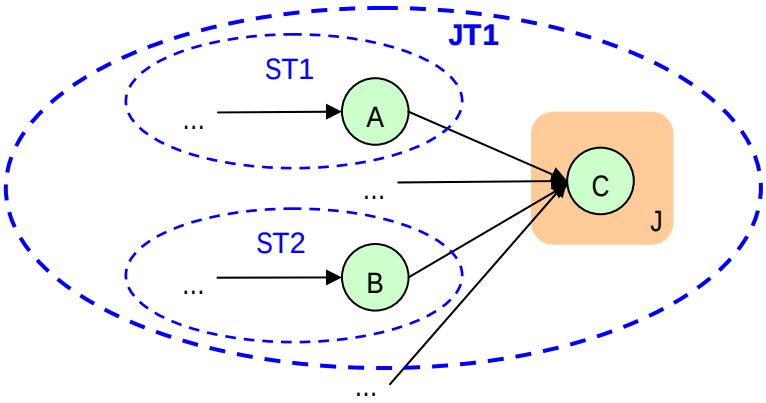

Abbildung 21: Join Transaktion schematisch

In der Abbildung 21 ist die Join Transaktion JT1 durch die Schachtelung von Sequenziellen Transaktionen *ST1* und *ST2* dargestellt. Dabei befindet sich die Join Aktivität *C* auf dem eindeutig definierten physischen Netzwerkknoten *J*. Die Join Aktivität selbst (*C*) kann dabei ebenfalls an der Transaktion partizipieren, muss jedoch nicht.

Auch dieses Modell ist optimistisch, so dass im Normalfall nur ein geringer transaktionsbedingter Overhead zu erwarten ist. Die einzelnen Sequenziellen Transaktionen werden entweder erfolgreich ausgeführt oder scheitern, müssen jedoch ihren Status dem Join-Knoten nicht explizit mitteilen, da diese Information im Zuge der Synchronisation der einzelnen Ausführungspfade zum Join-Knoten gelangt. Auch wenn auf einzelnen Kontrollflusspfaden die Ausführung der Aktivitäten vor dem Erreichen des Join-Knotens eingestellt wird, erfordert Deadpath-Eliminierung, dass die Prozesskontrolle dieser Pfade den Join Knoten trotzdem erreichen muss (vgl. [Zap05]). Damit werden auch die

benötigten Informationen über den Transaktionsverlauf der Pfade an den Join-Knoten implizit überbracht.

Bei einer Join Transaktion wird angenommen, dass alle an der Join Transaktion beteiligte Prozessausführungsstränge in die jeweiligen Subtransaktionen eintreten und diese damit begonnen werden. Daraus folgt, dass keine Information über den Beginn der Subtransaktionen an den Join-Knoten weitergeleitet werden muss. Damit kann der Nachrichtenaustausch über unzuverlässige Verbindungen gering gehalten werden, was die Robustheit und Skalierbarkeit des Ansatzes erhöht. Ob die zugehörigen Subtransaktionen tatsächlich gestartet wurden, kann beim Eintreffen der einzelnen Kontrollflusspfade festgestellt werden.

Eine Join Transaktion ist eine globale Transaktion, die aus mehreren sequenziellen Subtransaktionen besteht. Dabei scheitert die gesamte Join Transaktion, wenn eine der beteiligten Sequenziellen Transaktionen endgültig scheitert. Dies ist dann der Fall, wenn entweder Zeitrestriktionen überschritten werden, oder:

1. Forward Recovery der Subtransaktion scheitert
2. Die Subtransaktion Backward Recovery durchgeführt hat
3. Eine vorgegebene Anzahl an Wiederholungen der Subtransaktion durchgeführt wurde

Wenn eine Subtransaktion endgültig scheitert, müssen auch alle anderen kompensiert werden. Dabei wird der Join-Knoten die Kompensation, der bereits abgeschlossenen Subtransaktionen, einleiten. Transaktional gesicherte Pfade, die erst zu einem späteren Zeitpunkt den Join-Knoten erreichen, werden entsprechend beim Eintreffzeitpunkt kompensiert. Dazu muss die Information über den Transaktionsausgang beim Join-Knoten unter Umständen längere Zeit vorgehalten werden, bis alle beteiligten Stränge angekommen sind bzw. die maximale Wartezeit überschritten wird.

Somit ist eine Join Transaktion ein effizientes Transaktionsmodell, das über Kontrollflussstrukturen mit Joins gebildet werden kann. Weiterhin können Join Transaktionen ineinander verschachtelt werden, so dass auch mehrere Joins in einer Transaktion möglich sind. Dies ist in der Abbildung 22 dargestellt.

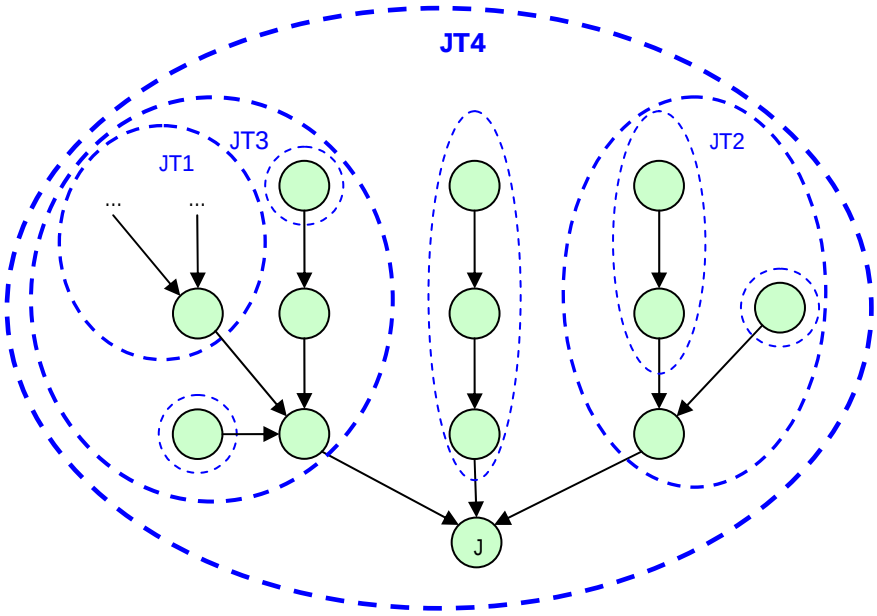

Abbildung 22: Verschachtelung von Join Transaktionen

In der Abbildung erkennt man fünf Sequenzielle Transaktionen und drei Join Transaktionen, die wiederum bei der unteren Join-Aktivität zu einer weiteren Join Transaktion (*JT4*) vereinigt werden. Dies Modell ist durchaus ablauffähig, allerdings hat man hier bereits mit dreizehn Aktivitäten ein relativ komplexes Modell. Verschachtelung von Join Transaktionen sollte deswegen möglichst flach gehalten werden.

Durch den Einsatz von so genannten Routing Aktivitäten, die keine Funktion außer der Kontrollflusssteuerung besitzen, ist es jedoch möglich auch die nicht mit einem Join vereinigten Pfade künstlich zu vereinen. In den meisten Fällen kann man damit nach jedem Split durch eine Routing Aktivität einen Join-Knoten nur für die Transaktionsabwicklung definieren. Die Anwendung von Routing Aktivitäten ermöglicht also eine effiziente Transaktionsunterstützung für eine weitere Klasse von Anwendungsfällen [LeRo99, Zap05].

Erweiterung J1: Hinzunahme eines nicht verbunden Pfades

Das Grundmodell einer Join Transaktion bietet eine effiziente Transaktionssicherung für Pfade, die ausschließlich in einem Join vereinigt werden. Wünschenswert ist aber auch die Unterstützung von allgemeineren Fällen, wie beispielsweise die Unterstützung von weiteren, nicht am Join teilnehmenden Pfaden.

Abbildung 23 skizziert diesen Fall. Zu erkennen ist, dass innerhalb einer Join Transaktion *JT1* drei Sequenzielle Transaktionen *ST1*, *ST2* und *ST3* modelliert sind, wobei aus der *ST3* kein Pfad zum Join Knoten führt.

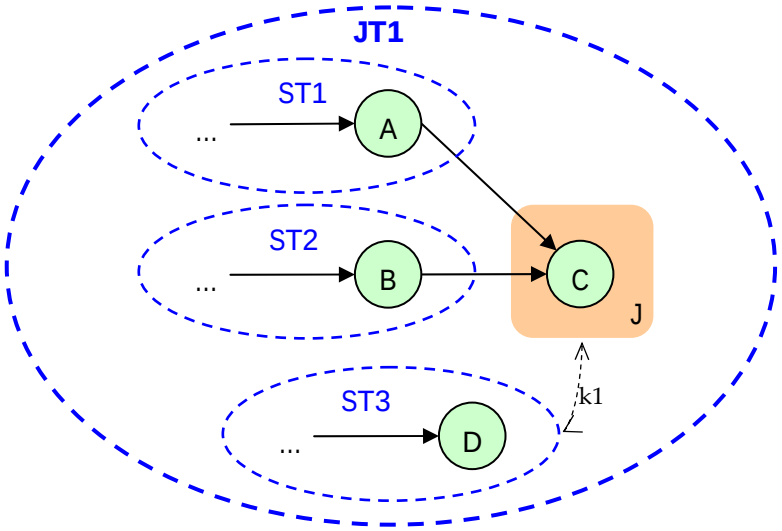

Abbildung 23: Nichtverbundener Pfad in einer Join Transaktion

Auch hierbei wird angenommen, dass alle Subtransaktionen gestartet werden, so dass der Subtransaktionsbeginn nicht explizit quittiert werden muss. Jedoch sollte bei einer solchen Modellierung berücksichtigt werden, dass nicht verbundene Pfade bei ihrer Ausführung nicht auf den Join-Knoten migrieren müssen, da sie am Join nicht partizipieren. Der Status, der nicht verbundenen Subtransaktionen, sowie kompensationsrelevante Informationen, sollten also in der Regel explizit übertragen werden (*k1*).

Weiterhin kann die Ausblendung der Annahme, dass alle an einer Join Transaktion beteiligten Knoten in jedem Fall gestartet werden, mehr Flexibilität bei der Transaktionsmodellierung bedeuten. Jedoch würde dies auch zu einem größeren Kommunikationsaufwand führen. Wenn generell das explizite Melden des Anfangs und des Endes von Subtransaktionen akzeptiert werden kann, dann schwindet die Bedeutung eines Join-Knoten in der Rolle eines Synchronisationspunktes. Ein vergleichbarer Kommunikationsaufwand wäre auch bei anderen synchronisationsfähigen Knoten anzutreffen, so dass ein Join-Knoten nicht unbedingt eine Besonderheit darstellt. Darauf basierte Transaktionsmechanismen werden im folgenden Abschnitt dargestellt.

4.4.4 Flexible Synchronisierte Transaktion

In vorhergehenden Abschnitten wurden effiziente Transaktionsmodelle für konkrete Kontrollflussstrukturen vorgestellt. Um der Anforderung T4 (Mächtigkeit der Transaktionsdefinition) vollständig gerecht zu werden, wird in diesem Abschnitt ein weiteres allgemeines kontrollflussunabhängiges Transaktionsmodell beschrieben. Auch in diesem Fall wird ein kompensationsbasierter Ansatz verfolgt, da sich dieser für lang andauernde Mobile Prozesse besser eignet (vgl. 6.2).

Das in diesem Abschnitt beschriebene Transaktionsmodell wird als *Flexible Synchronisierte Transaktion* (FST) bezeichnet. Es handelt sich dabei um eine weitere Variante der offen geschachtelten Transaktion, wobei die Sequenzielle und Join Transaktionen in diesem Modell als Subtransaktionen fungieren. Im Unterschied zur Join Transaktion ist die Synchronisation in FST an keine Kontrollflussstruktur gebunden. Die Synchronisation baut auf einem in der Umgebung vorhandenen Synchronisationsdienst auf. Dabei kann der Synchronisationsdienst beim Beginn der Prozessausführung anhand nichtfunktionaler Aspekte gefunden werden. Alternativ kann die Adresse des Dienstes bereits bei der Transaktionsmodellierung explizit angegeben werden.

Demgemäß sollten Synchronisationsdienste eher dauerhaft verfügbar und erreichbar sein. Im Allgemeinen sind diese Annahmen jedoch bei der Ausführung von Mobilen Prozessen nicht definiert, da Mobile Prozesse zum Ablauf keine bestimmte Infrastruktur voraus setzen. In Spezialfällen wie z.B. bei einem Join, werden jedoch implizit Geräte mit besonderen Fähigkeiten bezüglich der Erreichbarkeit und Leistung vorausgesetzt. So sollte ein Join-Knoten nach Möglichkeit dauerhaft erreichbar bleiben, damit eine Zusammenführung der einzelnen Prozesspfade nicht wiederholt zum Scheitern gezwungen wird. Diese Anforderungen beziehen sich auch auf den FST-Synchronisationsdienst.

Eine effiziente Unterstützung von Join Transaktionen und FSTs kann also nur mit einer Annahme über die besonderen Zuverlässigkeitsmerkmale der beteiligten Geräte geschehen. Diese implizit vorausgesetzte Annahme wird im Folgenden explizit als *Inhomogenitätsvoraussetzung* formuliert:

> *Für eine effiziente und vollständige Unterstützung der Ausführung von transaktionalen Mobilen Prozessen, sollte in der mobilen Umgebung mindestens ein, im Verhältnis zu den Übrigen, besonders zuverlässiger (erreichbarer und verfügbarer) Netzwerkknoten auffindbar sein.*

Diese Netzwerkknoten können als Join-Knoten sowie als Ausführungsorte für spezielle Dienste, wie z.B. ein Synchronisationsdienst, eingesetzt werden. Dabei ist keine Festlegung auf eine Technologie notwendig, je nach Situation, können verschiedene Netzwerknoten als relativ zuverlässig eingestuft werden. So ist beispielsweise die Erreichbarkeit und Zuverlässigkeit eines stationären Rechners mit WLAN, Bluetooth und Internet Anschlüssen im Vergleich höher als die der mitbetrachteten PDAs und Laptops. Die Funktion eines Synchronisationsdienstes, der auf einem solchen Knoten ablaufen soll, wird im nächsten Kapitel dargestellt, vorerst werden jedoch allgemeine Eigenschaften der Flexiblen Synchronisierten Transaktion erläutert.

Eine Flexible Synchronisierte Transaktion kann aus Sequenziellen sowie Join Transaktionen bestehen. Dabei können diese beliebig angeordnet werden ohne Überlappungen zu bilden. Dabei ist auch die Verschachtelung von Join Transaktionen erlaubt, da sie auf der FST-Ebene nicht sichtbar ist. Zusammen bilden diese Subtransaktionen eine globale Flexible Synchronisierte Transaktion. In der Abbildung 24 ist eine Flexible Synchronisierte Transaktion *T1* zu sehen, sowie ein Ausschnitt des Kommunikationsablaufs durch Nachrichtenaustausch *k1*, *k2*, *k3*. Dabei ist zu erkennen, dass diese aus vier Sequenztransaktionen besteht, wobei zwei davon in einer Join Transaktion aggregiert sind.

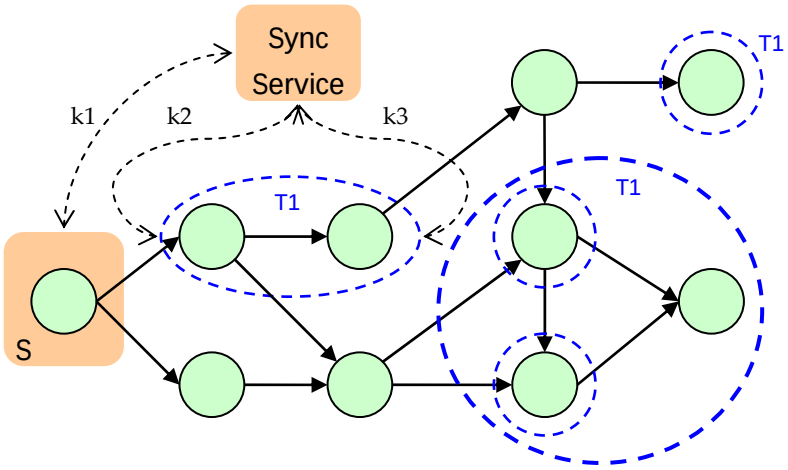

Abbildung 24: Beispiel einer Flexiblen Synchronisierten Transaktion

Alternativ könnte die gleiche Transaktion mit sechs Sequenztransaktionen abgebildet werden, wobei jede in der FST beteiligte Aktivität in einzelnen Subtransaktionen liegen würde. Diese Modellierung würde aber zu einem erhöhten Kommunikationsaufwand zwischen den Prozessteilnehmern und dem Synchronisationsdienst führen, was in einer mobilen Umgebung nach Möglichkeit zu vermeiden ist. Dies liegt daran, dass die Subtransaktionen weitgehend autonom sind und nur bei ihrem Start und Ende, oder in einem Fehlerfall, mit dem Synchronisationsdienst kommunizieren müssen. Im Folgenden wird das zu Grunde liegende Kommunikationsprotokoll genauer erläutert.

Bei der Verwaltung von FST müssen für die im Prozess definierten FSTs zunächst die Transaktionskontexte erzeugt werden. Dieser Vorgang kann gleich nach Prozessbeginn auf dem Initialgerät stattfinden. Dabei wird der Synchronisationsdienst gesucht. Wenn der Dienst erreichbar ist, können die Transaktionskontexte erzeugt werden. Der Ablauf dazu wird in der Abbildung 25 dargestellt und in der Abbildung 24 als *k1* demonstriert. Als Erstes wird für jede im Prozess modellierte FST die *createContext*-Operation des Synchronisationsdienstes aufgerufen, wobei die Anzahl der in einer FST definierten Subtransaktionen übergeben wird. Daraufhin antwortet der Synchronisationsdienst im Normalfall mit einer Bestätigung und dem Transaktionskontext für die Transaktion. Mehrere FSTs eines Prozesses können von unterschiedlichen Synchronisationsdiensten verwaltet werden.

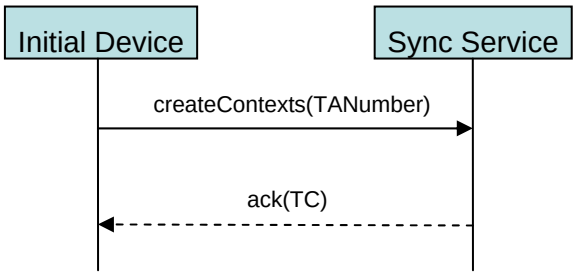

Abbildung 25: FST Coordination Protocol

Die Anmeldung und Erzeugung von Kontexten kann auch zu einem späteren Zeitpunkt stattfinden, entscheidend dabei ist, dass es früh genug stattfinden soll, so dass alle Subtransaktionen eindeutige Kontexte ihrer übergeordneten FSTs erfahren. Dabei kann ein Kontext aus der Adresse bzw. *URI* des Synchronisationsdienstes und einem FST-Indentifikator (*FST-ID*) bestehen. Nach seiner Erzeugung, wird er mit einer FST verknüpft und mit den übrigen Prozessdaten in der Prozessbeschreibung gespeichert. So wird der Transaktionskontext bei allen Migrationen mitgeführt.

Wenn die Prozessausführung eine Subtransaktion starten möchte, wird dies dem Synchronisationsdienst mitgeteilt (vgl. *k2* in Abb. 24). Der dabei anfallende Nachrichtenaustausch ist in der Abbildung 26 dargestellt. Das aktuelle Gerät ruft die Operation *registerForSync* des Synchronisationsdienstes auf, wobei der Transaktionskontext *TC* der FST übergeben wird. Der Synchronisationsdienst notiert, dass im Rahmen der globalen FST mit dem Kontext *TC* eine neue Subtransaktion gestartet wird. Dabei überprüft er, ob diese FST bereits gescheitert ist. Wenn das nicht der Fall ist, wird mit einer Bestätigung geantwortet und ein Subtransaktionskontext *SubTC* erzeugt, welcher mit übergegeben wird. Im anderen Fall antwortet der Dienst mit einem Abbruchbefehl, welcher die Subtransaktionsausführung beendet.

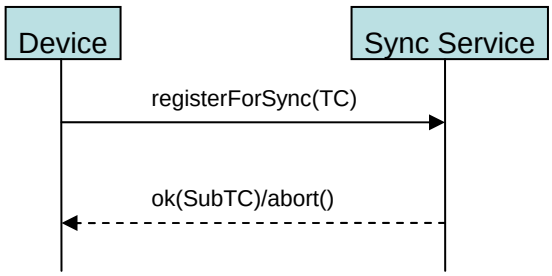

Abbildung 26: Anmeldung zum Beginn einer Subtransaktion

Wenn die Subtransaktion erfolgreich durchgeführt wurde, wird das dem Synchronisationsdienst mitgeteilt (Vgl. *k3* in Abb. 24). In der Abbildung 27 ist zu sehen, dass dabei jeweils der Kontext der Subtransaktion, sowie die erforderliche Kompensationsinformation *CopmInf* an den Synchronisationsdienst weitergegeben wird.

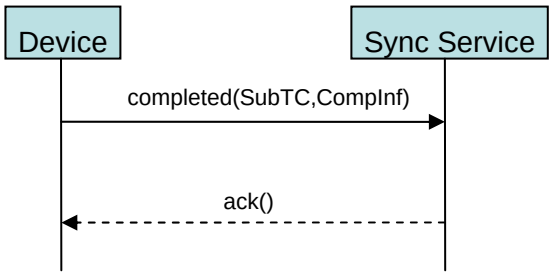

Abbildung 27: Abschluss einer Subtransaktion

Die Kompensationsinformation stellt dabei eine Beschreibung des Kompensationsprozesses für diese Subtransaktion dar. Im diskreten Fall wird diese Beschreibung dynamisch erzeugt, im globalen Fall wird entsprechend der statische Ausschnitt der Originalprozessdefinition verwendet. Dies dient der Erhaltung der kompensationsrelevanten Information. Im Abbruchsfall kann der Synchronisationsdienst die Kompensationsprozesse einleiten. Damit wird die Verantwortung über die Kompensierung an den Synchronisationsdienst übertragen, der nach der Inhomogenitätsvoraussetzung eine größere Chance besitzt die Kompensation erfolgreich durchzuführen.

Wie bereits beschrieben, können Subtransaktionen scheitern, dabei wird dies dem Koordinator explizit mitgeteilt oder der Synchronisationsdienst stellt es über definierte Zeitrestriktionen fest. Wenn eine der Subtransaktionen einer FST scheitert, muss die gesamte FST scheitern, um die Atomarität zu sichern. In diesem Fall werden für alle bereits ausgeführten Subtransaktionen die gespeicherten Kompensationsprozesse gestartet. Alle Subtransaktionen der gescheiterten FST, die zu beginnen beabsichtigen, werden durch den Synchronisationsdienst abgebrochen.

Weiterhin ist es möglich das Grundmodell der Flexiblen Synchronisierten Transaktion insbesondere die Kommunikationsprotokolle noch weiter zu optimieren. Da jedoch diese Optimierungsmöglichkeiten anwendungsabhängig sein können, werden sie an dieser Stelle nicht betrachtet.

4.5 Gegenüberstellung und Bewertung der Modelle

In vorherigen Abschnitten wurden drei Transaktionsmodelle vorgestellt, welche zusammen eine flexible Transaktionsunterstützung für verteilt ausgeführte Mobile Prozesse bieten. An dieser Stelle werden diese Modelle gegenübergestellt und bewertet.

Sequenzielle Transaktion kann dabei als Basis der entwickelten Transaktionsunterstützung angesehen werden. Kompensation und Wiederholungen finden auf der Ebene der Sequenziellen Transaktion statt, so dass ihre Eigenschaften für Join und Flexible Synchronisierte Transaktion bestimmend sind. In der Tabelle 4 sind die Eigenschaften der einzelnen Transaktionsmodelle den Anforderungen gegenübergestellt.

Dabei ist zu erkennen, dass Sequenzielle Transaktion die Anforderungen M1 (Toleranz der Knotenausfälle) und M2 (Geringer Kommunikationsaufwand) voll erfüllt. Sequenzielle Transaktion erfordert keine Kommunikation, da sie keine Parallelität zulässt und die Transaktionsunterstützung sich immer auf nur einem Gerät befindet. Wenn dabei der ausführende Knoten ausfällt, kann die Transaktionsunterstützung beim wiederholten Anlauf einfach fortgesetzt werden.

Weiterhin setzt eine Sequenzielle Transaktion keine speziellen Eigenschaften der Netzwerkarchitektur voraus, was der Anforderung P1 (Abstraktion von der Netzwerkstruktur) gänzlich entspricht. Auch die Migration (Anforderung P2) wird unterstützt. Auch nach Übertragung der Prozessbeschreibung von einem Gerät auf ein Anderes, kann die Transaktion weiter fortgesetzt werden. Ebenfalls wird die Anforderung P3 (Effiziente Unterstützung langer Transaktionen) unterstützt, da keine Sperren verwendet werden.

Die Atomarität (T1) wird bei Sequenziellen Transaktionen durch die Kompensationsroutine definiert. Wodurch auch die Konsistenz (T2) zum Teil gewährleistet wird. Sie ist nur zum Teil erfüllt, da eine Sequenzielle Transaktion keine Garantien für die sichere Ausführung des Kompensationsprozesses geben kann. Weiterhin kann Sequenzielle Transaktion geschützte sowie nichtgeschützte Dienste unterstützen, was der Anforderung T3 entspricht.

Alle Eigenschaften der Sequenziellen Transaktion werden jedoch durch den Nachteil der eingeschränkten Flexibilität bei der Transaktionsdefinition bestimmt, deshalb wird die Anforderung T4 (Mächtigkeit von Transaktionsdefinitionen) nicht erfüllt.

Tabelle 4: Gegenüberstellung der entwickelten Modelle

	M1	M2	P1	P2	P3	T1	T2	T3	T4
Sequenzielle Transaktion	+	+	+	+	+	+	+/-	+	-
Join Transaktion Grundmodell	+	+	+	+	+	+	+/-	+	+/-
Flexible Synchronisierte Transaktion	-	+/-	+/-	+	+	+	+/-	+	+

+ Wird unterstützt, - Wird nicht unterstützt, +/- Teilweise Unterstützung

Join Transaktionen bestehen aus mehreren Sequenziellen Transaktionen, wobei sie nur eine globale Abbruchentscheidung synchronisieren. Damit erben Join Transaktionen alle Eigenschaften der Sequenziellen Transaktionen, zeichnen sich aber durch die Möglichkeiten der Abbildung von parallelen Kontrollflusspfaden aus. Allerdings sind diese an die Kontrollstruktur Join gebunden, so dass die Anforderung T4 nur teilweise erfüllt ist (vgl. Tabelle 4).

FST erfüllen dagegen die Anforderung T4 (Mächtigkeit der Transaktionsdefinition) voll, da beliebige Transaktionsstrukturen erlaubt sind. Allerdings wird bei FST ein Synchronisationsdienst vorausgesetzt, wodurch die Anforderung P1 (Abstraktion von der Netzwerkstruktur) nicht ganz erfüllt ist. Außerdem kann ein längerer Ausfall des Koordinators zu ungewollten Ergebnissen führen, so dass die M1 Anforderung (Toleranz der Knotenausfälle) nicht erfüllt wird. Schließlich wird auch die Anforderung M2 (Geringer Kommunikationsaufwand) nicht mehr ganz erfüllt, da zwischen dem Synchronisationsdienst und den für Subtransaktionen verantwortlichen Geräten kommuniziert werden muss.

Durch vorgestellte Modelle erhalten Mobile Prozesse eine effiziente und skalierbare Transaktionsunterstützung. Fallweise können verschiedene Transaktionsmodelle eingesetzt werden. Während für eine einfache Atomaritätssicherung der auf einander folgender Aktivitäten, die Sequenzielle Transaktion einen effizienten Schutz bietet, können durch FSTs komplexe Prozessabschnitte oder gar ganze Prozesse transaktional geschützt werden. Dem Modellierer stehen also drei konfigurierbare und erweiterbare Transaktionsmodelle zur Verfügung. Durch deren geschickte Kombinierung kann ein adäquater Transaktionsschutz für verschiedene Anwendungssituationen geboten werden.

5 Entwurf der Middleware-Komponenten

In diesem Kapitel werden Entwürfe der Softwarekomponenten beschrieben, welche die Realisierung der im vorherigen Kapitel konzipierten Transaktionsmodelle darstellen. Diese Komponenten sollen die DEMAC-Middleware um den Transaktionsschutz bei der Ausführung von verteilt ausgeführten Mobilen Prozessen erweitern. Zunächst wird die Architektur der DEMAC-Middleware beschrieben. Danach wird die Funktionsweise von den zwei entwickelten Softwarekomponenten erläutert und im Rahmen der DEMAC-Middleware eingeordnet. Anschließend werden die benötigten Erweiterungen der *Demac Process Description Language* (*DPDL*) [Zap05] beschrieben, welche für Modellierung Mobiler Prozesse in der DEMAC-Middleware eingesetzt wird.

5.1 DEMAC-Architektur

In diesem Abschnitt werden die Ziele des DEMAC-Projekts erläutert, sowie eine grobe Darstellung der Systemarchitektur der DEMAC-Middleware gegeben. Im Rahmen des DEMAC-Projekts wird eine kontextbasierte Kooperation zwischen mobilen Geräten untersucht. Durch die Zusammenarbeit der mobilen Geräte (mit ihren spezifischen Fähigkeiten) entstehende Synergieeffekte, sollen dabei für Anwender und ihre Aufgaben nutzbar gemacht werden. Lang andauernde Aufgaben, die von mobilen Geräten ausgeführt werden können, werden dabei durch Mobile Prozesse Abgebildet. In diesem Zusammenhang ist ein weiteres Ziel des DEMAC-Projektes die Konzipierung und Entwicklung einer Middleware-Plattform zur Ausführung von Mobilen Prozessen auf mobilen Geräten. Zudem sollen ein Metamodell zur Beschreibung Mobiler Prozesse mit dem zugehörigen Interpretern entwickelt werden [Kun05a].

Die DEMAC Systemarchitektur beinhaltet vier grundlegende Komponenten (siehe Abbildung 28), welche zusammen die Ausführung von mobilen Prozessen ermöglichen sollen. Im Folgenden werden diese Komponenten, welche als *DEMAC-Services* bezeichnet werden, kurz dargestellt.

Asynchronous Transport Service: Der *Asynchronous Transport Service* stellt Kommunikationsinfrastruktur für alle Komponenten und Applikationen der DEMAC-Architektur bereit. Dabei wird die Abstraktion von konkreten Transportprotokollen, wie TCP/IP, IrDA oder Bluetooth ermöglicht, so dass den Komponenten, die auf dem Transportservice aufsetzen, eine einheitliche Schnittstelle geboten wird, unabhängig von der verwendeten Kommunikationstechnologie. Weiterhin ermöglicht es der Transport Service, Nachrichten zu senden und zu empfangen. Dies geschieht asynchron, ermöglicht durch eine eindeutige Kennzeichnung der Nachrichten. Daher können die DEMAC-

Komponenten oder Applikationen sich beim Transport Service registrieren, damit sie bei der Ankunft einer erwarteten Nachricht automatisch benachrichtigt werden [Kun05a].

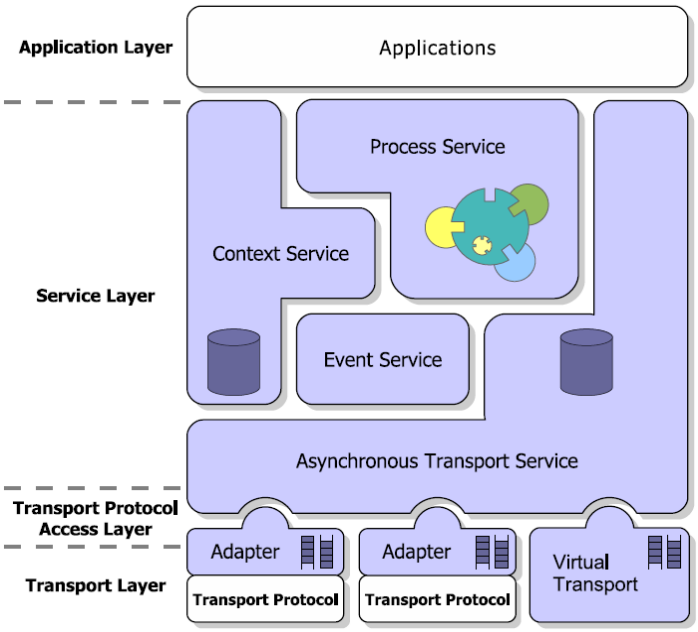

Abbildung 28: DEMAC Middleware-Architektur (aus [Kun05a])

Event Service: Spezielle Nachrichten (Events) mit der Information über die Änderungen des Eigenzustandes bzw. der Umweltsituation, werden vom *Event Service* verwaltet. Solche Zustandsänderungen z.B. ein Verbindungsabbruch werden als Events verpackt und an Interessenten weitergegeben. An entfernte Interessenten werden die Events mit Hilfe von Asynchronous Transport Service verschickt. Ein empfangendes externes Event wird in ein lokales umgewandelt und an den Context Service weitergegeben [Kun05a].

Contex Service: *Context Service* sammelt und verwaltet alle Informationen über den Kontext des Geräts (vgl. 2.3.1). Die Kontextinformation wird über Hardware Sensoren, ankommende Events oder durch einen expliziten Nachrichtenaustausch ermittelt. Die Komponenten oder Applikationen erhalten vom Context Service die benötigten Informationen. Dazu verwaltet der Service ein generisches und erweiterbares Kontextmodell, welches das gesamte Kontextwissen darstellt [Kun05a, Tur06].

Process Service: Der *Process Service* realisiert die Verwaltung und Ausführung von Mobilen Prozessen in der DEMAC-Architektur. Dabei ist der DEMAC Process Service modular aufgebaut und besitzt ein *Kern-* (*Core*) und ein *Basismodul* (*Base Module*) (vgl. Abbildung 29.) Das Kernmodul übernimmt dabei nur grundlegende Aufgaben, wie Empfang und Weiterleitung, sowie Speicherung von Prozessdefinitionen. Außerdem stellt es Schnittstellen für Anwendungen zur Verfügung, mit der die Prozessdefinitionen zur Abarbeitung an den

Process Service übergeben werden können. Das Basismodul baut auf den von dem Kernmodul bereitgestellten Funktionalitäten auf, wobei das Basismodul für die Interpretation und Ausführung von Prozessdefinitionen verantwortlich ist. Hierbei ist der Process Service bei der Suche nach Diensten bzw. nach geeigneten Geräten zur Prozessdelegierung auf den Context Service angewiesen. Außerdem kann das Basismodul durch so genannte Erweiterungsmodule (*Extension Module*) mit zusätzlichen Funktionalitäten angereichert werden (vgl. Abbildung 29) [Zap05, Kun05a].

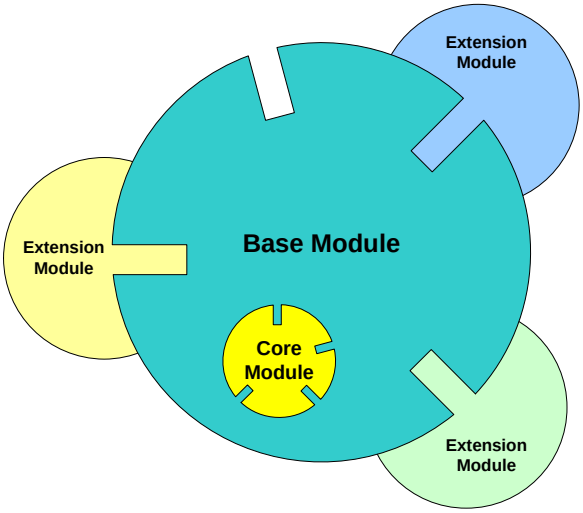

Abbildung 29: DEMAC Process Service (nach [Zap05])

Ein solcher modularer Aufbau des Process Services soll die Beteiligung möglichst vieler mobiler Geräte, in Abhängigkeit von ihrer Leistung, ermöglichen. So können leistungsschwache Geräte beispielsweise nur das Kernmodul implementieren, aber dadurch bereits einen Beitrag zur verteilten Prozessausführung leisten [Zap05].

5.2 Beschreibung entwickelter Komponenten

In diesem Abschnitt wird ein Entwurf des Transaktionsverwaltungssystems im Rahmen des DEMAC-Projektes dargestellt. Im Vordergrund stehen dabei die in Kapitel 4 entwickelten Transaktionsmodelle. Die Unterstützung dieser Transaktionsmodelle basiert dabei auf zwei Hauptkomponenten, die als Erweiterungsmodule des DEMAC Process Services entworfen sind: dem *Transaktionsverwaltungsmodul* und dem *Synchronisationsdienstmodul*. Als Nächstes werden diese Erweiterungsmodule einzeln vorgestellt und anschließend notwendige Anpassungen des Basismoduls beschrieben.

5.2.1 Entwurf eines Transaktionsverwaltungsmoduls

Das Transaktionsverwaltungsmodul (*Transaction Support Module*) übernimmt die Hauptaufgaben der Transaktionsverwaltung in Mobilen Prozessen. Die Unterstützung von Sequenziellen Transaktionen mit allen Erweiterungen, sowie das Grundmodel der Join Transaktion wird durch dieses Modul ermöglicht.

Dabei sieht der Entwurf des Moduls vor, dass die eigentliche Prozessausführung weiterhin im Basismodul stattfindet und nur die für eine Transaktionssicherung relevanten Aufgaben im Transaktionsverwaltungsmodul durchgeführt werden. Auch das Basismodul sollte seinerseits nicht direkt in die Transaktionssicherung eingreifen.

Um eine Transaktionsunterstützung durch das Transaktionsverwaltungsmodul zu ermöglichen, sollte das Basismodul das Auftreten aller, für eine Transaktionsverwaltung relevanten Ereignisse während der Prozessausführung, dem Transaktionsverwaltungsmodul mitteilen. Zu transaktionsrelevanten Ereignissen zählt der Beginn und das Ende der Ausführung einer transaktionsgeschützten Aktivität. Das Basismodul muss also die Transaktionsgrenzen aus der Prozessbeschreibung (vgl. 5.3) erkennen und den Anfang und das Ende der Ausführung einer Aktivität dem Transaktionsverwaltungsmodul mitteilen. Da die Basiskomponente es nicht in diesem Umfang bietet, gehören zum Entwurf eines Transaktionsverwaltungsmoduls auch entsprechend erweiternde Anpassungen des Basismoduls (vgl. 5.2.3).

Das Transaktionsverwaltungsmodul führt alle zur Transaktionssicherung notwendigen Schritte aus. Dabei verfolgt es, in welchem Stadium sich die einzelnen Transaktionen befinden. Es wird beispielsweise geprüft ob einige Transaktionen als abgeschlossen betrachtet werden können oder eventuell zurückgesetzt werden müssen. Weiterhin übernimmt die Transaktionsverwaltung die Sicherung des Start-Sets der gestarteten Transaktionen, falls für diese, Wiederholungsversuche vorgesehen sind. Das Start-Set und alle anderen Verwaltungsinformationen werden dabei in der Prozessbeschreibung gesichert, so dass auch bei Prozessmigration das Start-Set erhalten bleibt.

Ferner müssen alle Fehler bei der Ausführung von Aktivitäten und Prozessen erkannt werden. Fehler, die bei der Ausführung von Aktivitäten entstehen, werden vom Basismodul erkannt. Dabei erhalten alle Aktivitäten Markierungen, die ihren aktuellen Zustand kennzeichnen. In der Abbildung 30 sind alle möglichen Zustände und Übergänge zwischen den Aktivitäten dargestellt. Dabei stellen die drei unteren Aktivitäten *skipped*, *expired* und *in error* Fehlerzustände vor oder während der Ausführung der Aktivitäten dar.

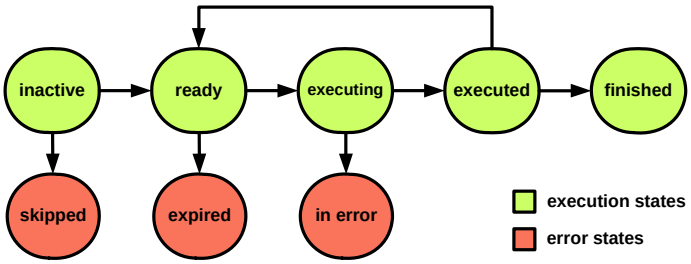

Abbildung 30: Lebenszyklus einer Aktivität (nach [LeRo99])

Aktivitäten werden als *skipped* markiert, wenn sie gar nicht bearbeitet werden sollen, also müssen diese Aktivitäten auch nicht bei der Kompensierung berücksichtigt werden. *Skipped*-Aktivitäten bewirken jedoch keinen Abbruch der Transaktionen. Wenn jedoch von dem Transaktionsausführungsmodul eine, in der Transaktion liegende, Aktivität mit der Markierung *expired* oder *in error* erscheint, muss die betroffene Transaktion kompensiert werden. Dabei kann eine Aktivität dann als *expired* markiert werden, wenn für sie definierte Zeitrestriktionen überschritten wurden. Als *in error* wird eine Aktivität beim Auftreten aller anderen Fehler markiert. Weiterhin können Transaktionen durch andere Bedingungen scheitern, so z.B. durch die Überschreitung von zeitlichen Restriktionen der Transaktionsausführung [LeRo99, Zap05].

Ein Backout-Schutz (vgl. Erweiterung S1 in 4.4.2) einer Sequenziellen Transaktion ermöglicht die Ausführung der, aus einer Transaktion führende, Pfade bis zum Abschluss der Transaktion zu verzögern. Um dies zu ermöglichen, kann das Transaktionsverwaltungsmodul beim Beginn einer Sequenziellen Transaktion alle betroffenen Aktivitäten außerhalb der Transaktion sperren und entweder beim erfolgreichen Transaktionsabschluss oder einem Transaktionsabbruch zur Ausführung wieder freigeben. Daraus folgt, dass der Aktivitätslebenszyklus um einen weiteren Zustand erweitert werden muss, welcher in der Prozessbeschreibung bei Migration mit übertragen wird. Dazu ist in der Abbildung 31 der erweiterte Lebenszyklus einer Aktivität zu sehen, welcher einen neuen Zustand „Locked“ aufweist.

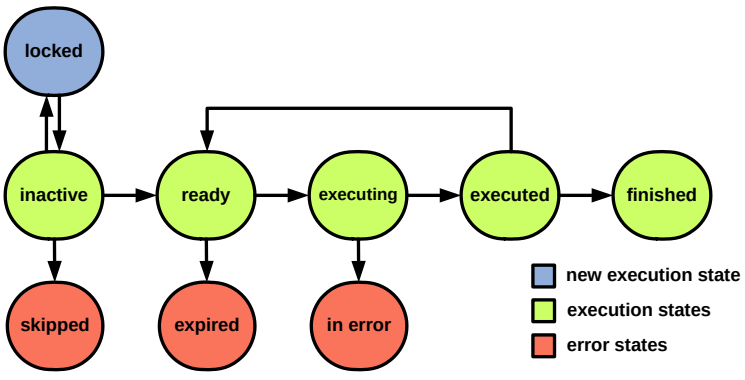

Abbildung 31: Erweiterter Lebenszyklus einer Aktivität

Aktivitäten, die sich in diesem Zustand befinden, werden als gesperrt betrachtet und von der Ausführung zeitweilig ausgenommen, bis sie vom Transaktionsverwaltungsmodul wieder als *inactive* markiert werden.

Zusammenfassend wird festgehalten, dass das Transaktionsverwaltungsmodul zwei wichtige Funktionalitäten besitzt: die Überprüfung der Transaktionszustände und Behandlung der Fehlerfälle. Außerdem sorgt es für die Speicherung des Start-Sets, und leitet die Kompensation und Wiederholungen ein. Bei der Verwaltung der Transaktionen mit parallelen Kontrollflusspfaden, ist es jedoch auf einen Synchronisationsdienst angewiesen, dessen Entwurf im nächsten Abschnitt dargestellt wird.

5.2.2 Entwurf des Synchronisationsdienstmoduls

Ein Synchronisationsdienstmodul (*Transaction Syncronisation Service Modul*) wird für die Unterstützung der Flexiblen Synchronisierten Transaktionen und für die Erweiterung J1 bei Join Transaktionen gebraucht. Dabei wird dieser Dienst im Rahmen der DEMAC-Middleware als ein weiteres Erweiterungsmodul des Process Services entworfen. Als ein Synchronisationsdienst wird demnach ein (mobiles) Gerät mit dem DEMAC Process Service und seiner Synchronisationsdienstmodul-Erweiterung betrachtet. Dabei können auf dem gleichen Gerät auch weitere Erweiterungsmodule zum Einsatz kommen, wie in der Abbildung 32 dargestellt. So können beispielsweise ein Synchronisationsdienstmodul und ein Transaktionsverwaltungsmodul auf einem Gerät vorkommen, was bei einer Join Transaktion mit der Erweiterung J1 der Fall ist.

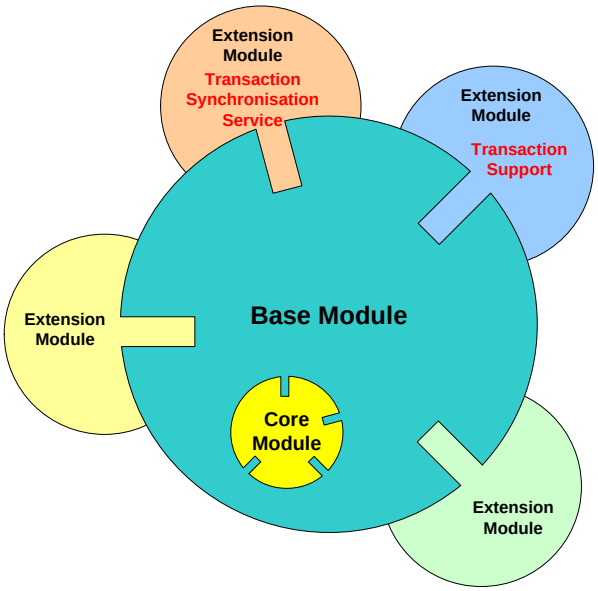

Abbildung 32: Process Service Architektur mit Transaktionsunterstützung

Die Hauptaufgabe des Synchronisationsdienstmoduls ist die Synchronisation der parallelen Kontrollflusspfade in einer Transaktion. Dabei kann die Abarbeitung einer Transaktion in drei Phasen unterteilt werden. In der ersten Phase werden die Transaktionen beim Synchronisationsdienst registriert. Dabei werden, die am Gerät ankommenden Nachrichten, von dem Assynchronous Transport Services zum Synchronisationsdienstmodul geleitet. Daraufhin vergibt das Synchronisationsdienstmodul den Transaktionen ihre eindeutigen Kennungen und verschickt diese mit Hilfe des Transport Services wieder zurück. Diese Kennungen werden auch in der lokalen Registratur (*Registry*-Komponente) des Synchronisationsdienstes gespeichert. Bei der Registrierung einer Transaktion wird auch die Anzahl der eingebetteten Subtransaktionen mitgegeben, sowie eventuell vorhandene Zeitgrenzen für die Transaktionsdauer. Diese Information wird ebenso gespeichert, da sie vom Synchronisationsdienstmodul in der zweiten Phase gebraucht wird.

Nach der Registrierungsphase einer Transaktion beginnt, aus der Sicht des Synchronisationsdienstes, die Phase der Transaktionsverwaltung. Dabei wertet das Synchronisationsdienstmodul die ankommenden Nachrichten aus und führt die Kontrolle der Zeitrestriktionen durch. Dabei erstellen die Geräte auf denen die Subtransaktionen abschließen Prozessbeschreibungen des zugehörigen Kompensationsprozesses und übertragen diese zum Synchronisationsdienst. Dabei legt das Synchronisationsdienstmodul diese Prozessdefinitionen in seiner Registratur ab. Optional kann die Überprüfung der übergebenen Prozessbeschreibungen durch das Basismodul durchgeführt werden.

Ferner können alle Informationen, die zur globalen FST Transaktion gehören, erst dann gelöscht werden, wenn diese erfolgreich abgeschlossen oder zurückgesetzt wurde. Dabei kann eine globale Transaktion als erfolgreich betrachtet werden, wenn alle ihre Subtransaktionen in vorgegebener Zeit erfolgreich abgeschlossen wurden oder sich von der Ausführung abgemeldet haben. Wenn jedoch eine der Subtransaktionen scheitert oder die Zeitrestriktionen überschritten wurden, beginnt die dritte Phase der Transaktionsverwaltung im Synchronisationsdienst – die Kompensation.

Die Kompensation besteht dabei im Grunde aus der Weitergabe der gespeicherten Prozessdefinitionen an das Basismodul. Das Basismodul führt dann die Kompensationsprozesse aus oder leitet sie an kompetentere Geräte weiter.

5.2.3 Anpassungen des Basismoduls

Wie bereits in 5.2.1 erwähnt sind auch im Basismodul Anpassungen notwendig. Vor allem sieht der Entwurf vor, dass das Basismodul die Transaktionsgrenzen sowie die Aktivitäten innerhalb derer identifiziert. Bevor der Status einer transaktional geschützten Aktivität auf

executed wechselt und nachdem er wieder verlassen wird, übergibt das Basismodul die Kontrolle an das Transaktionsverwaltungsmodul. Dieses ermöglicht die Ausführung der Prozesse bzw. Navigation durch den Kontrollfluss, von der Transaktionssicherung zu trennen.

Weiterhin müssen bei der Transaktionsunterstützung verschiedene Dienstarten unterscheidbar sein (vgl. 3.6.3). Beispielsweise können an einer Transaktion mit diskreter Kompensation nur solche Dienste teilnehmen, die auch eine entsprechende Kompensationsoperation anbieten (vgl. 4.4.1). Bisher wurden die Dienstarten in der DEMAC-Middleware jedoch nicht unterschieden. Die Ausführung einer konkreten automatisierten Aktivität erfolgt über ein generisches *ServiceObject*. Codebeispiel 1 zeigt seine Schnittstelle, welche die Methode *execute* enthält. Die Methode *execute* repräsentiert dabei den Aufruf einer Geschäftsoperation am dahinter stehenden Dienst.

```
public interface ServiceObject {
    public DataField execute(FormalParameter[] fp, DataField[] df)
    public boolean exceptionOccured();
    public boolean connectionResetOccured();
}
```

Codebeispiel 1: Interface ServiceObject (aus [Zap05])

Weiterhin sind die Methoden *exceptionOccured* und *connectonResetOccured* zur Fehlerbehandlung vorhanden, wobei Verbindungsfehler von den anderen explizit unterschieden werden. Der hier vorgestellte Entwurf sieht vor, dass das *ServiceObject* erweitert wird. Neue Funktionalitäten werden durch ein zusätzliches Interface *CompensabeleServiceObject* (vgl. Codebeispiel 2) vorgegeben, welches das *ServiceObject* implementiert.

```
public interface CompensableServiceObject implements ServiceObject
{
    public DataField compensate(FormalParameter[] fp,
                        DataField[] df,
                        String uuid) throws IllegalStateException;
    public boolean isCompensable();
    public boolean getServiceUUID();
}
```

Codebeispiel 2: Interface CompensableServiceObject

Dieses Interface repräsentiert einen ungeschützten Dienst. Wobei mit Methode *isCompensable* abgefragt werden kann, ob der *ServiceObject* tatsächlich einen ungeschützten und damit kompensierbaren Dienst repräsentiert. Hinzu kommt die Methode *compensate*, welche zum Aufruf einer Kompensierungsoperation an einem ungeschützten Dienst genutzt werden kann. Bei einem falschen Gebrauch, wenn also der Dienst gar nicht kompensierbar ist, wird eine *IllegalStateException* geworfen. Außerdem erwartet die Methode *compensate* eine *UUID* (*Universal Unique Identifier*) des Dienstes. Welche bereits vorher, direkt vor dem Aufruf der Methode *execute*, mit Methode *getServiceUUID* abgefragt wird.

Beim Aufruf der Methode *compensate* sucht der DEMAC Context Service einen konkreten Dienst anhand der übergebener UUID. Dies sichert zu, dass die Kompensationsoperation an dem gleichen Dienst aufgerufen wird, an dem bereits vorher die zugehörige Geschäftsoperation aufgerufen wurde. Dabei müssen bei diskreter Kompensation die UUIDs der benutzten Dienste während der Transaktionsausführung vorgehalten werden.

5.3 Erweiterung des DPDL-Schemas

In [Zap05] wurde die *Demac Process Description Language* (*DPDL*) zur Definition verteilt ausgeführten Mobilen Prozesse entwickelt, wobei DPDL auf XPLD (vgl. [WfMC05]) basiert. Dabei werden bei DPDL einige zusätzliche Mechanismen eingeführt, welche die Leichtgewichtigkeit und Flexibilität der Prozessbeschreibung unterstützten sollen. Das DPDL-Meta-Modell ermöglicht die Beschreibung von Aktivitäten und Transitionen zwischen ihnen, wobei die Aktivitäten entweder atomar sind oder als Platzhalter für Subprozesse oder Folgen von Aktivitäten dienen können. Weiterhin definiert das DPDL-Schema eine spezielle Blockstruktur *Transaction*, durch die ein transaktionsgeschützter Prozessfluss abgebildet werden kann [Zap05].

Die Beschreibungsmöglichkeiten der DPDL reichen jedoch in der gegebenen Form für die Unterstützung der in Kapitel 4 konzipierten Transaktionsmodelle nicht aus. Im Folgenden werden, die Defizite des DPDL-Schemas in der Version 1.0[12], in Bezug auf die Transaktionsdefinition, genauer beschrieben. Dabei werden die vorgeschlagenen Erweiterungen des DPDL-Schemas dargestellt, welche die Abbildbarkeit von den konzipierten Transaktionsmodellen ermöglichen sollen.

In DPDL können Transaktionen durch mehrere Elemente beschrieben werden. Dabei werden die Transaktionen als spezielle transaktionale Aktivitäten gesehen, welche durch eine Blockstruktur *Transaction* repräsentiert werden. Dazu kann ein *Activity*-Element des

[12] Vgl. http://vsis-www.informatik.uni-hamburg.de/projects/demac/dpdl1.0.xsd

DPDL-Schemas, unter anderem, durch eine *TransactionActivity* repräsentiert werden (vgl. Schemaauszug 3).

```
<xsd:element name="Activity">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element ref="dpdl:Description" minOccurs="0"/>
			<xsd:choice>
				<xsd:element ref="dpdl:Route"/>
				<xsd:element ref="dpdl:Implementation"/>
				<xsd:element ref="dpdl:BlockActivity"/>
				<xsd:element ref="dpdl:TransactionActivity"/>
				<xsd:element ref="dpdl:LoopActivity"/>
			</xsd:choice>
			…
		</xsd:sequence>
		…
	</xsd:complexType>
</xsd:element>
```

Schemaauszug 3: Ausschnitt des *Activity* Elements, DPDL Meta-Modell (nach [Zap05])

TransactionActivity verweist ihrerseits jedoch auf das Element *Transaction* (vgl. Schemaauszug 4). In dieser Form bringt diese, durch *TransactionActivity* geschaffene, Indirektion jedoch keine Vorteile, denn in dem *TransactionActivity*-Element sind keine weiteren Attribute vorhanden, welche die Verbindung zwischen der Aktivität und der Transaktion parametrisieren könnten.

```
<xsd:element name="TransactionActivity">
	<xsd:complexType>
		<xsd:attribute name="TransactionId" type="xsd:string"
		use="required"/>
	</xsd:complexType>
</xsd:element>
```

Schemaauszug 4: *TransactionActivity* Element, DPDL Meta-Modell (nach [Zap05])

Das Element *Transaction* beschreibt die eigentliche Transaktion als einen ganzen Prozessausschnitt, der unter einer Transaktionskontrolle ausgeführt werden soll (vlg. Schemaauszug 5). Dabei enthält das *Transaction*-Element alle dazugehörigen Aktivitäten (*ActivityRefs*) und Transitionen (*Transitions*), welche den Prozessfluss innerhalb der Transaktion beschreiben. Ferner ist die Definition eines *Compensation*-Elementes möglich,

das die Kompensation beschreiben kann. Weiterhin ist die Transaktion durch die Attribute: *Id*, Name, *Type* sowie durch die *InitActivity* zur Angabe von Initialknoten und einer *Start Activity*, welche die aktuell auszuführende Aktivität festhält, beschrieben. Durch das *Retries*-Element ist es möglich zu definieren, wie oft beim Scheitern die Transaktion wiederholt werden soll. *ExtendedAttributes* dienen der Erweiterung der Sprache. Mit Hilfe dieser Konstrukte kann eine Transaktion als eine Art Subprozess bzw. als eine geschlossene Folge von Aktivitäten repräsentiert werden. Dabei wird die so definierte Transaktion letztendlich durch nur eine Aktivität dargestellt, so dass bei diesem Ansatz eine Transaktionsdefinition mit einer Kontrollflussdefinition vermischt wird.

```
<xsd:element name="Transaction">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="dpdl:Retries" minOccurs="0"/>
                  <xsd:element ref="dpdl:ActivityRefs" minOccurs="0"/>
                  <xsd:element ref="dpdl:Transitions" minOccurs="0"/>
                  <xsd:element ref="dpdl:Compensation" minOccurs="0"/>
                  <xsd:element ref="dpdl:ExtendedAttributes"
                              minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="Id" type="xsd:string" use="required"/>
            <xsd:attribute name="StartActivity" type="xsd:string"
                  use="required"/>
            <xsd:attribute name="InitActivity" type="xsd:string"
                  use="required"/>
            <xsd:attribute name="Name" type="xsd:string"/>
            <xsd:attribute name="Type" type="xsd:string"/>
      </xsd:complexType>
</xsd:element>
```

Schemaauszug 5: *Transaction* Element, DPDL Meta-Modell (nach [Zap05])

Diese Modellierungsweise ist jedoch nicht flexibel genug und widerspricht der Anforderung T4 (Mächtigkeit der Transaktionsdefinition) (vgl. 3.2.3), wonach die Transaktionsdefinition möglichst mächtig und flexibel sein sollte. Die Modellierung von ganzen Transaktionen „in einer Prozessaktivität“ verfehlt demnach den eigentlichen Nutzen einer Transaktionsunterstützung für Mobile Prozesse. Eine effektive Transaktionsunterstützung sollte einem bereits modellierten Kontrollfluss einen transaktionalen Schutz bieten und nach Möglichkeit den Kontrollfluss selbst nicht beeinflussen. So dass eine Entkopplung nicht funktionale Aspekte des Transaktionsschutzes von der fachlichen Aspekten der Kontrollflussmodellierung möglich wird. Die in Kapitel 4 vorgestellten Transaktionsmodelle erfüllen diese Anforderungen.

Aus diesem Grund können die bereits vorhandenen Elemente nicht ohne Anpassungen verwendet werden. Dabei reicht eine einfache Aufzählung, der in einer Transaktion beteiligten Aktivitäten, aus, um dadurch die Transaktionsgrenzen zu markieren zu können. Das bedeutet, dass man die Indirektion und die Verbindung von Aktivität zur Transaktion nicht braucht, so das *TransactionActivity*-Element verworfen werden kann. Auch das *Transaction* Element muss entsprechend angepasst werden.

Die Anzahl der Wiederholungsversuche *Retries* sowie das Element *ActivityRefs* werden weiterhin gebraucht. Allerdings braucht man in der Transaktion keine *Transitionen*, da die Transaktionsausführung den eigentlichen Prozess nicht ausführt, sondern nur absichert. Aus demselben Grund können auch die als „required" definierten Attribute, *InitActivity* und *StartActivity*, nicht übernommen werden. Das daraus resultierende Element *Transaction* ist im Schemaauszug 6 zu sehen.

Dabei ist zu erkennen, dass auch einige Elemente und Attribute hinzugefügt oder umbenannt wurden. So wurde beispielsweise das *Name*-Attribut zur *Description* umbenannt, denn ein Name ist für eine Transaktion nicht relevant, die Identifikation der Transaktion ist bereits durch das Attribut *Id* gewährleistet. Außerdem hat das angepasste Element *Transaction* einen Attribut *URI*, welches einen URI eines Synchronisationsdienstes (oder eines Koordinators) angibt. Attribute *initTime* und *maxDuration* werden zur Angabe von Zeitrestriktionen benötigt. Dabei gibt das Attribut *maxDuration*, welches mit dem Typ *xsd:duration* definiert ist, die eigentliche maximale Dauer der Transaktionsabwicklung an. Das Attribut *initTime* wird dabei während der Ausführung, am Beginn der Transaktion gesetzt. Durch die Gegenüberstellung der Differenz beider Attribute mit der aktuellen Zeit kann die Transaktionsdauer überprüft werden. Das Attribut *StrategyId* kann zur defintion von nichtfunktionalen Aspekten der Transaktion benutzt werden. So können z.B. Kriterien für die Auswahl eines Synchronisationsdienstes definiert werden.

Das hinzugekommene Attribut *BackoutProtected* wird bei den Sequenziellen Transaktionen mit der Erweiterung S1, sowie bei den Join Transaktionen gebraucht. Falls dieses Attribut den Booleschen Wert „true" aufweist, bedeutet dies, dass die Aktivitäten außerhalb der Transaktion nur dann gestartet werden, wenn die Transaktion erfolgreich ausgeführt wurde. Vordefiniert ist dabei der Wert „false", der auch dann gilt, wenn dieses Attribut nicht explizit angegeben wird. In diesem Fall kann der Kontrollfluss die Transaktion auch vor einem erfolgreichen Ende verlassen.

```
<xsd:element name="Transaction">
      <xsd:complexType>
            <xsd:sequence>
                  <xsd:element ref="dpdl:ActivityRefs" minOccurs="1"/>
                  <xsd:element ref="dpdl:Compensation"
                        minOccurs="0" maxOccurs="unbound"/>
                  <xsd:element ref="dpdl:Transaction"
                        minOccurs="0" maxOccurs="unbound"/>
                  <xsd:element ref="dpdl:TransactionStartSet"
                        minOccurs="0" maxOccurs="1"/>
                  <xsd:element ref="dpdl:ExtendedAttributes"
                        minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="Id" type="xsd:string" use="required"/>
            <xsd:attribute name="StrategyId" type="xsd:string"/>
            <xsd:attribute name="Description" type="xsd:string"/>
            <xsd:attribute name="Type" type="xsd:TransactionType"
                  use="required"/>
            <xsd:attribute name="URI" type="xsd:anyURI"/>
            <xsd:attribute name="ForwardRetries"
                  type="dpdl:PositiveInteger"/>
            <xsd:attribute name="Retries" type="dpdl:PositiveInteger"/>
            <xsd:attribute name="CompensationType"
                  type="dpdl:CompensationType"/>
            <xsd:attribute name="initTime" type="xsd:dateTime"/>
            <xsd:attribute name="maxDuration" type="xsd:duration"/>
            <xsd:attribute name="BackoutProtected" type="xsd:boolen"/>
      </xsd:complexType>
</xsd:element>
```

Schemaauszug 6: Das angepasste Element *Transaction*

Weiterhin wurde das Element *Transaction* so erweitert, dass es einen Element *TransactionStartSet* und beliebig viele Elemente *Transaction* enthalten kann. Ein *Transaction* Element kann dadurch andere *Transaction* Elemente einschließen, wodurch die Modellierung von geschachtelten Transaktionen möglich wurde. Das Element TransactionStartSet kann zur Speicherung eines Start-Sets verwendet werden und wird am Ende dieses Abschnitts genauer erläutert.

Zur Kennzeichnung des Transaktionstyps wurde das Attribut *Type* in seinem Wertebereich von beliebigen, auf die in Kapitel 4 dargestellten Transaktionsmodelle, eingeschränkt. Dazu wurde ein *TransactionType* explizit mit den gültigen Werten *SEQUENCE*, *JOIN*, *FST*

und *ATOMIC* definiert (vgl. Schemaauszug 7), welche entsprechend für Sequenzielle, Join, Flexible Synchronisierte Transaktionen und eine Atomare Sphäre stehen.

```
<xsd:simpleType name="TransactionType">
	<xsd:restriction base="xsd:NMTOKEN">
		<xsd:enumeration value="SEQUENCE"/>
		<xsd:enumeration value="JOIN"/>
		<xsd:enumeration value="FST"/>
		<xsd:enumeration value="ATOMIC"/>
	</xsd:restriction>
</xsd:simpleType>
```

Schemaauszug 7: Definition von *TransactionType*

Zur Behandlung der Kompensation wurde das Element *Transaction* um ein Attribut *CompensationType* erweitert, welches durch einen eigenen Typ repräsentiert wird, dieser ist im Schemaauszug 8 zu sehen.

```
<xsd:simpleType name="CompensationType">
	<xsd:restriction base="xsd:NMTOKEN">
		<xsd:enumeration value="GLOBAL"/>
		<xsd:enumeration value="DISCRETE"/>
		<xsd:enumeration value="RERUN"/>
	</xsd:restriction>
</xsd:simpleType>
```

Schemaauszug 8: Definition von *CompensationType*

Kompensation findet immer auf der Ebene einer Sequenziellen Transaktion statt, deswegen muss dieses Attribut bei den übrigen Transaktionsarten nicht angegeben werden. Im Falle der globalen Kompensation (*GLOBAL*) wird der Kompensationsvorgang bereits zum Modellierungszeitpunkt festgelegt. Bei einer diskreten Kompensation (*DISCRETE*) wird das *Compensation*-Element erst zur Laufzeit dynamisch erzeugt. Die Definition des *CompensationTypes* lässt jedoch auch den Wert *RERUN* zu. Dies ist eine weitere Möglichkeit der Behandlung eines Transaktionsabbruchs, wobei keine Kompensation stattfindet, sondern die Transaktion einfach neu gestartet wird.

Schließlich wurde das Element *Retries* zu einem Attribut umdefiniert. Außerdem wurde die Steuerung der Wiederholungsversuche um ein optionales Attribut *ForwardRetries* erweitert, welches die Anzahl der Forward Recovery Versuche bei einer Sequenziellen Transaktion angibt. Der Typ der Attribute *Retries* und *ForwardRetries* ist eine positive Ganzzahl, die durch den Typ *PositiveInteger* definiert ist (vgl. Schemaauszug 9).

```
<xsd:simpleType name="PositiveInteger">
	<xsd:restriction base="xsd:integer">
		<xsd:minInclusive value="0"/>
	</xsd:restriction>
</xsd:simpleType>
```

Schemaauszug 9: Definition von *PositiveInteger*

Ein *Transaction*-Element kann ein *TransactionStartSet*-Element besitzen, welches bei der Transaktionsausführung zur Abspeicherung des Start-Sets benutzt werden kann (vgl. Schemaauszug 6). Das Start-Set einer Transaktion beinhaltet die Eingangsparameter der Aktivitäten, die einen oder mehrere außerhalb der Transaktion eingehende Knoten besitzen. Wenn eine Sequenzielle Transaktion wiederholbar ist (Attribut *Retries* mit einem Wert größer als null), sollte das Start-Set stets gesichert werden, so dass die Wiederholung mit gültigen Werten stattfinden kann. Das Start-Set wird mit vielen anderen Prozessparametern in der Prozessdefinition festgehalten, so dass Prozesse auch nach Migration vollständig bleiben. Die zur Definition des Start-Sets benötigen Elemente sind im Schemaauszug 10 dargestellt.

```
<xsd:element name="TransactionStartSet">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element ref="dpdl:ActivityDataSet"
				minOccurs="0" maxOccurs="unbound"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:element>
<xsd:element name="ActivityDataSet">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element ref="dpdl:DataField"
				minOccurs="0" maxOccurs="unbound"/>
		</xsd:sequence>
	</xsd:complexType>
	<xsd:attribute name="ActivityRefId" type="xsd:string"
				use="required"/>
</xsd:element>
```

Schemaauszug 10: Elemente *TransactionStartSet* und *ActivityDataSet*

Nach dieser Definition kann das Element *TransactionStartSet* eine beliebige Menge von Elementen beinhalten. Das *ActivityDataSet*-Element enthält die eigentlichen Parameter der Aktivitäten, welche durch das DPDL-Element *DataField* repräsentiert werden. Außerdem verweist das *ActivityDataSet*-Element durch das *ActivityRefId*-Attribut auf die zugehörige Aktivitätsreferenz *ActivityRef.*

Zum Schluss wird auch das *ActivityRef*-Element selbst in seinen Attributen angepasst. Um die in 5.2.1 beschriebene Erweiterung des Lebenszyklus der Aktivitäten zu ermöglichen, wird das Attribut *State* um den entsprechenden möglichen Wert *LOCKED* erweitert. Die für die diskrete Kompensation notwendige Aufbewahrung der UUID Dienste wird in einem optionalen Attribut *usedUUID* abgelegt.

Mit diesen angepassten Elementen können alle spezifizierten Transaktionen modelliert werden. Dabei müssen Modellierer bzw. die Modellierungswerkzeuge die transaktionsspezifischen Kontrollflussregeln einhalten. Dieses gilt insbesondere für Kontrollstrukturen, abhängige Sequenzielle und Join Transaktionen sowie Atomare Sphären.

6 Schlussbetrachtung

Im Rahmen dieser Arbeit wurden die Anforderungen an eine Transaktionsunterstützung in verteilt ausgeführten Mobilen Prozessen ausgearbeitet. Anhand dieser wurden verschiedene Transaktionskonzepte auf ihre Eignung für den Einsatz in Mobilen Prozessen analysiert. Diese Untersuchung ergab, dass keines der analysierten Konzepte für einen direkten Einsatz in Mobilen Prozessen geeignet ist. Es hat sich dabei ebenfalls herausgestellt, dass die Kompensationsbasierten Transaktionskonzepte, wie zum Beispiel Sagas oder die Kompensationssphäre als Basis für die Entwicklung der geeigneten Transaktionsmodelle dienen können.

Auf dieser Grundlage wurden drei Transaktionsmodelle mit ihren Erweiterungen vorgestellt: Sequenzielle Transaktion, Join Transaktion und Flexibel Synchronisierte Transaktion. Zur Umsetzung der Transaktionsunterstützung, basierend auf diesen Modellen, wurden zwei Erweiterungsmodule der Basiskomponenten des DEMAC Process Services entworfen und vorgestellt. Außerdem wurde die Prozessbeschreibungssprache an die Besonderheiten der vorgestellten Transaktionen angepasst. Eine komplette Betrachtung aller Aspekte der Transaktionsunterstützung ist jedoch aufgrund ihres Umfanges im Rahmen der vorliegenden Diplomarbeit nicht möglich. In diesem Kapitel werden daher weitere Entwicklungsmöglichkeiten aufgezeigt sowie das Ergebnis der Arbeit zusammengefasst.

6.1 Ausblick

Das entwickelte Transaktionskonzept sichert die Ausführung der verteilt ausgeführten Mobilen Prozesse. Dabei könnten die vorgestellten Verfahren und Komponenten in vielerlei Hinsicht erweitert werden.

Um die Zuverlässigkeit der Ausführung der Kompensationsprozesse zu erhöhen, könnte das Synchronisationsdienstmodul die Kompensationsprozesse selbst in eine globale Flexible Synchronisierte Transaktion einschließen und sich dabei als Synchronisationsdienst anführen. Dadurch wird der Synchronisationsdienst implizit über den Verlauf des von ihm gestarteten Kompensationsprozesses informiert. Bei dessen Scheitern kann er zum Beispiel den gesamten Prozess oder nur seine gescheiterten Teile wiederholen, was von der Strategie und der Transaktionsstruktur abhängt. Das Synchronisationsdienstmodul müsste dazu um einen effizienten Mechanismus der automatischen Modellierung von Transaktionsgrenzen erweitert werden. Weiterhin müssten die Wiederholungsstrategien genauer spezifiziert werden.

Außerdem könnte der Synchronisationsdienst selbst verteilt realisiert werden. Dieses würde möglicherweise seine Erreichbarkeit in einer unzuverlässigen und dynamischen mobilen Umgebung verbessern. Dabei müssen die verschiedenen Verteilungsszenarien auf ihre Eignung überprüft werden. Beispielsweise wäre ein dynamischer Verbund von Synchronisationsdiensten, in welchen die einzelne Synchronisationsdienste ihre internen Verwaltungsinformationen unter einander austauschen, denkbar.

Zudem könnten auch Erweiterungen für spezifische Einsatzbereiche entwickelt werden. Bei einem Einsatz der Mobilen Prozesse in einer Umgebung mit datenzentrischen, geschützten Diensten, wo Kompensationsverfahren möglicherweise nicht einsetzbar sind, wäre zu untersuchen, ob eine Atomare Sphäre, in Verbindung mit einer temporalen Aussetzung der Migrationsfähigkeit, zur Transaktionssicherung geeignet wäre.

6.2 Zusammenfassung und Ergebnis

Mobile Prozesse haben zum Ziel die Fähigkeiten der mobilen Geräte durch ihre Kooperation untereinander zu erweitern. Komplexe Aufgaben können durch Mobile Prozesse abgebildet werden und in einer mobilen Umgebung verteilt ausgeführt werden, wobei die beteiligten Geräte den, ihren Fähigkeiten entsprechenden, Beitrag leisten können. Dabei werden mobile Geräte und Netze, durch den technischen Fortschritt getrieben, immer leistungsfähiger. Es kann angenommen werden, dass diese Entwicklung auch in naher Zukunft anhalten wird, so dass Mobile Prozesse immer komplexere Aufgaben übernehmen können.

Verteilt ausgeführte Mobile Prozesse sollen, wie ihre stationären Pendants, transaktional geschützt ablaufen können. In dieser Arbeit wurde dazu ein Konzept entwickelt, das eine flexible, erweiterbare Transaktionssicherung bietet. Das Transaktionssicherungskonzept sieht dabei drei Transaktionsmodelle vor, wobei Sequenzielle Transaktion die Grundlage der entwickelten Transaktionsunterstützung darstellt. Sie lässt zwar keine parallelen Abläufe bei der Transaktionsausführung zu, dafür kann sie mit nur einem geringen Mehraufwand für die Prozessausführung realisiert werden. Kennzeichnend für die Sequenzielle Transaktion ist, dass bei ihrer Ausführung keine Synchronisation zwischen den Geräten notwendig ist. Die Join sowie die Flexible Synchronisierte Transaktionen basieren auf der Sequenziellen Transaktion und haben keine eigenen Kompensationsmechanismen. Während eine Join Transaktion eingeschränkt die Parallelität bei der Transaktionsausführung zulässt, ermöglicht die Flexible Synchronisierte Transaktion die flexibelste Variante der Transaktionsdefinition und ihrer Ausführung. Die Flexible Synchronisierte Transaktion setzt jedoch die Anwesenheit eines Synchronisationsdienstes voraus.

Im Zuge der Entwicklung der Transaktionsmodelle wurden auch Software Komponenten entworfen, welche die DEMAC Middleware um die Transaktionsunterstützung erweitern sollen. Das Transaktionsverwaltungsmodul ist die Hauptkomponente und ermöglicht die Ausführung aller Transaktionen. Zudem wird das Synchronisationsdienstmodul zusätzlich zum Beispiel für die Unterstützung der Flexiblen Synchronisieren Transaktion gebraucht. Außerdem wurde das DPDL-Schema um spezielle Konstrukte zur Transaktionsdefinition sowie zur Speicherung des Transaktionsausführungsstatus erweitert.

Die entwickelte Transaktionsunterstützung erlaubt es, die Transaktionen unabhängig von der Definition des Kontrollflusses zu beschreiben. Für die Transaktionsmodellierung stehen die drei Transaktionsarten mit ihren spezifischen Eigenschaften zur Auswahl, wobei die Modelle auch kombiniert zum Einsatz kommen können. Die meisten der aufgestellten Anforderungen werden von den einzelnen Transaktionsmodellen erfüllt, so ermöglichen Sequenzielle Transaktionen maximale Effizienz, während die Flexiblen Synchronisierten Transaktionen die größte Modellierungsflexibilität bieten und Join Transaktionen etwas von beidem haben. Durch die entsprechende Kombination der Modelle kann eine zielgerechte Transaktionsunterstützung modelliert werden.

Zusammenfassend lässt sich festhalten, dass verteilt ausgeführte Mobile Prozesse grundsätzlich mit einer Kompensationsbasierten Transaktionsunterstützung erweitert werden können. Ein flexibler, effizienter und den unterschiedlichen Leistungsfähigkeiten der mobilen Geräte angepasster Transaktionsschutz kann durch eine geschickte Kombination der verschiedenen Transaktionsmodelle erreicht werden.

Literaturverzeichnis

[Aal03] Van der Aalst, W.: Patterns and XPDL: *A critical evaluation of the xml process definition language*, Technical Report FIT-TR.2003-06, Queensland University of Technology, 2003.

[AaHe04] Van der Aalst, W, Van Hee, K.: *Workflow Management: Models, Methods, and Systems*, The MIT Press Cambridge, Massachusetts, London, ISBN:0262011891, 2004.

[All06] Allen, P.: Service Orientation: Wining Strategies and Best Practices, Cambridge university Press, New York, ISBN:0521843367, 2006.

[ACD+03] Andrews, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Tricovick, I.; Weerawarana, S.: *Business Process Execution Language for Web Services*, Specification 1.1, IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2003.

[ACKM04] Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V.: *Web Services: Concepts, Architectures and Applications*, Springer Verlag, Berlin Heidelberg, ISBN:3540440089, 2004.

[Ben04] Bengel, G.: *Verteilte Systeme*, Auflage 3, Vieweg, ISBN: 3528257385, 2004.

[BeNe97] Bernstein, P. A.; Newcomer, E.: *Principles of Transaction Processing*, Morgan Kaufman Publischers Inc, San Francisco, ISBN:1558604154, 1997.

[BiNe84] Birrell, A. D.; Nelson, B. J.: *Implementing Remote Procedure Calls* in *ACM Transactions on Computer Systems*, 2(1), Seiten 29-59, 1984.

[Bro96] Brown, P. J.: *The stick-e document: a Framework for Creating Context-Aware Applications*, in *Proceedings of electronic Publishing 96,* Seiten 259-272, 1996.

[CiRu04] Chichocki, A.; Rusinkiewicz, M.: *Providing Transactional Properties for Migrating Workflows.* In *Mobile Networks and Applications 9*, Kluwer Academic Publischers, Seite 473-480, 2004.

[CEM02] Capra,L.; Emmerich, W.; Mascolo, C.: *Middleware for Mobile Computing* in Gregori, E.; Anastasi, G.; Basagni, S: Networking 2002 Tutorial Papers, Volume 2497, Seiten 20-58, 2002.

[CES71] Coffman, E. G.; Elphick, M.; Shoshani, A.: *System Deadlocks* in *ACM Computing Surveys (CSUR)*, Band 3, Seiten 67-78, ACM Press, New York, ISSN:03600300, 1971.

[Dav78] Davies, C. T.: *Data processing spheres of control*, IBM Systems Journal, 17(2), Seiten: 179-198, 1978.

[DeAb00] Dey, A. K.; Abowd, G. D.: *Towards a Better Understanding of Context and Context-Awareness*, in *Proceedings of the CHI 2000 Workshop on "The What, Who, Where, When, Why and How of Context-Awareness"*, Georgia Institute of Technology, http://www.smartech.gatech.edu/bitstream/1853/3464/5/00-18e.pdf (Abruf: 01.02.2006), 2000.

[DHB97] Dunham, M. H.; Helal, A.; Balakrishnan, S.: *A Mobile Transaction Model That Captures Both the Data and Movement Behavior*, in *Mobile Networks and Applications*, 2(2), Seiten 149-162, 1997.

[DJMZ05] Dostal, W.; Jeckle, M; Melzer, I; Zengler, B.: Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis, Elsevier GmbH, München, ISBN:3827414571, 2005.

[DuKu99] Dunham, M. H.; Kumar, V: *Impact of Mobility on Transaction Management* in *Procceedings of the 1th ACM international workshop on Data engineering for wireless and mobile access*, Seiten 14-21, Seattle, Waschington, ISBN:1581131755, 1999.

[ElDu98] Elmagarmid, A.; Du, W.: Workflow Management: *State of the Art Versus State of the Products* in Dogac, A.; Kalinichenko, L.; Özsu M. T.; Sheth, A. (Herrausgeber:), Workflow Management Systems and Interoperability, Seiten: 1-17, Springer, Berlin, ISBN:3540644113, 1998.

[Emm00] Emmerich, W.: *Software Engineering and Middleware: A Roadmap*, in The Future of Software Engineering – 22nd International Conference on Software Engineering (ICSE2000), ACM Press, Seiten 117-129, 2000.

[Fai85] Fairley, R.: *Software Engineering Concepts.* McGraw-Hill Companies, ISBN:0070199027, 1985.

[FoZa94] Forman, G. H.; Zahorjan J.: *The Challenges of Mobile Computing*, IEEE Computing 25(4), Seiten 38-47, 1994.

[GeTs98] Georgakopoulos, D.; Tsalgatidou, A.: *Technology and Tools for Comprehensive Business Process Lifecycle Management*, in Dogac, A.; Kalinichenko, L.; Özsu M. T.; Sheth, A. (Herrausgeber:), Workflow Management Systems and Interoperability, Seiten: 356-395, Springer, Berlin, ISBN:3540644113, 1998.

[GMS87] Garcia-Molina, H.; Salem, K.: Sagas, in *Proceedings of the ACM Conference on Management of Data*, Seiten 249-259, 1987.

[Gra81] Gray, J.: *The Transaction concept: Virtues and limitations*, in *Proceedings of the International Conference on Very Large Data Bases*, Seiten 144-154, Cannes, 1981.

[GrRe93] Gray, J.; Reuter, A.: *Transaction Processing: Concepts and technique,* Morgan Kaufman Publishers, Inc., San Mateo, ISBN:1558601902, 1993.

[HaCh04] Hammer, M.; Champy, J.: *Reengineering the Corporation: Amanfesto for Business Revolution*, HarperCollins Publishers, New York, ISBN:0060559535, 2004.

[HaRe83] Haerder, T.; Reuter, A.: *Principles of Transaction-Oriented Database Recovery* in *ACM computing Surveys*, 15(4), Seiten: 287-317, ACM Press, New York, ISSN:03600300, 1983.

[HTK05] Höpfner, H.; Türker, C.; König-Ries, B.: *Mobile Datenbanken und Informationssysteme*, Dpunkt Verlag, ISBN:389864264, 2005.

[ImBa94] Imielinski, T.; Badrinath B. R.: *Mobile Wireless Computing: Chalenges in Data Management*, in Comunications of ACM 37(10), Seiten 18-28, ACM Press, New York, ISSN:00010782, 1994.

[ISO95] International Standarts, *Open Distributed Processing reference Model*, ISO/IEC IS 10746, Seite 5, 1995.

[KLS90] Korth, H. F.; Levy, E.; Silberschatz, A.: *A Formal Apporach to Recovery by Compensating Transactions*, Technical Report, UMI Order Number: CS-TR-90-14, University of Texas at Austin, 1990.

[Kun05] Kunze, C. P.: *Unterstützung mobiler Prozesse im Mobile Computing,* in *Techischer Bericht zum 1. GI/ITG KuVS Fachgespräch Energiebewusste Systeme und Methoden,* Seiten 42-47, 2005.

[Kun05a] Kunze, C. P.: DEMAC: *A Distributed Environment for Mobility-Aware Computing*, in Ferscha, A.; Mayrhofer, R.; Strang, T.; Linnhoff-Popien, C.; Dey, A.; Butz,A.; Schmidt A. (Herrausgeber:), *Adjunct Proceedings of the Third International Conference on Pervasive Computing,* Seiten 115-121, Oesterreichische Computer Gesellschaft, 2005.

[KZL06] Kunze, C. P.; Zaplata, S.; Lamersdorf, W.: *Mobile Process Description and Execution,* in *Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems,* Springer Verlag, 2006.

[KZL07] Kunze, C. P.; Zaplata, S.; Lamersdorf, W.: *Abstrakte Dienstklassen zur Realisierung mobiler Prozesse*, in *Proceedengs of Kommunkation in Verteilten Systemen* (KiVS), Gesellschaft für Informatik, 2007.

[LeRo99] Leyman, F.; Roller, D.: *Production workflow: concepts and techniques,* Prentice Hall PTR, Upper Saddle River, ISBN:0130217530, 1999.

[Ley95] Leymann, F.: *Supporting Business Transactions via Partial Backward Recovery,* in *Workflow Management Systems*, in *Proceedings of the BTW*, Seiten 51-70, Springer Verlag, Berlin, 1995.

[Ley96] Leymann, F.: *Workflows make objects really useful,* in Proceedings of the 6th Internal workshop on High Perfomance Transaction Systems, Asilomar, 1996.

[Ley97] Leyman, F.: *Transaktionsunterstützung für Workflows*, in Informatik: Forschung und Entwicklung, Band 12, Srpinger Verlag, Berlin Heidelberg, 1997.

[Lum05] Mumio, M.: *Telekomunikation in Europa.* In *Industrie Handel und Diensleistungen* 8/2005, EUROSTAT, ISSN: 15614832, http://epp.eurostat.cec.eu.int/portal/page?_pageid=1073, 46587259&_dad=portal&_schema=PORTAL&p_product_code=KS-NP-05-008 (Abruf: 24.05.2006), 2005.

[MLMB+06] MacKenize, C. M.; Laskey K.; McCabe, F.; Brown, P. F.; Metzt, R.; Hamilton, B. A.: *Reference Model for Service OrientedArchitecture 1.0,* OASIS Committee Specification 1, 2006.

[Mos81] Moss, J.E.B.: *Nested Transactions: An approach to reliable Computing*. MIT, 1981.

[Mül02] Müller-Wilken, S.: Mobile Geräte in verteilten Anwendungsumgebungen, Dissertation am Fachbereich Informatik, Universität Hamburg, http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/312/Diss_M%FCller-Wilken.pdf (Abruf: 30.01.2007), 2002.

[NoMa02] Norin, R.; Marin, M.: Workflow Process Definition Interface – xml process definition language, Specification WFMC-TC-1025, workflow Management Coalition, 2002.

[Pan99] Pandya, R.: *Mobile and Perosnal Comunication Systems and Services*, IEEE Press series on digital & mobile communication, New York, ISBN 0780347080, 1999.

[Pap03] Papazoglou, M. P.: *Service-Oriented Computing: Concepts, Characteristics and Directions* in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Seite 3, IEEE Computer Society, Washington, 2003.

[Pop98] Pope, A.: *The Corba Reference Guide: Understanding the Common Object Request Broker Architecture*, Addison-Wesley, Boston, 1998.

[Rie05] Von Riegen, M.: Transactionale *Koordination dynamischer Prozesse in Grid-Umgebungen,* Diplomarbeit, Universität Hamburg, 2006.

[Rot02] Roth, J.: *Mobile Computing: Grundlagen, Technik, Konzepte.* Dpunkt Verlag, Heidelberg, ISBN:3898641651, 2002.

[Rot05] Roth, J.: *Mobile Computing: Grundlagen, Technik, Konzepte.* Dpunkt Verlag, Heidelberg, Auflage 2, ISBN:3898643662, 2005.

[RuSh95] Rusinkiewicz, M.; Sheth, A.: *Specification and Execution of Transactional Workflows*, Modern database systems: the object model, interoperability, and beyond, Seiten 592-620, ACM Press /Addison-Wesley Publishing Co., New York, ISBN:0201590980, 1995.

[Sat96] Satyanarayanan, M.: *Fundamental Challenges in Mobile Computing in Symposium on Principles of Distributed Computing.* ACM Press, New York, http://www-2.cs.cmu.edu/afs/cs/project/coda-www/ResearchWebPages/docdir/podc95.pdf (Abruf:16.11.2006), 1996.

[ShRu93] Sheth, A.; Rusinkiewicz, M.: *On Transactional Workflows* in *Bulletin of the IEEE Technical Committee on Data Engineering*, 16(2), Seiten 37-40, 1993.

[SNKP94] Satyanarayanan, M.; Noble, B.; Kumar, P.; Price, M.: *Application-Aware Adaptation for Mobile Computing*, in *Proceedings of the 6th ACM SIGOPS in Dagstuhl*, Carnegie Mellon University, 1994.

[TaSt02] Tanenbaum, A. S.; Van Steen, M.: *Distributed Systems: Principles and Paradigms*, Prentice Hall, Upper Saddle River, ISBN: 0130888931, 2002.

[Tur06] Turjalei, M.: *Integration von Context-Awareness in eine Middleware für mobile Systeme* Diplomarbeit, Universität Hamburg, http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/407/TurjaleiDiplomarbeit.pdf (Abruf: 14.02.07), 2006.

[Wei91] Weiser, M.: The computer for the Twenty-First Century, Scientific American, 265 (3), Seiten 94-104, 1991.

[Wei93] Weiser, M.: *Ubiquitos Computing*, in *IEEE Computer Hot Topics*, 1993.

[WfMC99] WfMC Workflow Management Coaliton: Terminology & Glossary, Ausgabe 3, Workflow Management Coalition, http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf (Abruf 12.02.2007), 1999.

[WfMC05] WfMC Workflow Management Coaliton: *Workflow Managemen Coalition Workflow Standard: Process Definition Interface – XML Process Defintion Language 1.13 Final*, Workflow Managemen Coalition, http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-09-07_xpdl_2.pdf, 2005.

[Zap05] Zaplata, S.: *Prozessintegration in Middleware für Mobile Systeme* Diplomarbeit, Universität Hamburg, http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/305/Diplomarbeit_ZA_041005.pdf (Abruf: 14.02.06), 2005.

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Codebeispiele

Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Außerdem erkläre ich, dass ich mit der Einstellung dieser Diplomarbeit in den Bestand der Bibliotheken der Universität Hamburg einverstanden bin.

Hamburg, den

[Alexander Holbreich]