Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Verteilte Systeme und Informationssysteme

Diplomarbeit

April 2009

Realisierung von erweiterten Transaktionskoordinatoren auf Basis von WS-Coordination

Stefan Fink

3fink@informatik.uni-hamburg.de
Studiengang Informatik
Matr.-Nr. 5288669

Erstgutachter: Prof. Dr.-Ing. Norbert Ritter
Zweitgutachter: Dr. Axel Schmolitzky

„The best thing about standards is that there are so many to choose from.“
– Andrew Tanenbaum

Inhaltsverzeichnis

Abkürzungsverzeichnis

2PC	Two-Phase Commit Protocol
AP	Application Program
API	Application Programming Interface
B2B	Business-to-Business
BACC	BusinessAgreementWithCoordinatorCompletion, Protokoll von WS-BA
BAPC	BusinessAgreementWithParticipantCompletion, Protokoll von WS-BA
BPEL	Business Process Execution Language, auch WS-BPEL oder BPEL4WS
BTP	Business Transaction Protocol
CRM	Communication Resource Manager
Ctx	Koordinationskontext
DBS	Database System
DTP	Distributed Transaction Processing
EPR	Enpoint Reference
HTTP	Hypertext Transfer Protocol
IRI	Internationalized Resource Identifier
MAP	Message Addressing Properties
MDBS	Multidatabase System
OASIS	Organization for the Advancement of Structured Information Standards
QoS	Quality of Service
RM	Resource Manager
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol (vor Version 1.2)
Sub-TA	Sub-Transaction
TA	Transaktion
TL-TA	Top-Level-Transaction
TM	Transaction Manager
TX	Schnittstelle zwischen AP und TM bei X/Open DTP
UBR	UDDI Business Registry
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WS	Web-Service
WS-ACID	Web Services ACID Specification, Teil von WS-TXM
WS-AT	Web Services AtomicTransaction
WS-BA	Web Services BusinessActivity

WS-BA-I	Web Services BusinessActivity Initiator
WS-BP	Web Services Business Process Specification, Teil von WS-TXM
WS-C	Web Services Coordination
WS-CAF	Web Services Application Composite Framework
WS-CDL	Web Services Choreography Description Language
WS-CF	Web Services Coordination Framework Specification, Teil von WS-CAF
WS-Context	Web Services Context Specification, Teil von WS-CAF
WS-LRA	Web Services Long Running Action Specification, Teil von WS-TXM
WS-TX	Web Services Transactions, Sammelbegriff für WS-C, WS-AT und WS-BA
WS-TXM	Web Services Transaction Management, Teil von WS-CAF
WSDL	Web Services Description Language
XA	Schnittstelle zwischen TM und RM bei X/Open DTP
XA+	Schnittstelle zwischen TM und CRM bei X/Open DTP
XML	Extensible Markup Language

1 Einleitung

Unternehmensprozesse lassen sich mit Hilfe betrieblicher Informationssysteme als Workflows modellieren. Immer häufiger bilden dabei *serviceorientierte Architekturen* (SOA) auf Basis von *Web-Services* (WS) die technische Grundlage für eine Umsetzung. Dies begründet sich durch die Verfügbarkeit von freien, allgemein akzeptierten Standards wie XML [BPSM+06], SOAP [GHM+03a], WSDL [CCMW01] und UDDI [CHRR04], die eine herstellerübergreifende und plattformunabhängige Realisierung gewährleisten. Darüber hinaus wird durch die lose Kopplung der autonomen Dienste ein hoher Grad an Modularität und Flexibilität gewonnen.

Die Komposition von Web-Services zu aggregierten Anwendungen erfordert neben der Dienstnutzung vor allem die Festlegung einer Abfolge für Service-Aufrufe, um auf diese Weise eine Prozesslogik zu verwirklichen. Die Ansätze zur Realisierung sind hierbei geprägt durch die beiden Paradigmen *Orchestrierung* und *Choreographie* [Pel03]. Das Konzept der Orchestrierung steht für die Integration von Services zu einem ausführbaren Prozess, der sich mit ablauffähigen Beschreibungssprachen formulieren und in einer Workflow-Engine zur Ausführung bringen lässt. Im Zentrum steht eine Ablaufumgebung, die die Abwicklung des Prozesses sowie den Nachrichtenfluss steuert. Auf Grund der Orchestrierung durch eine Ablaufumgebung, wird diese Art der Komposition meist innerhalb der Grenzen einer Organisation eingesetzt, da hier Vertrauensaspekte eine tragende Rolle spielen. Im Gegensatz zur Orchestrierung verfolgt der Ansatz der Choreographie eine unternehmensübergreifende Kollaboration von Diensten zur Erlangung eines gemeinsamen Ziels. Eine Choreographie beschreibt den Nachrichtenaustausch zwischen autonomen, gleichberechtigten Kooperationspartnern. Der Fokus liegt also auf der Koordination zulässiger Nachrichtenabfolgen, so dass hierdurch gewissermaßen ein globales Kommunikationsprotokoll spezifiziert wird. Die Anwendungslogik der Teilnehmer bleibt hierbei verborgen.

Eine bedeutende Anforderung von geschäftlichen Anwendungen ist ferner ein verlässliches Übereinkommen zwischen beteiligten Partnern – bzw. der sie repräsentierenden Dienste – bezüglich durchgeführter Geschäftsabschlüsse. Auf der technischen Ebene haben sich *Transaktionen* (TA) als Standardmittel zur konsistenten Überführung von Prozesszuständen etabliert. Die dahinterstehenden Konzepte sind wohlverstanden und in gängigen Systemumgebungen umgesetzt, so dass sich die Integrität der Daten durch eine geeignete Fehlerbehebung sicherstellen lässt.

1.1 Motivation

Transaktionale Abläufe auch für Choreographien von Web-Services verfügbar zu machen, kann als besondere Herausforderung angesehen werden, da im Gegensatz zu or-

chestrierten Web-Services-Prozessen keine zentrale Instanz existiert, die Zugriff auf die Prozesslogik hat und daher verhältnismäßig einfach die Transaktionskontrolle übernehmen kann. Im Rahmen von Choreographien sind die beteiligten Partner durch autonome Dienste gegeben, so dass die Durchsetzung der transaktionalen Absicherung ohne Kenntnis von Anwendungsdetails erfolgen muss.

Im Web-Services-Umfeld existieren für die Realisierung von transaktionalen Abläufen u.a. die Erweiterungen *WS-AtomicTransaction* (WS-AT, [LW07]) und *WS-BusinessActivity* (WS-BA, [FL07]). Diese Standards ermöglichen sowohl kurzlebige ACID-Transaktionen (WS-AT) als auch langlebige Business-Transaktionen (WS-BA). Als Voraussetzung wird in beiden Fällen ein Koordinator benötigt, der mit Hilfe von *WS-Coordination* (WS-C, [FJ07]) zwischen den choreographierten Teilnehmern vermittelt. Der wesentliche Ablauf der Transaktionskontrolle nach WS-C im serviceorientierten Computing (SOC) entspricht dem *X/Open-DTP-Modell* [Ope96], bei dem eine TA durch einen Initiator gestartet wird und in deren Verlauf auf verteilte Ressourcen zugegriffen werden kann. Ein Koordinator vermittelt zwischen den Teilnehmern und leitet mit dem 2PC-Protokoll den konsistenten Abschluss der Arbeit ein, sobald der Initiator ihn dazu veranlasst.

An diesem Ablauf wird deutlich, dass nur der Initiator den Abschluss bzw. Abbruch einer TA einleiten kann, da er implizit als kontrollierender Teilnehmer betrachtet wird. Für Orchestrierungen ist diese Annahme zutreffend, da der Daten-, Nachrichten- und Kontrollfluss ohnehin durch eine zentrale Instanz gesteuert wird. Die Orchestrierungs-Engine verfügt mit der Prozesslogik also über die notwendige Information zur erfolgreichen Beendigung der Transaktionen. In Service-Choreographien können dagegen Situationen auftreten, in denen der Initiator nicht über den Abschluss einer TA entscheiden kann, da sich das Anwendungswissen auf alle Kooperationspartner verteilt. Zudem werden die Teilnehmer durch autonome Dienste repräsentiert, die nach dem Geheimnisprinzip gekapselt sind, so dass entsprechende Information nicht ohne Weiteres zugreifbar ist.

Diese Betrachtung offenbart zwei Aspekte desselben Problems. Einerseits hat der Initiator möglicherweise nicht genügend Anwendungswissen, um die *Demarkation* einzuleiten; andererseits verfügt er jedoch über das Privileg zur Demarkation, weshalb auch kein anderer Teilnehmer in der Lage sein kann, die TA abzuschließen. Das dargestellte Koordinationsprotokoll WS-C ermöglicht demzufolge keine explizite Trennung von der *Rolle* eines Teilnehmers und seinem *Privileg* zur Prozessbeendigung. Eine Lösung für diesen Fall besteht in der *Abgabe des Demarkationsrechts* an den Koordinator sowie in dessen Benachrichtigung über die Zustände der Teilnehmer im Verlauf der TA, so dass dieser die Aufgabe zur Demarkation erfüllen kann.

Eine weitere Herausforderung betrifft die mangelnde Fehlertoleranz transaktionaler Abläufe. So wird bei Auftreten eines Fehlers oder durch den Austritt von Teilnehmern in der Regel der gesamte Prozess abgebrochen und die geleistete Arbeit ist verloren.

Das Projekt *Transactional Activity Control for the Grid* (TracG, [HRR07], [RHR08]) am Arbeitsbereich VSIS untersucht Möglichkeiten zur Beseitigung der geschilderten Proble-

me, indem die Rolle des Koordinators gestärkt wird. Das Fundament hierfür bildet eine Menge von Inferenzregeln, mit deren Hilfe der Koordinator weitestgehend autonom entscheiden kann, wann der Prozess abgeschlossen werden darf und welche Teilnehmer ihre Ergebnisse festschreiben müssen bzw. verwerfen können. Ferner setzt das Projekt die etablierten Spezifikationen WS-C und WS-AT in einer prototypischen Java-Implementation um. Generische WS-AT-Clients bilden hierbei die zu koordinierenden Teilnehmer.

1.2 Zielsetzung

Im Zentrum des TracG-Projekts steht die erweiterte Transaktionskontrolle durch einen koordinatorbasierten Ansatz auf Grundlage von WS-Coordination. Daher wird die von TracG eingeführte regelbasierte Koordinierung von Teilnehmern eingehend diskutiert. Dies beinhaltet einerseits die Erweiterung von WS-C, um das für eine Koordination notwendige Anwendungswissen dem Koordinator verfügbar zu machen und andererseits einen durch den Koordinator zu implementierenden Mechanismus zur Auswertung der Koordinationsprimitive. Der Koordinator soll dadurch in die Lage versetzt werden, die lokalen Sichten der Teilnehmer zu einer globalen Sicht zu aggregieren und auf dieser Basis autonom über den Abschluss des Prozesses entscheiden zu können.

Das Ziel der Arbeit ist daher, zunächst die Spezifikation WS-BusinessActivity zu implementieren. Ausgehend von der vorhandenen Implementation von WS-Coordination sollen nachfolgend die diskutierten Erweiterungen zur Stärkung des Koordinators realisiert werden. Hierzu gehören eine detaillierte Bekanntmachung von Diensten sowie die regelbasierte Koordinierung.

Da besonders im Bereich der langlebigen Business-Transaktionen ein Scheitern der bereits erfolgten Arbeit unbedingt zu vermeiden ist, wird in diesem Kontext vor allem das Ausscheiden von Teilnehmern diskutiert. Es werden Aspekte untersucht, die zu einem robusteren Transaktionsverhalten führen und bei tolerierbaren Fehlern helfen, einen Prozessabbruch zu vermeiden. Insbesondere gilt es, die *Vitalität*[1] eines Teilnehmers in den regelbasierten Entscheidungsvorgang des Koordinators zu integrieren. Aufgabe wird sein, das erstellte Konzept für die Vitalität auszuweiten und den Regelsatz ggf. entsprechend anzupassen.

In einer weiterführenden Betrachtung wird auf die Klassifikation von Teilnehmern bzgl. ihres Abschlussverhaltens in Business-TA eingegangen. Grundsätzlich sind hierbei die Varianten *Do-Compensate*, *Validate-Do* und *Provisional-Final* möglich, die sich hinsichtlich der internen Realisierung von Abbruch und Festschreibung unterscheiden. Nach Furniss und Green von der Firma Choreology [FG05] unterstützt die WS-BA-Spezifikation derzeit nur Do-Compensate, wodurch eine Kompensation in vielen Fällen unnötig aufwendig ist. Ein alternativer Vorschlag von Choreology zur Beseitigung dieser Einschränkung wird untersucht und für eine Umsetzung im erarbeiteten Konzept überprüft.

[1] Notwendigkeit eines Teilnehmers zu einem erfolgreichen Abschluss der TA beizutragen.

1.3 Eingrenzung des Themas

Der Lebenszyklus eines dynamischen Prozesses umfasst neben der Durchführungsphase, in der ggf. auch die Transaktionsabwicklungen stattfinden, außerdem eine Planungs- und eine Vereinbarungsphase, in denen geeignete Dienste lokalisiert und entsprechend der gestellten Anforderungen komponiert werden. Trotz dieser integralen Anforderungen von dynamischen Prozessen, werden diese Phasen nicht ausführlicher betrachtet. Ebenso wird die Aushandlung nicht-funktionaler Qualitätseigenschaften (QoS), wie Sicherheit und Zuverlässigkeit, nicht eingehender beleuchtet.

Durch das in dieser Arbeit vorgestellte Konzept zur Stärkung der zentralen Position des Koordinators bzgl. der Koordinationslogik erübrigt sich die Propagierung des Rechts zur Prozessbeendigung an andere Teilnehmer weitestgehend. Ein explizites Management des Demarkationsrechts braucht damit nicht umgesetzt zu werden, was in komplexen Szenarien zu einer Reduktion des Koordinationsaufwands auf Seiten der Teilnehmer führt.

Workflows lassen sich mit Hilfe geeigneter Beschreibungssprachen spezifizieren. Für Orchestrierungen kann die *Web Services Business Process Execution Language* (WS-BPEL, [AAA+07]) genutzt werden und für Choreographien die *Web Services Choreography Description Language* (WS-CDL, [BBF+05]). Um die im TracG-Projekt vorgestellten Konzepte für diese Standards verfügbar zu machen, ist ggf. eine Anpassung der Spezifikationen erforderlich. Diese Thematik wurde bereits in der Diplomarbeit von Jorge Homann [Hom06] behandelt und ist daher nicht Gegenstand der vorliegenden Ausarbeitung.

Jenseits der transaktionalen Absicherung kann die Anforderung auftreten, sicherstellen zu müssen, dass ein Kooperationspartner die garantierten Zusicherungen auch wirklich erbracht hat. Dieser Fragestellung widmet sich das Projekt *Enforcement of Steps* (EoS, [RZ07]), das ebenfalls am Arbeitsbereich VSIS durchgeführt wird.

1.4 Aufbau der Arbeit

Im Anschluss an diesen einleitenden Abschnitt werden im zweiten Kapitel die technischen Grundlagen für das zu erarbeitende Konzept vorgestellt. Zunächst wird die Web-Service-Technologie mit den wesentlichen Standards eingeführt. Dazu gehören das XML-basierte Nachrichtenformat SOAP, die Schnittstellenbeschreibungssprache WSDL und der UDDI-Verzeichnisdienst. Nachfolgend werden Web-Services im Umfeld geschäftlicher Anwendungen betrachtet und Implikationen herausgearbeitet, die sich durch die methodisch unterschiedlichen Ansätze von Orchestrierung und Choreographie ergeben.

Eine weitere wesentliche Grundlage für diese Arbeit ist das Prinzip der Transaktion. Es werden die ACID-Eigenschaften kurzlebiger Transaktionen vorgestellt und Methoden zur verteilten Transaktionsabwicklung, wie das X/Open-DTP-Modell und das 2PC-Protokoll, erläutert. Darüber hinaus werden einige erweiterte Transaktionsmodelle eingeführt, die interessante Aspekte für eine transaktionale Verarbeitung von Web-Service-

Prozessen liefern. Es folgt die Vorstellung bestehender Standards für die Durchführung von Transaktionen mit Web-Services und ein abschließender Vergleich.

Das dritte Kapitel liefert eine Analyse von WS-Coordination und stellt den Ansatz des TracG-Projekts im Detail vor. Zu den thematisierten Aspekten der transaktionalen Koordinierung gehören die Bedeutung des Initiator-Koordinator-Protokolls und das Privileg, den Transaktionabschluss einleiten zu können und zu dürfen. Im zweiten Teil wird beschrieben, wie es dem Koordinator durch die Nutzung zusätzlicher Information gelingt, eine Transaktion selbstständig zu koordinieren. In diesem Kontext werden auch erweiterte Anforderungen vorgestellt, die zur Steigerung der Fehlertoleranz und Erhöhung der Flexiblität der Transaktion beitragen.

Im vierten Kapitel wird ein konkretes Konzept zur Integration der erarbeiteten Anforderungen in WS-BusinessActivity vorgestellt. Hierzu gehören die Verwaltung der Teilnehmer und von Protokollzuständen sowie deren Aggregation zu einem Koordinatorzustand. Es wird beschrieben, welche Information der Koordinator über Teilnehmerdienste benötigt und wie diese übermittelt werden kann. Hierfür werden Protokollerweiterungen vorgestellt, die zumeist den Koordinationskontext ergänzen oder aber neue Nachrichten definieren. Eine wesentliche Voraussetzung für die Autonomie des Koordinators ist die Kenntnis der Prozessausdehnung sowie der abzuschließenden oder verzichtbaren Teilnehmer. Das Konzept beschreibt, wie dieses Wissen für eine Entscheidung bzgl. des Prozessabschlusses genutzt wird. Das Gerüst eines Algorithmus konkretisiert das Konzept im Kontext der spezifizierten Protokollnachrichten.

Ferner werden erweiterte Anforderungen wie Vitalität, Abhängigkeiten zwischen Teilnehmern und Implementationsmuster von Diensten diskutiert sowie ihre Integrationsfähigkeit in das entwickelte Koordinatorkonzept analysiert.

Die realisierte Implementation wird im fünften Kapitel vorgestellt. Neben einer Architekturübersicht werden interessante Aspekte der Implementierung veranschaulicht. Hierzu gehören das Persistieren der Aktivität, die Verarbeitung asynchron eintreffender Nachrichten, der Zustandswechsel des Koordinators oder die Verwendung einer Inferenz-Engine zur Auswertung der Regelmenge.

Das sechste Kapitel fasst die Ergebnisse zusammen und bewertet die erhaltenen Resultate. Ein abschließender Ausblick gibt Hinweise auf weiterführende Themen.

2 Technische Grundlagen

Die technischen Voraussetzungen für das Verständnis dieser Arbeit werden hauptsächlich durch Web-Services (s. Abschn. 2.1) und Transaktionen gebildet (s. Abschn. 2.3). Hinzu kommen Konzepte, die Web-Services in die Umgebung geschäftlicher Anwendungen eingliedern (s. Abschn. 2.2) sowie Spezifikationen, die eine Beziehung zu Transaktionen herstellen (s. Abschn. 2.4). Die folgenden Abschnitte geben einen Überblick über die notwendigen Technologien und führen ein in verwendete Begriffe und Modelle.

2.1 Serviceorientierte Architekturen mit Web-Services

Eine *serviceorientierte Architektur* (SOA) ist das abstrakte Konzept einer verteilten Softwarearchitektur, die auf vorhandene Technologien, wie XML oder HTTP, zurückgreift [DJMZ05]. Sie ist gekennzeichnet durch den modularen Aufbau aus voneinander unabhängigen, wiederverwendbaren Komponenten, welche ihre Funktionalitäten in Form von Diensten (Services) bereitstellen [HRS05].

Zu den wichtigsten Aspekten dieses Paradigmas zählt die lose Kopplung der Services. Darunter wird das dynamische Suchen, Finden und Einbinden von Diensten zur Laufzeit verstanden. Voraussetzung für diese dynamische Bindung sind maschinenlesbare, einheitlich beschriebene Schnittstellen sowie Verzeichnisdienste, die eine effiziente Suche ermöglichen. Mit der Kapselung der zu Grunde liegenden Technologie hinter einheitlichen Schnittstellen wird das Geheimnisprinzip umgesetzt und damit eine Trennung von Implementationsdetails und Verwendung eines Services erreicht, was in Verbindung mit der Nutzung gängiger Transportprotokolle zu einer hohen Plattform- und Implementationsunabhängigkeit führt. Die Kapselung hat auch eine Reduktion der Komplexität zur Folge und ermöglicht außerdem die Bereitstellung der Funktionalitäten von Legacy-Systemen [HRS05]. Verteilte Anwendungen lassen sich somit einfacher, schneller und kostengünstiger realisieren.

Mit einer SOA wird die Automatisierung von Kommunikationsabläufen verfolgt. Daher steht das Zusammenspiel von Maschinen bzw. Diensten im Zentrum der Betrachtung. Es lassen sich drei unterschiedliche Rollen identifizieren (siehe Abb. 2.1):

Service Provider: Der Dienstanbieter stellt einen Dienst zur Verfügung, indem er die zugehörige Dienstbeschreibung in einem Verzeichnis veröffentlicht (Publish). Sie enthält eine vollständige, maschinenlesbare Spezifikation des Dienstes. Dazu gehören Festlegungen der Schnittstellen, der Nachrichtenformate sowie ggf. textuelle Funktionsbeschreibungen des Dienstes.

Service Registry: Das Dienstverzeichnis ermöglicht die Veröffentlichung und das Auffinden von Diensten. In Analogie zu einem Branchenbuch werden Dienstbeschrei-

bungen nach verschiedenen Bereichen kategorisiert, so dass sich die Suche vereinfacht (Find).

Service Requestor: Nachdem ein geeigneter Dienst ausfindig gemacht wurde, bindet sich der Dienstnutzer an den Dienst, indem er direkt mit dem Dienstanbieter kommuniziert (Bind).

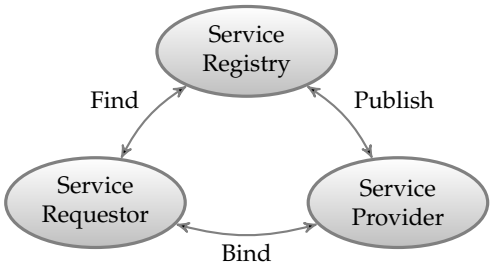

Abbildung 2.1: Schema einer serviceorientierten Architektur (vgl. [Rit05]).

Eine SOA lässt sich prinzipiell mit allen dienstbasierten Techniken konstruieren. Nach Kossmann und Leymann [KL04] bieten *Web-Services* (WS) jedoch den Vorteil, dass sie auf XML-basierten Spezifikationen beruhen. Entscheidend für ihren Erfolg ist ferner die Verzahnung der verwendeten Technologien sowie die umfangreiche Unterstützung durch namhafte Hersteller. Die Autoren ergänzen zwei weitere wichtige Eigenschaften von Web-Services: Zum einen die Identifizierbarkeit durch URIs und ihre Autonomie gegenüber den Dienstnutzern. Zusammenfassend ergeben sich für Web-Services folgende wesentliche Merkmale:

- WS haben eine maschinenlesbare, selbstbeschreibende Schnittstelle.
- Die Kommunikation geschieht mittels XML-basierter SOAP-Nachrichten.
- Verwendung finden gängige Transportprotokolle, wie z.B. HTTP oder SMTP.
- WS sind eindeutig durch URIs identifizierbar.
- WS sind autonom gegenüber den Dienstnutzern, d.h., Qualitätseigenschaften (QoS) werden ggf. durch weitere Vereinbarungen geregelt.

Ein Dienst kann auch Funktionalitäten kapseln, die möglicherweise von anderen Anbietern bereitgestellt werden. Man bezeichnet ihn dann als zusammengesetzten Dienst. Für einen Dienstnutzer ist diese Komposition nicht ersichtlich.

Abbildung 2.2 zeigt eine Veranschaulichung der verwendeten Technologien im Web-Services-Umfeld als Schichtmodell.[1] Die Basis bilden dabei die Datenübertragungsprotokolle der untersten Schicht (Transport). Prinzipiell können hier beliebige Protokolle eingesetzt werden, denn die Spezifikationen machen diesbezüglich keine Vorgaben. Am

[1] Ähnliche Varianten eines Web-Services-Stacks finden sich in der Literatur.

weitesten verbreitet ist jedoch das „Hypertext Transfer Protocol“ (HTTP), da dieses Protokoll die meisten Firewalls und Proxies ungefiltert passieren kann [ACKM04].[2]

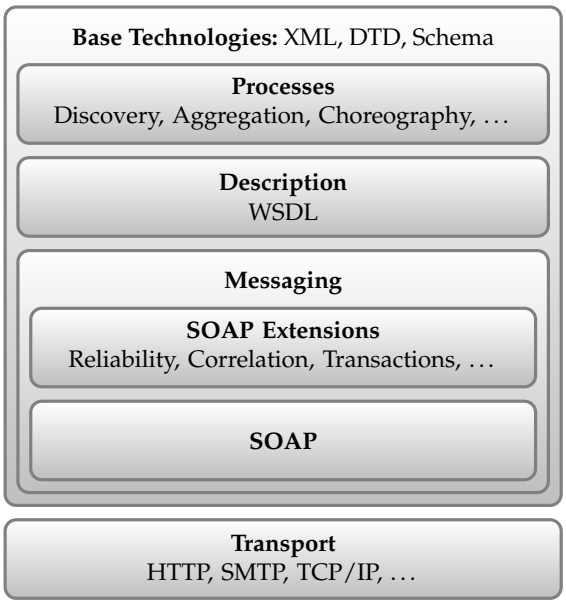

Abbildung 2.2: Der Web-Services-Stack (in Anlehnung an [BHM+04]).

Mit der „Extensible Markup Language“ (XML) [BPSM+06] vom W3C wird die plattformunabhängige Basis (Base Technologies) für die Protokolle und Nachrichten bereitgestellt. XML dient der Beschreibung von Datentypen und Strukturen. Neben der Kernsyntax von XML sind hierfür vor allem das XML-Information-Set [CT04], das XML-Schema [FW04] und die XML-Namespaces [BHLT06] relevant (vgl. [BHM+04]).

Die Nachrichtenformatierung (Messaging) findet mit Hilfe des XML-basierten SOAP-Standards statt (vgl. Abschnitt 2.1.1). Auf diese Weise wird die Unabhängigkeit vom darunterliegenden Transportprotokoll garantiert. Auf Grund der Erweiterbarkeit von SOAP existieren eine Reihe von Zusatzspezifikationen (SOAP Extensions), die SOAP um Funktionalitäten ergänzen, welche von SOAP selbst nicht definiert werden. Hierzu gehören beispielsweise Sicherheits- und Zuverlässigkeitsaspekte oder Transaktionssteuerung.

Oberhalb der Messaging-Schicht ist die Beschreibung der Web-Services mit WSDL angeordnet (Description). Die WSDL-Dokumente enthalten Informationen zur Funktionalität der WS sowie zum Zugriff auf sie (vgl. Abschnitt 2.1.2).

Die oberste Schicht im Web-Services-Stack bildet die Prozessebene (Processes), wozu in dieser Darstellung auch das Auffinden von Web-Services in UDDI-Verzeichnissen (vgl. Abschnitt 2.1.3) gezählt wird (Discovery). Außerdem sind hier die Komposition von WS zu Orchestrierungen sowie Choreographien einzuordnen (siehe auch Kapitel 2.2).

[2] Nach [DJMZ05] liegt die Verbreitung von HTTP bei etwa 90 %.

2.1.1 SOAP

SOAP[3] [GHM+03a] ist ein Standard vom W3C und liegt gegenwärtig in der Version 1.2 vor. SOAP ist XML-basiert und dient der strukturierten Übertragung von Daten. Zur SOAP-Spezifikation gehören im Kern folgende Vorgaben (vgl. [GHM+03b], [GHM+03c]):

- Ein Nachrichtenformat zur Darstellung von Information in XML,
- Modelle zur Verarbeitung und zur Erweiterung von SOAP-Nachrichten,
- Regeln zur Bindung an verschiedene Transportprotokolle und
- ein Mechanismus für entfernte Prozeduraufrufe (RPC) via SOAP-Nachrichten.

Wie sich auch Binärdaten in SOAP-Nachrichten übertragen lassen, legen zusätzliche Spezifikationen fest.

Struktur von SOAP-Nachrichten

Das XML-Dokument einer SOAP-Nachricht besteht aus dem Wurzelelement `Envelope` sowie den darin gekapselten Elementen `Header` und `Body` (vgl. Abb. 2.3). Die Angabe eines SOAP-Headers ist optional. Dagegen ist der SOAP-Body obligatorisch, da er die eigentlichen Daten enthält. Kindelemente von `Header` und `Body` werden weiter in Header-Blocks und Body-Sub-Elements unterteilt.

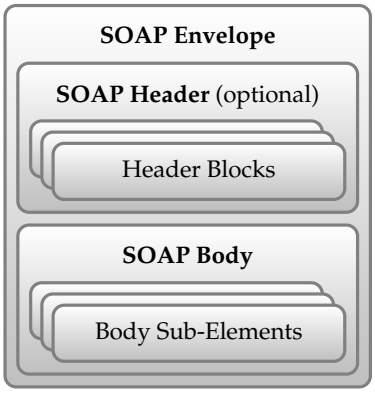

Abbildung 2.3: Schema einer SOAP-Nachricht [ML07].

Der Inhalt des SOAP-Headers ist nicht Teil der Spezifikation. Er dient gewissermaßen als Container für Erweiterungen, die den SOAP-Standard um zusätzlichen Funktionen ergänzen [DJMZ05]. Jeder einzelne Header-Block muss einen XML-Namensraum definieren, der durch einen URI referenziert wird.

Eine SOAP-Nachricht wird bei der Übertragung durch mehrere sog. SOAP-Knoten geleitet. Dazu zählen der Sender, der Empfänger und die Zwischenstationen. Jeder dieser Knoten nimmt stets eine oder mehrere Rollen ein, mit denen die Header-Blöcke über das

[3] In früheren Versionen handelte es sich bei „SOAP“ um ein Akronym mit der Bedeutung „Simple Object Access Protocol“. Dies wurde jedoch mit Erscheinen der aktuellen Version aufgegeben [GHM+03b].

`role`-Attribut assoziiert sind. SOAP-Knoten dürfen nur dann die Header-Blöcke verarbeiten, wenn diese für ihre Rolle bestimmt sind. Neben den vordefinierten Rollen des SOAP-Standards (siehe Tab. 2.1) können weitere anwendungsspezifische Rollen vereinbart werden.

Tabelle 2.1: Vordefinierte Rollen des SOAP-Standards [GHM+03b].

Rolle	Beschreibung
`next`	jeder Zwischenknoten und der Empfänger
`none`	kein Knoten
`ultimateReceiver`	der Empfänger

Über das optionale boolesche Attribut `mustUnderstand` kann angegeben werden, ob der jeweilige Knoten den Header-Block auswerten können muss. Falls diese Forderung zutrifft und der Knoten sie nicht erfüllen kann, wird die Verarbeitung der Nachricht abgebrochen und eine Fehlermeldung zurückgesendet.

Fehlerbehandlung

Bei Auftreten eines Fehlers in einem beliebigen Knoten wird ein `Fault`-Element als einziges Kindelement des Bodies an den ursprünglichen Sender zurückgesendet. Das `Fault`-Element hat die obligatorischen Kindelemente `Code` und `Reason`, die den Fehlercode (siehe Tab. 2.2) und eine textuelle Beschreibung des Fehlers enthalten.

Tabelle 2.2: SOAP Fault-Codes [GHM+03b].

SOAP Fault Code	Beschreibung
`VersionMismatch`	Versionskonflikt
`MustUnderstand`	der Knoten kann einen Header-Block nicht auswerten
`DataEncodingUnknown`	unbekannte Codierung
`Sender`	Fehler beim Sender
`Receiver`	Fehler beim Empfänger

Erweiterungen von SOAP

Bei der Entwicklung von SOAP wurde besonders auf Einfachheit und Erweiterbarkeit des Standards geachtet. Als Konsequenz existieren diverse SOAP-Erweiterungen, die zusätzliche Eigenschaften ergänzen. Nachfolgend wird die Erweiterung WS-Addressing vorgestellt, da sie die Grundlage für eine asynchrone Kommunikation in langlebigen Prozessen mit HTTP bildet. Genau diese Funktionalität wird später benötigt.

WS-Addressing Durch die Wahl des Transportprotokolls wird gleichzeitig eine Entscheidung über synchrone bzw. asynchrone Kommunikation getroffen. So unterstützt

HTTP beispielsweise keine asynchrone Datenübertragung. Folglich bestimmt das Transportprotokoll in gewisser Weise den nutzbaren SOAP-Funktionsumfang, was nach Dostal et al. [DJMZ05] klar im Widerspruch zum ursprünglichen Gedanken von SOAP steht.

Desweiteren wird ebenfalls der Service-Endpunkt vom Transportprotokoll spezifiziert, so dass auch die Adressinformation nicht unabhängig formuliert werden kann. Dies betrifft vor allem das Routing und die Nachrichtenidentität. Die ursprünglich von BEA, IBM, Microsoft, SAP und Sun entwickelte und mittlerweile vom W3C weitergeführte SOAP-Erweiterung *WS-Addressing* [GHR06a] soll diese Mängel beseitigen.

Hierzu definiert WS-Addressing die Konzepte der Endpunktreferenzen (Endpoint References, EPR) und Eigenschaften der Nachrichtenadressierung (Message Addressing Properties, MAP) sowie deren Abbildung auf Header-Elemente von SOAP-Nachrichten (vgl. [GHR06b]).

EPRs sind Datenstrukturen, die als einziges obligatorisches Element die Adressinformation des Endpunktes als absoluten IRI [DWSM05] enthalten. Daneben sind die optionalen Elemente `ReferenceParameters` und `Metadata` erlaubt, um zusätzliche Informationen für den Endpunkt formulieren zu können (vgl. Listing 2.1).

Listing 2.1: XML-Infoset der „Enpoint References “ (aus [GHR06a]).

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?
  <wsa:Metadata>xs:any*</wsa:Metadata>?
</wsa:EndpointReference>
```

Das Konzept der MAPs erweitert die Nachrichten um diverse Eigenschaften, die direkten Einfluss auf das Routing nehmen. Sie erlauben es, neue Kommunikationsformen zu verwenden, wie z.B. asynchrone Interaktionen oder Publish/Subscribe-Verfahren. Listing 2.2 zeigt die mit den MAPs eingeführten SOAP-Header-Elemente als XML-Infoset.

Listing 2.2: XML-Infoset der „Message Addressing Properties“ (aus [GHR06a]).

```
<wsa:To>xs:anyURI</wsa:To> ?
<wsa:From>wsa:EndpointReferenceType</wsa:From> ?
<wsa:ReplyTo>wsa:EndpointReferenceType</wsa:ReplyTo> ?
<wsa:FaultTo>wsa:EndpointReferenceType</wsa:FaultTo> ?
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:MessageID>xs:anyURI</wsa:MessageID> ?
<wsa:RelatesTo RelationshipType="xs:anyURI"?>
  xs:anyURI
</wsa:RelatesTo> *
<wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?
```

Nach [DJMZ05] kann WS-Addressing durch die Beseitigung der eingangs erwähnten Schwächen eine hohe Relevanz zugeschrieben werden. Dies zeigt sich sowohl an

den auf WS-Addressing aufbauenden Standards (z.B. WS-Notification, WS-Policy) als auch an der Anzahl verfügbarer Implementationen. Die Autoren erwarten eine zukünftige Integration von WS-Addressing in SOAP und WSDL. Zumindest für WSDL scheint sich diese Annahme zu bestätigen, da WSDL 2.0 eine Endpoint-Komponente beinhaltet, die in Verbindung mit anderen WSDL-2.0-Komponenten zur Spezifikation von Service-Endpunkten verwendet werden kann [GHR06a].

2.1.2 WSDL – Web Services Description Language

Die XML-basierte *Web Services Description Language* (WSDL) ist ein weiterer Standard vom W3C und liegt derzeit in Version 2.0 vor [BL07]. Diese Arbeit basiert jedoch noch auf Version 1.1 [CCMW01], da die OASIS-Standards, die die Grundlage dieser Arbeit bilden (vgl. Kap. 2.4.3), noch die alte WSDL-Version verwenden. Zudem existieren WSDL-Konverter, die zwischen den Versionen umwandeln.[4] Nachfolgend wird daher WSDL 1.1 beschrieben.

WSDL dient der neutralen Definition von WS-Schnittstellen. Dabei wird die abstrakte Beschreibung der Funktionalität von der Beschreibung der konkreten technischen Details zum Aufruf des WS unterschieden. Abbildung 2.4 illustriert diese Struktur einer WSDL-Schnittstelle.

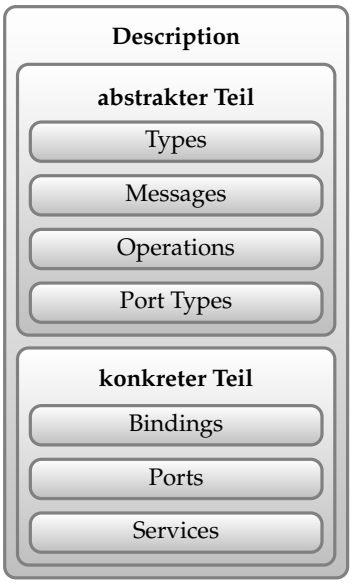

Abbildung 2.4: Struktur einer WSDL-Spezifikation (nach [ACKM04]).

Im abstrakten Teil werden die verwendeten Typen, Nachrichten und Operationen des Dienstes unabhängig von einem Protokoll oder einem Endpunkt beschrieben. Der abstrakte Teil besteht aus folgenden Elementen:

[4] Das W3C stellt einen webbasierten Konverter zur Umwandlung von WSDL 1.1 nach WSDL 2.0 unter `http://www.w3.org/2006/02/WSDLConvert.html` bereit.

Types: Typdefinitionen der verwendeten Datenstrukturen, meistens auf Basis von XML-Schema [FW04]. Es ist jedoch möglich, andere Typsysteme zu verwenden.

Messages: Beschreibung der Nachrichten unter Verwendung der definierten Typen.

Operations: Abstrakte Spezifikation der Dienstfunktionalität. Operationen senden und empfangen Nachrichten, wobei WSDL vier Interaktionsmuster unterstützt:

- **One-way:** Die Operation empfängt eine einzelne Nachricht ohne eine Antwort zu senden (asynchron).
- **Request-response:** Die Operation empfängt eine Anfrage und sendet eine Antwortnachricht (synchron).
- **Solicit-response:** Die Operation sendet eine Anfrage und wartet auf eine Antwortnachricht (synchron).
- **Notification:** Die Operation sendet eine einzelne Nachricht ohne auf eine Antwort zu warten (asynchron).

Port Types: Menge der unterstützten Operationen eines oder mehrerer Endpunkte.

Der konkrete Teil einer WSDL-Beschreibung setzt die im abstrakten Teil beschriebenen Elemente in Beziehung zu verwendeten Transportprotokollen und Kodierungen. Außerdem wird die URI des Dienstes spezifiziert. Folgende Elemente sind zulässig:

Bindings: Spezifikation eines konkreten Protokolls und Datenformats für die definierten Port Types. Sie beschreibt, wie eine Operation aufgerufen wird und wie die Nachricht kodiert sein muss.

Ports: Konkrete Endpunkte werden als Kombination einer Netzwerkadresse und eines Bindings definiert.

Services: Stellen Ports bereit und definieren damit Mengen logisch zusammenhängender Endpunkte.

Eine WSDL-Beschreibung legt die strukturellen Eigenschaften eines Web-Services fest. WSDL-Definitionen sind maschinenlesbar und eignen sich daher besonders gut für die Veröffentlichung in Dienstverzeichnissen. Bei der Entwickung von Web-Services dienen diese Schnittstellenbeschreibungen als Basis zur Erzeugung von Code-Skeletten.

2.1.3 UDDI – Universal Description, Discovery and Integration

Die aktuelle Version 3.0.2 des Verzeichnisdienstes *Universal Description, Discovery and Integration* (UDDI) [CHRR04] wurde im Oktober 2004 von der Standardisierungsorganisation OASIS verabschiedet. UDDI beinhaltet ein Datenmodell zur Spezifizierung von Diensten, einen leistungsfähigen Namens- und Verzeichnisdienst sowie verschiedene SOAP-Schnittstellen für den Zugriff auf das Verzeichnis.

Die Vision von UDDI sieht vor, beliebige Unternehmen und ihre Dienste mit Hilfe von Anwendungen in öffentlichen UDDI-Verzeichnissen, den sog. *UDDI Business Registries* (UBR), zu registrieren und auffinden zu können [DJMZ05]. Zur Erprobung der Technologie wurden UBRs von Microsoft, IBM und SAP betrieben, inzwischen jedoch wieder eingestellt.[5] Häufig werden UDDI-Verzeichnisse dagegen firmenintern eingesetzt und nur bestimmte Dienste quasiöffentlich zur Verfügung gestellt.

Aufbau von UDDI-Verzeichnissen

Die Struktur von UDDI-Verzeichnissen ist vergleichbar mit dem Aufbau von Telefonbüchern ([ACKM04],[DJMZ05]). Sie besteht aus den Kategorien:

White Pages: Dieser Abschnitt enthält Informationen über Unternehmen, welche Web-Services anbieten. Eine Analogie hierzu bildet das Telefonbuch.

Yellow Pages: Sie ordnen die verfügbaren Dienste nach Geschäftsfeldern, vergleichbar mit den Gelben Seiten.

Green Pages: Die technischen Einzelheiten für den Aufruf des Dienstes werden mit den Green Pages bereitgestellt. Meist enthalten sie Referenzen auf Dienstbeschreibungen, die beim Diensterbringer hinterlegt sind. Die Green Pages sind primär für den menschlichen Nutzer gedacht, der im Vorfeld einer Anwendungsentwicklung nach geeigneten Services sucht (design time discovery).

Service Type Registration: Diese Kategorie repräsentiert das maschinenlesbare Pendant zu den Green Pages und ermöglicht das dynamische Binden von Diensten (runtime binding). Beide Kategorien verweisen aufeinander.

Entsprechend dieser Verzeichnisse, besteht das UDDI-Datenmodell im Kern aus vier dazu korrespondierenden Datenstrukturen.

UDDI-Schnittstellen

UDDI-Verzeichnisse werden auf dreierlei Weise genutzt: Dienstanbieter veröffentlichen ihre Dienste, Dienstnutzer suchen Dienste und andere Verzeichnisse gleichen Daten ab. Für diese Aufgaben stehen diverse SOAP-APIs zur Verfügung. Zwingend schreibt der Standard folgende APIs vor:

UDDI Inquiry API: Diese Schnittstelle ermöglicht die Suche im Verzeichnis sowie das Abfragen von Informationen über einen bestimmten Service. Die API ist vorgesehen für UDDI-Browser und Dienstnutzer.

[5] Siehe hierzu die von Microsoft herausgegebene FAQ unter `http://uddi.microsoft.com/about/FAQshutdown.htm`.

UDDI Publication API: Mit Hilfe dieser Schnittstelle werden Informationen über Web-Services in einem Verzeichnis veröffentlicht oder aktualisiert. Daher ist die API für Dienstanbieter gedacht.

In den letzten Abschnitten wurden die Grundbausteine serviceorientierter Architekturen eingeführt, wodurch die Beschreibung der WS-Interfaces (WSDL) und des Nachrichtenformats (SOAP) sowie die Veröffentlichung (UDDI) in Dienstverzeichnissen ermöglicht wird. Diese Standards bilden die technische Grundlage für die Komposition von Web-Services zu Geschäftsprozessen.

2.2 Workflows und Web-Services

Die Produktion von Waren und das Erbringen von Dienstleistungen setzen sich üblicherweise aus vielen Einzelschritten, sog. *Aktivitäten*, zusammen. Die zielgerichtete Abfolge dieser Aktivitäten zur Erstellung einer Unternehmensleistung wird als *Geschäftsprozess* bezeichnet. In Bezug auf Softwaresysteme, die diese Aufgabe unterstützen und den Ablauf der Prozesse weitestgehend automatisieren, wird von einem Geschäftsprozess auch als Arbeitsablauf bzw. *Workflow* gesprochen.

Die Komposition von Geschäftsprozessen auf Basis von Web-Services beinhaltet zwei Aspekte, die als *Orchestrierung* und *Choreographie* bezeichnet werden. Unter einer Orchestrierung wird die Komposition von Web-Services zu einem Gesamtprozess verstanden, der unter der Kontrolle einer Geschäftsdomäne abläuft. Dagegen bezeichnet eine Choreographie die Koordination von Prozessen unterschiedlicher Kooperationspartner, wie sie insbesondere im unternehmensübergreifenden Bereich (B2B) von Bedeutung ist. In Abbildung 2.5 wird dieser Zusammenhang veranschaulicht.

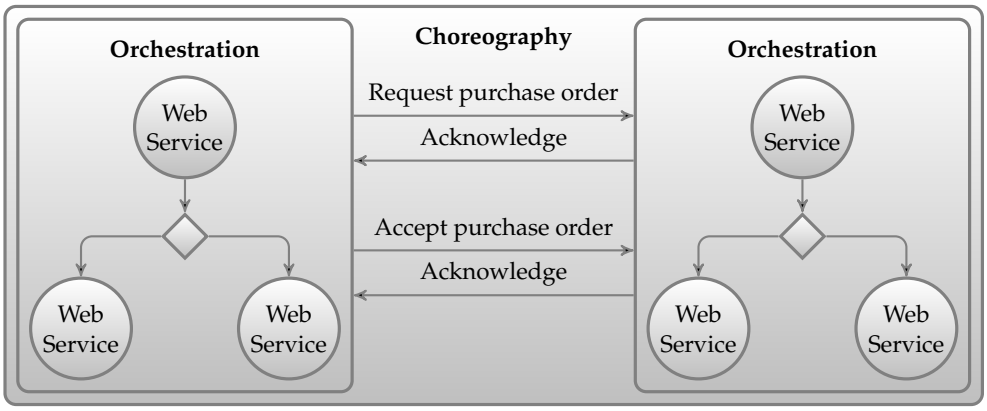

Abbildung 2.5: Orchestrierung versus Choreographie [Pel03].

2.2.1 Konversation

Eine *Konversation* beschreibt die konkrete Kommunikation zwischen Diensten. Ihr liegt dabei immer ein gemeinsamer Kontext der beteiligten Kommunikationspartner zu Grun-

de, der den Zustand der Konversation wiedergibt [ACKM04]. Im „Web Services Glossary“ des W3C [HB04] findet sich folgende Definition:

> *„A Web service conversation involves maintaining some state during an interaction that involves multiple messages and/or participants.“*

Demnach sind verteilte Transaktionen (vgl. Kap. 2.3) über Konversationen abzuwickeln, da es für einen konsistenten Abschluss mehrerer Teilnehmer erforderlich ist, ihren Zustand zu kennen.

2.2.2 Choreographie

Eine *Choreographie* beschreibt die Kommunikation zwischen autonomen, gleichberechtigten Kooperationspartnern. Der Fokus liegt dabei auf dem Zusammenspiel zulässiger Nachrichtenabfolgen zwischen den Teilnehmern. Es wird gewissermaßen ein globales, öffentliches Protokoll spezifiziert, dessen interne Realisierung auf Seiten einzelner Partner nicht Gegenstand der Choreographie ist. Darüber hinaus existiert auch keine zentrale Instanz, die die Kommunikation steuert. Vielmehr reagieren die Teilnehmer selbstständig auf eintreffende Nachrichten.

Im Gegensatz zu einer Konversation bezieht sich eine Choreographie jedoch nicht auf konkrete Teilnehmer, sondern stets auf Rollen, wie z.B. Bank, Kunde oder Firma. Diese Rollen werden in einer Konversation von konkreten Diensten oder Anwendungen eingenommen [RS04]. Für die Darstellung von Choreographien können Zustandsdiagramme, Aktivitätsdiagramme oder Sequenzdiagramme eingesetzt werden [ACKM04]. Das W3C definiert den Begriff der Choreographie sehr ausführlich [HB04]:

> *„1. A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.“*
>
> *„2. Web Services Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services, applications or human beings. Transactions among Web Services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be a choreography.“*

Im Hinblick auf Transaktionen entspricht ein abstraktes Transaktionsprotokoll genau dem Teil der Choreographie, der zwischen Koordinator der Transaktion und Teilnehmern stattfindet (vgl. Zustandsgraphen in Abschnitt 2.4.3). Der restliche Teil der Choreographie wird durch die Nachrichten der Teilnehmer untereinander gebildet.

Choreographien lassen sich mit Hilfe von Beschreibungssprachen spezifizieren – als Beispiel sei die Sprache WS-CDL genannt (vgl. [BBF+05], [BDO05]). Dies soll hier jedoch nicht weiter vertieft werden.

2.2.3 Behavioral Interface

Die Schnittstellen der Kooperationspartner in einer Choreographie werden als *Behavioral Interfaces* oder als abstrakter Prozess bezeichnet. Sie legen die Reihenfolge und Abhängigkeiten von Interaktionen der einzelnen Teilnehmer aus ihrer jeweiligen Perspektive fest und ergänzen damit strukturelle Schnittstellenbeschreibungen (structural interface), wie sie von WSDL unterstützt werden. Im Unterschied zur Choreographie wird das kommunikative Verhalten aus dem Blickwinkel *eines* Partners beschrieben. Die Schnittstelle umfasst also keine komplette Interaktion – dies würde die Betrachtung von mindestens zwei Parteien erfordern – sondern stets nur die Aktionen eines einzelnen Partners. Genau wie bei Choreographien, werden auch hier keine internen Abläufe eines Teilnehmers wiedergegeben. Nur sein nach außen sichtbares Verhalten ist relevant.

In einer gegebenen Konversation kann eine Rolle in der Choreographie mit mehreren verhaltensbeschreibenden Schnittstellen verknüpft werden und daher auch mit mehreren WSDL-Interfaces. Beispielsweise kann die Rolle „Kunde“ aus den Operationen bestehen, sich über die Verfügbarkeit einer Ware zu informieren oder diese Ware zu bestellen. Umgekehrt können mehrere abstrakte Prozesse die Bedingungen einer Rolle in einer Choreographie erfüllen [BDO05], d.h., es kann mehrere Kunden in einer Choreographie geben.

2.2.4 Orchestrierung

Eine *Orchestrierung* beschreibt immer die Sicht *eines* Kooperationspartners [Pel03]. Sie definiert die Komposition von Aktivitäten zu einem Gesamtprozess, der eine neue Funktionalität implementiert und so einen geschäftlichen Mehrwert generiert [DJMZ05]. Es handelt sich um einen ausführbaren Prozess, der von einer Orchestrierungs-Engine gesteuert wird und der sowohl mit internen als auch mit externen Web-Services interagieren kann. Das W3C definiert den Begriff wie folgt (aus [HB04]):

> *„An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.“*

Eine Orchestrierung beschreibt alle Kontroll- und Datenstrukturen, alle Abhängigkeiten sowie die Ausführungsreihenfolge der Kommunikationsabläufe, die für die Prozessmodellierung relevant sind. Dazu gehören auch interne Abläufe, die bei Teilnahme an einer Choreographie nicht bekannt gemacht werden müssen oder sollen.

Orchestrierungen können wiederum als Web-Service zur Verfügung gestellt werden (Aggregation). Durch eine stufenweise Komposition lässt sich so von den Funktionen der Komponenten abstrahieren und auf diese Weise eine Reduktion der Komplexität erreichen [ACKM04].

Eine Beschreibungssprache, die für Orchestrierungen zur Verfügung steht, ist die Business Process Execution Language (BPEL) [AAA+07].

2.2.5 Abgrenzung und Diskussion

Die eingeführten Begriffe korrelieren sehr stark miteinander und überschneiden sich zum Teil. Diese engen Beziehungen können jedoch bei der Modellierung von Web-Service-Kompositionen genutzt werden. Nach Barros et al. [BDO05] ergeben sich folgende Verwendungsmöglichkeiten für Choreographien:

- Generierung der abstrakten Prozesse, die jeder Teilnehmer bereitstellen muss, um in einer Choreographie mitzuwirken.
- Validierung von verhaltensbeschreibenden Schnittstellen gegen eine Choreographie und die daraus ableitbare Befähigung eines Teilnehmers an der Choreographie partizipieren zu können.

Auf ähnliche Weise lassen sich abstrakte Prozessbeschreibungen einsetzen. Sie können als Ansatz zur Realisierung von ausführbaren Prozessen dienen, indem sie einen Entwicklungsrahmen liefern, der zu einer vollständigen Orchestrierung erweitert werden kann. Umgekehrt können für Orchestrierungen Konsistenzprüfungen zu einer gegebenen verhaltensbeschreibenden Schnittstelle durchgeführt werden [BDO05].

Aus diesen Zusammenhängen ergeben sich zwei Sichtweisen. Ausgehend von einer Choreographie kann von einer Top-Down-Perspektive gesprochen werden, in der die Choreographie als Ausgangspunkt dient, verhaltensbeschreibende Schnittstellen zu spezifizieren, die wiederum verwendet werden können, um neue Services zu entwickeln bzw. bestehende Services anzupassen. Die Bottom-Up-Perspektive abstrahiert dagegen von einem ausführbaren Prozess über die abstrakten Prozesse zu einer Choreographie. Dies erweist sich als nützlich bei der Analyse der gesamten Kommunikation [DKLW07]. Abbildung 2.6 verdeutlicht diese Beziehungen.

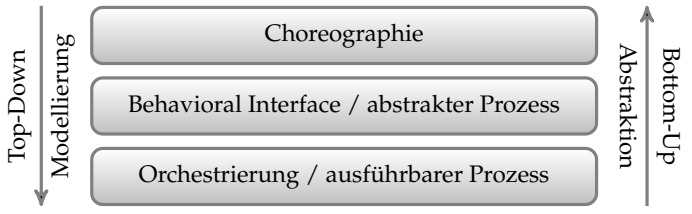

Abbildung 2.6: Beziehungen zwischen Choreographie, Behavioral Interface und Orchestrierung.

Wie später noch zu sehen ist, kommen bei der Abwicklung verteilter Transaktionen Koordinatoren zum Einsatz. Entsprechend den Perspektiven von Choreographie und Orchestrierung, gibt es verschiedene Möglichkeiten sie zu realisieren.

Ein Koordinator als Bestandteil einer Orchestrierungs-Engine kennt die Anwendungslogik der Steuerungsinstanz sowie die Zustände der Aktivitäten zu jedem Zeitpunkt

der Prozessausführung. Die Orchestrierungs-Engine ist Initiator und Koordinator einer Transaktion zugleich und kann daher auch stets darüber entscheiden, wann eine Transaktion abzuschließen ist bzw. welche Aktivitäten festschreiben oder abbrechen sollen.

Obwohl Choreographien ohne zentrale Steuerungsinstanz ablaufen, werden zur Transaktionsabwicklung auch hier Koordinatoren benötigt. Diese steuern das Verhalten der Teilnehmer jedoch nicht unmittelbar, denn sie haben keinen Einfluss auf deren Kontrollfluss, wie es bei Orchestrierungen der Fall ist. Dennoch sind sie auf Daten der Anwendungslogik angewiesen, um zu einer Entscheidung bzgl. Abschluss oder Abbruch einer Transaktion zu gelangen. Die in dieser Arbeit angestrebte Autonomie von Transaktionskoordinatoren erfordert daher eine Erweiterung bestehender Protokolle, um diese Anforderung leisten zu können. Es ist wichtig zu verstehen, dass Transaktionskoordinatoren in einer Choreographie nicht den gesamten Prozess lenken – sie sammeln auf Basis der ausgetauschten Nachrichten lediglich Daten, die die Transaktion betreffen und reagieren entsprechend des implementierten Protokolls mit eigenen Nachrichten.

2.3 Transaktionen

Zu den Grundanforderungen vieler Systeme gehören konsistenzerhaltende Datenmanipulationen und Toleranz gegenüber Systemstörungen oder -ausfällen. Transaktionen haben sich als geeignetes Instrument erwiesen, diese Aufgabe zu erfüllen.

2.3.1 Klassisches Transaktionskonzept

Eine *Transaktion* (TA) ist definiert als ununterbrechbare Folge von Befehlen bzw. Änderungen, die die Datenbank von einem logisch konsistenten Zustand in einen neuen logisch konsistenten Zustand überführt. Als fundamentale Bedingungen zur Konsistenzsicherung bei Transaktionen gelten die sog. *ACID-Eigenschaften*, die nachfolgend kurz erläutert werden:

Atomarität (Atomicity): Eine Transaktion wird entweder vollständig durchgeführt oder sie hat, bei Auftreten eines Fehlers, keinerlei Wirkung.

Konsistenz (Consistency): Eine Transaktion überführt die Datenbasis von einem konsistenten Zustand in einen (neuen) konsistenten Zustand. Möglicherweise inkonsistente Zwischenergebnisse sind nach außen nicht sichtbar.

Isolation: Parallel ausgeführte Transaktionen beeinflussen sich nicht gegenseitig, d.h. sie laufen in einem „logischen Einbenutzerbetrieb“ ab.

Durchgesetzt wird diese logische Isolation einer TA beispielsweise mit Hilfe von Sperrverfahren. Die Serialisierbarkeit einer TA dient dabei als Korrektheitskriterium, wobei die Einhaltung unterschiedlicher Isolationsgrade vor Mehrbenutzeranomalien verschiedenartiger Ausprägung schützt.

Dauerhaftigkeit (Durability): Nach erfolgreichem Abschluss einer TA sind ihre Änderungen dauerhaft gespeichert. In Fehlersituationen muss diese Eigenschaft mit Hilfe von Logging- und Recovery-Maßnahmen durchgesetzt werden.

Das Transaktionsparadigma bildet ein Verarbeitungsrahmen für die Einhaltung semantischer Integritätsbedingungen. Sind diese nach Ausführung der TA verletzt, müssen die von der TA vorgenommenen Änderungen wieder rückgängig gemacht werden.

Wesentliche Maßnahmen des Transaktionssystems zur Sicherung der Datenintegrität betreffen daher das Logging und Recovery zur Wiederherstellung des jüngsten transaktionskonsistenten Zustands und damit der Einhaltung von Atomarität und Dauerhaftigkeit sowie die Synchronisation nebenläufiger TA zur Sicherung der Isolation.

2.3.2 X/Open DTP

X/Open Distributed Transaction Processing (DTP) [Ope96] beschreibt ein Modell der Open Group für die verteilte Transaktionsverarbeitung in offenen Systemen. Die im Jahre 1993 erstmals veröffentlichte Spezifikation definiert Schnittstellen und Protokolle sowie vier grundlegende, funktionale Komponenten (siehe auch Abb. 2.7):

Anwendungsprogramm: Mit Aufrufen an den Transaktionsmanager startet die *Anwendung* (Application Program, AP) eine Transaktion und leitet auch deren Beendigung oder Abbruch ein. Angeforderte Dienste eines (entfernten) Ressourcenmanagers werden innerhalb eines transaktionssicheren Kontextes ausgeführt.

Transaktionsmanager: Der *Transaktionsmanager* (TM) erstellt einen Kontext, verwaltet die Transaktion und koordiniert ihren Abschluss oder ihre Rücksetzung.

Ressourcenmanager: Die *Ressourcenmanager* (RM) gewähren den Zugang zu den eigentlichen Daten. Sie repräsentieren beispielsweise Datenbanken oder Dateisysteme.

Kommunikationsmanager: Im Fall einer verteilten Anwendung übernehmen *Kommunikationsmanager* (Communication Resource Managers, CRM) die Abwicklung der Kommunikation zwischen mehreren TMs.

Die wesentlichen Schnittstellen des Modells sind das TX-Interface zwischen Applikation und Transaktionsmanager sowie das XA-Interface zwischen Transaktionsmanager und Ressourcenmanager. Für verteilte Anwendungen wird zusätzlich das XA+-Interface zwischen TM und CRMs eingesetzt.

Der Ablauf einer Transaktion nach X/Open DTP wird durch die Anwendung initiiert, indem sie den lokalen Transaktionsmanager aufruft. Dieser erstellt einen Transaktionskontext, durch den nachfolgende Aufrufe und Protokollnachrichten eindeutig einer Transaktion zugeordnet werden können. Wenn die Anwendung den Dienst eines Ressourcenmanagers anfordert, wird dieser Kontext übergeben und der RM kann sich mit

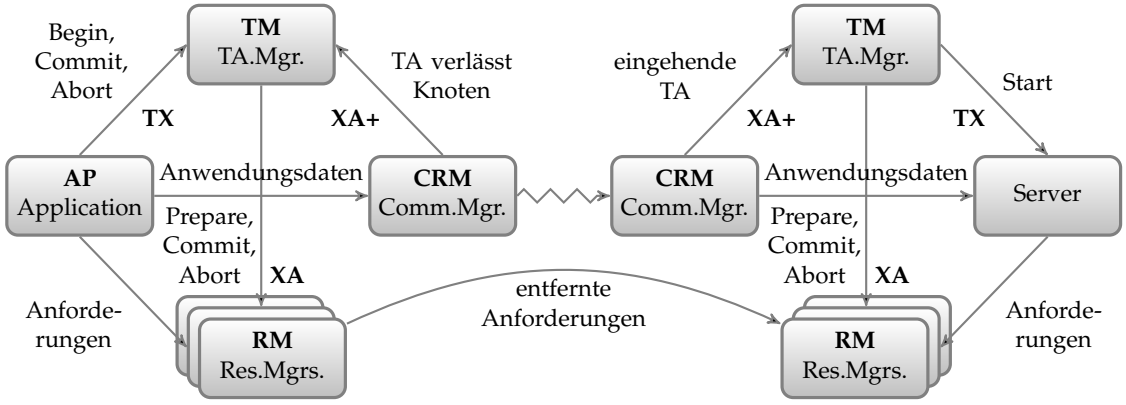

Abbildung 2.7: Das X/Open-DTP-Modell (nach [GR93]).

Hilfe des Kontextes beim TM registrieren. Fordert die Anwendung oder der Ressourcenmanager einen entfernten Dienst an, wird die Kommunikation mit Hilfe der Kommunikationsmanager abgewickelt. Auf diese Weise wird eine Hierarchie von TMs gebildet, die jeweils ihre lokalen RMs kontrollieren. Der TM der initiierenden Anwendung dient als übergeordneter Koordinator und leitet auf deren Veranlassung hin das hierarchische 2PC-Protokoll ein, um die Transaktion konsistent zu beenden. Sind nur lokale RMs beteiligt, wird das einfache 2PC-Protokoll verwendet. Eine Beschreibung des 2PC-Protokolls wird im nächsten Abschnitt gegeben (siehe Kapitel 2.3.3).

Für eine ausführliche Darstellung der Schnittstellen und Protokolle von X/Open DTP sei neben der Spezifikation [Ope96] auch auf Gray und Reuter [GR93] verwiesen.

2.3.3 Zwei-Phasen-Commit-Protokoll (2PC)

Zur Erfüllung der Atomaritätseigenschaft von (verteilten) Transaktionen wird das *Zwei-Phasen-Commit-Protokoll* (2PC) eingesetzt. Das Protokoll stellt sicher, dass alle Teilnehmer einer Transaktion die durchgeführten Änderungen entweder festschreiben (Commit) oder zurücksetzen (Rollback). Es ist auch für zentralisierte Systeme relevant, um einen Konsens unter verschiedenen Prozessen herbeizuführen.

Die Durchführung des 2PC-Protokolls wird durch einen Koordinator gesteuert, der durch einen der beteiligten Knoten gestellt wird. Meistens handelt es sich dabei um den Inititator der Transaktion. Der Ablauf der beiden Protokollphasen wird im Folgenden kurz geschildert (vgl. [GMUW02]):

Phase I: Diese sog. Voting- oder Prepare-Phase fordert die Teilnehmer einer Transaktion T auf, zu einer Entscheidung bzgl. Commit oder Abort zu kommen.

1. Der Koordinator leitet den Abschluss von T ein, indem er eine $\langle$`Prepare` $T\rangle$-Nachricht an alle Teilnehmer versendet.
2. Nach Erhalt von $\langle$`Prepare` $T\rangle$ entscheiden die Teilnehmer, ob sie die Transaktion T abschließen können. Möglicherweise muss ein Teilnehmer noch lokale

Aktivitäten abwarten, bevor er sich in seiner Entscheidung festlegen kann. Die Teilnehmer senden nun entweder

- $\langle$`Ready` $T\rangle$, wenn sie einem Commit von T zustimmen oder
- $\langle$`Failed` $T\rangle$, wenn sie die Transaktion T abbrechen wollen. Nach dem Versenden der Failed-Nachricht bricht ein Teilnehmer unmittelbar ab.

Phase II: Die zweite Phase beginnt, nachdem alle Teilnehmer ihre Antwort übermittelt haben. Sendet ein Teilnehmer keine Antwort, wird dies nach Verstreichen eines angemessenen Timeouts als Abbruch gewertet.

1. Der Koordinator sendet
   - $\langle$`Commit` $T\rangle$, wenn er von allen Teilnehmern $\langle$`Ready` $T\rangle$ erhalten hat oder
   - $\langle$`Abort` $T\rangle$, wenn er von mindestens einem Teilnehmer $\langle$`Failed` $T\rangle$ erhalten hat.
2. Alle Teilnehmer beenden die Transaktion entsprechend der Anweisung des Koordinators, quittieren dies im Falle von $\langle$`Commit` $T\rangle$ mit einer abschließenden Bestätigung und geben ihre Sperren wieder frei.

Das 2PC-Protokoll ist ein blockierendes Protokoll, da es Teilnehmer blockiert, die einem Commit zugestimmt haben, jedoch bislang kein Commit vom Koordinator erhalten haben. Dies kann auftreten, wenn eine Nachricht durch einen Netzwerkfehler verloren geht oder einer der Kommunikationspartner zwischenzeitlich ausgefallen war, so dass er eine Nachricht nicht entgegennehmen konnte. Um in derartigen Fehlersituationen eine geeignete Recovery durchführen zu können, vermerkt jeder Teilnehmer seine Zustandsübergänge sicher in einem Log.

In der hierarchischen Ausprägung des 2PC-Protokolls sind die Teilnehmer baumartig strukturiert und die Kommunikation eines Teilnehmers findet nur mit seinen direkt übergeordneten bzw. untergeordneten Knoten statt. Das Besondere ist, dass die Knoten, die weder Wurzel noch Blatt des Baumes sind, nun zwei Rollen einnehmen müssen. Gegenüber ihrem Elternknoten verhalten sie sich wie ein gewöhnlicher Teilnehmer und gegenüber ihren Kindknoten übernehmen sie die Rolle des Koordinators. Der Wurzelknoten ist der Hauptkoordinator, der das 2PC-Protokoll unverändert ausführt. Die Subkoordinatoren leiten die Protokollnachrichten an ihre Kindknoten weiter und aggregieren deren Commit-Entscheidungen zu einer Antwort an den jeweils übergeordneten Koordinator.

Eine sehr ausführliche Darstellung des 2PC-Protokolls und seiner Variationen findet sich in [WV02].

2.3.4 Diskussion von ACID-Transaktionen

ACID-Transaktionen entsprechen dem Modell der *flachen Transaktion*. Sie können nicht weiter strukturiert werden und müssen daher in Fehlersituationen komplett zurückgesetzt werden [GR93]. Außerdem werden Sperren zur Absicherung parallel ausgeführter

TA erst nach dem Commit wieder freigegeben. Diese beiden Eigenschaften sichern die Atomarität und Isolation, was für kurzlebige TA in zentralisierten Systemen genau so gewünscht ist. Zur Sicherung der ACID-Eigenschaften in verteilten Umgebungen kann das oben vorgestellte X/Open-DTP-Modell zusammen mit dem 2PC-Protokoll eingesetzt werden.

Für langlebige Aktivitäten ist es dagegen nicht akzeptabel, dass die während einer Transaktion verrichtete Arbeit im Fehlerfall komplett verloren geht. Außerdem kann das Sperren von Ressourcen für die gesamte Dauer einer TA dazu führen, dass es zu einem Leistungseinbruch durch Sperrkonflikte und Verklemmungen kommt.

In der Literatur werden diverse Transaktionsmodelle vorgestellt, die sich hauptsächlich in ihrer internen Struktur und der damit verbundenen Granularität bzgl. bestimmter Konsistenzeigenschaften unterscheiden (siehe [GR93], [WV02]). Nachfolgend werden die für diese Arbeit relevanten Modelle vorgestellt.

2.3.5 Geschachtelte Transaktionen

Die Aufteilung einer Transaktion in eine baumförmige Hierarchie aus Subtransaktionen (Sub-TA) wird als *geschlossen geschachtelte Transaktion* bezeichnet. Die Wurzel des Baumes bildet dabei die sog. Top-Level-Transaktion (TL-TA) [GR93].

Das lokale Commit einer Sub-TA hat hierbei zunächst keine Wirkung, außer dass die Ergebnisse dem Elternknoten sichtbar gemacht werden (die Sperren werden weiterhin gehalten). Umgekehrt führt ein Rollback dazu, dass alle Kinder einer Sub-TA ebenfalls zurückgesetzt werden – nicht jedoch die gesamte TA! Das Commit der TL-TA erfolgt erst, wenn alle Sub-TA erfolgreich abgeschlossen haben. Für die TL-TA gelten damit weiterhin ACID-Bedingungen, so dass sich bzgl. der Problematik flacher TA nichts ändert.

Dies verbessert sich erst, wenn die Commit-Regel für Sub-TA die vorzeitige Sperrfreigabe erlaubt. Ergebnisse werden dann vor Abschluss der TL-TA sichtbar, d.h. die Isolation ist gelockert. In diesem Fall spricht man von einer *offenen Schachtelung*.

Die frühzeitige Sperrfreigabe senkt zwar die Wahrscheinlichkeit von Konflikten, macht jedoch ein Rollback unmöglich, da zwischenzeitlich andere Transaktionen die Ergebnisse verändert haben könnten. Daher wird im Fehlerfall das Konzept der *Kompensation* angewandt. Kompensierende Transaktionen setzen die Effekte einer Transaktion auf semantischer Ebene zurück, d.h. es wird eine inverse Operation ausgeführt, um die Wirkung der Transaktion rückgängig zu machen. Problematisch sind dabei insbesondere Aktivitäten, die Auswirkungen in der realen Welt haben, welche nicht umkehrbar sind.

2.3.6 Flex-Transaktionen

Durch die Integration mehrerer autonomer und möglicherweise heterogener Datenbanksysteme (DBS) werden sog. *Multidatenbanksysteme* (MDBS) gebildet. Die Struktur einer MDBS-Transaktion unterteilt sich hierbei in die globale Ebene des MDBS und in die lokale Ebene der DBS-Knoten, d.h., die globale Transaktion ist zweifach geschachtelt.

Da die erfolgreiche Beendigung einer globalen Transaktion wesentlich von den autonomen DBS abhängt, ist eine größere Unabhängigkeit von einzelnen Knoten gewünscht. Mit den flexiblen Transaktionen, auch *Flex-Transaktionen* genannt, schlagen Elmargarmid et al. in [ELLR90] daher ein erweitertes Transaktionskonzept vor, mit dem Subtransaktionen genauer spezifiziert werden können, um so auch in Fehlersituationen ein deutlich robusteres Transaktionsverhalten durchzusetzen. Im Folgenden werden die Aspekte der Erweiterungen vorgestellt.

Funktionsreplikation

Die Autoren gehen davon aus, dass für Subtransaktionen häufig funktionsäquivalente Alternativen zur Verfügung stehen, die bei Auftreten eines Fehlers oder bei Zeitüberschreitungen als gleichwertiger Ersatz dienen können. Das Vorhandensein entsprechender Sub-TA muss vom Nutzer in einer Transaktion angegeben werden. Hierbei soll es auch möglich sein, Präferenzen für Sub-TA festzulegen, so dass die Reihenfolge, in der substituierende Sub-TA eingesetzt werden, durch eine Ordungsrelation gegeben ist.

Auf diese Weise ist es möglich, die Auswirkungen einer abgebrochenen Sub-TA gering zu halten und ein Scheitern der globalen Transaktion zu vermeiden. Die globale Transaktion zeigt somit ein fehlertolerantes Verhalten.

Gemischte Transaktionen

Ein weiterer Vorschlag besteht darin, innerhalb derselben globalen Transaktion sowohl kompensierbare als auch nicht-kompensierbare Subtransaktionen zuzulassen. Eine Sub-TA kann bei diesen *gemischten Transaktionen* mit einer kompensierenden Transaktion assoziiert werden, die deren Effekt semantisch rückgängig macht. Dies hat den Vorteil, dass individuelle Ergebnisse frühzeitig sichtbar gemacht werden und die DBS-Knoten gehaltene Sperren wieder freigeben können. Die Atomarität der globalen Transaktion wird hiermit aufgehoben und auch das Granulat der Isolation wird auf das Niveau der Untertransaktionen gesenkt. Kompensierbare Sub-TA werden somit offen geschachtelt. Mehrotra et al. [MRSK92] klassifizieren Subtransaktionen wie folgt:

kompensierbar: Eine abgeschlossene Untertransaktion kann durch eine kompensierende Transaktion semantisch rückgängig gemacht werden.

wiederholbar: Eine Subtransaktion kann durch wiederholtes Ausführen möglicherweise doch noch zu einem erfolgreichen Abschluss gebracht werden.

pivot: Die Subtransaktion ist weder wiederholbar noch kompensierbar.

Für die Ausführungsreihenfolge dieser Transaktionstypen schlagen die Autoren vor, zuerst die kompensierbaren TA auszuführen, da sich diese rückgängig machen lassen. Nach der Ausführung der Pivot-Transaktionen kann die globale TA dagegen nicht mehr zurückgesetzt werden – sie bilden also gewissermaßen den Wendepunkt in der Ausfüh-

rungsreihenfolge. Die als letztes ausgeführten wiederholbaren TA können nun solange aufgerufen werden, bis die globale TA erfolgreich beendet worden ist.

Temporale Bedingungen

Bei Multidatenbanksystemen entscheidet ein DBS-Knoten autonom, wann er die Sub-TA ausführt. Seitens der globalen Transaktion können jedoch zeitliche Anforderungen bestehen, die für ein erfolgreiches Abschließen der TA eingehalten werden müssen.

Es lassen sich verschiedene temporale Bedingungen formulieren. Zunächst kann mit Hilfe eines temporalen Prädikats der genaue Ausführungszeitpunkt einer Sub-TA angegeben werden. Mit einer zeitlichen Wertefunktion lässt sich zusätzlich ausdrücken, wann der Abschluss einer Transaktion noch sinnvoll ist bzw. ab welcher zeitlichen Schranke die Ausführung der TA unnütz ist. Darüber hinaus kann über einfache Ordnungsrelationen die Ausführungsreihenfolge von Subtransaktionen festgelegt werden.

Anhand dieser zeitlichen Vorgaben für die Transaktionen ist es möglich, einen Ablaufplan für die optimale Abarbeitung der Transaktionen aufzustellen.

2.3.7 Business-Transaktionen

Die transaktionale Ausführung von Workflows (siehe Kap. 2.2) wird auch als *Business-Transaktion* bezeichnet. Business-Transaktionen können alle bisher vorgestellten Konzepte in sich vereinen und erfordern größtmögliche Flexibilität bei der Koordinierung. Sheth und Rusinkiewicz [SR93] sowie Alonso et al. [AAA+96] diskutieren mögliche Anforderungen an transaktionale Workflows. Zusammenfassend werden folgende Eigenschaften genannt:

- Offene Schachtelung und dadurch Lockerung der Isolation sowie Verwendung von Kompensationsmechanismen.
- Komposition unterschiedlicher Transaktionstypen: kurz- oder langlebig, ACID-TA oder Flex-TA, usw.
- Abhängigkeiten zwischen Subtransaktionen, die entweder von vornherein feststehen oder sich dynamisch ergeben.
- Ausführung von Subtransaktionen nur unter bestimmten Voraussetzungen.
- Selektives Abschließen oder Zurücksetzen bzw. Kompensieren von Sub-TA, also Lockerung der Atomarität und gemischte Ergebnisse.

Business-Prozesse sind in der Regel langlebig und bestehen aus vielen autonomen Teilnehmern, die zwischen zwei Aufrufen nicht kontrolliert werden können. Sie laufen in verteilten Umgebungen ab und koppeln daher meistens heterogene Komponenten. Vor diesem Kontext müssen Transaktionen als Koordination autonomer Teilnehmer verstanden werden [AAA+96].

2.4 Web-Services und Transaktionen

Mit dem Aufkommen serviceorientierter Architekturen und ihrem zunehmenden Einsatz in betrieblichen Anwendungen, wird von Web-Services auch die transaktionssichere Durchführung von Arbeitsabläufen verlangt. Um diese Aufgabe leisten zu können, sind zwingend Mechanismen erforderlich, mit denen die Korrelation von Nachrichten und Aktivitäten der lose gekoppelten Dienste realisiert werden kann.

Zusätzliche Relevanz hat die Umsetzung unterschiedlicher Transaktionsformen. Neben der Unterstützung kurzlebiger ACID-Transaktionen konzentrieren sich die Anforderungen dabei vor allem auf die transaktionale Absicherung langlebiger Geschäftsprozesse. Die in diesen Bereichen etablierten Spezifikationen des Web-Services-Umfelds werden nachfolgend kurz vorgestellt.

2.4.1 BTP – Business Transaction Protocol

Das *Business Transaction Protocol* (BTP) [FDF+04] von OASIS liegt in der aktuellen Version 1.1 seit 2004 vor. BTP verwendet XML und kann daher auch mit SOAP eingesetzt werden. Allerdings wurde es nicht speziell für Web-Services entwickelt [LF03].

Das mit BTP definierte Protokoll hat zwei Phasen mit den Koordinator-Nachrichten `Prepare` und `Confirm` bzw. `Cancel`. Damit trotz eines einzigen Protokolls zwischen ACID-Transaktionen und langlebigen Transaktionen unterschieden werden kann, werden zusätzlich zwei Transaktionstypen definiert:

Atom: Transaktionen dieses Typs erfüllen die Atomaritätseigenschaft in dem Sinne, dass die Teilnehmer konsistent mit `Confirm` oder `Cancel` benachrichtigt werden. Aufgrund unterschiedlicher Implementationsmuster (siehe nachfolgender Abschnitt), impliziert dies jedoch keine Transaktion nach ACID-Kriterien!

Cohesion: Dieser Transaktionstyp eignet sich besonders für langlebige Transaktionen, da durch Lockerung der Isolation ein vorzeitiger Abschluss möglich wird. Zusätzlich wird die Atomarität aufgehoben, so dass den Teilnehmern derselben TA sowohl ein Abbruch mit `Cancel` als auch die erfolgreiche Beendigung mit `Confirm` gestattet wird. Die Entscheidung hierüber trifft der Koordinator mit Hilfe der Anwendungslogik.

Das Protokoll unterstützt im Wesentlichen die Rollen *Superior* und *Inferior* sowie einige davon abgeleitete Rollen. Der Superior entspricht dem Koordinator und der Inferior einem gewöhnlichen Teilnehmer. Mit Hilfe der anderen Rollen wird unter anderem die *Interposition* von Koordinatoren entsprechend eines hierarchischen Transaktionsbaumes realisiert.

Externe Effekte und Implementationsmuster

In der BTP-Spezifikation wird betont, dass Teilnehmern bzgl. der Implementation des Transaktionsabschlusses keine Vorgaben gemacht werden. Je nach Art der Anwendung kann unterschiedliches Commit-Verhalten der Teilnehmer bei Eintreffen der `Prepare`-, `Confirm`- und `Cancel`-Nachrichten sinnvoll sein. Besonders wenn die externen Auswirkungen einer Business-Transaktion schwierig durch eine Kompensation zu revidieren sind, ist ein vorläufiger Zustand wünschenswert, der einfacher zurückzusetzen ist.

Es werden diverse Implementationsmuster sowie die internen Effekte der Teilnehmer auf ankommende Nachrichten motiviert. Tabelle 2.3 zeigt diese Muster, wie sie von der Firma Choreology zusammengefasst werden (vgl. [Fur04], [FG05]). Die Implementation von ACID-Kriterien kann hierbei als Reinform von Provisional-Final angesehen werden.

Tabelle 2.3: Implementationsmuster aus BTP.

Muster	`Prepare`	`Confirm`	`Cancel`
Provisional-Final	do, mark pending	mark final	delete
Validate-Do	validate, log	do	forget
Do-Compensate	do, log	forget	reverse

Im Rahmen einer Warenbestellung ist es für einen Lieferanten beispielsweise angebracht, die Datenänderungen bei einem `Prepare` als „vorübergehend“ zu markieren und erst im Falle eines positiven Abschlusses (`Confirm`) als endgültiges Ergebnis festzuschreiben. Bei einer `Cancel`-Nachricht können die Änderungen dann einfach gelöscht werden. Dieses interne Verhalten wird durch das Muster Provisional-Final ausgedrückt.

Der reine Kompensationsansatz läuft dagegen nach dem Muster Do-Compensate ab. Dabei werden bei Eingang einer `Prepare`-Nachricht alle Änderungen komplett durchgeführt und zusätzliche Informationen gespeichert, die im Falle einer `Cancel`-Nachricht für die Kompensation herangezogen werden. Die `Confirm`-Nachricht löst schließlich eine Löschung der Kompensationsinformation aus, wobei keine Änderung der eigentlichen Daten mehr erfolgt.

Für den Initiator und das Protokoll sind alle Implementierungen äquivalent, jedoch können sich zwischen den Vertragspartnern, die durch die Dienste repräsentiert werden, im Falle eines Abbruchs unterschiedlich gravierende Folgen ergeben. Diese Thematik wird später wieder aufgegriffen und die Implementationsmuster werden dann unter dem Gesichtspunkt der Koordination genauer untersucht (vgl. Abschnitt 3.2.3).

2.4.2 WS-CAF – Web Services Composite Application Framework

Das *Web Service Composite Application Framework* (WS-CAF) [BCH+03] wurde speziell für den Einsatz mit Web-Service konzipiert und basiert daher auf WSDL und SOAP. Ursprünglich geht die Entwicklung u.a. auf die Firmen Arjuna, Oracle und Sun zurück; sie liegt jedoch inzwischen in der Verantwortung eines OASIS-Gremiums. Die letzten Ände-

rungen der Dokumente stammen aus den Jahren 2005-2007 und definieren die aktuelle Version 1.0. WS-CAF ist unterteilt in drei Spezifikationen:

- *Web Services Context* (WS-Context) [LNP07] stellt ein leichtgewichtiges Framework zur Kontexterstellung und -verwaltung sowie zur Erzeugung von Aktivitäten bereit. Der Kontext enthält neben der obligatorischen Kontext-Id weitere Elemente, wie eine optionale Liste von Sub-Aktivitäten oder die Gültigkeitsdauer der Kontextinformation. Über die eindeutige Kontext-Id gelingt die Zuordnung von Teilnehmern zu Aktivitäten. Ferner kann der Kontext anwendungspezifische Daten aufnehmen, die für die Korrelation benötigt werden, wie beispielsweise die URI des Koordinators einer Transaktion.

- *Web Services Coordination Framework* (WS-CF) [LNP05] basiert auf WS-Context und bietet Mechnanismen zur Kontexterweiterung, eine zuverlässige Nachrichtenübermittlung sowie die Koordinationsfunktionalität von WS-CAF. Hierfür sind in WS-CF die Rollen eines Koordinators und der Teilnehmer vorgesehen. Daneben gibt es den Koordinationsdienst, der unterschiedliche Protokolle zur Transaktionsbeendigung, wie beispielsweise 2PC, einsetzen kann. WS-CF unterstützt auch die Interposition von Koordinatoren.

- *Web Services Transaction Management* (WS-TXM) bündelt drei verschiedene Transaktionsspezifikationen, die die Funktionalität von WS-Context und WS-CF einsetzen und für verschiedene Transaktionsmodelle nutzbar macht. Der nächste Abschnitt gibt einen Überblick über die Transaktionstypen.

Der modulare Aufbau von WS-CAF ermöglicht es Anwendungen einen genau auf ihre Anforderungen zugeschnittenen Funktionsumfang zu verwenden. Insbesondere kann durch die ausschließliche Nutzung von WS-Context ein Kontext verwaltet werden, ohne dass eine Koordination benötigt wird.

WS-TXM – Web Services Transaction Management

Web Services Transaction Management (WS-TXM) fasst die folgenden Spezifikationen zur Transaktionsunterstützung zusammen: *Web Services ACID* (WS-ACID) [LNP06a], *Web Services Long Running Action* (WS-LRA) [LNP06c] und *Web Services Business Process* (WS-BP) [LNP06b]. Im Folgenden werden diese kurz erläutert:

WS-ACID: Dieser Transaktionstyp implementiert die ACID-Kriterien unter Nutzung des 2PC-Protokolls. Eine Besonderheit ist die für Teilnehmer bestehende Möglichkeit, nach erfolgreicher Prepare-Phase autonom entscheiden zu können, ob sie ein Commit oder ein Rollback durchführen. Dies kann zu sog. heuristischen Ergebnissen führen, wenn die Teilnehmer zu einer anderen Entscheidung kommen als der Koordinator. Um dennoch die Atomarität der ACID-Anforderung zu gewährleisten, muss der Koordinator alle Teilnehmer zu einem konsistenten, atomaren Abschluss dirigieren. Teilnehmer, die für sich bereits eigenmächtig die zweite Phase

des 2PC-Protokolls eingeleitet haben, müssen daher in der Lage sein, ihre Aktionen entsprechend zu revidieren.

WS-LRA: Mit diesem Transaktionstyp werden langlebige Aktivitäten realisiert, deren Teilaufgaben kompensierbar sein müssen. Die Kompensation geschieht mit Hilfe eines *Compensators*, den ein beteiligter Service zusätzlich beim Koordinator registrieren muss.

WS-BP: Dieses Modell vereint kurzlebige ACID-Transaktionen und langlebige Prozesse zu Workflows. Alle Teilnehmer werden Geschäftsdomänen zugeordnet und die Koordination beschränkt sich auf den Austausch zwischen diesen Domänen. Jede Domäne besitzt ihren eigenen Koordinator, so dass hier das Prinzip der Interposition zur Anwendung kommt. Auf Grund der Kombination verschiedener Transaktionstypen beinhaltet die Propagation globaler Koordinatorentscheidungen in die Domänen der Subkoordinatoren also im Wesentlichen eine Abbildung auf domänenspezifische Transaktionstypen (WS-ACID, WS-LRA).

Das Hauptziel von WS-CAF ist die Unterstützung verschiedener Transaktionsmodelle, wobei sowohl Orchestrierungen mit BPEL als auch Web-Service-Choreographien möglich sein sollen. Durch die schichtweise Realisierung des Frameworks werden eine generische Kontextverwaltung und Dienstkoordinierung geschaffen, die sich anwendungsspezifisch nutzen lassen. Die Spezifikationen bauen also aufeinander auf und erhöhen so schrittweise die Mächtigkeit des Frameworks.

2.4.3 WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity

WS-Coordination (WS-C) [FJ07] beschreibt ein erweiterbares Framework, um verteilten Anwendungen die Korrelation ihrer Aktivitäten zu ermöglichen. Die Spezifikationen *WS-AtomicTransaction* (WS-AT) [LW07] und *WS-BusinessActivity* (WS-BA) [FL07] definieren darauf basierend zwei Koordinationstypen, die dieses Framework nutzen und Protokolle zur Abwicklung kurz- und langlebiger Transaktionen zur Verfügung stellen.

Historisch geht die Entwicklung auf Entwürfe von IBM, Microsoft und BEA zurück, die schließlich bei OASIS eingereicht und mittlerweile als Standard verabschiedet wurden. Häufig wird auch die zusammenfassende Bezeichnung *Web Services Transaction* (WS-TX) für die drei Standards verwendet, die seit Juli 2007 in Version 1.1 vorliegen. WS-TX basiert ebenso wie WS-CAF auf den Beschreibungssprachen WSDL und SOAP.

WS-Coordination

Die Hauptaufgabe von WS-Coordination besteht in der Erstellung und Verwaltung von Koordinationskontexten, die der Zuordnung von Teilnehmern zu Aktivitäten dienen. WS-C spezifiziert zu diesem Zweck einen *Activation-Service*, der die Erzeugung des Kontextes übernimmt sowie einen *Registration-Service*, über den sich Dienste für die Teilnahme an einer Aktivität registrieren können. Desweiteren sind für jeden unterstützten

Koordinationstyp sog. *Protocol-Services* vorgesehen. Diese sind jedoch nicht in WS-C beschrieben, sondern werden über zusätzliche Spezifikationen erklärt. OASIS stellt hierfür die Typen WS-AT und WS-BA bereit (siehe unten). Der beschriebene Aufbau eines Koordinators ist in Abbildung 2.8 dargestellt.

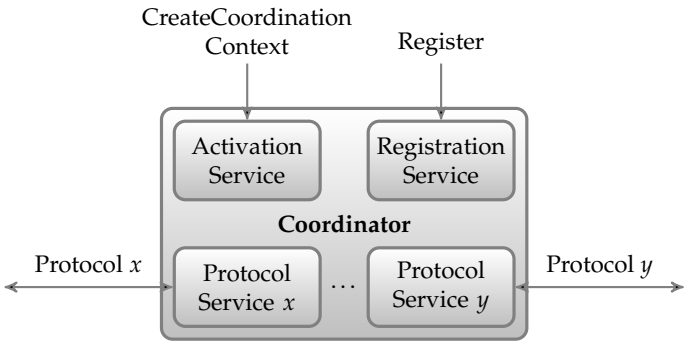

Abbildung 2.8: Aufbau eines Koordinators nach WS-C (aus [FJ07]).

Um eine Koordination zwischen Teilnehmern zu etablieren, sind zwei Schritte erforderlich. Zunächst muss ein Koordinationskontext erstellt werden und anschließend können sich Teilnehmer für diesen Kontext registrieren. Das Sequenzdiagramm in Abb. 2.9 veranschaulicht den zugehörigen Nachrichtenfluss zwischen den beteiligten Akteuren.

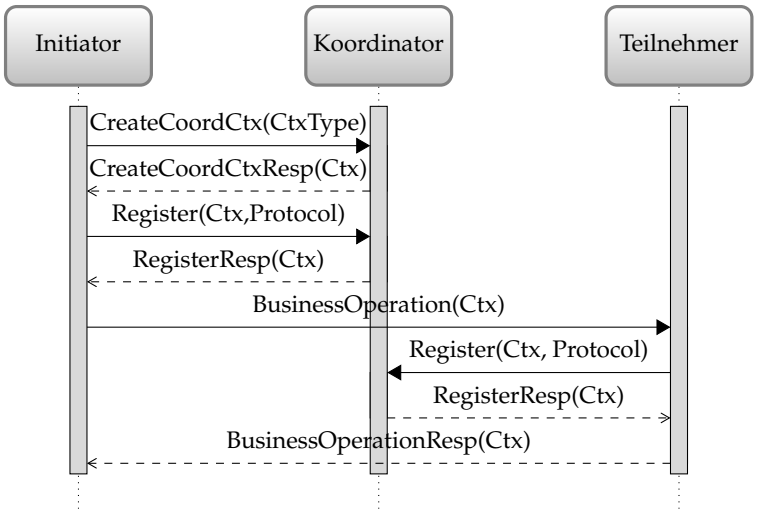

Abbildung 2.9: Sequenzdiagramm WS-Coordination.

Die Kontexterstellung wird durch den Initiator einer Transaktion unter Angabe des Koordinationstyps mit einer `CreateCoordinationContext`-Nachricht am Activation-Service des Koordinators ausgelöst. Der Koordinator erzeugt daraufhin eine neue Aktivität des angegebenen Typs und sendet den korrespondierenden Kontext im SOAP-Body an die Anwendung (Initiator) zurück.

Nachfolgend wird dieser Kontext von der Anwendung im SOAP-Header der Nachrichten an andere Dienste propagiert, um sie in den koordinierten Ablauf mit einzubeziehen. Teilnehmer treten dann der Aktivität bei, indem sie sich für ein bzgl. des Koordina-

tionstyps zulässiges Protokoll anmelden. Sie senden dazu eine entsprechend parametrisierte `Register`-Nachricht an den Registration-Service des Koordinators. Der Endpunkt dieses Services ist als URI im Kontext enthalten.

Teilnehmer können sich auch bei einem anderen als den im Kontext angegebenen Koordinator anmelden. Dazu übermitteln sie den vorhandenen Kontext als `Current-Context`-Element an einen Koordinator ihrer Wahl, der sich daraufhin als Teilnehmer bei dem im Kontext angegebenen Registration-Service registriert (Interposition).

Der Initiator meldet sich üblicherweise für ein Completion-Protokoll an, mit dem später die Transaktionsbeendigung eingeleitet und das Resultat an die Anwendung übermittelt wird (nicht abgebildet).

Kontext Der eindeutige Koordinationskontext dient der Kennzeichnung zusammengehöriger Nachrichtenfolgen. Um Nachrichten einer Konversation entsprechend identifizieren zu können, wird der Kontext daher stets im SOAP-Header mitgesendet. Der Kontext enthält die folgenden, obligatorischen Elemente (vgl. [CCJL04]).:

- `Identifier`: Gibt eine Id zur Identifikation der Aktivität an, deren Struktur nicht weiter definiert wird. Der Koordinator muss bei ihrer Erzeugung für die Eindeutigkeit der Id sorgen.
- `CoordinationType`: Der Koordinationstyp wird im Kontext als eindeutige URI angegeben und gruppiert für diesen Typ zulässige Protokolle. Von OASIS vordefinierte Koordinationstypen sind WS-AtomicTransaction und WS-BusinessActivity.
- `RegistrationService`: Dieses Element spezifiziert den Registration-Service des zuständigen Koordinators als Endpunktreferenz. Siehe auch WS-Addressing im Abschnitt 2.1.1.

Darüber hinaus ist das optionale Sub-Element `Expires` erlaubt, das die Gültigkeitsdauer des Kontextes in Millisekunden angibt sowie weitere, anwendungsspezifische Elemente. Listing 2.3 zeigt einen Kontext der realisierten Implementation.

Listing 2.3: Koordinationskontext mit `participantId` als zusätzlichem Element.

```
1  <coord:CoordinationContext
2      xmlns:coord="http://docs.oasis-open.org/ws-tx/wscoor/2006/06">
3    <coord:Identifier>
4      urn:uuid:3B68E538-D58F-4ABB-BC2D-91F1ACC1B7B4
5    </coord:Identifier>
6    <coord:Expires>0</coord:Expires>
7    <coord:CoordinationType>
8      http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome
9    </coord:CoordinationType>
10   <coord:RegistrationService>
11     <addressing:Address
```

```
12          xmlns:addressing="http://www.w3.org/2005/08/addressing">
13        http://localhost:8080/axis2/services/RegistrationService
14      </addressing:Address>
15    </coord:RegistrationService>
16    <vsis:participantId
17        xmlns:vsis="http://coordination.vsis.de/participants">
18      2A4A4DE0-8D2B-4373-A382-7C92BFF6D269
19    </vsis:participantId>
20  </coord:CoordinationContext>
```

Es ist wichtig zu verstehen, dass mit WS-Coordination allein kein besonderer Nutzen verbunden ist, da es nur den Rahmen für die Koordinationstypen bildet. Wesentlich an WS-C ist die Erweiterbarkeit über Protocol-Services und zusätzliche Kontextelemente, die mit den Koordinationstypen definiert werden. In den folgenden Abschnitten werden hierzu die Standards WS-AtomicTransaction und WS-BusinessActivity vorgestellt.

WS-AtomicTransaction

WS-AtomicTransaction (WS-AT) [LW07] definiert einen Koordinationstyp auf Basis von WS-Coordination, der die Teilnehmer zu einem einheitlichen Transaktionsabschluss nach ACID-Maßgabe führt. Dafür werden im Kern zwei Protokolle verwendet, die im Folgenden erläutert werden.

Completion-Protokoll Das *Completion-Protokoll* (siehe Abb. 2.10) wird zwischen der Anwendung (Initiator) und Koordinator eingesetzt. Über dieses Protokoll teilt der Initiator dem Koordinator mit, ob die Transaktion abgeschlossen werden soll (Commit) oder ob die TA zurückgesetzt werden soll (Rollback). Der Koordinator übermittelt der Applikation anschließend den Ausgang der Transaktion.

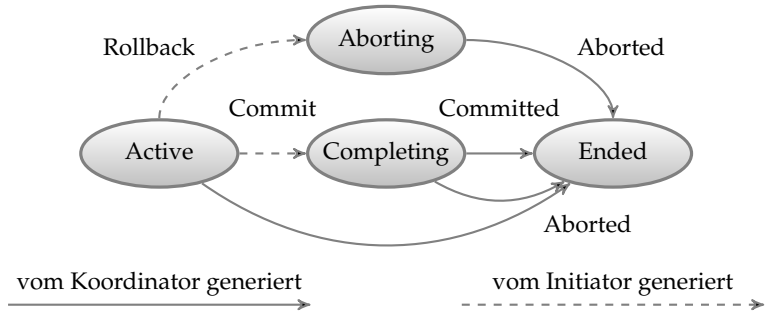

Abbildung 2.10: WS-AtomicTransaction – Completion (vgl. [LW07]).

2PC Das 2PC-Protokoll (vgl. Abschnitt 2.3.3) wird zwischen dem Koordinator und den Teilnehmern eingesetzt, nachdem vom Initiator die Anweisung zur Beendigung der TA gegeben wurde. 2PC unterscheidet zwischen Teilnehmern mit flüchtigem Speicher, wie z.B. Caches, die nur lesend an einer TA teilnehmen und Ressourcen mit permanentem

Speicher und definiert dafür die Protokollvarianten *Volatile 2PC* und *Durable 2PC*. Um den Abschluss der festschreibenden Teilnehmer zu optimieren, treten Teilnehmer des Volatile 2PC zuerst in die Prepare-Phase ein. Die Optimierung besteht darin, dass hierdurch weniger zu koordinierende Teilnehmer für das Durable 2PC existieren, so dass dieses schneller abgewickelt werden kann. Zudem können nur lesende Teilnehmer, sofern sie nicht am Ausgang der TA interessiert sind, frühzeitig mit einer `ReadOnly`-Nachricht aussteigen, da sie ohnehin keine dauerhaften Daten festzuschreiben haben. Anschließend werden die für Durable 2PC registrierten Teilnehmer koordiniert zu einer Terminierung geführt. Die Protokolle werden durch das Zustandsdiagramm in Abb. 2.11 dargestellt.

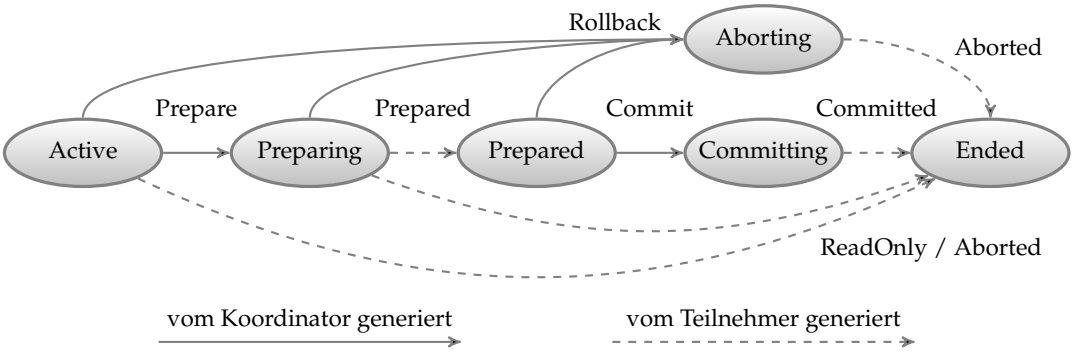

Abbildung 2.11: WS-AtomicTransaction – 2PC (vgl. [LW07]).

WS-BusinessActivity

WS-BusinessActivity (WS-BA) [FL07] erweitert WS-Coordination und erlaubt es, transaktionale Workflows zu koordinieren. Die Abwicklung der langlebigen Prozesse findet in einer offenen Schachtelung statt, so dass ggf. Kompensationsmaßnahmen eingeleitet werden müssen.

WS-BA definiert die beiden Koordinationstypen *AtomicOutcome* (obligatorisch) und *MixedOutcome* (optional) sowie die Protokolle *BusinessAgreementWithParticipantCompletion* (BAPC, siehe Abb. 2.12) und *BusinessAgreementWithCoordinatorCompletion* (BACC, siehe Abb. 2.13). Es sind alle Kombinationen von Koordinationstypen und Protokollen möglich. Die Koordination nach AtomicOutcome fordert alle Teilnehmer auf, entweder einheitlich mit `Close` oder mit `Compensate` abzuschließen. Diese Vorgabe muss bei MixedOutcome nicht eingehalten werden. Innerhalb derselben Aktivität sind hier sowohl Teilnehmer erlaubt, die mit `Closed` abschließen, als auch Teilnehmer, die mit `Compensated` kompensieren.

Die beiden Protokolle BAPC und BACC unterscheiden sich nur dahingehend, dass die Teilnehmer bei BAPC selbstständig mit einer `Completed`-Nachricht signalisieren, wann sie mit ihrer Arbeit fertig sind. Bei BACC muss hingegen der Koordinator initiativ werden und die Teilnehmer zum Beenden ihrer Arbeit mit einer `Complete`-Nachricht auffordern. Daher ist in diesem Fall auch ein weiterer Zustand `Completing` vorgesehen. Wenn sich schließlich alle Teilnehmer im `Completed`-Zustand befinden, kann der

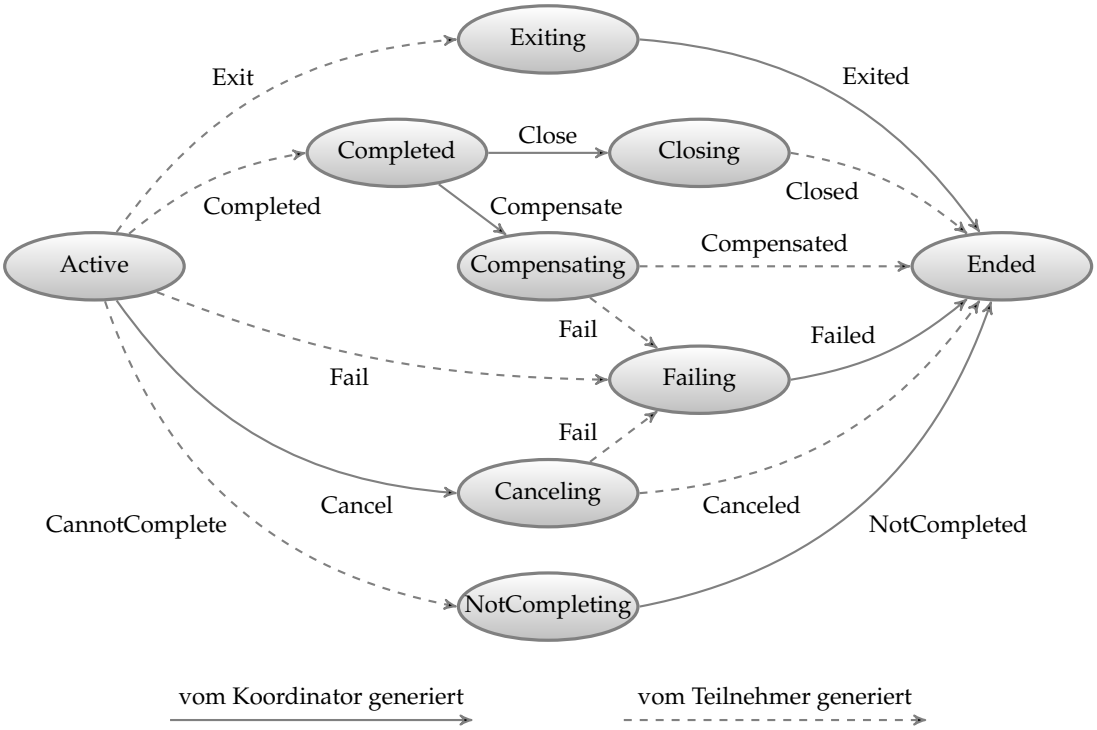

Abbildung 2.12: WS-BusinessActivity – BusinessAgreementWithParticipantCompletion (vgl. [FL07]).

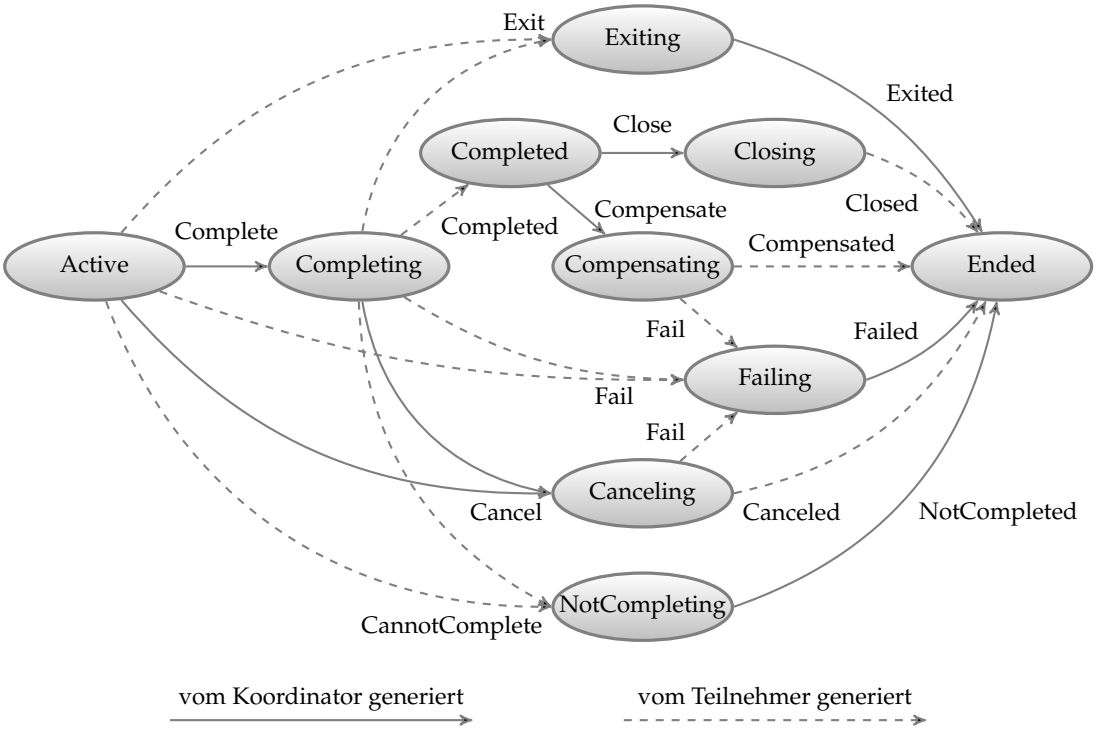

Abbildung 2.13: WS-BusinessActivity – BusinessAgreementWithCoordinatorCompletion (vgl. [FL07]).

Koordinator entscheiden, wie das Protokoll weitergeführt werden muss.

Bemerkenswert ist, dass WS-BA kein Completion-Protokoll spezifiziert, wie es bei WS-AT der Fall ist. Die Kommunikation zwischen Anwendung und Koordinator bleibt also offen, so dass der Koordinator entweder in die Anwendung integriert werden muss oder ein zusätzliches Protokoll implementiert werden muss.

2.4.4 Zusammenfassung und Vergleich

Prinzipiell eignen sich alle vorgestellten Spezifikationen für die transaktionale Koordination von Web-Services. BTP ist jedoch als einziger Standard nicht direkt auf diese Aufgabe zugeschnitten, was in einer möglichen Umsetzung zu erheblichem Mehraufwand führen würde [LF03]. Zudem besteht BTP nur aus einer einzigen Schicht, die einen zweiphasigen Protokollansatz umfasst, mit dem sowohl ACID-Transaktionen als auch Business-Prozesse unterstützt werden. Daher muss die interne Realisierung auf Teilnehmerseite unspezifiziert bleiben. Für die Abwicklung des Protokolls hinsichtlich des unterstützten TA-Modells ist die Implementation eines adäquaten Musters vorgesehen (vgl. Tab. 2.3).

Gegenüber dem monolithischen Ansatz von BTP ist der modulare Aufbau von WS-CAF und WS-C/WS-TX besser geeignet, um benötigte Funktionalitäten fein abgestimmt einzusetzen. Außerdem wird dadurch die Entwicklung zusätzlicher Komponenten erleichtert, so dass BTP für die Realisierung erweiterter Koordinatoren nicht sehr attraktiv erscheint. Hinsichtlich der Transaktionssteuerung bieten alle Spezifikationen einen vergleichbaren Funktionsumfang. Tabelle 2.4 gibt einen Überblick über die drei Standards.

Tabelle 2.4: Vergleich der Spezifikationen (vgl. [LF03] und [Kra04]).

	WS-C, WS-AT, WS-BA	**WS-CAF**	**BTP**
Ursprung	Microsoft, IBM, BEA	Arjuna, Fujitsu, IONA, Oracle, Sun	OASIS
Organisation	OASIS	OASIS	OASIS
Jahr	2007	2005-2007	2004
Version	1.1	1.0	1.1
Technologien	WSDL, SOAP	WSDL, SOAP	XML
Architektur	zwei Schichten	drei Schichten	monolithischer Ansatz
ACID-TA	ja	ja	ja
langlebige TA	ja	ja	ja
Mixed Outcome	ja	ja	ja
Interposition	ja	ja	ja
Muster	ACID, Do-Compensate	ACID, Do-Compensate	alle Muster

Für die Realisierung erweiterter Transaktionskoordinatoren sind auf Grund ihres modularen Aufbaus vor allem WS-CAF und WS-C/WS-TX interessant. Jenseits der recht ausgewogenen Leistungsmerkmale beider Spezifikationen ist zu bemerken, dass nach der Verabschiedung von WS-Context 1.0 als OASIS-Standard im April 2007, die Arbeit des Technischen Kommitees von WS-CAF für beendet erklärt wurde.[6] Im Zuge dessen

[6]Siehe hierzu das letzte Posting auf der WS-CAF-Mailing-Liste unter `http://lists.oasis-open.`

wurden sämtliche Mailing-Listen geschlossen und es sind seither keine Aktivitäten um WS-CAF mehr zu beobachten. Goethals et al. glauben eine Neuorientierung der OASIS-Organisation zu Gunsten von WS-C/WS-TX erkennen zu können, so dass die Zukunft von WS-CAF als „unklar“ bezeichnet werden darf [GSLV07].

Neben der aktiveren Entwicklung in der jüngsten Vergangenheit und der aussichtsreicheren Zukunft von WS-C/WS-TX, muss für diese Arbeit auch berücksichtigt werden, dass sie im Rahmen eines bestehenden Projekts stattfindet, in dem man sich bereits früher für einen Ansatz auf Basis von WS-C/WS-TX entschieden hat und inzwischen einen Koordinator für WS-AT umsetzen konnte.

Die weiterführenden Betrachtungen der nächsten Kapitel beschränken sich daher auf die Standards WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity.

org/archives/ws-caf/200705/msg00000.html.

3 Anforderungsanalyse und Evaluation

Die Koordinierung transaktionaler Workflows in serviceorientierten Umgebungen setzt die Unterstützung kurz- und langlebiger Aktivitäten voraus. Geeignete Spezifikationen wurden in Kapitel 2.4 vorgestellt.

Im Folgenden werden Schwierigkeiten herausgearbeitet, die sich mit dem Einsatz von WS-Coordination und der zugehörigen Koordinationstypen WS-AtomicTransaction und WS-BusinessActivity ergeben. Es werden verschiedene Lösungsansätze untersucht und ihre Eignung hinsichtlich eines autonomen Koordinationsdienstes beleuchtet. Anschließend werden erweiterte Anforderungen der Transaktionssteuerung vorgestellt sowie deren Potential die Autonomie des Koordinators zu erhöhen.

3.1 Analyse von WS-Coordination

Die Transaktionssteuerung der vorgestellten Web-Service-Protokolle basiert im Wesentlichen auf dem X/Open-DTP-Modell, in dem die TA durch einen Initiator gestartet wird und in deren Verlauf auf verteilte Ressourcen zugegriffen werden kann. Ein Koordinator vermittelt zwischen den Teilnehmern und führt die Transaktion zu einem konsistenten Abschluss bzgl. des durchgeführten Koordinationstyps, sobald der Initiator ihn dazu veranlasst. Nachfolgend wird analysiert, in welchen Situationen dieses Modell an seine Grenzen stößt.

3.1.1 Completion-Protokoll

Diesem Abschnitt wird das folgende einfache Szenario einer Reisebuchung zu Grunde gelegt: Im Rahmen einer Urlaubsplanung werden von einem Reisebüro (Initiator) Anfragen an verschiedene Fluggesellschaften gestellt. Das Ziel besteht darin, den preiswertesten Flug zu buchen. Die maßgebliche Anwendungslogik befindet sich im Initiator und entspricht in diesem Fall der trivialen Fallunterscheidung, den niedrigsten Wert für die Flugkosten zu ermitteln. Dementsprechend wird anschließend dieser Flug gebucht, während alle anderen Angebote verworfen werden.

Für WS-Coordination existieren die Koordinationstypen WS-AtomicTransaction und WS-BusinessActivity zur Koordinierung kurzlebiger ACID-Transaktionen bzw. langlebiger Business-Transaktionen (vgl. Abschnitt 2.4.3). Im weiteren soll die Frage erörtert werden, wie sich das Beispiel der Reisebuchung mit diesen Protokollen umsetzen lässt.

Das geforderte Verhalten für WS-AT wäre, die billigste Fluggesellschaft festschreiben zu lassen (Commit) und alle anderen abzubrechen (Abort). Da WS-AT jedoch nur atomare Ergebnisse (AtomicOutcome) zulässt, kann es hier nicht eingesetzt werden. Gemischte Ergebnisse (MixedOutcome) werden dagegen von WS-BA unterstützt, jedoch definiert

WS-BA kein Completion-Protokoll, wie es bei WS-AT der Fall ist. Das Verhalten zwischen Initiator und Koordinator bleibt damit unspezifiziert. Gleichwohl benötigt der Koordinator die Information über festzuschreibende Teilnehmer, da er die Transaktion sonst nicht zu einem Abschluss führen kann. Für dieses Dilemma bestehen grundsätzlich zwei Lösungsmöglichkeiten:

Integrationslösung: Initiator und Koordinator werden zu einer Einheit zusammengefasst. Dadurch hat der Koordinator direkten Zugriff auf die Anwendungsentscheidungen und kann den Transaktionsablauf entsprechend koordinieren. Die Nachteile dieser Lösung bestehen in der engen Kopplung beider Instanzen und der anwendungsspezifischen Realisierung des Koordinators. Dies steht im Widerspruch zum SOA-Gedanken von lose gekoppelten Komponenten, die ihre Funktionalität dienstbasiert erbringen.

Kommunikationslösung: Der Koordinator wird als eigenständige Instanz realisiert und muss mit Hilfe eines Initiator-Koordinator-Protokolls über Anwendungsentscheidungen informiert werden. In [RHR08] werden zwei Möglichkeiten beschrieben, wie dies umgesetzt werden kann:

Proxy-Konzept: Der Koordinator agiert als Vermittler zwischen Initiator und Teilnehmern, indem er die Anweisungen des Initiators auf das Koordinationsprotokoll abbildet. Als Ergebnis liefert er den Ausgang der Transaktion zurück. Eigenständige Entscheidungen trifft der Koordinator hierbei nicht.

Das Completion-Protokoll von WS-AT ist nach diesem Konzept spezifiziert. Für WS-BA wurde dieser Ansatz beispielsweise mit dem *WS-BusinessActivity-Initiator-Protokoll* (WS-BA-I) von Erven et al. umgesetzt (vgl. [EHHZ07]).

Callback-Konzept: Der Koordinator steuert die Transaktionsabwicklung weitestgehend selbst und erfragt im Fall von anwendungsspezifischen Koordinationsentscheidungen über ein Callback-Interface des Initiators das weitere Vorgehen.

Dieser Ansatz sorgt für eine größere Entkopplung des Initiators vom Koordinationsprozess als das Proxy-Konzept, da der Initiator den Prozess nicht aktiv vorantreiben muss. Implementierungen sind nicht bekannt [RHR08].

Die vorgestellten Konzepte gestatten unterschiedliche Grade der Entkopplung von Initiator und Koordinator. Während die Komponenten bei der Integrationslösung nur rein begrifflich getrennt werden, führt die Kommunikationslösung zu einer wirklichen Separation der Akteure hinsichtlich ihrer zugedachten Aufgaben. Das Callback-Konzept bietet dabei in erster Linie technische Vorteile gegenüber dem Proxy-Konzept.

3.1.2 Management des Demarkationsprivilegs

Das Initiator-Koordinator-Modell geht implizit davon aus, dass der Initiator einer Transaktion stets auch derjenige Teilnehmer ist, der den Abschluss der Transaktion einleitet. Diesen Vorgang bezeichnet man auch als *Demarkation*. Dabei ist es irrelevant, ob der Initiator dem Koordinator seine Entscheidung aktiv im Sinne des Proxy-Konzepts oder reaktiv als Beantwortung einer Anfrage über das Callback-Interface mitteilt. Der Initiator bleibt der einzige demarkationsberechtigte Teilnehmer.

Mit dieser Annahme sind jedoch mehrere Voraussetzungen verbunden. Zum einen wird unterstellt, dass der initiierende Teilnehmer über ausreichend Anwendungswissen verfügt, um die Demarkationsentscheidung treffen zu können. Wie bereits dargelegt (vgl. Abschnitt 2.2.5), kann dies im Rahmen von Orchestrierungen erwartet werden. Bei choreographierten Transaktionen muss hingegen von autonomen Teilnehmern ausgegangen werden, deren Interna nicht zugreifbar sind. Bekannt ist nur das im Rahmen der Choreographie festgelegte Kommunikationsverhalten.

Desweiteren muss berücksichtigt werden, dass Dienste in kollaborativen Interaktionen als Repräsentanten von realen Institutionen agieren können. Sie unterliegen damit also auch denselben rechtlichen Beschränkungen und besitzen die gleichen Befugnisse wie menschliche Vertreter dieser Organisationen. Ferner sind häufig Zuständigkeiten zwischen Niederlassungen zu beachten, so dass Teilnehmer an die Authorisationsstruktur eines Unternehmens gebunden sind. Es besteht also die Möglichkeit, dass gewisse Entscheidungen einfach nicht getroffen werden *dürfen*.

Ein technischer Aspekt, der zusätzlich gegen die Konzentration des Demarkationsprivilegs beim Initiator spricht, ist der Einsatz mobiler Geräte. So ist es denkbar, dass der Initiator eine Transaktion anstößt, während er online ist, im weiteren Verlauf auf Grund von fehlender Konnektivität jedoch nicht in der Lage ist, sein Recht zur Demarkation wahrzunehmen.

Weder das Completion-Protokoll von WS-AtomicTransaction noch seine Pendents für WS-BusinessActivity spezifizieren, welche Teilnehmer registrierungsberechtigt sind. Vermutlich wird angenommen, dass der Initiator der einzige demarkationsberechtigte Teilnehmer ist und dieses Privileg stets erfüllen kann. Das Hauptproblem besteht somit in der impliziten Verknüpfung der *Rolle* des Initiators mit seinem *Privileg*, den Transaktionsabschluss zu eröffnen. Wünschenswert wäre daher die Trennung beider Aspekte und die Einführung einer expliziten Rollen- und Privilegienverwaltung. Diese Thematik wurde bereits eingehend von Denis Rathig in seiner Diplomarbeit behandelt (vgl. [Rat04]). Bezogen auf die Demarkation schlägt Rathig zwei Formen der Rechteverwaltung vor:

Teilung: Das Privileg zur Demarkation kann von mehreren Teilnehmern gleichzeitig eingenommen werden. In diesem Szenario ist der Initiator einer Transaktion zunächst der einzige demarkationsberechtigte Teilnehmer. Über einen parametrisierbaren Activation-Service des Koordinators kann er jedoch angeben, welche Teilnehmer zusätzlich das Privileg von ihm anfordern dürfen. Diese Daten werden fortan als

Bestandteil des erzeugten Koordinationskontextes weitergereicht. Trifft die Privilegienanforderung eines Teilnehmers beim Initiator ein, entscheidet dieser, ob das Recht zur Demarkation erteilt wird.

Weitergabe: Das Demarkationsprivileg wird vom ursprünglichen Inhaber an einen anderen Teilnehmer abgetreten, der daraufhin diese Rolle übernimmt. Auch hier ist vorerst der Initiator im Besitz des Privilegs. Aber schon bei Erstellung des Koordinationskontextes wird festgelegt, an welchen Teilnehmer das Recht übergehen soll. Der neue demarkationsberechtigte Teilnehmer wird mit Hilfe des weitergeleiteten Kontextes von seinem Privileg in Kenntnis gesetzt.

Grundsätzlich ermöglichen diese Methoden auch anderen Teilnehmern als dem Initiator, den Abschluss einer Transaktion einzuleiten. Ungeklärt bleibt jedoch die Frage nach der Bestimmung von Teilnehmern, die über ausreichendes Anwendungswissen verfügen, um mit dem Demarkationsprivileg ausgestattet zu werden. Insbesondere bei komplexen Szenarien kann sich die zuverlässige Auswahl geeigneter Teilnehmer als Problem erweisen.

3.1.3 Diskussion der Ansätze

Die oben beschriebenen Ansätze verdeutlichen, wie sehr die Transaktionssteuerung vom Initiator abhängt. Im Prinzip ist stets der Initiator im Besitz des Demarkationsrechts und steuert die Transaktion durch Befehle an den Koordinator. WS-BA spezifiziert hierfür nicht einmal ein entsprechendes Protokoll, so dass an dieser Stelle proprietäre Entwicklungen zum Einsatz kommen müssen oder beide Komponenten gleich vollends integriert werden müssen. Diese enge Verbindung von Initiator und Koordinator läuft jedoch dem SOA-Paradigma einer losen Kopplung von Diensten zuwider. Es besteht das Risiko, dass die Interoperabilität als Zielvorgabe einer serviceorientierten Architektur für den Bereich der transaktionalen Workflows verloren geht.

Das Projekt *Transactional Activity Control for the Grid* (TracG, [HRR07], [RHR08]) untersucht daher Möglichkeiten zur Überwindung der geschilderten Probleme, indem die Rolle des Koordinators gestärkt wird. Im Kern umfasst das TracG-Projekt ein Koordinations-Framework, das den Koordinator befähigt, weitestgehend autonom über den Prozessabschluss zu entscheiden und darüber, welche Teilnehmer ihre Ergebnisse festschreiben müssen bzw. verwerfen können. Der Koordinator aggregiert hierfür die lokalen Sichten der Teilnehmer zu einer globalen Sicht des Prozesses und fällt auf dieser Grundlage die Entscheidung über den Transaktionsabschluss.

Das Demarkationsrecht wird somit dem Koordinator übertragen, wodurch auch ein Completion-Protokoll zur Einleitung des Transaktionsabschlusses überflüssig wird. Es wird allenfalls noch ein Mechanismus zur abschließenden Übergabe des Transaktionsergebnisses an den Initiator benötigt. Die Stärkung der zentralen Position des Koordinators senkt dabei deutlich die Komplexität der Prozesslogik eines Teilnehmers, insbesondere

die des Initiators, weil kein Teilnehmer mehr die Qualifikation zur Demarkation besitzen muss. Darüber hinaus trägt dieser Ansatz zur Erhöhung der Interoperabilität bei, da der Koordinator im Sinne einer SOA als Dienstleister für eine transaktionale Absicherung fungiert. Trotz der Zentralisierung der Koordinationssteuerung werden vor allem choreographierte Workflows lose gekoppelter Teilnehmer unterstützt, wenn unabhängige Koordinatoren anhand funktionaler Kriterien auswählbar sind.

Der Einsatz proprietärer Completion-Protokolle sowie die explizite Weitergabe des Demarkationsrechts werden im Folgenden als mögliche Alternative angesehen, für den Fall, dass eine autonome Transaktionssteuerung durch den Koordinator nicht in Frage kommt. Diese Situation kann beispielsweise eintreten, wenn sich in Dienstverzeichnissen kein geeigneter Koordinator finden lässt, der alle Belange der beteiligten Teilnehmer unterstützt.

Nachfolgend wird das in TracG vorgeschlagene Konzept der *autonomen Koordination* vorgestellt, dass darauf beruht, dem Koordinator das Demarkationsprivileg zuzusprechen und ihn mit der notwendigen Information zu versorgen, selbstständig über den Transaktionsabschluss entscheiden zu können. Es werden ferner erweiterte Charakteristika von Transaktionen sowie deren Auswirkungen auf das TracG-Konzept untersucht, um erforderliche Koordinatoren genauer spezifizieren zu können.

3.2 Autonome Koordination

Um einen transaktionalen Prozess autonom koordinieren zu können, müssen zwei wesentliche Fragen vom Koordinator beantwortet werden können:

- *Wann* muss die Abschlussphase der Transaktion eingeleitet werden?

  Der Koordinator muss hierfür entscheiden können, wann die Arbeit einer Transaktion erledigt ist und keine weiteren Teilnehmer mehr aufgerufen werden, die sich für die Teilnahme an der Transaktion registrieren können.

- *Wer* muss die Ergebnisse festschreiben und wer muss sie verwerfen?

  Die Beantwortung dieser Frage hängt maßgeblich vom Koordinationstyp ab, also ob es sich um AtomicOutcome oder MixedOutcome handelt. Bei MixedOutcome muss der Koordinator die für den Prozess *notwendigen* Teilnehmer kennen, um diese selektiv festschreiben zu lassen.

In WS-AtomicTransaction sind beide Aspekte klar durch das Completion-Protokoll geregelt. Um den Prozess zu einem erfolgreichen Abschluss zu dirigieren, führt der Koordinator das 2PC-Protokoll unter allen Teilnehmern aus, sobald der kontrollierende Teilnehmer mit Hilfe des Completion-Protokolls die Beendigung signalisiert (vgl. Abschnitt 2.4.3).

In WS-BusinessActivitiy kann ein Teilnehmer dem Koordinator entweder selbstständig den Abschluss seiner lokalen Arbeit mitteilen (BusinessAgreementWithParticipant-

Completion, BAPC) oder er wird vom Koordinator angewiesen, seine lokale Arbeit zu beenden und in einen Wartezustand einzutreten (BusinessAgreementWithCoordinator-Completion, BACC). In beiden Fällen wartet der Teilnehmer anschließend auf ein finales `Close` oder `Compensate`, um seine Arbeit festzuschreiben oder zu verwerfen (vgl. Abb. 2.12, 2.13 in Abschnitt 2.4.3).

Um ein erfolgreiches Ergebnis von Business-Aktivitäten zu erzielen, brauchen nicht alle Teilnehmer festzuschreiben. Es reicht aus, wenn die *notwendigen* Teilnehmer den Prozess fehlerfrei durchlaufen (MixedOutcome). Allerdings kann der Koordinator auf Grund des fehlenden Completion-Protokolls weder wissen, wann in die Abschlussphase eingetreten werden soll, noch besteht die Möglichkeit ihm mitzuteilen, welcher Teilnehmer bei MixedOutcome notwendig für einen erfolgreichen Abschluss ist.

Auf alleiniger Grundlage der WS-BA-Spezifikation kann also bestenfalls der Koordinationstyp AtomicOutcome realisiert werden, wenn als Kriterium für den Eintritt in die Abschlussphase das Erreichen des `Completed`-Zustands aller registrierten Teilnehmer herangezogen wird. Hierbei bleibt jedoch ungewiss, ob sich bereits alle Teilnehmer des Prozesses registriert haben. Die Eliminierung dieses Zweifels und die Ermöglichung von MixedOutcome-Szenarien erfordern zusätzliche Mechanismen, um den Koordinator mit der fehlenden Information zu versorgen. Der Koordinator muss generell beurteilen können, wann der Prozess in die Abschlussphase eintreten kann und für die Terminierung mit MixedOutcome-Ergebnissen braucht er Informationen über die Notwendigkeit von Teilnehmern. Beides ist nicht ohne zusätzlichen Aufwand möglich.

Bestehende Ansätze, in denen der Koordinator explizit über den Zustand der Teilnehmer informiert wird, arbeiten analog zu dem Terminierungsprotokoll von WS-AT (vgl. [EHHZ07], [VGZ+05]). Mit dem TracG-Projekt wird dagegen ein Vorschlag unterbreitet, wie der Koordinator eigenständig entscheiden kann, wann der Prozess zu beenden ist und welche Teilnehmer festschreiben müssen. Er benötigt dafür stets genaue Kenntnis der Teilnehmerzustände, um den Fortschritt des Gesamtprozesses beurteilen zu können.

3.2.1 Status der Teilnehmer und des Prozesses

Die Hauptaufgabe des Koordinators ist die Herbeiführung eines Transaktionsabschlusses gemäß der Protokollgraphen aus Abschnitt 2.4.3. Hierfür muss er die Zustände der registrierten Teilnehmer einer Transaktion kennen, um Nachrichten entsprechend des Protokolls versenden zu können.

Die Protokollgraphen und die in den Spezifikationen enthaltenen Tabellen für Zustandsübergänge (vgl. [LW07], [FL07]) geben jedoch nur Aufschluss darüber, wie der Nachrichtenaustausch zwischen dem Koordinator und *einem* Teilnehmer abläuft. Der Fortschritt des Prozesses und der Koordinatorzustand richten sich dagegen nach den Zuständen *aller* Teilnehmer. So kann der Koordinator beispielsweise erst in den Zustand `Completed` wechseln, nachdem er von allen für BAPC oder BACC registrierten Teilnehmern eine `Completed`-Nachricht erhalten hat (vgl. Abb. 2.12, 2.13). Der Prozess befindet

sich damit häufig in einem Zustand, der die Teilnehmerzustände bündelt und im Protokollgraph nicht direkt vorkommt. Wichtig ist dabei, dass vom Koordinator alle Nachrichten akzeptiert werden, die von Teilnehmern in den jeweiligen Zuständen versandt werden können.

Die hier beschriebene Vorgehensweise entspricht der typischen Realisierung eines Protokolls als Zustandsautomat. Damit hat der Koordinator allerdings noch keine Entscheidungsautonomie gewonnen bzgl. der eingangs aufgeworfenen *Wann*- und *Wer*-Fragen.

3.2.2 Aufrufbaum

Wie oben bereits angedeutet, kann der Koordinator erst im Prozess fortfahren, wenn *alle* Teilnehmer einen bestimmten Zustand erreicht haben. Das Problem dabei ist, dass der Koordinator keine Informationen darüber besitzt, wieviele Teilnehmer ein Prozess haben wird – er kennt nur die Anzahl der Teilnehmer, die sich bereits registriert haben! Zur eigenständigen Beantwortung der *Wann*-Frage benötigt der Koordinator jedoch die Gesamtzahl *aller* Teilnehmer, die sich für einen Prozess registrieren werden.

Diese Information muss dem Koordinator also irgendwie übergeben werden. Die Idee hierfür ist, dass jeder Teilnehmer, angefangen beim Initiator, dem Koordinator bei der Registrierung mitteilt, wieviele Dienste er im weiteren Verlauf aufrufen wird. Umgekehrt muss jeder neue Teilnehmer, der sich registriert, die Id desjenigen Teilnehmers angeben, der ihn aufgerufen hat. Auf diese Weise kann der Koordinator sukzessive auf die Gesamtzahl der zu erwartenden Teilnehmer schließen und sobald diese erreicht ist, kann im Protokoll fortgefahren werden, da nun keine weiteren Registrierungen mehr zu erwarten sind.

In einer Choreographie besitzt der Koordinator überdies keine Einsicht in die Aufrufe zwischen den Teilnehmern. Erst wenn sich neue Teilnehmer mit einem bekannten Kontext anmelden, kann er einen Zusammenhang zu einer bestehenden Aktivität herstellen. Mit der zuvor beschriebenen Eltern-Id ist es dem Koordinator nun zusätzlich möglich, einen *Aufrufbaum* zu erzeugen, der ihm Aufschluss über die strukturellen Zusammenhänge zwischen den Teilnehmern gibt. Darin erscheint der Initiator als Wurzel, da er bei seiner Registrierung keine Eltern-Id angegeben haben kann, und Teilnehmer, die keine weiteren Dienste aufrufen, bilden die Blätter des Baumes. Alle anderen Teilnehmer stellen innere Knoten des Baumes dar. Abbildung 3.1 zeigt die beispielhafte Darstellung eines Aufrufbaumes, in dem der Web-Service *a* (WS *a*) der Initiator der Transaktion ist.

Diese Struktur der Web-Service-Aufrufe wird vom Koordinator zur Beantwortung der *Wer*-Frage bei einer Transaktion mit gemischtem Ergebnis benötigt. Der Koordinator verwaltet dazu die Menge der festzuschreibenden Teilnehmer und die Menge der abzubrechenden Teilnehmer, welche im Folgenden als *Complete-Menge* und *Cancel-Menge* bezeichnet werden. Für einen erfolgreichen Abschluss des Business-Prozesses müssen die Teilnehmer die Transaktion entsprechend ihrer Mengenzugehörigkeit beenden, wobei alle Teilnehmer anfangs der Complete-Menge angehören.

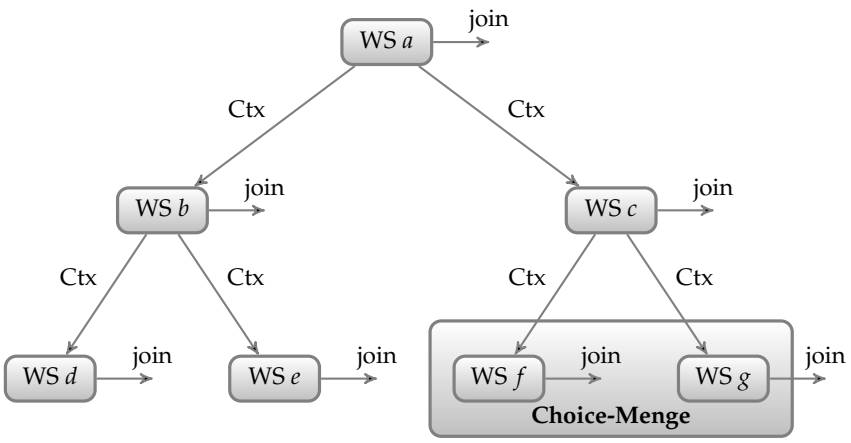

Abbildung 3.1: Beispiel eines Aufrufbaumes von Web-Services.

Zur Modellierung von alternativen Auswahlmöglichkeiten, wie im Beispiel der Flugbuchung aus Abschnitt 3.1.1, führt der Koordinator zusätzlich sog. *Choice-Mengen* (vgl. Abb. 3.1). Bei Teilnehmern einer Choice-Menge ist zunächst unklar, wer am Ende für den Abschluss ausgewählt wird. Wenn die Anwendungslogik des Elternknotens schließlich entschieden hat, welcher Teilnehmer den Vorzug bekommt, wird dies dem Koordinator gesondert mitgeteilt. Dieser verschiebt dann die nicht ausgewählten Teilnehmer in die Cancel-Menge. Die Mengenzugehörigkeiten werden durch die Hierarchie des Aufrufbaumes entsprechend der Eltern-Kind-Linie nach unten propagiert. Wenn also ein innerer Knoten in die Cancel-Menge verschoben wird, gilt dies auch für alle seine Kinder und die durch sie aufgerufenen Dienste.

Alle Mengen werden im Verlauf des Prozesses stets vom Koordinator aktualisiert, so dass er eigenständig in der Lage ist, den Abschluss der Transaktion einzuleiten, nachdem alle Service-Aufrufe stattgefunden haben und alle Choice-Mengen entschieden sind. In diesem Modell sind die Teilnehmer für anwendungsspezifische Entscheidungen zuständig, die vom Koordinator selbstständig in die technische Abwicklung der transaktionalen Koordination umgesetzt werden. Hierfür ist kein Completion-Protokoll erforderlich. Außerdem benötigt kein Teilnehmer eine globale Sicht des Prozesses – diese Information wird ausschließlich vom Koordinator aggregiert.

Das beschriebene Verfahren setzt voraus, dass jeder aufgerufene Teilnehmer der Transaktion auch beitritt und diese nicht eigenmächtig wieder verlassen kann. Zur flexibleren Gestaltung der Prozessverarbeitung wird nachfolgend untersucht, welche erweiterten Anforderungen an das Modell bestehen können und welche Auswirkungen diese auf die Koordination haben.

3.2.3 Erweiterte Anforderungen an die autonome Koordinierung

In Abschnitt 2.3.7 wurden Eigenschaften von Business-Transaktionen zusammengefasst, die in der Langlebigkeit der Prozesse und in der Autonomie teilnehmender Dienste begründet sind. Vornehmlich dienen sie der flexiblen Konfiguration transaktionaler Work-

flows, um eine Erhöhung der Fehlertoleranz sowie eine Verminderung von Leistungseinbußen auf Seiten der Teilnehmer zu bewirken. Die Mechanismen zur Durchsetzung dieser Ziele wurden bereits teilweise im Rahmen der erweiterten Transaktionsmodelle vorgestellt und sollen anschließend in Bezug auf die autonome Koordinierung präzisiert werden.

Bisher kann das Modell lediglich die bedingte Auswahl zwischen mehreren Teilnehmern mit Hilfe der Choice-Mengen abbilden. Hiermit wird jedoch keine größere Robustheit in Fehlersituationen erzielt, sondern es ist nur ein Mechanismus, um Entscheidungen des Elternknotens in MixedOutcome-Szenarien durchzusetzen. Welche erweiterten Kriterien einen Beitrag zur Erhöhung der Flexibiliät und der Fehlerrobustheit leisten können, wird nun untersucht.

Vitalität

Das präsententierte Modell beruht auf einer offen geschachtelten Hierarchie von Web-Service-Aufrufen, wobei angenommen wurde, dass jeder Teilnehmer notwendig für die gesamte Transaktion ist. Dies ist eine Eigenschaft von Subtransaktionen, die auch als *Vitalität* bezeichnet wird und erstmals in [GMGK+90] eingeführt wurde. Darin werden folgende Stufen der Vitalität unterschieden:

vital: Der erfolgreiche Abschluss einer *vitalen* Subtransaktion ist notwendig für den erfolgreichen Abschluss der Elterntransaktion. Umgekehrt muss bei einem Abbruch einer vitalen Subtransaktion, auch die Elterntransaktion abbrechen. Wenn in einer Aufrufhierarchie alle Knoten vital sind, setzen sich die Abbrüche fort bis zur Top-Level-Transaktion, die dann ebenfalls scheitert.

nicht-vital: Teilnehmer gelten als *nicht-vital*, wenn ihr Scheitern keinen Abbruch des Elternknotens provoziert.

Wenn eine Reisebuchung eine Subtransaktion zur Bestellung eines Leihwagens vorsieht, kann dies im Allgemeinen als nicht-vital angesehen werden, da vor Ort stets auch ein Taxi genommen werden kann. Dagegen sind Teiltransaktionen, wie „Buche Flug“ oder „Buche Hotel“ vital.

Durch die Einführung dieser Eigenschaften von Teilnehmern, kann die Fehlertoleranz des gesamten Workflows gesteigert werden, da der Abbruch eines Teilnehmers nicht mehr zwangsläufig zum Scheitern des Prozesses führt. Der Abbruch eines Elternteilnehmers hat jedoch weiterhin den Abbruch der Subtransaktionen zur Folge.

In einer Implementation müssen alle Attribute für einen Dienst bei seinem Aufruf vom Elternknoten annotiert werden. Ähnlich wie bei der Angabe von Choice-Mengen kann die Information mit dem erweiterbaren Kontext transportiert werden, so dass die Spezifikationen nicht verletzt werden. Ein aufgerufener Teilnehmer leitet den Kontext bei seiner Anmeldung schließlich an den Koordinator weiter, so dass dieser in der Lage ist, die Transaktionsabwicklung entsprechend anzupassen.

Abhängigkeiten

Üblicherweise werden Subtransaktionen derselben Ebene parallel ausgeführt, um die Effizienz der Prozessabarbeitung zu erhöhen. Für einen Prozess kann es jedoch sinnvoll sein, kausale Abhängigkeiten einzuführen, die eine sequentielle oder alternative Ausführung von Teiltransaktionen gestatten [Elm92]. Zur Modellierung von Ausführungsabhängigkeiten werden bei Flex-Transaktionen [ELLR90] die folgenden Abhängigkeiten unterschieden:

positive Abhängigkeit: Durch benutzerdefinierte *Präzedenzen* von Teilnehmern, darf eine Aktivität nicht gestartet werden, bevor ihr Vorgänger erfolgreich beendet wurde. Für den Prozess bedeutet dies eine sequentielle Ausführung der Teiltransaktionen.

Bezogen auf das Beispiel der Reisebuchung sollte ein Auto z.B. erst dann gemietet werden, wenn der Flug gebucht werden konnte.

In diese Kategorie fallen damit auch Kompensationstransaktionen, denn auch sie werden nur dann ausgeführt, wenn die assoziierte Transaktion erfolgreich war. Allerdings besitzen sie eine andere Semantik, da sie die Wirkung ihrer Vorgängertransaktion aufheben.

negative Abhängigkeit: Der Aufruf alternativer Teilnehmer bei einem Fehler eines assoziierten und ersetzbaren Teilnehmers wird auch als als *Kontingenz*[1] [Elm92] oder als Funktionsreplikation (vgl. Abschnitt 2.3.6) bezeichnet. Alternative Teilnehmer sind in der Lage, einen ähnlichen Dienst zu erfüllen wie ihr gescheiterter Vorgänger, so dass sie diesen substituieren können (Bsp. Zimmerbuchung).

Allgemein werden Alternativen genutzt, wenn eine Sub-TA mit einem `fail` fehlschlägt. Welche Ursache zum Scheitern führt, ist dagegen wieder durch den Benutzer festzulegen. Eine Hotelbuchung kann beispielsweise mißlingen, weil das Hotel voll ist, oder weil ein bestimmtes Zimmer belegt ist.

Das wesentliche Problem dieser Anforderung ist, dass der Koordinator auf Grund seiner beschränkten Sicht des Prozesses nicht in der Lage ist, selbstständig Dienstaufrufe durchzuführen – diese werden stets von den Teilnehmern initiiert. Die Funktion des Koordinators konzentriert sich allein auf die technische Ablaufsteuerung der Transaktionen, um eine Menge von Teilnehmern zu einem kollektiven Transaktionabschluss zu bewegen.

Lösungsansätze für die Modellierung von Ausführungsabhängigkeiten müssen den Koordinator daher entweder befähigen, doch selbst Dienste aufrufen zu können oder aber diese Aufgabe beim Teilnehmer zu belassen und eine unterstützende Koordinierung anzubieten.

Im ersten Fall ist die Lösung vergleichbar mit der in Abschnitt 3.1.1 diskutierten Integrationslösung. Jedoch müsste die Integration sogar über den Initiator und Koordinator

[1] Vom englischsprachigen *contingency*.

hinaus auf alle Teilnehmer ausgeweitet werden, da sämtlichen Teilnehmern die Formulierung von Abhängigkeiten ermöglicht werden soll. Wie bereits geschildert, läuft dieser Weg dem Verständnis von serviceorientierten Architekturen zuwider und soll in dieser Arbeit nicht weiter verfolgt werden.

In einem unterstützenden Ansatz bleibt der Koordinator dagegen als eigene Einheit bestehen und muss über alle vorhandenen Abhängigkeiten informiert werden, so dass dies in der Koordinierung entsprechend berücksichtigt werden kann. So darf beispielsweise kein Prozessabbruch eingeleitet werden, wenn es für einen gescheiterten Dienst Alternativen gibt. Eine konzeptuelle Untersuchung der Realisierungsmöglichkeiten soll im nächsten Kapitel darüber Aufschluss geben, wie ausgedehnt die Unterstützung von Abhängigkeiten eines dienstbasierten Koordinators ausfallen kann.

Temporale Bedingungen

Wie im Rahmen der Flex-TA bereits beschrieben (vgl. 2.3.6), ist die Ausführung von Teiltransaktionen oftmals nur in engen zeitlichen Grenzen sinnvoll. Die Buchung eines Fluges sollte beispielsweise unbedingt vor dem Abflugtermin erfolgen.

Um ein geeignetes Zeitintervall zu spezifizieren, reicht es aus, eine untere und eine obere Schranke jeweils durch Zeitmarken anzugeben, zwischen denen eine Sub-TA durchgeführt werden soll. In WS-Coordination ist allerdings nur das Kontextelement `Expires` zur Angabe von oberen Schranken vorgesehen, mit dem die Gültigkeitsdauer des gesamten Kontextes festgelegt werden kann. Eine präzisere Steuerung der Teilnehmer erfordert für jeden Teilnehmer zwei Schranken.

Zur Durchsetzung der temporalen Beschränkungen reicht ein ereignisbasierter Koordinator, der nur bei Eintreffen einer Nachricht aktiv wird, nicht mehr aus. Für die regelmäßige Überprüfung der zeitlichen Schranken sind zeitbasierte Mechanismen erforderlich, die eine Reaktion des Koordinators auslösen, falls eine Bedingung nicht eingehalten wird.

Implementationsmuster

Die bereits in Abschnitt 2.4.1 eingeführten Implementationsmuster beschreiben die konkrete Umsetzung des Transaktionsabschlusses seitens der Teilnehmer. Diese Muster werden nachfolgend für eine Verwendung in WS-BusinessActivity verdeutlicht:

Provisional-Final: Der Teilnehmer setzt die von der Anwendung geforderte Leistung um, markiert sie als „vorläufig“ und macht sie sichtbar. Danach befindet er sich im Zustand `Completed`. Bei Erhalt einer `Close`-Nachricht werden die provisorischen Daten als „endgültig“ markiert bzw. als Resultat festgeschrieben. Im Gegensatz dazu führt eine `Compensate`-Nachricht zur Löschung der vorläufigen Daten, so dass der alte Zustand wieder hergestellt ist.

Ein typischer Anwendungsfall für eine derartige Implementation ist das Einholen von Angeboten für eine geplante Bestellung und der anschließende Bestellvorgang.

Validate-Do: Vor Erreichen des `Completed`-Zustands überprüft der Dienst die beabsichtigten Änderungen auf Validität und speichert sie – die Änderungen werden jedoch nicht umgesetzt. Die Umsetzung erfolgt erst nachdem die `Close`-Nachricht empfangen wurde. Sendet der Koordinator dagegen eine `Compensate`-Nachricht, werden die Einträge über durchzuführende Änderungen gelöscht.

Do-Compensate: Nach diesem Muster realisierte Teilnehmer führen die Änderungen vollständig durch, machen sie sichtbar und speichern Kompensationsinformationen für den Fall, dass die Änderungen zurückgenommen werden müssen. Eine `Close`-Nachricht vom Koordinator führt zur Löschung dieser Kompensationsdaten, während nach dem Empfang einer `Compensate`-Nachricht mit Hilfe dieser Daten alle durchgeführten Änderungen semantisch rückgängig gemacht werden.

Die Umsetzung von ACID-Kriterien entspricht dem Muster Provisional-Final, wobei die vorläufigen Effekte nicht sichtbar gemacht werden.

Nach Furniss und Green [FG05] sind Teilnehmer von WS-BusinessActivity auf Grund eines fehlenden Zustandsüberganges von `Closing` zu `Failing` ausschließlich auf das Implementationsmuster Do-Compensate festgelegt (vgl. Zustandsgraphen in Abschnitt 2.4.3). Ein Dienst, der nach dem Muster Provisional-Final realisiert ist, muss aber in der Lage sein, die geforderte Durchführung abzubrechen. Im Beispiel des Bestellangebots käme dies einer Zurücknahme des Angebots ggf. mit einer nachfolgenden Angebotserneuerung gleich. Im Implementationsmuster Validate-Do kann es dagegen passieren, dass validierte, mögliche Änderungen zu einem späteren Zeitpunkt nicht mehr durchführbar sind und deswegen abgebrochen werden müssen. In beiden Fällen müssten die Teilnehmer mit einer `Fail`-Nachricht die Ausführung beenden können. Da dies in WS-BA jedoch nicht vorgesehen ist, kann nur das Muster Do-Compensate implementiert werden. Hierbei sind im Zustand `Closing` nur unkritische Änderungen zu erwarten, wie das Verwerfen der Kompensationsinformation, die kein Scheitern der TA zur Folge haben können und daher nicht die Integrität des Datenbestands bedrohen. Infolgedessen ist kein entsprechender Übergang in den Zustand `Fail` vorgesehen.

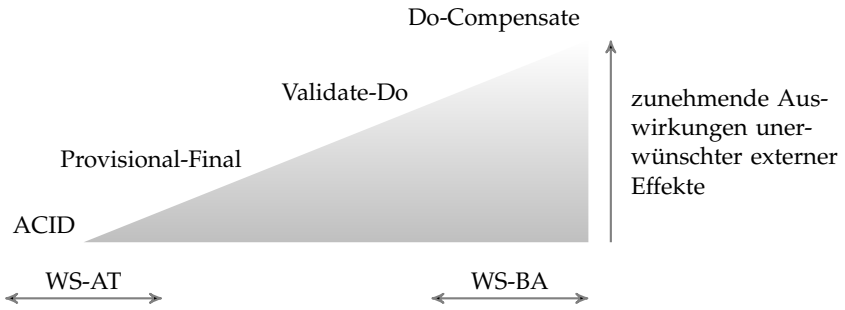

Abbildung 3.2: Spektrum der Business-Transaktionen (aus [FG05]).

Die vorgestellten Implementationsmuster haben unterschiedlich weitreichende Konsequenzen, was in Abb. 3.2 veranschaulicht wird. Insbesondere Do-Compensate, als das einzige von WS-BA unterstützte Muster, kann erhebliche externe Effekte nach sich ziehen. Dies wirkt sich vor allem bei der Kompensation negativ aus, da in diesem Fall alle Auswirkungen in ihrer gesamten Tragweite unwirksam gemacht werden müssen. Im Extremfall kann dies dazu führen, dass im Rahmen einer Bestellung gelieferte Waren wieder zurückgeschickt werden müssen. Derartig gravierende Auswirkungen treten mit den Mustern Provisional-Final und Validate-Do nicht auf, da in beiden Fällen keine festgeschriebenen Änderungen in der ersten Phase durchgeführt werden. Stattdessen werden durchzuführende Änderungen entweder nur vorgemerkt oder auf ihre Durchführbarkeit hin überprüft. Die tatsächlichen Änderungen am Datenbestand geschehen erst in der zweite Phase, nachdem der Koordinator die `Close`-Nachricht verschickt hat.

Ähnlich wie andere vorgestellte Mechanismen können weitere Implementationsmuster neben Do-Compensate die Auswirkungen gescheiterter Transaktionen mildern. Um zusätzliche Muster zu ermöglichen, muss im Wesentlichen nur der Zustandsgraph um eine `Fail`-Kante von `Closing` zu `Failing` erweitert werden (vgl. Abb. 3.3). Mit Ausnahme der Transitionstabellen bzgl. dieser neuen Kante ändert sich für den Koordinator hierbei zunächst nichts, da die Muster auf Teilnehmerseite umgesetzt werden. Das Protokoll bleibt sonst unverändert.

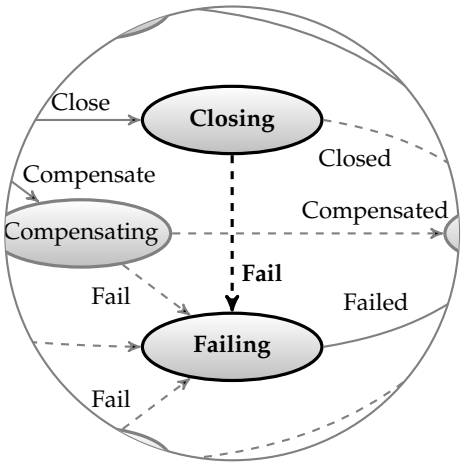

Abbildung 3.3: WS-BusinessActivity – neuer Zustandsübergang mit `Fail`-Nachricht.

Die Einführung der neuen `Fail`-Kante hat allerdings einen Einfluss auf die transaktionalen Zusicherungen des Protokolls. Denn mit Erreichen des `Completed`-Zustands geben Teilnehmer eine Garantie ab, die zugesagten Dienste auch erbringen zu können bzw. schon erbracht zu haben (bei Do-Compensate). Dieser Verantwortung können sie sich nun jedoch über die neue `Fail`-Nachricht im Zustand `Closing` entziehen. Auf diese Weise würde das Protokoll seinen Zweck also nicht mehr erfüllen.

Es kann daher nur nicht-vitalen Teilnehmern oder ersetzbaren Teilnehmern mit deklarierten Alternativen gestattet werden, die Muster Provisional-Final oder Validate-Do zu

implementieren. Scheitert ein nicht-vitaler Teilnehmer im Zustand `Closing`, kann der Prozess fortgeführt werden, wenn eine `Fail`-Nachricht nicht grundsätzlich als schwerwiegendes, zum Prozessabbruch führendes Ereignis interpretiert wird. Für ersetzbare Teilnehmer können die als negative Abhängigkeit deklarierten Alternativteilnehmer herangezogen werden. Do-Compensate-Teilnehmern ist es weiterhin nicht erlaubt, im Zustand `Closing` zu Scheitern, was auf Grund ihrer Realisierung typischerweise einzuhalten ist. Damit der Koordinator beurteilen kann, für welche Teilnehmer die neue `Fail`-Kante erlaubt ist, müssen alle Dienste ihr Implementationsmuster bei ihrer Registrierung annotieren.

3.2.4 Zusammenfassung

Für die autonome Koordinierung transaktionaler Workflows müssen die wesentlichen Fragen nach dem Zeitpunkt des Transaktionsabschlusses und nach der Identität der festzuschreibenden Teilnehmer beantwortet werden. Das vorgestellte Modell eines Koordinators verwendet dazu einen Aufrufbaum der Teilnehmer, der mit Hilfe des Kontextes und erweiterter Registrierungsnachrichten dynamisch aufgebaut wird. Damit wird es möglich, auf die Gesamtzahl der zu erwartenden Teilnehmer zu schließen und infolgedessen den Zeitpunkt für die Einleitung der Terminierungsphase zu bestimmen. Durch die gleichzeitige Verwaltung von Complete-, Cancel- und Choice-Mengen lassen sich MixedOutcome-Szenarien umsetzen.

Um auch in Fehlersituationen ein möglichst robustes Verhalten realisieren zu können und die Abbruchwahrscheinlichkeit des Prozesses zu minimieren, reicht die rein strukturelle Information des Aufrufbaumes nicht aus. Hierfür ist es erforderlich, dem Koordinator Eigenschaften der Teilnehmer mitzuteilen, die sich direkt auf die Ablaufsteuerung auswirken können und den Koordinator in die Lage versetzen, den Prozess derart zu steuern, dass der Arbeitsverlust auf Teilnehmerseite gering bleibt. Potentielle Teilnehmerattribute, die hierfür in Betracht kommen und in diesem Kapitel diskutiert wurden, sind nachfolgend noch einmal zusammenfassend aufgeführt:

- Vitalität eines Teilnehmers, d.h. seine Notwendigkeit für einen erfolgreichen Transaktionsabschluss
- Abhängigkeiten zwischen Teilnehmern zur Modellierung von Präzedenzen und Alternativen
- Temporale Bedingungen für den Aufruf eines Teilnehmers
- Implementierungsmuster bzgl. des Transaktionsabschlusses eines Teilnehmers

Im folgenden Kapitel soll untersucht werden, wie sich diese Anforderungen umsetzen lassen, um eine optimale Koordinierung des Transaktionsablaufs zu realisieren.

4 Konzept eines autonomen Koordinators

Das aktuelle Kapitel befasst sich mit der Erarbeitung eines konkreten Konzeptes für die weitreichende autonome Koordinierung von Business-Transaktionen auf Basis von WS-Coordination und WS-BusinessActivity. Im Zentrum der Betrachtung steht dabei die Umsetzung der grundlegenden Mechanismen, die zu dieser Autonomie beitragen sowie deren Einbindung in die spezifizierten Protokollabläufe.

Darüber hinaus werden Integrationsmöglichkeiten weiterführender Aspekte zur Steigerung der Flexibilität und Fehlertoleranz für das entwickelte Koordinatorkonzept untersucht und ein richtungsweisender Entwurf für diese Erweitungen vorgeschlagen.

4.1 Autonomie durch regelbasierte Koordinierung

Der Koordinator eines transaktionalen Workflows ist verantwortlich für die Steuerung des Prozessablaufs, um die Teilnehmer zu einem konsistenten Abschluss zu dirigieren. Die Erfüllung dieser Aufgabe erfordert die Verwaltung und Verarbeitung unterschiedlicher Informationen, die nachfolgend detailliert vorgestellt werden.

4.1.1 Verwaltung der Teilnehmerzustände und des Prozesses

Die Koordination einer Transaktion beginnt für den Koordinator mit dem Empfang einer `CreateCoordinationContext`-Nachricht vom Initiator der Transaktion. Der Koordinator erzeugt daraufhin eine Aktivität eines in der Nachricht angegebenen Koordinationstyps und sendet den korrespondiereden `CoordinationContext` zurück an den Initiator. Nachfolgend müssen sich alle zu koordinierenden Teilnehmer – einschließlich des Initiators – explizit für ein bezüglich des Kontexttyps zulässiges Protokoll beim Koordinator registrieren.

Identität der Teilnehmer

Zur Erfüllung des Transaktionsprotokolls sind in jedem Zustand von BAPC und BACC ausschließlich die durch WS-BA spezifizierten Nachrichten zwischen Koordinator und Teilnehmer erlaubt (vgl. Abb. 2.12, 2.13). Der Koordinator muss also in der Lage sein, einzelne Teilnehmer zu unterscheiden, um das Protokoll entsprechend der Spezifikation fortzuführen. Da WS-Coordination nicht definiert, wie dies umgesetzt werden soll, bleiben grundsätzlich die beiden Möglichkeiten, den Teilnehmern selbst zu gestatten an sich eine Kennung zu vergeben oder es dem Koordinator zu überlassen für die Unterscheidbarkeit der Dienste zu sorgen. In diesem Entwurf wird es die Aufgabe des Koordinators sein, eindeutige Teilnehmer-Ids zu vergeben, da er auch bereits für die Erzeugung des Kontextes und dessen Identifikation verantwortlich ist.

Die Kennungen werden vom Koordinator bei der Registrierung von Teilnehmern erzeugt und mit der `RegisterResponse`-Nachricht an die Teilnehmer übermittelt. Fortan müssen sie von den Teilnehmern als `participantId`-Element eines erweiterten Koordinationskontextes in *allen* Nachrichten mitgesendet werden, so dass ihre Identifikation gewährleistet ist. Dies gilt somit auch für Nachrichten an andere Teilnehmer, die für den Koordinator verborgen sind. Daraus folgt, dass sich Teilnehmer zunächst registrieren müssen, bevor sie weitere Dienste aufrufen können, um vom Koordinator eine Teilnehmer-Id zugewiesen zu bekommen und diese im erweiterten Kontext bei Aufruf anderer Teilnehmer weiterzuleiten. Siehe hierzu auch das Listing 2.3 eines erweiterten Koordinationskontextes auf Seite 32.

Mit der Einführung der Teilnehmer-Ids wird die Voraussetzung geschaffen, Teilnehmer einer Transaktion eindeutig in der Aktivität zu speichern. Ihre individuellen Protokollzustände werden ebenfalls gesichert und garantieren damit die korrekte Fortsetzung des Protokolls. Darüber hinaus können mit Hilfe der Ids leicht weitere Attribute eines Teilnehmers verwaltet werden, die bei der Teilnehmerregistrierung einmalig annotiert werden müssen.

Protokollsteuerung und Zustandswechsel des Koordinators

Wie bereits in Abschnitt 3.2.1 angedeutet, lassen sich die Zustände des Protokolls, wie sie in den Zustandsgraphen abgebildet sind, nicht ohne weiteres auf den Koordinator übertragen, da der Nachrichtenaustausch im Allgemeinen nicht ausschließlich zwischen dem Koordinator und einem einzigen Teilnehmer stattfindet. Vielmehr müssen die Teilnehmerzustände aller beteiligten Kommunikationspartner zu einem Gesamtzustand des Koordinators aggregiert werden, um auf dieser Grundlage über den weiteren Verlauf des Prozesses zu bestimmen. Unter dem *Protokollzustand* wird daher im Folgenden der Zustand *eines* Teilnehmers bzw. der Fortschritt im Protokoll zwischen Koordinator und genau einem Teilnehmer verstanden. Der *Koordinator-* oder *Prozesszustand* beschreibt dagegen die Aggregation aller Teilnehmerzustände aus Sicht des Koordinators zu einem globalen Transaktionszustand.

Die Notwendigkeit zur Aggregation von Teilnehmerzuständen wird bereits in einem Prozess mit nur zwei Teilnehmern deutlich, die sich in unterschiedlichen Zuständen befinden. Wenn beispielsweise ein Teilnehmer mit seiner Arbeit fertig ist und sich im Protokollzustand `Completed` befindet, während der zweite Teilnehmer mit der Durchführung seiner Aufgabe gescheitert ist und dem Koordinator eine `Fail`-Nachricht gesendet hat, wirft dies die Frage auf, welcher Zustand nun für den Koordinator gilt – `Completed` oder `Failed`? Außer bei Verzweigungen des Protokollgraphen, treten ähnliche Situationen auch dann auf, wenn zwar alle Teilnehmer dieselben Zustände durchlaufen – jedoch zu unterschiedlichen Zeitpunkten. Offenbar ist es also eher die Regel als die Ausnahme, dass sich Teilnehmer in unterschiedlichen Zuständen befinden. Mit zunehmender Anzahl von Teilnehmern verstärkt sich dieser Effekt sogar noch, so dass eine eindeutige

Abbildung aller Teilnehmerzustände in einen Prozesszustand benötigt wird, der zweifelsfrei das weitere Vorgehen des Koordinators ausdrückt.

Der Koordinator kann in genau denjenigen Protokollzuständen in die Weiterführung des Prozesses eingreifen, in denen die Teilnehmer auf eine Antwort von ihm warten. Diese Zustände bieten sich daher an, das Protokoll in Phasen aufzuteilen, die die Grundlage für spätere Koordinatorzustände bilden. Insbesondere der Protokollzustand `Completed` trennt den Prozessdurchlauf in eine vorbereitende Phase und in eine Abschlussphase. Desweiteren leiten Protokollzustände, die über Ausstiegs- und Abbruchnachrichten erreicht werden können, im Allgemeinen eine Phase des Abbruchs ein. Wird noch ein Endzustand zur Signalisierung des Transaktionsergebnisses eingeführt, lässt sich ein Koordinator für jeden Koordinationstyp mit jeweils vier Zuständen realisieren. Die Abbildungen 4.1 und 4.2 zeigen die entwickelten Zustandsdiagramme der Koordinatoren für AtomicOutcome und MixedOutcome. Ein Prozessablauf beginnt hierbei stets im Zustand `PREPARING` und verläuft im Erfolgsfall über den Zustand `FINISHING` zum Endzustand `ENDED`. Bei einem Prozessabbruch wird in den Endzustand dagegen über den Zwischenzustand `ABORTING` eingetreten. Eine detaillierte Beschreibung der Koordinatorzustände wird in den folgenden Absätzen gegeben.

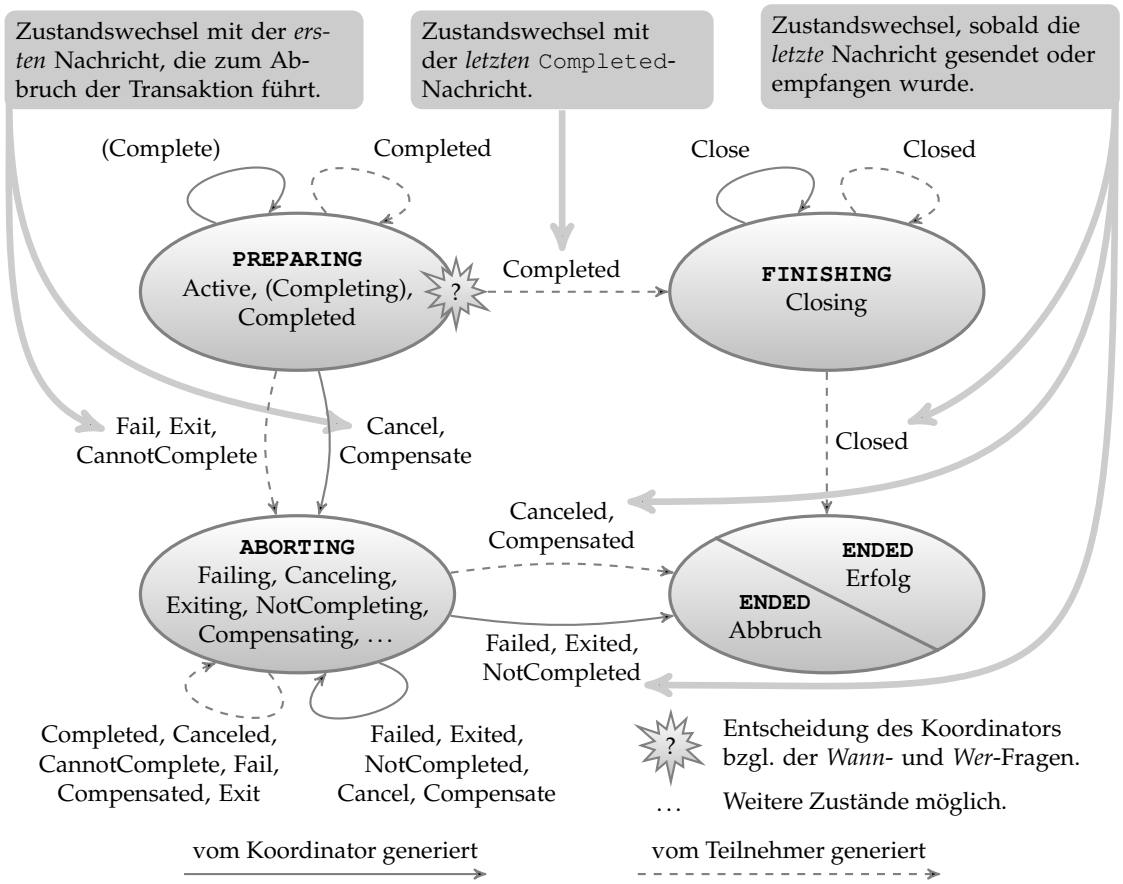

Abbildung 4.1: WS-BusinessActivity – Zustände des AtomicOutcome-Koordinators für die Protokolle BAPC und (BACC).

Aus den Abbildungen ist ersichtlich, dass sich zwar die Prozesszustände gleichen, je-

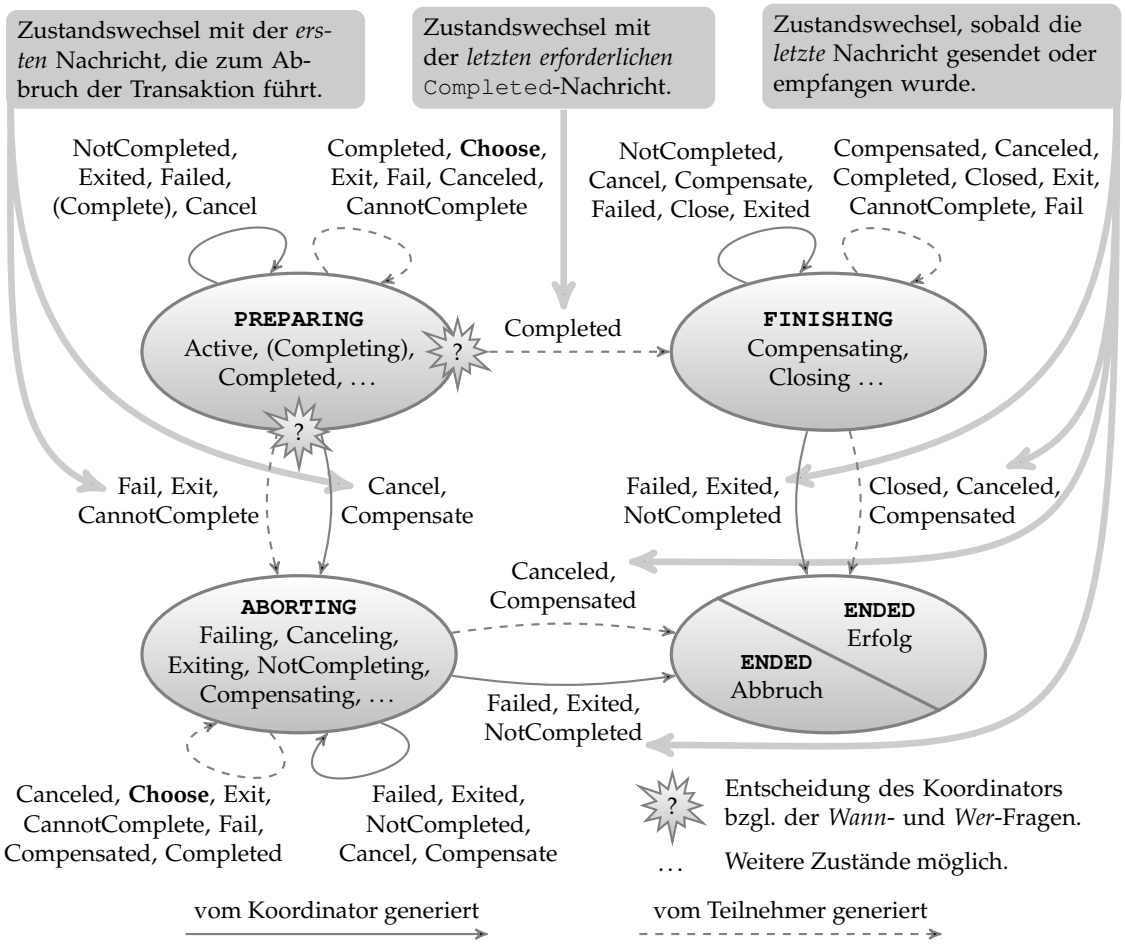

Abbildung 4.2: WS-BusinessActivity – Zustände des MixedOutcome-Koordinators für die Protokolle BAPC und (BACC).

doch teilweise unterschiedliche Nachrichten für einen Zustandsübergang verantwortlich sind. Da der jeweilige Koordinator die Protokollzustände aller Teilnehmer aggregiert, werden in der Regel auch mehrere Nachrichten empfangen oder gesendet *ohne* den Zustand zu wechseln, was durch reflexive Schleifen deutlich gemacht ist. Ein Zustandswechsel wird stets durch die erste oder letzte Nachricht einer Folge von Nachrichten verursacht, so dass eine klare Abgrenzung der Koordinatorzustände möglich wird. Im Einzelnen haben die Koordinatoren folgende Zustände:

`PREPARING`: Der Koordinatorzustand `PREPARING` besteht hauptsächlich aus den Protokollzuständen `Active`, `Completed` und ggf. `Completing` (bei BACC). In dieser Phase muss der Koordinator alle Nachrichten akzeptieren, die Teilnehmer in den zugehörigen Protokollzuständen versenden können. Dabei muss er selbstverständlich den gespeicherten Protokollzustand des Teilnehmers mit einbeziehen, so dass eine protokollgerechte Kommunikation zu Stande kommt. In diesem Sinne sind die Abbildungen als eine vergröberte Wiedergabe des Protokolls zu verstehen, da die Teilnehmerzustände nicht dargestellt werden. Dass weitere Protokollzustände möglich sind, beispielsweise wenn ein Teilnehmer den Prozess mit `Exit` verlässt,

wird in Abbildung 4.2 mit „. . . “ angedeutet.

Sofern keine Nachricht zum Prozessabbruch geführt hat, endet die erste Phase bzw. der Koordinatorzustand `PREPARING`, sobald der letzte notwendige Teilnehmer den Zustand `Completed` erreicht hat. Wann dies der Fall ist, entspricht der bereits zuvor aufgeworfenen Kernfrage nach dem Zeitpunkt für die Einleitung des Transaktionsabschlusses und ist ein wesentlicher Aspekt für die Autonomie des Koordinators. Zusätzlich muss der Koordinator die Frage nach der Identität der abzuschließenden und zu kompensierenden Teilnehmer entscheiden können, bevor er in den Zustand `FINISHING` eintreten kann, was die zweite maßgebliche Problemstellung bzgl. der Autonomie des Koordinators ausmacht. Während sich diese Frage für den AtomicOutcome-Koordinator auf die Feststellung beschränkt, ob sich alle Teilnehmer im `Completed`-Zustand befinden oder mindestens ein Teilnehmer abgebrochen hat, muss der MixedOutcome-Koordinator hinsichtlich der jeweiligen Teilnehmer entscheiden, ob der Prozess fortgeführt werden kann oder abgebrochen werden muss. Die genauen Kriterien, die zur Beantwortung der *Wann*- und *Wer*-Frage führen, werden weiter unten in diesem Abschnitt erläutert.

`FINISHING`: Im Prozesszustand `FINISHING` wickelt der Koordinator die *erfolgreiche* Beendigung des Protokolls ab, da bereits zuvor alle Teilnehmer durch Erreichen des `Completed`-Zustands signalisiert haben, dass sie ihre Arbeit erfolgreich abschließen konnten. In AtomicOutcome-Szenarien entspricht `FINISHING` daher genau dem Protokollzustand `Closing` (vgl. Abb. 4.1), da alle Teilnehmer konsistent abschließen müssen. Im MixedOutcome-Koordinator umfasst `FINISHING` zusätzlich den Protokollzustand `Compensating`, um selektive Ergebnisse realisieren zu können (vgl. Abb. 4.2).

Der Wechsel in den Zustand `FINISHING` geschieht unmittelbar nach Empfang der letzten erforderlichen `Completed`-Nachricht. Bei AtomicOutcome ist zuvor ausschließlich der Zeitpunkt zu klären, wann der Zustandswechsel durchgeführt werden kann, während im Fall von MixedOutcome zusätzlich überprüft werden muss, welche Teilnehmer für den Abschluss notwendig sind (siehe unten).

Da BAPC und BACC eine `Fail`-Nachricht im Protokollzustand `Compensating` erlauben, muss der MixedOutcome-Koordinator im Zustand `FINISHING` ebenfalls darauf reagieren können. Darüber hinaus sind wieder andere Zustände einzelner Teilnehmer möglich, wenn diese unnötig für den Prozessabschluss sind und vom Koordinator mit `Cancel` zum Abbruch bestimmt wurden oder sie bereits selbst den Prozess verlassen haben.

`ABORTING`: Der Zustand `ABORTING` bedeutet den Abbruch des Prozesses. Er wird vom Koordinator eingenommen, sobald dieser im Zustand `PREPARING` von einem für die Transaktion notwendigen Teilnehmer die Nachricht `Fail`, `CannotComplete` oder `Exit` erhält. Die übrigen Teilnehmer werden dann mit `Cancel` zum Abbruch

bewegt oder, falls sie sich bereits in `Completed` befinden, mit `Compensate` zur Kompensation ihrer Teil-TA angeregt. Für den Fall, dass der Koordinationskontext verfällt, weil das mit `Expires` angegebene Verfallsdatum überschritten wurde, darf der Koordinator den Prozess selbst mit `Cancel` oder `Compensate` abbrechen.

Ausschlaggebend für den Abbruch des Prozesses ist, dass das Transaktionsziel nicht mehr erreicht werden kann. Bei AtomicOutcome trifft dies bereits zu, sobald der erste Teilnehmer ausscheidet – gleichgültig mit welcher Nachricht. Bei MixedOutcome kann der Prozess dagegen weiterlaufen, wenn der Teilnehmer nichts zum Transaktionsergebnis beiträgt oder es möglicherweise noch zu früh ist, um über einen Abbruch zu entscheiden. Der Koordinator bleibt dann zunächst im Zustand `PREPARING` und wickelt nur für den ausscheidenden Teilnehmer das Protokoll ab, bis sich dieser im `Ended`-Zustand befindet. Die frühzeitig in `PREPARING` ausgeschiedenen Teilnehmer werden jedoch in einer Liste gesichert, um später feststellen zu können, ob sie für das Transaktionsergebnis relevant gewesen wären. Sollte dies der Fall sein, muss der Prozess doch noch abgebrochen werden. Anderenfalls kann der Prozess zu einem erfolgreichen Ende geführt werden, ohne durch das Ausscheiden des Teilnehmers einen Abbruch provoziert zu haben.

Auch an dieser Stelle spielt die Beantwortung der *Wer*-Frage für den Koordinierung von MixedOutcome eine entscheidende Rolle. Anders als vor dem Eintritt in `FINISHING` besteht hier jedoch die Möglichkeit, dass für einen Teilnehmer noch nicht bestimmt werden kann, ob er notwendig für die Transaktion ist. Dies tritt beispielsweise ein, wenn der ausscheidende Teilnehmer zu einer Choice-Menge gehört, deren Entscheidung noch aussteht. In diesem Fall muss der Koordinator den Zustand `PREPARING` beibehalten, bis er beurteilen kann, ob der Austritt des Teilnehmers zum Abbruch des Prozesses führt.

Die Protokolle von WS-BA lassen – ähnlich dem Scheitern einer Kompensation – auch den Fehlschlag eines Teilnehmers im Zustand `Canceling` zu, was wiederum keinen Zustandswechsel des Koordinators auslöst, da er sich ohnehin in `ABORTING` befindet. Die Nachricht kann daher weitestgehend ignoriert werden, so dass der Koordinator auf die `Fail`-Nachricht nur mit einer protokollgemäßen Bestätigung zu antworten braucht. Auch für den Fall, dass sich der Koordinator noch im Zustand `PREPARING` befindet, hat die `Fail`-Nachricht eines Teilnehmers im Zustand `Canceling` keine Bedeutung, da der Koordinator den betreffenden Teilnehmer bereits für den Abbruch vorgesehen hat und der Prozess dadurch nicht gefährdet wird. Die möglichen Interpretationen von Ausstiegsnachrichten werden im nachfolgenden Abschnitt ausführlich diskutiert.

`ENDED`: Der Koordinator wechselt in den Zustand `ENDED`, wenn die letzte Nachricht gesendet oder empfangen wurde, d.h. wenn sich alle registrierten Teilnehmer ihrerseits im Protokollzustand `Ended` befinden. Je nach dem, ob der Koordinator den

Endzustand über `FINISHING` oder `ABORTING` erreicht hat, signalisiert der Zustand den erfolgreichen Abschluss oder ein Scheitern des Prozesses.

Mit Erreichen des Endzustands sind alle Kommunikationspartner über das Ende der Aktivität informiert, so dass sie die Protokollinstanz für den aktuellen Koordinationskontext löschen können.

Auf Grund des fehlenden Initiator-Koordinator-Protokolls kann der Initiator der Transaktion allerdings nicht über den Ausgang des Prozesses informiert werden. Am Ergebnis der Transaktion interessierte Teilnehmer können sich jedoch mit dem Service-Aufruf `getStatus` eigenständig beim Koordinator über den Ausgang informieren.

Aus der Perspektive des Koordinators besitzt ein vollständiger Protokolldurchlauf bei dieser Realisierung also lediglich vier Zustände. Der jeweils aktuelle Prozesszustand wird vom Koordinator als Attribut der Aktivität abgelegt, die über die Id des Koordinationskontextes eindeutig zu referenzieren ist.

Obwohl der Protokollzustand eines Teilnehmers bereits als Eigenschaft seines Teilnehmerobjekts vorgehalten wird, ist es für eine effiziente Abfrage der Teilnehmer eines Protokollzustands sinnvoll, die Protokollzustände zusätzlich als Listen von Teilnehmer-Ids zu repräsentieren, welche genau die Ids derjenigen Teilnehmer enthalten, die sich gegenwärtig im jeweiligen Zustand befinden. Dies erleichtert insbesondere die konkrete Überprüfung bestimmter Protokollzustände auf das Vorhandensein von Teilnehmern. Bei einem Zustandswechel eines Teilnehmers bedeutet dies jedoch, dass der Koordinator sowohl das Teilnehmerobjekt als auch beide beteiligten Zustandslisten aktualiseren muss.

Bei einem Vergleich der Koordinatormodelle für AtomicOutcome und MixedOutcome fällt auf, dass die Trennungsschärfe der Zustände und Zustandsübergänge im Atomic-Modell deutlicher ausgeprägt ist. So kann der Atomic-Koordinator ausschließlich über `Close`-Nachrichten in den Zustand `FINISHING` eintreten, da dieser äquivalent zum Protokollzustand `Closing` ist. Im Gegensatz dazu werden im MixedOutcome-Modell unter Umständen mehrere Protokollzustände zum Koordinatorzustand `FINISHING` aggregiert, wobei `Failing` und `Compensating` je nach Koordinatorentscheidung auch zum Prozesszustand `ABORTING` gezählt werden können. Entsprechend der Aggregation von Protokollzuständen, sind auch mehrere Nachrichten möglich, die der Koordinator für die Zustandsübergänge berücksichtigen muss.

Es sei hier darauf hingewiesen, dass Abbildung 4.2 (teilweise auch Abb. 4.1) nicht alle möglichen Protokollzustände von Teilnehmern in den Koordinatorzuständen wiedergibt, sondern nur die „typischen“ Protokollzustände. Ein Beispiel hierzu: Der Koordinator befinde sich im Zustand `PREPARING` und erhalte von einem nicht erforderlichen Teilnehmer eine `Exit`-Nachricht. Der betreffende Teilnehmer wird dann protokollgemäß in den `Ended`-Zustand geführt; der Prozesszustand bleibt jedoch `PREPARING`, da der Ausstieg des Teilnehmers irrelevant für den Prozessabschluss ist. In einer vollständigen

Darstellung müssten daher auch die Protokollzustände `Exiting` und `Ended` im Koordinatorzustand `PREPARING` abgebildet werden und – analog – die Zustände `Canceling`, `NotCompleting` und `Failing`. Sie sind jedoch nicht signifikant für diesen Zustand und daher wurde aus Gründen der Übersichtlichkeit darauf verzichtet, sie einzuzeichnen. Gleichwohl liefern die Kantenbeschriftungen einen Hinweis auf alle weiteren mögliche Zustände. Die dargestellten Protokollzustände sind gewissermaßen *charakteristisch* für den jeweiligen Koordinatorzustand.

Interpretation der Ausstiegsnachrichten

Die Protokolle aus WS-BusinessActivity gestatten es den Teilnehmern durch unterschiedliche Nachrichten den Transaktionsprozess zu verlassen (vgl. [FL07]). Mit `Exit` teilt ein Dienst dem Koordinator mit, dass er nicht bereit ist, die Transaktion fortzusetzen. Dabei wird jegliche verbleibende Arbeit vom Teilnehmer verworfen und bereits durchgeführte Aktionen widerrufen. Eine `CannotComplete`-Nachricht informiert den Koordinator über einen auf Teilnehmerseite aufgetretenen Fehler, der den erfolgreichen Abschluss der Operation verhindert. Die Störung konnte allerdings abgefangen werden und die Effekte der laufenden Prozessinstanz lassen sich unwirksam machen. Der Empfang einer `Fail`-Nachricht setzt den Koordinator hingegen darüber in Kenntnis, dass der Teilnehmer in den Zuständen `Active`, `Completing` (BACC), `Compensating` oder `Canceling` gescheitert ist, der Stand der durchgeführten Arbeit jedoch unbestimmt bleibt.

Nach Maßgabe der Spezifikation hat der Koordinator den Erhalt der jeweiligen Ausstiegsnachricht in jedem Fall mit einer protokollgemäßen Bestätigung zu quittieren und den Teilnehmer anschließend zu „vergessen“. Trotz der unterschiedlichen Ursachen für einen Austritt der Teilnehmer bleibt als gemeinsame Konsequenz, dass die Dienste für den weiteren Verlauf des Prozesses nicht mehr zur Verfügung stehen. Der Koordinator muss dann entscheiden, ob eine erfolgreiche Durchführung der Transaktion noch möglich ist. Für AtomicOutcome-Transaktionen kann dies stets verneint werden.

Prozesse des MixedOutcome-Typs können dagegen weitergeführt werden, wenn Teilnehmer mit `Exit` oder `CannotComplete` die Transaktion verlassen, sofern der ausgeschiedene Dienst irrelevant für das Transaktionergebnis ist, da dies bei `Exit` ohne Fehlersituation aus eigenem Antrieb heraus geschieht und der aufgetretene Fehler bei `CannotComplete` vom Teilnehmer behandelbar ist.

Bricht der Teilnehmer dagegen mit einer `Fail`-Nachricht ab, ist ein gravierender Fehler aufgetreten und sein Zustand undefiniert. Dies ist besonders kritisch, wenn sich der Teilnehmer im Zustand `Compensating` befindet, da nun unklar ist, ob er durchgeführte Änderungen zurücknehmen konnte oder nicht. Leider gibt die WS-BA-Spezifikation keine Auskunft darüber, welchen Effekt dies auf den Prozess haben muss. Es wird lediglich definiert, dass der Koordinator mit `Failed` zu antworten hat und anschließend beide Partner den Vorfall „vergessen“ sollten. Die Auswirkungen auf den Prozess und die anderen Teilnehmer bleiben also unspezifiziert und sind damit anwendungsabhängig.

Grundsätzlich ergeben sich auch bei Erhalt von `Fail`-Nachrichten die beiden Möglichkeiten, den Prozess entweder weiterzuführen, falls ein positiver Abschluss des gescheiterten Teilnehmers nicht notwendig für das Transaktionsergebnis ist oder die Transaktion insgesamt abzubrechen.

Fortführung: Soll der Prozess nach Erhalt einer `Fail`-Nachricht weitergeführt werden, ergeben sich die geringsten Auswirkungen für die Koordinierung, wenn Teilnehmer im Zustand `Compensating` oder `Canceling` scheitern. In beiden Fällen ist der positive Abschluss nicht relevant für das Transaktionsergebnis, so dass sich hier keine Änderung des Prozesszustands ergibt. Dennoch sollte bei einem `Fail` im Zustand `Compensating` eine nachfolgende Untersuchung des des Vorfalls stattfinden, um eventuelle Schäden abzuwenden.

Alle anderen Ausstiegsnachrichten können einen Prozessabbruch zur Folge haben. Die Möglichkeiten des Koordinators dies zu verhindern, beschränken sich auf die Auswertung der Choice-Mengen und Vitalität der Teilnehmer sowie weiterer in Abschnitt 3.2 vorgestellter Mechanismen zur Erhöhung der Prozessstabilität. Aus diesem Grund kann der Koordinator hier nicht, wie von der Spezifikation gefordert, sämtliche Information des ausgeschiedenen Teilnehmers löschen, sondern er muss sich stattdessen alle Teilnehmer merken, die den Prozess verlassen haben, um diese später mit den abzuschließenden Teilnehmern der Complete-Menge abzugleichen. Denn erst wenn bereits abgebrochene Teilnehmer in der Complete-Menge enthalten sind, ist ein Prozessabbruch notwendig.

Abbruch: Wird der Empfang einer `Fail`-Nachricht immer als schwerwiegender Vorfall betrachtet, ist der Prozess abzubrechen. Dabei sind `Exit`- oder `CannotComplete`-Nachrichten weiterhin erlaubt, da sich die Teilnehmer hierbei nicht in einem undefinierten Zustand befinden. In Abbildung 4.2 kann dieses Verhalten durch Einfügen einer `Fail`-Kante vom Zustand `FINISHING` in den Zustand `ABORTING` erzielt werden.

Um einem Teilnehmer zu gestatten, mit einer `Fail`-Nachricht auch im Zustand `Compensating` einen Gesamtabbruch auszulösen, muss der Koordinator sicherstellen, dass mit dem Versenden der `Close`-Nachrichten erst *nach* dem Empfang der letzten `Compensated`-Bestätigung begonnen wird. Ansonsten kann nicht garantiert werden, dass wirklich alle Teilnehmer noch in der Lage sind, abzubrechen oder zu kompensieren.

Über das Verhalten des Koordinators muss bei allen Teilnehmern im Vorwege Klarheit herrschen, so dass ein zur Anwendung passender Koordinator ausgewählt werden kann. Alternativ sind Koordinatoren denkbar, deren Verhalten für die Behandlung der Ausstiegsnachrichten parametrisierbar ist. Dem Koordinator müsste die gewünschte Verhaltensweise dabei über eine entsprechende Deklaration bei Erstellung des Koordinationskontextes mitgeteilt werden. Eine Parametrisierung des Koordinators ließe sich im Ko-

ordinator einfach realisieren – das primäre Problem betrifft eher die Aushandlung der Parameter zwischen den Teilnehmern vor Prozessbeginn.

In dieser Arbeit wird das Hauptaugenmerk auf eine möglichst robuste Prozessdurchführung gelegt, so dass Transaktionen von den hier entwickelten Koordinatoren nur dann abgebrochen werden, wenn keine Aussicht mehr besteht, den Prozess zu einem erfolgreichen Ende zu führen. Insbesondere findet kein Abbruch des Prozesses bei Erhalt einer `Fail`-Nachricht statt, wenn kein positiver Abschluss des Teilnehmers für den Prozess erforderlich ist.

4.1.2 Verwaltung der Prozessausdehnung

Der Koordinator kann erst in den Zustand `FINISHING` wechseln, wenn sich *alle* Teilnehmer, die für einen erfolgreichen Transaktionsabschluss benötigt werden, im Protokollzustand `Completed` befinden. Er muss also die Gesamtzahl aller Teilnehmer kennen, die der Prozess in seiner maximalen Ausdehnung haben wird. Hierfür ist es erforderlich, dass die Teilnehmer bei ihrer Registrierung bekanntgeben, wieviele weitere Teilnehmer sie während des Prozesses aufrufen werden. Der Koordinator erwartet diese Angabe als zusätzliches Element `subsequentCalls` eines erweiterten Koordinationskontextes.

Der Vorteil dieser Art der Datenübermittlung besteht darin, dass wie bei der Zuweisung der Teilnehmer-Ids keine zusätzlichen Nachrichten benötigt werden. Alle erforderlichen Daten werden im Kontext bereits vorhandener Nachrichten aus WS-Coordination transportiert.

Mit Hilfe der Teilnehmer-Ids sowie der Anzahl künftig aufgerufener Dienste kann der Koordinator bei der Registrierung des n-ten Teilnehmers die Anzahl der bereits registrierten Teilnehmer $P_{\text{reg}}(n)$ sowie die Anzahl der erwarteten Teilnehmer $P_{\text{exp}}(n)$ durch folgende Rekurrenzgleichungen bestimmen:

$$\begin{aligned} P_{\text{reg}}(0) &= 0 \\ P_{\text{reg}}(n) &= P_{\text{reg}}(n-1) + 1 \end{aligned} \tag{4.1}$$

$$\begin{aligned} P_{\text{exp}}(0) &= 1 \\ P_{\text{exp}}(n) &= P_{\text{exp}}(n-1) + \texttt{subsequentCalls}_n \end{aligned} \tag{4.2}$$

Die Anzahl der registrierten Teilnehmer $P_{\text{reg}}(n)$ wird also mit 0 initialisiert und bei jeder Registrierung um 1 inkrementiert. Dagegen wird die Anzahl der erwarteten Teilnehmer $P_{\text{exp}}(n)$ hauptsächlich durch die schrittweise Summierung der $\texttt{subsequentCalls}_n$ gebildet, welche die Anzahl der Dienste angeben, die der n-te Teilnehmer noch aufrufen wird. Der Initiator wird hierbei über die Initialisierung von $P_{\text{exp}}(0)$ mit dem Wert 1 berücksichtigt.

Die Werte werden bei jeder Registrierung miteinander verglichen und wenn schließlich eine Übereinstimmung von $P_{\text{reg}}(n)$ und $P_{\text{exp}}(n)$ vorliegt, ist die maximale Teilneh-

merzahl der Transaktion erreicht. Es sind also keine Teilnehmer mehr zu erwarten, die dem Prozess beitreten werden und es gilt:

$$P_{\text{reg}}(n) = P_{\text{exp}}(n) \tag{4.3}$$

Dem Koordinator steht nun eine erste Bedingung zur Verfügung, um über den Zeitpunkt für die Einleitung der zweiten Prozessphase zu entscheiden. Für die Koordination nach AtomicOutcome hat der Koordinator damit bereits genügend Informationen, um den Prozess fortzusetzen. In MixedOutcome-Szenarien reicht dies allein jedoch nicht aus, da möglicherweise noch abzuschließende Teilnehmer aus Choice-Mengen bestimmt werden müssen.

4.1.3 Verwaltung struktureller Information

Die Festlegung von festzuschreibenden und abzubrechenden Teilnehmern in Transaktionen des MixedOutcome-Typs erfordert eine globale Sicht auf den Prozess, da die Abhängigkeiten zwischen den Dienstaufrufen die Entscheidungsgrundlage bzgl. Erfolg oder Abbruch der Transaktion liefern. Diese Information erhält der Koordinator mit Hilfe des in Abschnitt 3.2.2 beschriebenen Aufrufbaums, den er bei Registrierung eines Teilnehmers entsprechend erweitert. Die Voraussetzung hierfür stellt die bei der Registrierung eines Dienstes mitgesendete `callerId` dar, mit der ein neuer Teilnehmer angibt, von wem er aufgerufen wurde. Die zuvor aufgestellte Forderung, nach der in jeder Nachricht die Teilnehmer-Id des Senders enthalten sein muss, ermöglicht einem aufgerufenen Dienst zu erkennen, wer ihn aufgerufen hat und dies an den Koordinator weiterzuleiten.

Choice-Mengen

Ein weiterer Aspekt hierbei ist die Auswahl von Teilnehmern einer Choice-Menge, mit der ein Dienstnutzer dem Koordinator auf Basis seiner Anwendungslogik mitteilen kann, welche Teilnehmer für den weiteren Prozess relevant sind und welche Teilnehmer abgebrochen werden können (vgl. hierzu Abb. 3.1 auf S. 46). Die Weiterleitung der für die Koordination benötigten Daten erfolgt abermals nach dem bekannten Prinzip: Der Dienstnutzer teilt dem aufgerufenen Teilnehmer mit, dass dieser Mitglied einer Choice-Menge ist und bei der darauffolgenden Registrierung des Teilnehmers, leitet dieser die Information an den Koordinator weiter. Die eigentliche Auswahl der abzuschließenden Teilnehmer erfolgt mit einer separaten `Choose`-Nachricht des Aufrufers, in der alle ausgewählten Teilnehmer aufgeführt sind.

Um die `Choose`-Nachricht generieren zu können, muss der Dienstnutzer wiederum die Teilnehmer-Ids der durch ihn eingebundenen Dienste kennen. Daher ist es wichtig, dass sich der aufgerufene Dienst registriert und auf Zuteilung seiner `participantId` wartet *bevor* er dem Dienstnutzer antwortet. In der Antwortnachricht des Teilnehmers für die aufgerufene Anwendungsfunktion ist dann neben dem Operationsergebnis zusätz-

lich die eigene Teilnehmer-Id enthalten, so dass der Aufrufer in der Lage ist, eindeutig zwischen mehreren Ergebnissen auszuwählen.

Für die Auswahl der Teilnehmer wird vereinbart, dass es pro deklarierter Choice-Menge stets genau *eine* `Choose`-Nachricht gibt, in der möglicherweise auch mehrere Teilnehmer ausgewählt werden. Dies hat den Vorteil, dass der Koordinator exakt feststellen kann, wieviele `Choose`-Nachrichten noch ausstehen und wann eine endgültige Entscheidung über den Abschluss der Teilnehmer getroffen werden kann.

In Abb. 4.2 ist die `Choose`-Nachricht durch Fettdruck hervorgehoben, da sie eine Erweiterung darstellt und nicht von WS-BusinessActivity spezifiziert wird. Das Modell zeigt außerdem, dass die `Choose`-Nachricht nur im Koordinatorzustand `PREPARING` erwartet wird, da sie für die Fortführung des Prozesses eine Schlüsselrolle einnimmt. Ein Beispiel für eine `Choose`-Nachricht ist in Listing 4.1 gegeben.

Listing 4.1: Choose-Nachricht zur Auswahl eines Teilnehmers.

```
1 <soapenv:Body>
2   <ns2:Choose xmlns:ns2="http://coordination.vsis.de/wsba/"
3       ChoiceSet="4272A44F-8427-4333-B74C-5DA18A223A21">
4     <ns2:ParticipantId>
5       C8186C05-B6C2-46CE-8206-19C00E9C66EC
6     </ns2:ParticipantId>
7   </ns2:Choose>
8 </soapenv:Body>
```

Im Listing ist ersichtlich, dass Choice-Mengen ebenfalls durch eine Id gekennzeichnet werden. Dies ermöglicht die Verwaltung mehrerer Choice-Mengen eines Dienstnutzers und damit die korrekte Zuordnung der `Choose`-Nachrichten zu korrespondierenden Choice-Mengen.

Inferenzregeln

Für die Entscheidungsfindung des Koordinators wird die Gesamtzahl der Teilnehmer eines Prozesses in eine Cancel-Menge sowie eine Complete-Menge eingeteilt. Die disjunkten Mengen beschreiben das potentielle Abschlussverhalten der Teilnehmer auf Grundlage ihres aktuellen Protokollzustands und geben jederzeit Auskunft über abzuschließende Dienste bzw. über zu verwerfende Teilergebnisse. Als Voreinstellung wird jeder Teilehmer bei seiner Registrierung der Complete-Menge zugeordnet, da zunächst angenommen werden kann, dass ein aufgerufener Dienst seine Leistung auch erbringen wird. Im Verlauf des Prozesses werden die Mengenzugehörigen der Teilnehmer fortlaufend aktualisiert, so dass der Koordinator technisch stets in der Lage ist, die Prozessbeendigung einzuleiten, da die Teilnehmer mit ihrer Mengenzugehörigkeit signalisieren, wie sie den Prozess für einen erfolgreichen Transaktionsabschluss beenden müssen. Da eine frühzeitige Auswertung der Complete- und Cancel-Menge jedoch Teilnehmer unberücksichtigt

lässt bzw. unnötig zum Festschreiben ihrer Ergebnisse dirigiert, sofern der Prozess noch nicht seine maximale Ausdehnung erreicht hat oder noch Choice-Mengen entschieden werden müssen, sollte der Koordinator warten, bis diese Voraussetzungen gegeben sind.

Auf Basis der Interaktionen zwischen den Teilnehmern kann die Mengenzuordnung vom Koordinator häufig vollkommen autonom vorgenommen werden, ohne dabei auf Anwendungslogik der Teilnehmer angewiesen zu sein. Im Zusammenhang mit Choice-Mengen ist es dagegen erforderlich, anwendungsspezifisch zwischen den Diensten auszuwählen. Diese Entscheidung liegt jedoch in alleiniger Verantwortung des Dienstnutzers, der seine Selektion mit Hilfe der `Choose`-Nachricht an den Koordinator übermittelt. Auf diese Weise bleibt der Koordinator anwendungsneutral und realisiert ausschließlich die technischen Aspekte der Transaktionssteuerung.

Im TracG-Projekt wurden grundlegende Regeln identifiziert, die sich auf die Struktur des Aufrufbaumes abstützen und die Zuordnung von Teilnehmern zur Complete- oder Cancel-Menge gestatten (vgl. [RHR08]):

$$\forall a \forall b \forall z (\text{invocation}(a,b,z) \wedge \text{completion}(a,z) \wedge \neg(\text{partOfChoice}(b,z))) \rightarrow \text{completion}(b,z) \tag{4.4}$$

$$\forall a \forall b \forall z (\text{invocation}(a,b,z) \wedge \text{completion}(a,z) \wedge \text{partOfChoice}(b,z) \wedge \text{choose}(b,z)) \rightarrow \text{completion}(b,z) \tag{4.5}$$

Danach kann ein Teilnehmer b der Complete-Menge zugeordnet werden, wenn er innerhalb des Kontextes z von einem Teilnehmer a aufgerufen wird, der bereits zur Complete-Menge gehört (vgl. Regel 4.4). Wird Teilnehmer b dagegen als Element einer Choice-Menge von a aufgerufen, wird er nur dann der Complete-Menge zugeordnet, wenn er auch ein durch eine `Choose`-Nachricht ausgewählter Teilnehmer der Choice-Menge ist (vgl. Regel 4.5).

$$\forall a \forall b \forall z (\text{invocation}(a,b,z) \wedge \text{cancel}(a,z)) \rightarrow \text{cancel}(b,z) \tag{4.6}$$

$$\forall a \forall b \forall z (\text{invocation}(a,b,z) \wedge \text{partOfChoice}(b,z) \wedge \neg(\text{choose}(b,z))) \rightarrow \text{cancel}(b,z) \tag{4.7}$$

Umgekehrt wird Teilnehmer b der Cancel-Menge zugeordnet, wenn der Dienstnutzer bereits der Cancel-Menge angehört (vgl. Regel 4.6) oder wenn b als Teilnehmer einer Choice-Menge von a aufgerufen wurde, aber nicht für den Abschluss ausgewählt wird (vgl. Regel 4.7).

Die Regeln bilden die Struktur des Aufrufbaumes nach, indem Mengenzugehörigkeiten von der Wurzel entlang der Zweige in Richtung der Blätter propagiert werden. Dies ist insbesondere dann wichtig, wenn ein Teilnehmer von der Complete-Menge in die Cancel-Menge verschoben wird, weil er kein ausgewählter Teilnehmer einer Choice-Menge ist (vgl. Regel 4.7). Durch Regel 4.6 wird gleichzeitig der gesamte Teilbaum, zu dem der verschobene Teilnehmer die Wurzel bildet, der Cancel-Menge zugeordnet.

Obwohl Choice-Mengen innerhalb von Nachrichten mit einer Id referenziert werden,

ist diese bei Anwendung der Regeln nicht relevant. Die Ids helfen dem Koordinator lediglich festzustellen, welche Choice-Mengen ausgewählt wurden und vor allem, wie viele `Choose`-Nachrichten noch ausstehen. Auf der Ebene der Inferenzregeln ist dagegen entscheidend, *ob* ein Teilnehmer ausgewählt wurde oder nicht – welcher Choice-Menge er angehört, ist nebensächlich.

Die Auswertung der Complete- und Cancel-Menge beendet den Koordinatorzustand `PREPARING` und legt fest, welche Teilnehmer abschließen oder kompensieren müssen. Im Zustand `FINISHING` erhalten Teilnehmer der Complete-Menge entsprechend eine `Close`-Nachricht und Teilnehmer der Cancel-Menge eine `Compensate`-Nachricht.

Für den Fall, dass Teilnehmer abgebrochen haben, die sich nun in der Complete-Menge befinden, kann das Prozessziel nicht mehr erreicht werden. Der Koordinator wechselt in den Zustand `ABORTING` und sendet allen Teilnehmern im Protokollzustand `Completed` eine `Compensate`-Nachricht. Der Prozess wird damit abgebrochen.

4.1.4 Gerüst eines Algorithmus

Im Folgenden wird ein ereignisbasierter Entwurf eines Koordinators vorgestellt, der aktiv wird, sobald das Eintreffen einer Nachricht ein Ereignis auslöst.

Nach der Spezifikation werden Duplikate von Nachrichten ignoriert und verlorengegangene Nachrichten erneut gesendet. Dagegen lösen nicht protokollgemäße Nachrichten einen Fehler aus, so dass das Protokoll abbricht. Diese Ereignisse werden im nachfolgenden Entwurf jedoch nicht berücksichtigt, da der Fokus an dieser Stelle auf die Zustandsübergänge des Koordinators gelegt wird sowie auf Nachrichten, die für das Fortschreiten des Prozesses von Bedeutung sind. In einer späteren Implementierung ist die Behandlung von Duplikaten und verlorengegangenen oder falschen Nachrichten selbstverständlich zu integrieren.

`PREPARING`

Nach der Erstellung eines Koordinationskontextes befindet sich der Koordinator im Zustand `PREPARING`, wo der Prozess aufgebaut wird, indem sich Teilnehmer registrieren. Das Ziel ist, die Teilnehmer in den `Completed`-Zustand zu dirigieren.

Registrierung eines Teilnehmers:

- Hinzufügen des Teilnehmers zur Aktivität,
- Setzen des Teilnehmerzustands auf `Active`,
- Hinzufügen des Teilnehmers zur `activeList`,
- Zuordnung des Teilnehmers zur Complete-Menge sowie
- Inkrementierung der Anzahl registrierter Teilnehmer $P_{\text{reg}}(n)$ um 1.

- Falls in der `Register`-Nachricht `subsequentCalls` gesetzt ist:
  - Inkrementierung der Anzahl erwarteter Teilnehmer $P_{\text{exp}}(n)$ um die Anzahl `subsequentCalls`.
- Bei MixedOutcome:
  - Falls in der `Register`-Nachricht `ChoiceSet` gesetzt ist:
    * Aktualisierung der Regelmenge und
    * Inkrementierung der Anzahl zu entscheidender Choice-Mengen.
- Wenn das Protokoll BACC ist, sich der Koordinator noch im Zustand `PREPARING` befindet und keine Teilnehmer mehr erwartet werden, also $P_{\text{reg}}(n) = P_{\text{exp}}(n)$ gilt:
  - Senden einer `Complete`-Nachricht an jeden Teilnehmer der `activeList`,
  - Aktualisierung des Teilnehmerzustands auf `Completing` und
  - Verschieben des Teilnehmers in die `completingList`.
  - Bei MixedOutcome:
    * Cancel- und Complete-Mengen vorab auswerten und
    * Teilnehmern der Complete-Menge, die in sich in der `activeList` befinden, eine `Complete`-Nachricht senden, sie in die `completingList` verschieben und ihren Teilnehmerzustand auf `Completing` aktualisieren sowie
    * Teilnehmern der Cancel-Menge, die in sich in der `activeList` befinden, eine `Cancel`-Nachricht senden, sie in die `cancelingList` verschieben und ihren Teilnehmerzustand auf `Canceling` aktualisieren.

Empfang einer `Completed`-Nachricht:

- Aktualisierung des Teilnehmerzustands auf `Completed` und
- Verschieben des Teilnehmers in die `completedList`.
- Bei AtomicOutcome:
  - Falls der Koordinatorzustand `PREPARING` ist, keine Teilnehmer mehr erwartet werden, also $P_{\text{reg}}(n) = P_{\text{exp}}(n)$ gilt, und die `completingList` sowie die `activeList` leer sind:
    * Wechsel des Koordinatorzustands nach `FINISHING`.
  - Sonst:
    * Beibehaltung des Koordinatorzustands `PREPARING`.
- Bei MixedOutcome:

  - Falls der Koordinatorzustand `PREPARING` ist, keine Teilnehmer mehr erwartet werden, also $P_{\text{reg}}(n) = P_{\text{exp}}(n)$ gilt, und keine Entscheidungen bzgl. Choice-Mengen mehr ausstehen:
    * Auswertung der Cancel- und Complete-Mengen.
    * Falls die `abortedList` nicht leer ist und sich mindestens ein Teilnehmer der `abortedList` in der Complete-Menge befindet:
      · Wechsel des Koordinatorzustands nach `ABORTING`.
    * Falls die `abortedList` leer ist und sich alle Teilnehmer der Complete-Menge in der `completedList` befinden:
      · Wechsel des Koordinatorzustands nach `FINISHING`.
  - Sonst:
    * Beibehaltung des Koordinatorzustands `PREPARING`.

Empfang einer `Exit`-, `CannotComplete`- oder `Fail`-Nachricht:

- Bei AtomicOutcome:
  - Wechsel des Koordinatorzustands nach `ABORTING`.
- Bei MixedOutcome:
  - Aktualisierung des Teilnehmerzustands auf `Exiting`, `NotCompleting` oder `Failing`,
  - Verschieben des Teilnehmers in die `abortedList` und
  - Quittieren der Nachricht mit `Exited`, `NotCompleted` oder `Failed`.
  - Falls der Koordinatorzustand `PREPARING` ist, keine Teilnehmer mehr erwartet werden, also $P_{\text{reg}}(n) = P_{\text{exp}}(n)$ gilt, keine Entscheidungen bzgl. Choice-Mengen mehr ausstehen und die `activeList` sowie die `completingList` leer sind:
    * Auswertung der Cancel- und Complete-Mengen.
    * Falls die `abortedList` nicht leer ist und sich mindestens ein Teilnehmer der `abortedList` in der Complete-Menge befindet:
      · Wechsel des Koordinatorzustands nach `ABORTING`.
    * Sonst:
      · Wechsel des Koordinatorzustands nach `FINISHING`.
  - Sonst:
    * Beibehaltung des Koordinatorzustands `PREPARING`.

Empfang einer `Canceled`-Nachricht (nur bei MixedOutcome):

- Aktualisierung des Teilnehmerzustands auf `Ended` und
- Verschieben des Teilnehmers in die `abortedList`.

Empfang einer `Choose`-Nachricht (nur bei MixedOutcome):

- Aktualisierung der Regelmenge und
- Dekrementierung der Anzahl zu entscheidender Choice-Mengen.

`FINISHING`

In `FINISHING` wird der Prozess erfolgreich beendet. Die Teilnehmer werden angewiesen, den Prozes abzuschließen oder ggf. zu kompensieren.

Nach einem Wechsel zu `FINISHING`:

- Bei AtomicOutcome:
  - Senden einer `Close`-Nachricht an jeden Teilnehmer der `completedList`,
  - Aktualisierung des Teilnehmerzustands auf `Closing` und
  - Verschieben des Teilnehmers in die `closingList`.
- Bei MixedOutcome:
  - Senden von `Close`- und `Compensate`-Nachrichten an Teilnehmer entsprechend ihrer Zugehörigkeit zur Complete- oder Cancel-Menge,
  - Aktualisierung der Teilnehmerzustände auf `Closing` oder `Compensating` und
  - Verschieben der Teilnehmer in die `closing`- oder die `compensatingList`.

Empfang einer `Closed`-Nachricht:

- Aktualisierung des Teilnehmerzustands auf `Ended` und
- Löschen des Teilnehmers aus der `closingList`.
- Bei AtomicOutcome:
  - Falls der Koordinatorzustand `FINISHING` ist und die `closingList` leer ist:
    - Wechsel des Koordinatorzustands nach `ENDED_SUCCESS`
  - Sonst:
    - Beibehaltung des Koordinatorzustands `FINISHING`.
- Bei MixedOutcome:
  - Falls der Koordinatorzustand `FINISHING` ist und die `closingList` und die `compensatingList` leer sind:

        * Wechsel des Koordinatorzustands nach `ENDED_SUCCESS`
    - Sonst:
        * Beibehaltung des Koordinatorzustands `FINISHING`.

Empfang einer `Compensated`-Nachricht (nur bei MixedOutcome):

- Aktualisierung des Teilnehmerzustands auf `Ended` und
- Löschen des Teilnehmers aus der `compensatingList`.
- Falls der Koordinatorzustand `FINISHING` ist und die `compensatingList` und die `closingList` leer sind:
    - Wechsel des Koordinatorzustands nach `ENDED_SUCCESS`
- Sonst:
    - Beibehaltung des Koordinatorzustands `FINISHING`.

Empfang einer `Fail`-Nachricht (nur bei MixedOutcome):

- Aktualisierung des Teilnehmerzustands auf `Failing`,
- Löschen des Teilnehmers aus der `compensatingList` und
- Senden einer `Failed`-Nachricht.
- Falls der Koordinatorzustand `FINISHING` ist und die `compensatingList` und die `closingList` leer sind:
    - Wechsel des Koordinatorzustands nach `ENDED_SUCCESS`
- Sonst:
    - Beibehaltung des Koordinatorzustands `FINISHING`.

`ABORTING`

In `ABORTING` wird der Prozess abgebrochen. Alle Teilnehmer müssen entsprechend ihres Protokollzustands über das Scheitern des Prozesses informiert werden.

Nach einem Wechsel zu `ABORTING`:

- Falls eine Teilnehmernachricht zum Abbruch geführt hat:
    - Aktualisierung des Protokollzustands des den Abbruch auslösenden Teilnehmers auf `Exiting`, `NotCompleting` oder `Failing`,
    - Löschen des Teilnehmers aus der `activeList` oder der `completingList` und

  - Quittieren der Abbruchnachricht mit `Exited`, `NotCompleted` oder `Failed`.

- Senden einer `Cancel`-Nachricht an jeden Teilnehmer der `activeList` oder der `completingList` (BACC) sowie einer `Compensate`-Nachricht an jeden Teilnehmer der `completedList`,
- Aktualisierung ihrer Protokollzustände auf `Canceling` oder `Compensating` und
- Verschieben dieser Teilnehmer in die `cancelingList` oder `compensatingList`.

Empfang einer `Canceled`- oder `Compensated`-Nachricht:

- Aktualisierung des Teilnehmerzustands auf `Ended` und
- Löschen des Teilnehmers aus der `cancelingList` oder der `compensatingList`.
- Falls der Koordinatorzustand `ABORTING` ist und die `cancelingList` und die `compensatingList` leer sind:
  - Wechsel des Koordinatorzustands nach `ENDED_FAILURE`
- Sonst:
  - Beibehaltung des Koordinatorzustands `ABORTING`.

Empfang einer `Exit`-, `CannotComplete`- oder `Fail`-Nachricht:

- Aktualisierung des Teilnehmers auf `Exiting`, `NotCompleting` oder `Failing`,
- Löschen des Teilnehmers aus der `activeList` oder der `completingList` und
- Quittieren der Ausstiegsnachricht mit `Exited`, `NotCompleted` oder `Failed`.
- Falls der Koordinatorzustand `ABORTING` ist und die `cancelingList` und die `compensatingList` leer sind:
  - Wechsel des Koordinatorzustands nach `ENDED_FAILURE`
- Sonst:
  - Beibehaltung des Koordinatorzustands `ABORTING`.

Empfang einer `Completed`-Nachricht:

- Aktualisierung des Teilnehmerzustands auf `Completed` und
- Verschieben des Teilnehmers in die `completedList`.
- Aktualisierung des Teilnehmerzustands auf `Canceling`,
- Verschieben des Teilnehmers in die `cancelingList` und

- Senden einer `Cancel`-Nachricht an den Teilnehmer.
- Falls der Koordinatorzustand `ABORTING` ist und die `cancelingList` und die `compensatingList` leer sind:
  - Wechsel des Koordinatorzustands nach `ENDED_FAILURE`
- Sonst:
  - Beibehaltung des Koordinatorzustands `ABORTING`.

Empfang einer `Choose`-Nachricht (nur bei MixedOutcome):

- Ignorieren der Nachricht.

`ENDED_SUCCESS` und `ENDED_FAILURE`

Der Prozess wurde entsprechend des Koordinatorzustands erfolgreich oder durch einen Abbruch beendet. Abschließend ist eine Benachrichtigung des Initiators oder eines zuvor vereinbarten Teilnehmers denkbar, um das Ergebnis der Transaktion zu übermitteln.

4.1.5 Beispielszenario Flugbuchung

Das bereits in Abschnitt 3.1.1 vorgestellte Beispielszenario einer Flugbuchung wird nun erneut herangezogen, um die realisierten Konzepte zu veranschaulichen. Das Ziel besteht wieder darin, den Flug bei der preisgünstigsten Fluggesellschaft zu reservieren und alle anderen Angebote zu verwerfen.

Abbildung 4.3 zeigt ein Sequenzdiagramm der Buchung mit dem Initiator der Transaktion (Reisebüro), einem Koordinator sowie zwei Fluggesellschaften. Dargestellt ist eine Buchung mit dem WS-BA-Protokoll BusinessAgreementWithParticipantCompletion (BAPC). Neu eingeführte Parameter und Nachrichten, die für die autonome Koordinierung benötigt werden, sind fett hervorgehoben.

Nachdem der Koordinationskontext erstellt wurde, registriert sich der Initiator als erster Teilnehmer und übergibt als Parameter `subsequentCalls` die Anzahl der Dienste, die er im weiteren Verlauf aufrufen wird. In diesem Fall ist der entsprechende Wert 2. Mit der Antwort des Koordinators bekommt der Initiator seine Id zugewiesen (`yourId`), die er fortan in jeder Nachricht als `participantId` mitsenden muss.

Der Initiator sendet zur Buchung eines Fluges nachfolgend zwei Business-Nachrichten `BookFlightRequest` an Fluglinien, in denen er sich mit seiner `participantId` identifiziert. In diesen Nachrichten ist außerdem der Parameter `choiceSet` gesetzt, um den Diensten zu signalisieren, dass sie als Teil einer Choice-Menge aufgerufen werden.

Daraufhin registrieren sich die Dienste der Fluglinien unter Angabe der `callerId`, die hier der `participantId` des Initiators entspricht, und der Choice-Menge, zu der sie gehören (`choiceSet`). Der Parameter `subsequentCalls` enthält in diesem Fall den Wert 0 oder kann vollständig entfallen, da die Fluglinien nicht beabsichtigen, weitere

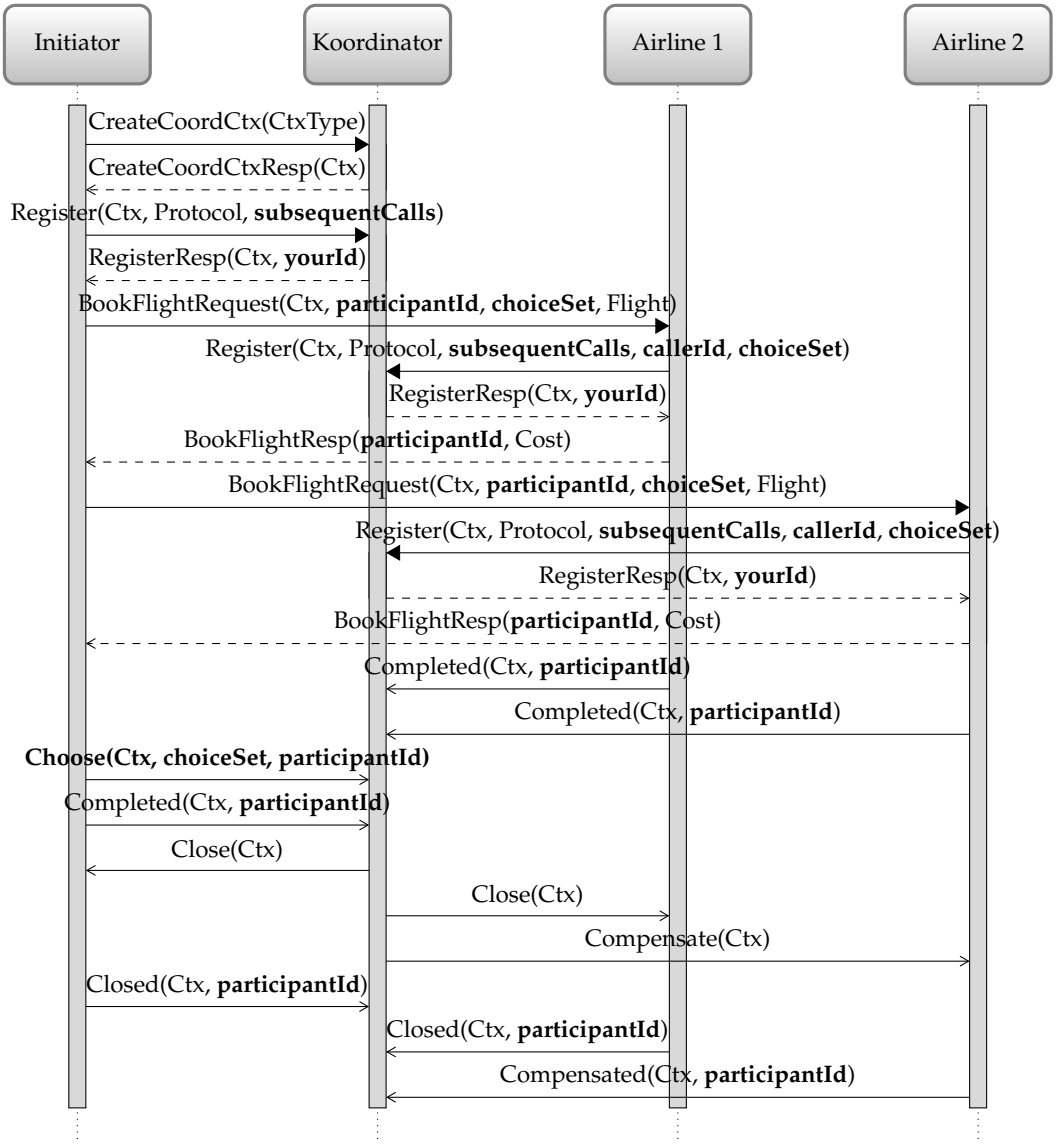

Abbildung 4.3: Sequenzdiagramm einer Flugbuchung mit BAPC.

Dienste in die Transaktion einzubinden. Nach der Registrierung senden sie eine Antwort bzgl. der Flugbuchung an den Aufrufer und teilen dem Koordinator mit der `Completed`-Nachricht mit, dass sie mit der Durchführung der Operation fertig sind.

Nach Empfang der Antwortnachrichten wählt der Initiator die abzuschließenden und zu verwerfenden Dienste aus und teilt dies dem Koordinator per `Choose`-Nachricht mit. Anschließend sendet auch der Initiator eine `Completed`-Nachricht und signalisiert damit, dass er seine Arbeit beendet hat. Gemäß der vom Initiator getroffenen Auswahl führt der Koordinator das Protokoll mit `Close`- oder `Compensate`-Nachrichten an die Teilnehmer fort und nach einer Bestätigung der Teilnehmer über den durchgeführten Abschluss ist das Protokoll beendet.

4.2 Umsetzung der erweiterten Anforderungen

Über die Basisfunktionalität der Transaktionskoordination hinausgehend, ist es wünschenswert zusätzliche Mechanismen zu integrieren, die zu einem robusteren Transaktionsverhalten führen oder eine erweiterte Funktionalität ermöglichen. Die in Abschnitt 3.2.3 vorgestellten Eigenschaften zur Steigerung der Fehlertoleranz und der Flexibilität von transaktionalen Workflows werden daher nachfolgend erneut aufgegriffen, konzeptionell konkretisiert und hinsichtlich der Umsetzung für einen dienstbasierten Koordinator untersucht.

4.2.1 Vitalität

Vitalität ist eine Eigenschaft aufgerufener Dienste, die die Notwendigkeit eines erfolgreichen Abschlusses des Dienstes für das gesamte Transaktionsergebnis ausdrückt. Die Erweiterung des Modells um vitale und nicht-vitale Teilnehmer erfordert zunächst eine entsprechende Annotation der Dienste. Diese Anforderung kann nur der Dienstnutzer leisten, da er als einziger beurteilen kann, welche Dienste er zur Erfüllung seiner Operation benötigt. Da weiterhin davon auszugehen ist, dass Dienste im Regelfall vital sind, sollte dies als Voreinstellung stets gelten und nicht ausdrücklich deklariert werden müssen. Nur wenn ein Dienst nicht-vital für einen Aufrufer ist, muss dies explizit bekanntgemacht werden.

Der Koordinator erlangt abermals über bereits existierende Nachrichten Kenntnis von der fehlenden Vitalität eines Dienstes, indem zunächst der Dienstnutzer den aufgerufenen Teilnehmer über einen Parameter `nonVital` darüber informiert, dass er nicht relevant für das Transaktionergebnis ist. Diese Einstufung leitet der Teilnehmer mit seiner Registrierungsnachricht schließlich an den Koordinator weiter, so dass dies in der Koordination adäquat berücksichtigt werden kann.

Das wesentliche Instrument für die Entscheidung des Koordinators sind die in Abschnitt 4.1.3 vorgestellten Inferenzregeln (vgl. Regeln 4.4–4.7), mit denen die Teilnehmer der Complete-Menge oder der Cancel-Menge zugeordnet werden. Damit die Transaktion erfolgreich verläuft, müssen die Teilnehmer entsprechend ihrer Mengenzugehörigkeit abschließen, so dass mit der Complete- und Cancel-Menge gewissermaßen der „Soll"-Zustand der Transaktion ausgedrückt wird. Die Regeln basieren auf der Annahme, dass aufgerufene Dienste stets vital für die Transaktion sind, so dass der Abbruch eines Teilnehmers das Scheitern von MixedOutcome-Transaktionen zur Folge hat, sofern sich der ausgeschiedene Teilnehmer in der Complete-Menge befindet. Es wird nun gezeigt, wie die Einführung von nicht-vitalen Teilnehmern die Fehlertoleranz des Prozesses steigert, indem weitere Regeln eingeführt werden, die das Ausscheiden von Teilnehmern sowie deren Vitalität in Beziehung setzt.

Die bekannten Regeln 4.4–4.7 bleiben dabei zunächst bestehen und die Einteilung der Teilnehmer in die Complete- und Cancel-Menge wird unverändert durchgeführt (spä-

ter müssen die Regeln allerdings leicht erweitert werden). In einem weiteren Schritt wird nun jedoch zusätzlich geprüft, ob der Ausstieg eines Teilnehmers zu kaskadierenden Abbrüchen führt und schließlich den Prozess scheitern lässt. Hierfür wird das Prädikat aborted(a, z) eingeführt, mit dem das Scheitern oder der Ausstieg eines Teilnehmers a im Kontext z mit einer `Fail`-, `Exit`- oder `CannotComplete`-Nachricht formuliert wird. Der aus dem Ausstieg des Teilnehmers resultierende Zustand der Transaktion wird schließlich in der Close- und der Compensate-Menge festgehalten. Diese Mengen spiegeln damit den „Ist"-Zustand der Transaktion wider und zeigen direkt an, welche Teilnehmer eine `Close`-Nachricht und welche Teilnehmer eine `Compensate`-Nachricht erhalten müssen. Als Hilfsprädikate werden Propagate2Top und Propagate2Leaf eingeführt, um Knoten zu markieren, die bereits abgearbeitet wurden und so eine zyklische Anwendung der Regeln zu vermeiden.

$$\begin{aligned}\forall a \forall b \forall z(&\text{invocation}(a, b, z) \wedge \text{completion}(b, z) \wedge \text{vital}(b, z) \wedge \neg(\text{aborted}(a, z)) \\ &\wedge(\text{aborted}(b, z) \vee \text{propagate2Top}(b, z))) \rightarrow \text{propagate2Top}(a, z)\end{aligned} \tag{4.8}$$

$$\begin{aligned}\forall a \forall b \forall z(&\text{invocation}(a, b, z) \wedge \text{completion}(a, z) \wedge \neg(\text{aborted}(b, z)) \\ &\wedge(\text{aborted}(a, z) \vee \text{propagate2Top}(a, z) \vee \text{propagate2Leaf}(a, z))) \\ &\rightarrow \text{propagate2Leaf}(b, z)\end{aligned} \tag{4.9}$$

$$\forall a \forall z(\text{propagate2Top}(a, z) \vee \text{propagate2Leaf}(a, z)) \rightarrow \text{compensate}(a, z) \tag{4.10}$$

$$\forall a \forall z(\text{completion}(a, z) \wedge \neg(\text{aborted}(a, z)) \wedge \neg(\text{compensate}(a, z))) \rightarrow \text{close}(a, z) \tag{4.11}$$

Regel 4.8 drückt die Propagation eines Dienstverlustes in Richtung der Wurzel des Aufrufbaumes aus, indem der Elternknoten a mit Propagate2Top markiert wird, sofern ein aufgerufener vitaler Teilnehmer b der Complete-Menge angehört und seinen Dienst nicht mehr erbringen kann (aborted(b, z)) bzw. durch die Regeln für die Kompensation vorgesehen ist (propagate2Top(b, z)). Ist Teilnehmer a bereits selbst aus dem Prozess ausgestiegen, braucht er nicht zu kompensieren und wird daher von der Zuordnung zur Compensate-Menge ausgeschlossen ($\neg$(aborted(a, z))). Mit Regel 4.9 wird ein Ausstieg oder die Markierung zur Kompensation eines Teilnehmerdienstes im Aufrufbaum in Richtung der Blätter weitergeleitet, sofern der Kindknoten nicht bereits selbst abgebrochen hat. Die Markierungen pflanzen sich auf diese Weise in einem Teilbaum fort bis sie einen Elternknoten erreichen, der nicht-vital ist. In diesem Fall greift Regel 4.8 nicht und die Kaskade der Markierungen ist beendet. Anderenfalls wird der gesamten Aufrufbaum markiert, so dass alle Teilnehmer kompensieren müssen und die Transaktion damit als gescheitert gilt. Die dritte Regel 4.10 übernimmt die Zuordnung markierter Teilnehmer zur Compensate-Menge und Regel 4.11 ordnet schließlich alle verbleibenden Teilnehmer der Complete-Menge, also diejenigen Dienste, die weder ausgestiegen sind noch durch die Auswirkung eines Ausstiegs kompensieren müssen, der Close-Menge zu.

Die Regeln unterscheiden zwischen Teilnehmern, die den Prozess von sich aus mit einer `Fail`-, `Exit`- oder `CannotComplete`-Nachricht verlassen (aborted(a, z)) und den-

jenigen Teilnehmern, die als Folge der Regeln markiert wurden (propagate2Top(a, z) bzw. propagate2Leaf(b, z)). Damit ist der Vorteil verbunden, dass allen Teilnehmern der Compensate- oder der Close-Menge unmittelbar `Compensate`- oder `Close`-Nachrichten geschickt werden können, da bereits ausgeschiedene Teilnehmer in diesen Mengen nicht enthalten sind (vgl. Regeln 4.8 und 4.9).

Implizit wurde auch bisher nach ähnlichen Regeln entschieden – allerdings weniger formal. Im Entwurf des Algorithmus (vgl. Abschnitt 4.1.4) wird unter „Empfang einer `Exit`-, `CannotComplete`- oder `Fail`-Nachricht“ im Zustand `PREPARING` bereits beschrieben (vgl. Seite 68), wie ausgeschiedene Teilnehmer der Complete-Menge in die `abortedList` verschoben werden, um nachträglich einen Prozessabbruch einleiten zu können. Da im Algorithmus noch die Vitalität aller Teilnehmer vorausgesetzt wird, gilt der Prozess dort bereits als gescheitert, sobald mindestens ein Teilnehmer der Complete-Menge in der `abortedList` enthalten ist. Der Unterschied besteht nun darin, dass hier auch nicht-vitale Teilnehmer zugelassen sind, so dass die Überprüfung auf einen Abbruch des Prozesses komplexer wird, da nicht in jedem Fall eines Ausstiegs der gesamte Prozess scheitern muss. Daher wurde für die Feststellung eines Prozessabbruchs ein Regelsatz nach dem Vorbild der Entscheidungsregeln für die Complete- oder Cancel-Menge entwickelt.

Choice-Mengen und Vitalität

Die Einführung der Vitalität als neue Teilnehmereigenschaft führt zu der Frage, wie sich die Auswahl per `Choose`-Nachricht auf die Vitalität auswirkt. Für die alten Regeln gilt stets die Vitalität aller Teilnehmer, so dass durch die `Choose`-Nachricht ausgewählte vitale Teilnehmer einer Choice-Menge der Complete-Menge zugeordnet werden (vgl. Regel 4.5). Nicht ausgewählte vitale Teilnehmer werden entsprechend in die Cancel-Menge verschoben, d.h. die Auswahl der Teilnehmer ist in diesem Fall vorrangig und die Teilnehmer werden zu nicht-vitalen Diensten, die nicht abschließen sollen (vgl. Regel 4.7).

Unter Einbeziehung von nicht-vitalen Diensten müssen die Regeln neu betrachtet werden, da jetzt die Möglichkeit besteht, nicht-vitale Teilnehmer einer Choice-Menge für den Abschluss auszuwählen. a sei im Folgenden ein solcher Teilnehmer. Falls nun ein weiterer Teilnehmer b, der sich im Aufrufbaum unterhalb von a befindet, aus dem Prozess aussteigt, pflanzen sich nach den neuen Regeln 4.8 und 4.9 die Kompensationen im betroffenen Teilbaum fort, bis der nicht-vitale Zwischenknoten a diesen Vorgang in Richtung der Wurzel stoppt. Da a jedoch zur Auswahl der Choice-Menge gehört, muss er erfolgreich abschließen, was nun durch die zur Kompensation vorgesehenen Kindnoten nicht mehr möglich ist. Aus diesem Grund müssen die alten Regeln derart erweitert werden, dass sich die Aus- oder Abwahl von Teilnehmern einer Choice-Menge auch in deren Vitalitätseigenschaft niederschlägt. Die folgenden Regeln orientieren sich daher an den Regeln 4.5 und 4.7 und führen die explizite Zuweisung oder Aberkennung der Vitalität

ein:

$$\forall a \forall b \forall z(\text{invocation}(a,b,z) \wedge \text{completion}(a,z) \wedge \text{partOfChoice}(b,z) \wedge \text{choose}(b,z)) \rightarrow \text{vital}(b,z) \tag{4.12}$$

$$\forall a \forall b \forall z(\text{invocation}(a,b,z) \wedge \text{partOfChoice}(b,z) \wedge \neg(\text{choose}(b,z))) \rightarrow \neg(\text{vital}(b,z)) \tag{4.13}$$

Ein ausgewählter Teilnehmer einer Choice-Menge wird der Complete-Menge zugeordnet (vgl. Regel 4.5) und damit vital für den Prozess (vgl. Regel 4.12). Umgekehrt wird er der Cancel-Menge zugerechnet (vgl. Regel 4.7) und auf nicht-vital gesetzt (vgl. Regel 4.13).

Durch die Einführung von vitalen bzw. nicht-vitalen Diensten konnte der wesentliche Vorteil erreicht werden, dass nicht-vitale Dienste fehlschlagen dürfen, auch wenn der Aufrufer der Complete-Menge angehört. Darüber hinaus ist sogar ein Fehlschlag oder der Ausstieg von vitalen Diensten ohne Gefährdung des Prozesses möglich, wenn sich die Dienste in einem Teilbaum befinden, dessen Wurzel nicht-vital ist.

4.2.2 Diskussion von Abhängigkeiten

Die in Abschnitt 3.2.3 eingeführten Abhängigkeiten binden Dienstaufrufe an das Ergebnis einer zuvor ausgeführten Teiltransaktion. Hierbei unterscheidet sich die positive Abhängigkeit (Präzedenz), bei der eine Folgetransaktion nur dann ausgeführt wird, wenn die Vorgängertransaktion erfolgreich war, von der negativen Abhängigkeit (Kontingenz), die eine Folgetransaktion erst dann zur Ausführung bringt, wenn der Vorgänger gescheitert ist.

Es wurde bereits dargelegt, dass der in dieser Arbeit skizzierte Koordinator einen solchen Ablauf nur unterstützend begleiten kann, da er die erforderlichen Anwendungsdaten nicht kennt, um eigenständig Teilnehmer aufzurufen. Die Dienstaufrufe sequentiell auszuführender oder alternativer Teilnehmer müssen daher weiterhin vom Dienstnutzer ausgeführt werden. Die Aufgabe des Koordinators ist darin zu sehen, Präzedenzen oder Alternativdienste in der Koordinierung zu berücksichtigen, falls Teilnehmer diese deklariert haben.

Für die Annotation kann erneut der bewährte Transportmechanismus verwendet werden, so dass die Information von positiven und negativen Abhängigkeiten den Koordinator über die Zwischenstation der aufgerufenen Teilnehmern erreicht. Nachfolgend soll festgestellt werden, welche Informationen der Koordinator benötigt.

Präzedenz

Da die Präzedenz eine positive Abhängigkeit darstellt, wird der Aufruf einer Reihe von Teilnehmern, die in einer Präzedenzrelation stehen, nur dann vollständig durchgeführt, wenn kein Dienst dieser Folge scheitert oder den Prozess verlässt. Die Folge bricht also beim ersten Ausfall eines Teilnehmers ab. Daraus resultiert eine Differenz der mit

`subsequentCalls` bekanntgegebenen Anzahl erwarteter Dienstaufrufe und der Anzahl tatsächlich durchgeführter Aufrufe.

Sofern der ausgefallene Teilnehmer vital für den Prozess ist, kann die Transaktion unmittelbar abgebrochen werden – auch wenn der Prozess noch nicht seine maximale Ausdehnung erreicht hat. Handelt es sich dagegen um einen nicht-vitalen Teilnehmer, besteht für den Koordinator keine Veranlassung, den Prozess als gescheitert zu betrachten, da der Teilnehmer für einen erfolgreichen Abschluss nicht zwingend erforderlich ist. Es ergibt sich das Problem, dass der Koordinator die Registrierung weiterer Teilnehmer erwartet, die in der Präzedenzfolge nach dem ausgefallenen Dienst hätten aufgerufen werden sollen, der Dienstnutzer diese Dienste jedoch nicht mehr aufrufen wird, da die Folge durch den ausgefallenen Teilnehmer unterbrochen ist. Der Prozess kommt damit zum Stillstand.

Zur Vermeidung einer derartigen Situation benötigt der Koordinator neben der Angabe von `subsequentCalls`, also der Anzahl *aller* Teilnehmer, die ein Dienstnutzer aufrufen wird, zusätzlich für jede Präzedenzfolge die Anzahl derjenigen Teilnehmer, die zu der jeweiligen Folge gehören. Dies muss der Dienstnutzer dem Koordinator bei seiner Registrierung bekanntgeben. Für den Fall eines Ausstiegs oder Abbruchs von Teilnehmern einer Präzedenzfolge, kann der Koordinator dann ableiten, dass die restlichen Dienste der Folge nicht mehr aufgerufen werden und muss die Anzahl der erwarteten Teilnehmer P_{exp} entsprechend reduzieren. Hierfür ist es wiederum erforderlich, dass der Koordinator die Präzedenzfolgen eines Dienstnutzers unterscheiden und Teilnehmer diesen Folgen zuordnen kann. Daher muss der Dienstnutzer jedem dieser Teilnehmer anzeigen, dass er Element einer Folge von Dienstaufrufen ist. Zur Unterscheidung von Teilnehmerfolgen sollte hierfür wieder eine Id verwendet werden, die der Teilnehmer bei seiner Registrierung schließlich an den Koordinator weiterleitet.

Die Einhaltung einer bestimmten Reihenfolge von Dienstaufrufen liegt bei dieser Konzeption in der alleinigen Verantwortung des aufrufenden Teilnehmers. Der Koordinator benötigt daher keine Information über die Reihenfolge der Dienstaufrufe, sondern muss nur entscheiden können, wann mit keinen weiteren Registrierungen mehr zu rechnen ist. Dies wird durch die zweifelsfreie Zuordnung der Teilnehmer zu Präzedenzfolgen sowie der Anzahl der Teilnehmer einer Präzedenzfolge sichergestellt.

Neben diesen grundlegenden Überlegungen ist für eine konkrete Umsetzung die Untersuchung des Zusammenwirkens von Choice-Mengen und Präzedenzfolgen erfordlich sowie eine Festlegung der Kommunikation zwischen Dienstnutzer und Dienstanbier hinsichtlich der für Präzedenzfolgen relevanten Protokollzustände. So reicht es beispielsweise nicht aus, dass nur der Koordinator über einen Fehlschlag oder den Austritt eines Teilnehmers einer Präzedenzfolge informiert wird, sondern insbesondere braucht der Dienstnutzer diese Information, da die positive Abhängigkeit in diesem Fall den Abbruch einer Folge vorsieht.

Kontingenz

Im Gegensatz zur positiven Abhängigkeit bringt die negative Abhängigkeit Dienste zur Ausführung, wenn die Vorgängertransaktion gescheitert ist, so dass mit dieser Beziehung Alternativdienste eingebunden werden können. Da ausgefallene Dienste durch ihre Alternativen ersetzt werden, ändert dabei sich die Anzahl aktiver Teilnehmer im Prozess nicht. Daher ist es ausreichend, Dienste mit ihren Alternativen im Parameter `subsequentCalls` stets nur mit dem Wert 1 zu berücksichtigen – gleichgültig wieviele Alternativen es für einen Dienst gibt. In Überprüfungen, die auf die Liste ausgeschiedener Teilnehmer `abortedList` zugreifen, müssen substituierte Dienste entsprechend herausgefiltert werden.

Da der Koordinator selbst keine Dienste aufrufen kann, benötigt er keine Information über die zur Verfügung stehenden Alternativen für einen Teilnehmer – er muss jedoch wissen, *dass* es für einen Dienst einen Ersatz gibt, damit er im Fehlerfall den Prozess im Zustand `PREPARING` halten kann und nicht unnötigerweise den Prozess abbricht. Bei der Registrierung von Teilnehmern muss dies dem Koordinator über einen booleschen Parameter `hasAlternatives` mitgeteilt werden. Zusätzlich muss der Koordinator sich registrierende Alternativdienste erkennen können und natürlich wissen, für welchen Teilnehmer sie einen Ersatz darstellen. Beides lässt sich mit einem Parameter `isAlternativeFor=`⟨`participantId`⟩ realisieren.

Ein Problem bei diesem Ablauf betrifft die Übermittlung von Teilnehmer-Ids an den Dienstnutzer. Bisher wurde die Teilnehmer-Id von aufgerufenen Diensten nur bei der Auswahl von Choice-Mengen benötigt (vgl. Abschnitt 4.1.3), wobei die Id mit der Antwortnachricht des aufgerufenen Dienstes mitgesendet wurde. Doch bei Ausfall eines Teilnehmers wird diese Nachricht möglicherweise gar nicht mehr übertragen. Es wird daher eine zusätzliche Nachricht zwischen den Teilnehmern benötigt, die einerseits den Dienstnutzer darüber informiert, dass ein aufgerufener Dienst nicht mehr am Prozess teilnimmt und andererseits die Id des ausgeschiedenen Teilnehmers übermittelt.

Auch bei der Umsetzung von Alternativdiensten hat sich gezeigt, dass die erforderlichen Erweiterungen nicht allein die Kommunikation zwischen Koordinator und Teilnehmern berühren, sondern auch zusätzliche Nachrichten zwischen den Teilnehmern untereinander verlangen. Klärungsbedarf besteht außerdem bei der Kombination von Alternativdiensten und Präzedenzfolgen sowie bei der zusätzlichen Integration von Vitalitätseigenschaften und Choice-Mengen.

4.2.3 Temporale Aspekte

Der bisher konzipierte Koordinator arbeitet nachrichtenbasiert, d.h. eintreffende Nachrichten stellen Ereignisse dar, auf die der Koordinator nachrichtenspezifisch reagiert. Nach Ausführung seiner administrativen Aufgaben und möglicherweise dem Versenden von Protokollnachrichten an die Teilnehmer stellt er seine Aktivität ein und wartet auf das nächste durch eine Teilnehmernachricht ausgelöste Ereignis.

Zur Einhaltung von temporalen Beschränkungen werden zeitorientierte Ereignisse benötigt, die das Erreichen eines Zeitpunktes signalisieren. Diese Zeitpunkte können eine untere Schranke markieren, mit der ausgedrückt wird, dass eine Operation erst *ab* einem bestimmten Datum ausgeführt werden soll oder sie geben eine obere Schranke vor, mit der ein Verfallsdatum modelliert wird, so dass eine Transaktion *bis* zu dem angegeben Datum abgeschlossen sein muss. Zusätzlich lassen sich zeitliche Beschränkungen nach ihrem Geltungsbereich klassifizieren:

Global: Eine globale Schranke hat Auswirkungen auf den gesamten Prozess. Wird die durch die Schranke definierte Auflage verletzt, muss die Transaktion abgebrochen werden.

In WS-Coordination wird hierfür das optionale Kontextelement `Expires` spezifiziert, mit dem ein Verfallsdatum für den Koordinationskontext angegeben werden kann. Der Wert wird bei Erstellung des Kontextes festgelegt und falls er überschritten wird, darf der Koordinator den Prozess mit `Cancel` oder `Compensate` beenden, sofern er noch keinem Teilnehmer eine `Close`-Nachricht geschickt hat. Umgekehrt darf ein Teilnehmer den Prozess mit `Exit` verlassen, solange er noch kein `Completed` gesendet hat.

Lokal: Eine lokale Schranke hat nur Einfluss auf den Teil des Aufrufbaumes, in dem sie definiert wurde.

WS-Coordination gestattet die Modellierung von lokalen oberen Schranken mit dem `Expires`-Element, wenn der Teilbaum durch die Interposition eines zweiten Koordinators oder die Schachtelung eines weiteren Gültigkeitsbereichs (nested scope) in die globale Transaktion eingebunden wird.

Die Festlegung des `Expires`-Wertes erfolgt in Millisekunden. Fehlt diese Angabe, ist der Kontext in seinem Geltungsbereich unbegrenzt gültig. Die Einhaltung von unteren Schranken liegt in der alleinigen Verantwortung der Dienstnutzer, die ihre Dienstaufrufe so steuern müssen, dass sie nicht zu früh ausgeführt werden.

Für die Umsetzung von zeitbasierten Ereignissen wird eine Komponente benötigt, die den Ablauf von zeitlichen Schranken signalisiert, so dass der Koordinator den Zustand des Prozesses auf die Einhaltung dieser Bedingungen überprüfen und ggf. den Prozessabbruch einleiten kann.

4.2.4 Implementationsmuster

Wie bereits in Abschnitt 3.2.3 geschildert, kann die Unterstützung von weiteren Implementationsmustern neben Do-Compensate die Tragweite von Auswirkungen gescheiterter Transaktionen auf Teilnehmerseite verringern. Nach Furniss und Green [FG05] ist hierfür im Wesentlichen die Einführung einer zusätzlichen `Fail`-Kante zwischen den Zuständen `Closing` und `Failing` in den Protokollgraphen von WS-BusinessActivity

erforderlich (vgl. Abb. 3.3). Unter der Voraussetzung einer ausreichenden Annotation der Dienste, konnte als Ergebnis in Abschnitt 3.2.3 dargelegt werden, dass die Muster Provisional-Final und Validate-Do ausschließlich in nicht-vitalen Diensten oder bei Teilnehmern mit Alternativdiensten implementiert werden können, da anderenfalls die transaktionalen Zusicherungen nicht eingehalten werden. Weitere Auswirkungen der unterschiedlichen Implementationsmuster für die Koordinierung werden nachfolgend untersucht.

In der ersten Koordinationsphase vor dem `Completed`-Zustand können die annotierten Muster zusätzlich als Optimierungskriterium für den Protokollablauf herangezogen werden. Wenn nämlich Dienste, die sich verhältnismäßig leicht zurücksetzen lassen als erste in den `Completed`-Zustand dirigiert werden, ist der Aufwand für die Behandlung eines Fehlers am geringsten, da Do-Compensate-Dienste noch mit `Cancel` abgebrochen werden können und nicht kompensieren müssen. In der ersten Phase bzw. im `PREPARING`-Zustand des Koordinators führt dies zu folgender Reihenfolge bei der Koordinierung der Implementationsmuster:

$$\text{Provisional-Final} \quad \rightarrow \quad \text{Validate-Do} \quad \rightarrow \quad \text{Do-Compensate} \tag{4.14}$$

Diese Reihenfolge kann der Koordinator natürlich nur im Protokoll BACC bestimmen, da die Teilnehmer bei BAPC nach eigenem Ermessen in den `Completed`-Zustand eintreten.

Weitere Eigenschaften der Teilnehmerdienste und Ausführungsfolgen

Im Abschnitt über Flex-Transaktionen (vgl. 2.3.6) wurde im Rahmen der sog. gemischten Transaktionen eine Einteilung der Sub-TA in „kompensierbar“, „wiederholbar“ und „pivot“ vorgenommen. Diese Eigenschaften wurden als Grundlage für die Erstellung einer Ausführungsreihenfolge herangezogen, die hier noch einmal wiedergegeben wird:

$$\text{kompensierbar} \quad \rightarrow \quad \text{pivot} \quad \rightarrow \quad \text{wiederholbar} \tag{4.15}$$

Die Ausführung dieser Anordnung ist genau dann erfolgreich, wenn die Pivot-TA erfolgreich abgeschlossen werden kann. Falls es mehrere Pivot-TA in einer Ausführungsfolge gibt, muss über alternative Ausführungspfade sichergestellt werden, dass die Transaktion auch im Fehlerfall garantiert zu einem erfolgreichen Abschluss gelangt. Eine Flex-TA ist *wohlgeformt*, wenn alle Pivot-TA von mindestens einem (Alternativ-)Pfad gefolgt werden, der nur aus wiederholbaren Sub-TA besteht [HTKR05]. Sofern Alternativen für abgebrochene Sub-TA existieren, dürfen kompensierbare und wiederholbare Sub-TA auch verschachtelt werden – der Pfad zwischen zwei Pivot-TA muss jedoch ausschließlich kompensierbare Sub-TA enthalten [AAA+96]. Abbildung 4.4 zeigt ein Beispiel für eine wohlgeformte Flex-TA aus kompensierbaren TA k_i, Pivot-TA p_i und wiederholbaren TA w_i.

Da die diskutierten Implementationsmuster allesamt zurücksetzbar bzw. kompensier-

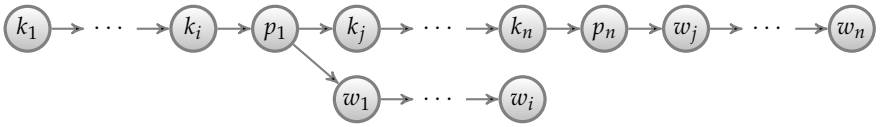

Abbildung 4.4: Wohlgeformte Flex-Transaktion (aus [HTKR05]).

bar sind, können sie in der Sub-TA-Ordnung von Flex-Transaktionen anstelle der kompensierbaren Sub-TA eingesetzt werden. Wird zusätzlich die optimierte Ausführungsreihenfolge innerhalb der Implementierungsmuster angewendet, ergibt sich insgesamt die Ausführungsfolge:

$$\begin{aligned} \text{Provisional-Final} \quad &\rightarrow \quad \text{Validate-Do} \quad \rightarrow \quad \text{Do-Compensate} \\ &\rightarrow \quad \text{pivot} \quad \rightarrow \quad \text{wiederholbar} \end{aligned} \tag{4.16}$$

Ein erfolgreicher Protokollablauf lässt sich in zwei Phasen bzw. die Koordinatorzustände `PREPARING` und `FINISHING` unterteilen. Wie oben beschrieben, werden in Phase I alle kompensierbaren TA nach ihrem Implementationsmuster unterschieden, um den Aufwand bei Abbrüchen in dieser Phase zu minimieren. Nach den kompensierbaren TA werden Pivot-TA abgearbeitet, die nicht kompensierbar und nicht wiederholbar sind. Sie müssen für den Fehlerfall wiederholbare Alternativpfade definieren, die garantiert zu einem Ende führen. Die als letztes abzuwickelnden wiederholbaren TA schließen die Phase I ab. Am Beginn der Phase II befinden sich alle Teilnehmer im Zustand `Completed`. Der Koordinator trifft auf Basis der aufgestellten Complete- und Cancel-Mengen und des Koordinationstyps nun die Entscheidung, welche Teilnehmer abschließen oder kompensieren müssen. Beim Abschluss mit `Close` ist wegen der neu eingeführten `Fail`-Kante wieder das Implementationsmuster zu berücksichtigen. Im Fehlerfall müssen ggf. Alternativen herangezogen werden, sofern die Vitalität des Teilnehmers dies erfordert. Die Reihenfolge ist in der zweiten Phase nicht mehr entscheidend.

4.3 Zusammenfassung

In diesem Kapitel konnte das Konzept eines autonomen Koordinators entwickelt werden, der durch die Verwaltung und Auswertung unterschiedlicher Daten selbstständig über den Prozessabschluss entscheiden kann. Im Einzelnen werden dafür folgende Informationen benötigt:

- Ids, Protokollzustände und Eigenschaften der Teilnehmer,
- die Anzahl der registrierten Teilnehmer P_{reg} und die Anzahl der erwarteten Teilnehmer P_{exp},
- strukturelle Abhängigkeiten zwischen den Dienstaufrufen sowie

- für den Abschluss ausgewählte Teilnehmer aus Choice-Mengen.

Hieraus bildet der nicht-atomare Koordinator mit einem Satz von Inferenzregeln die Complete- und Cancel-Mengen, die den anzustrebenden Transaktionsabschluss der Teilnehmer beschreiben und in den aggregierten Prozesszustand des Koordinators einfließen. Unter zusätzlicher Berücksichtigung der Vitalität von Teilnehmern werden weitere Regeln angewandt, die das Ausscheiden nicht-vitaler Teilnehmer ermöglichen.

Der erfolgreiche Transaktionsabschluss wird schließlich im Zustand `FINISHING` abgewickelt, indem die Teilnehmer entsprechend ihrer Mengenzugehörigkeiten in den Endzustand geführt werden. Der Zeitpunkt für den Prozessabschluss bestimmt sich für die beiden Koordinationstypen wie folgt:

- AtomicOutcome: Alle registrierten Teilnehmer befinden sich im Protokollzustand `Completed` und es stehen keine Registrierungen mehr aus.
- MixedOutcome: Alle registrierten Teilnehmer der Complete-Menge befinden sich im Protokollzustand `Completed`, alle Choice-Mengen wurden entschieden und es stehen keine Registrierungen mehr aus.

Im Rahmen von Transaktionen des MixedOutcome-Typs wurde auch die Semantik der `Fail`-Nachrichten diskutiert. Im Bestreben, ein möglichst robustes Transaktionsverhalten zu realisieren, interpretiert der in dieser Arbeit spezifizierte Koordinator `Fail`-Nachrichten wie `Exit` und `CannotComplete`. Eine nicht weiter ausgeführte Nachbehandlung des Fehlers wird hierbei vorausgesetzt.

Neben der Vitalität von Teilnehmern, die erfolgreich in das Koordinatorkonzept integriert werden konnte, wurden weitere Aspekte der Transaktionsteuerung betrachtet, die zur Erhöhung der Fehlertoleranz und der Flexibilität beitragen. Hierbei zeigte sich, dass der Fokus auf den Koordinator für die Berücksichtigung von Teilnehmerabhängigkeiten und unterschiedlichen Implementierungsmustern nicht mehr ausreicht. Es werden weiterführende Untersuchungen für den erfordlichen Nachrichtenaustausch zwischen den Teilnehmern benötigt.

Die Integration zeitlicher Vorgaben für den Beginn oder Abschluss von Transaktionen ist dagegen für die Koordination technisch unproblematisch, sofern ein Zeitgeber das Verstreichen von Terminen durch entsprechende Ereignisse signalisiert.

5 Implementierung des Koordinators

Im Verlauf dieser Arbeit konnten Koordinatoren für die Abwicklung langlebiger Transaktionen nach WS-BusinessActivity realisiert werden, mit denen die eingeführten Konzepte näher untersucht wurden und auf ihre Realisierbarkeit hin überprüft wurden. Nachfolgend werden die in Java entwickelten Business-Koordinatoren für die Koordinationstypen AtomicOutcome und MixedOutcome sowie die Erweiterungen für WS-Coordination im Detail vorgestellt. Dabei werden besondere Problemstellungen hervorgehoben.

5.1 Architektur

Als Grundlage für die Implementierung wurde auf eine vorhandene Implementation des TracG-Projektes zurückgegriffen, die einen Koordinator nach WS-AtomicTransaction sowie Dienste zur Erstellung und Verwaltung von Koordinationskontexten nach WS-Coordination umfasst. Eine Architekturübersicht, die auch die Integration der Implementation in das Web-Services-Schichtenmodell veranschaulicht, ist in Abbildung 5.1 dargestellt. Für die Unterstützung von Business-Transaktionen angepasste oder vollkommen neu entwickelte Komponenten sind in der Grafik mit einem schwarzen Rand gekennzeichnet.

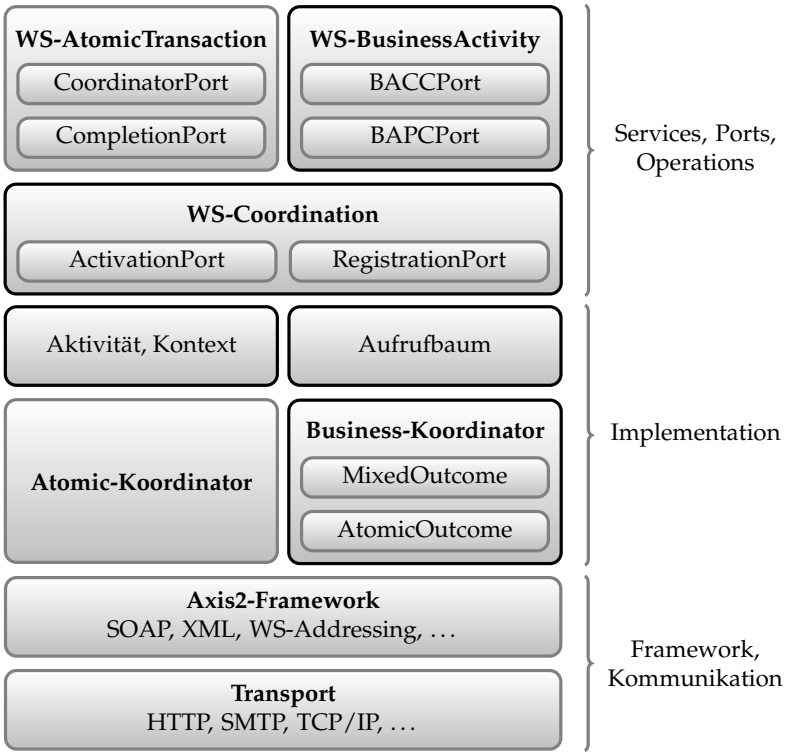

Abbildung 5.1: Die Architektur des Koordinators und seine Integration in den Web-Services-Stack.

Die Architektur nutzt das Axis2-Framework[1] der Apache Foundation, das sowohl eine Laufzeitumgebung für Web-Services als auch Bibliotheken für die Entwicklung von Web-Service-Clients bereitstellt (vgl. [FTW07]). Axis2 unterstützt unterschiedliche Transportprotokolle, wie HTTP oder SMTP, sowie beliebige Kommunikationsmuster für die Interaktion von Diensten.

Bei Erstellung eines neuen Koordinationskontextes über den in WS-Coordination spezifizierten Activation-Port wird ein Atomic-Koordinator oder in Abhängigkeit vom angegebenen Koordinationstyp ein Business-Koordinator für AtomicOutcome oder MixedOutcome erzeugt, der im weiteren Verlauf die Protokollnachrichten verarbeitet. Hierfür stellen die Koordinatoren spezifikationsgemäß Ports für die unterstützten Protokolle bereit. Die Verwaltung des Koordinatorzustands und der registrierten Teilnehmer wird über den Kontext der zugehörigen Aktivität garantiert. Strukturelle Information über den Prozess und hierarchische Abhängigkeiten der Teilnehmer speichert der Business-Koordinator zusätzlich in einem Aufrufbaum.

5.2 Implementierung des Koordinators

Alle Koordinatoren werden von der Basisklasse `CoordinatorImpl` abgeleitet, die wiederum das Interface `Coordinator` implementiert. Die Basisklasse stellt eine Methode zur Erzeugung des Koordinationskontextes bereit, die außerdem eine Aktivität mit den Parametern des Kontextes initialisiert. Mit Hilfe eines einfachen Speichers lassen sich diese Aktivitätsobjekte sichern und zu einem späteren Zeitpunkt über die Kontext-Id wiederherstellen. Da eine Aktivität die interne Repräsentation einer Transaktion ist, wird auf diese Weise sichergestellt, dass die Protokollnachrichten die richtige Instanz erreichen und der Zustand der Transaktion erhalten bleibt. Nach Beendigung einer Aktivität wird der Speicher wieder freigegeben.

Die von der Basisklasse abgeleiteten Koordinatoren implementieren das eigentliche Protokollverhalten. Insgesamt umfasst die Implementation drei Koordinatoren für die Protokolle aus WS-AT und WS-BA: Completion und Durable 2PC bzw. Volatile 2PC aus WS-AT werden durch die Klasse `AtomicCoordinatorImpl` implementiert. Die Klassen `AtomicBusinessCoordinatorImpl` sowie `MixedBusinessCoordinatorImpl` implementieren die Koordinationstypen AtomicOutcome und MixedOutcome und realisieren jeweils die WS-BA-Protokolle BAPC und BACC für den entsprechenden Koordinationstyp. Abbildung 5.2 veranschaulicht diese Ableitungshierarchie.

Aus dem Diagramm ist ersichtlich, dass es für jede Nachricht der Protokolle genau eine korrespondierende Methode hat, die bei Eintreffen einer entsprechenden Nachricht aufgerufen wird. Wie in der Konzeptbeschreibung dargelegt ist, führen die Koordinatoren außerdem Zustandslisten von Teilnehmern, um einfach entscheiden zu können, ob sich Teilnehmer in einem Zustand befinden und ggf. welche Teilnehmer dies sind. Ab-

[1] `http://ws.apache.org/axis2/`

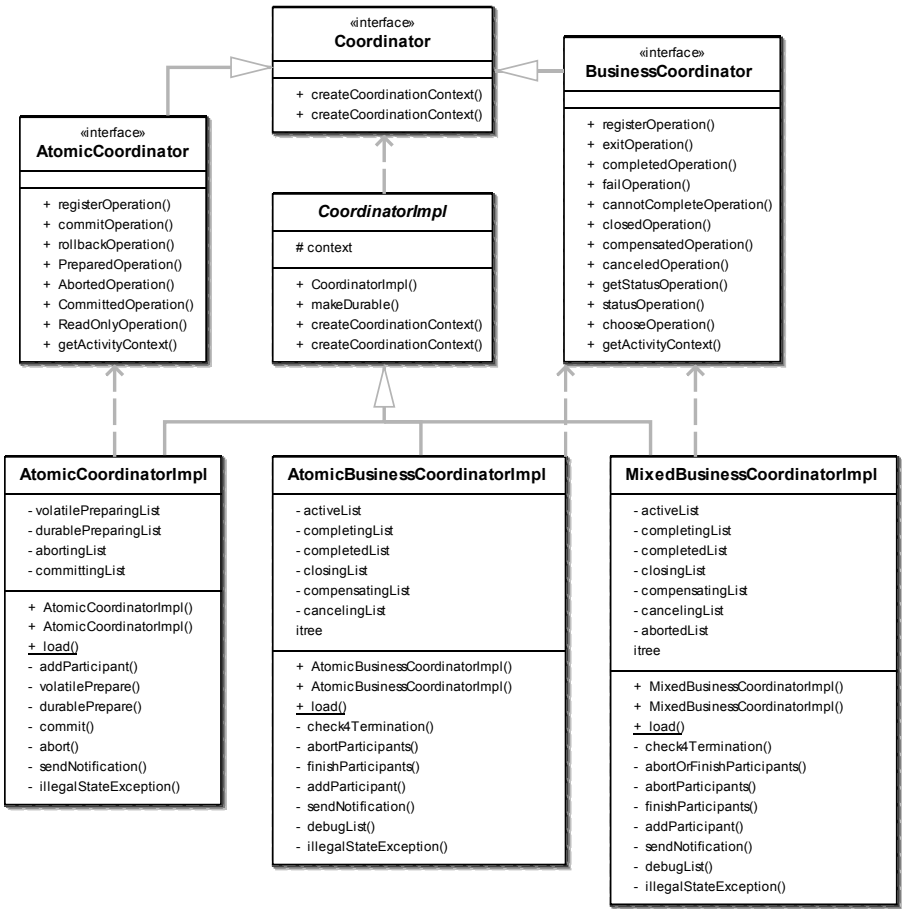

Abbildung 5.2: Ableitungshierarchie der Koordinatorklassen.

hängigkeiten, die durch Interaktionen von Teilnehmern entstehen, sowie die Anzahl der erwarteten und registrierten Teilnehmer verwalten die Business-Koordinatoren in einem Aufrufbaum.

Am Beispiel des Koordinators `AtomicBusinessCoordinatorImpl` wird nachfolgend veranschaulicht, wie diese Informationen konkret genutzt werden. Listing 5.1 zeigt einen Auszug des Quellcodes aus der Methode `finishParticipants` und demonstriert den Zustandswechsel des Koordinators nach `FINISHING`. Die Methode wird immer aufgerufen, wenn für den Koordinator die Möglichkeit zu einem Zustandswechsel besteht. Im gewählten Beispiel ist dies genau dann der Fall, wenn sich der Koordinator in `PREPARING` befindet und von Teilnehmern, die sich im Protokollzustand `Active` oder `Completing` (BACC) befinden, eine `Completed`-Nachricht erhält – der Aufruf kommt also aus der Methode `completedOperation` (siehe auch Abbildung 5.2).

Listing 5.1: Wechsel des Koordinatorzustands nach `FINISHING` für AtomicOutcome.

```
1 private void finishParticipants(String protocolType) throws AxisFault {
2   ...
3   Iterator<ParticipantInfo> iter = context.
4       getRegisteredParticipants(protocolType);
```

```
5   if (context.getStatus() == CoordinatorStatus.STATUS_BA_PREPARING) {
6     // Alle Teilnehmer in Completed und keine weiteren Registrierungen?
7     if (activeList.isEmpty() && completingList.isEmpty()
8         && !itree.hasPendingRegistrations()) {
9       // Setzen des Koordinatorzustands
10      context.setStatus(CoordinatorStatus.STATUS_BA_FINISHING);
11      while (iter.hasNext()) {
12        ParticipantInfo pInfo = iter.next();
13        if (pInfo.getStatus()==ParticipantStatus.STATUS_BA_COMPLETED) {
14          // Setzen des Teilnehmerzustands und Senden der Nachricht
15          pInfo.setStatus(ParticipantStatus.STATUS_BA_CLOSING);
16          sendNotification("close", pInfo.getEpr().getAddress()
17              .toString(), pInfo.getId(), protocolType);
18          // Teilnehmer in die closingList verschieben
19          completedList.remove(pInfo.getId());
20          closingList.add(pInfo.getId());
21        } else {
22          throw illegalStateException();
23        }
24      }
25    }
26  } else {
27    throw illegalStateException();
28  }
29  ...
30 }
```

Die Überprüfung der Bedingung für den Eintritt in den Zustand `FINISHING` erfolgt in den Zeilen 7 und 8. Entsprechend des aufgestellten Konzepts dürfen sich keine Teilnehmer mehr in `Active` oder `Completing` (BACC) befinden und es dürfen keine Registrierungen mehr ausstehen. Die Ausdehnung des Prozesses wird im Aufrufbaum verwaltet (vgl. Abschnitt 5.3), der für die Überprüfung nach Vollständigkeit die Methode `hasPendingRegistrations` anbietet.

Ist die Bedingung erfüllt, wechselt der Koordinator in den Zustand `FINISHING`, indem die Aktivität (`context`) aktualisiert wird. Bei der anschließenden Iteration durch die registrierten Teilnehmer wird ihr in der Aktivitität gespeicherter Zustand auf `Closing` gesetzt und jedem Teilnehmer eine `Close`-Nachricht gesendet. Außerdem werden die Teilnehmer von der `completedList` in die `closingList` verschoben.

Die über den Kontext referenzierbare Aktivität wird entsprechend der zu koordinierenden Transaktion erzeugt. Sie unterteilt sich strukturell in eine `AtomicActivity` und eine `BusinessActivity`, in der sowohl der Zustand des Koordinators als auch die Zustände der Teilnehmer gespeichert werden (vgl. Abb. 5.3). Die `BusinessActivity` ist zusätzlich für die Koordinationstypen `AtomicOutcome` und `MixedOutcome` parametrisierbar.

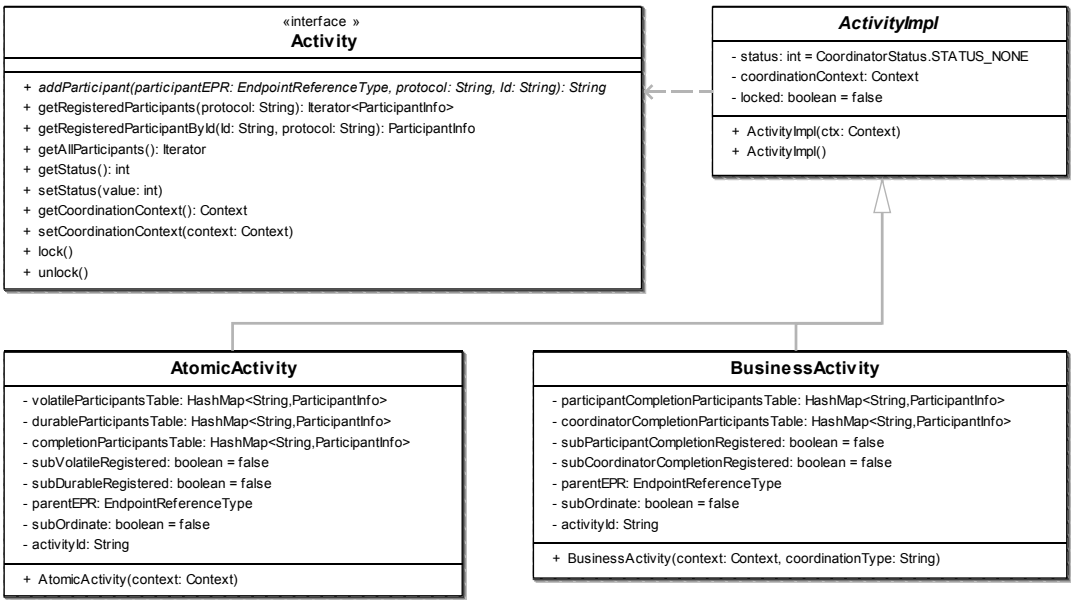

Abbildung 5.3: Ableitungshierarchie einer Aktivität.

Trifft eine neue Nachricht bei einem Protokolldienst ein, wird diese nicht unmittelbar vom Koordinator verarbeitet. Auf Grund der Asynchronität wurde stattdessen eine Warteschlange implementiert, die eintreffende Nachrichten zunächst zwischenspeichert und auf diese Weise den Koordintor synchronisiert. Abbildung 5.4 stellt diesen Vorgang als Sequenzdiagramm dar. Wenn ein Service die Nachricht empfängt, reiht er sie mit dem Methodenaufruf `queue` in die Warteschlange ein. Ein separater Worker-Thread leitet die Nachrichten später weiter an die zuständigen Koordinatoren, indem er die Nachricht aus der Warteschlange entfernt (`dequeue`), ihre Header-Elemente analysiert, mit Hilfe des Kontextes den assoziierten Koordinator lädt (`load(Context)`) und schließlich die Nachricht an die zuständige Methode des Koordinators weiterleitet.

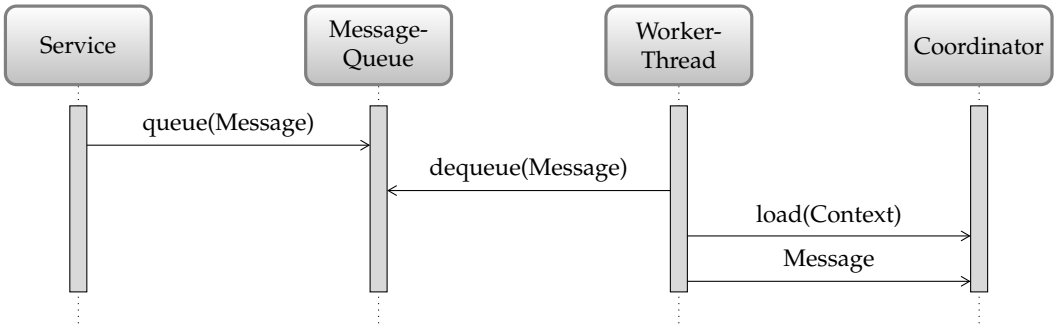

Abbildung 5.4: Sequenzdiagramm für die Nachrichtensynchronisation.

Durch diese Art der Nachrichtenverarbeitung wird einerseits der Empfang von Nachrichten von deren Verarbeitung entkoppelt und andererseits können so konkurrierende Änderungen des Kontextes oder der Teilnehmerlisten verhindert werden, da stets nur ein einziger Worker-Thread läuft und daher nur eine Nachricht zur Zeit verarbeitet wird.

5.3 Umsetzung der Inferenzregeln mit Mandarax

Die Umsetzung der Inferenzregeln (vgl. Regeln 4.4–4.7), mit deren Hilfe der Koordinator über den Abschluss von Teilnehmern entscheidet, wurde mit der Java-Klassenbibliothek Mandarax[2] [Die05] realisiert. Das mitterweile in Version 3.4 vorliegende Open-Source-Projekt implementiert eine Prädikatenlogik der ersten Ordnung [DH01] und ließ sich auf Grund seiner Java-Architektur problemlos in die Implemenation integrieren.

Die Repräsentation des Aufrufbaumes wird in der Klasse `InvocationTreeImpl` als Regelmenge umgesetzt, die mit Mandarax in einer Wissensbasis verwaltet wird. Die Klassen sind in Abbildung 5.5 dargestellt.

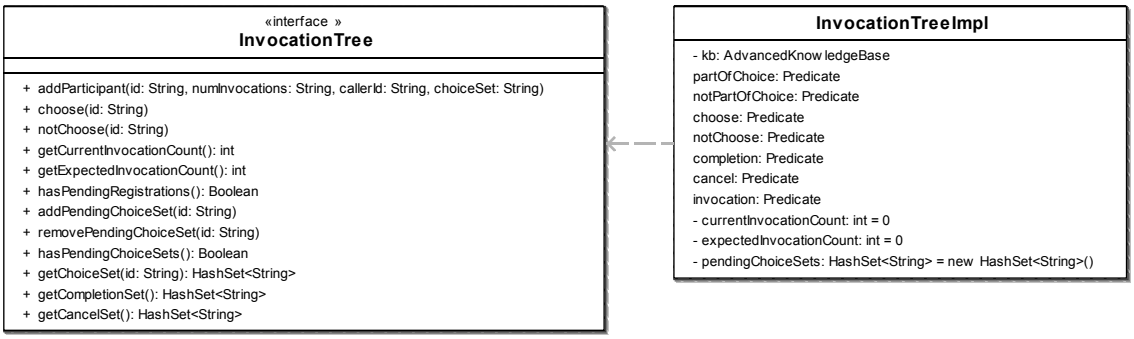

Abbildung 5.5: Klassendiagramm des Aufrufbaumes.

Beim Erzeugen der Instanz werden die Inferenzregeln über die Mandarax-API angelegt. Die Angabe des Kontextes z in den Regeln kann hierbei entfallen, da für jeden Kontext eine eigene Koordinatorinstanz mit zugehörigem Aufrufbaum erstellt wird. Listing 5.2 zeigt einen Ausschnitt der Initialisierung des Aufrufbaumes. Da Mandarax keine Negation von Prädikaten unterstützt, müssen für negierte Ausdrücke explizite Prädikate eingeführt werden (vgl. Zeilen 14, 17).

Listing 5.2: Initialisierung des Aufrufbaumes.

```
1  package de.vsis.coordination.coordinator.invocationtree;
2  import org.mandarax.kernel.*;
3  import org.mandarax.reference.AdvancedKnowledgeBase;
4  import org.mandarax.util.LogicFactorySupport;
5  ...
6  public class InvocationTreeImpl implements InvocationTree {
7    ...
8    public InvocationTreeImpl() {
9      Class[] struct1 = {String.class};
10     Class[] struct2 = {String.class, String.class};
11     // predicates
12     Predicate partOfChoice = new SimplePredicate("partOfChoice",
```

[2] http://mandarax.sourceforge.net

```
13        struct1);
14      Predicate notPartOfChoice = new SimplePredicate("notPartOfChoice",
15        struct1);
16      Predicate choose = new SimplePredicate("choose", struct1);
17      Predicate notChoose = new SimplePredicate("notChoose", struct1);
18      Predicate completion = new SimplePredicate("completion", struct1);
19      Predicate cancel = new SimplePredicate("cancel", struct1);
20      Predicate invocation = new SimplePredicate("invocation", struct2);
21      // variable terms
22      LogicFactorySupport lfs = new LogicFactorySupport();
23      Term a = lfs.variable("a");   // parent node
24      Term b = lfs.variable("b");   // child node
25      // rules
26      Rule rule1 = lfs.rule(
27        lfs.prereq(invocation, a, b), lfs.prereq(completion, a),
28        lfs.prereq(notPartOfChoice, b), lfs.fact(completion, b));
29      Rule rule2 = lfs.rule(
30        lfs.prereq(invocation, a, b), lfs.prereq(completion, a),
31        lfs.prereq(partOfChoice,b), lfs.prereq(choose, b),
32        lfs.fact(completion, b));
33      Rule rule3 = lfs.rule(
34        lfs.prereq(invocation, a, b), lfs.prereq(cancel, a),
35        lfs.fact(cancel, b));
36      Rule rule4 = lfs.rule(
37        lfs.prereq(invocation, a, b), lfs.prereq(partOfChoice, b),
38        lfs.prereq(notChoose, b), lfs.fact(cancel, b));
39      // setup knowledge base
40      kb = new AdvancedKnowledgeBase();
41      kb.add(rule1); kb.add(rule2); kb.add(rule3); kb.add(rule4);
42    }
43  ...
```

Neue Teilnehmer werden bei ihrer Registrierung mit der Methode `addParticipant` in die Wissensbasis aufgenommen, indem das Prädikat `completion` als Fakt gesetzt wird und sie auf diese Weise der Complete-Menge zugeordnet werden. Ferner werden die Instanzvariablen `currentInvocationCount` und `expectedInvocationCount` aktualisiert. Sie entsprechen der Anzahl bereits registrierter Teilnehmer P_{reg} und der Anzahl erwarteter Teilnehmer P_{exp}. Für die Auswahl von Teilnehmern einer Choice-Menge stehen auf Grund der fehlenden Unterstützung für negierte Prädikate die beiden Methoden `choose` und `notChoose` zur Verfügung. Entsprechend wird für ausgewählte Teilnehmer der Fakt `choose` und für nicht ausgewählte Teilnehmer der Fakt `notChoose` in die Wissensbasis geschrieben.

Da die Wissensbasis im Verlauf des Prozesses stets um neue Fakten erweitert wird und keine vorhandenen Regeln gelöscht werden, kann dies zu widersprüchlichen Aus-

sagen führen, wenn gleichzeitig mehrere Regeln und Fakten gültig sind. Für den Aufrufbaum trifft dies auf das Verschieben von Teilnehmern aus der Complete-Menge in die Cancel-Menge zu, denn Teilnehmer der Cancel-Menge befinden sich danach auch in der Complete-Menge.

Zwar sieht Mandarax in der Klasse `org.mandarax.util.comparators` verschiedene Strategien zur Priorisierung von Regeln und Fakten vor, diese bieten jedoch keine Bevorzugung der zuletzt hinzugefügten Regeln vor älteren Regeln. Daher wurde dieses Problem in der Implementation mit Bildung einer Mengendifferenz gelöst, indem die Teilnehmer der Cancel-Menge explizit aus der Complete-Menge entfernt wurden.

5.4 Erweiterungen der SOAP-Nachrichten

Die Implementierung des vorgestellten Koordinatorkonzepts setzt verschiedene Erweiterungen von WS-Coordination voraus. Zunächst müssen die Teilnehmer identifizierbar sein, um ein Protokoll implementieren zu können. Der Koordinator erzeugt daher bei einer Registrierung von Diensten eindeutige Teilnehmer-Ids und sendet diese mit der `RegisterResponse`-Nachricht den Teilnehmern als `yourId` zu (siehe Listing 5.3).

Listing 5.3: Übermittlung der Teilnehmer-Id vom Koordinator.

```
<ns2:RegisterResponse>
  <ns2:CoordinatorProtocolService>
    <Address>
      http://localhost:8080...
    </Address>
  </ns2:CoordinatorProtocolService>
  <vsis:yourId>
    16103ECC-3AA1-42B2-8218-DF716763E94C
  </vsis:yourId>
</ns2:RegisterResponse>
```

Zur Identifikation müssen Teilnehmer diese Id fortan als `participantId` eines erweiterten Kontextes in jeder Nachricht mitsenden:

Listing 5.4: Übermittlung der `participantId` an andere Teilnehmer.

```
<coord:CoordinationContext>
  <Identifier>urn:uuid:1F2838A8...</Identifier>
  <RegistrationService>
    <addressing:Address>
      http://localhost:8080...
    </addressing:Address>
  </RegistrationService>
  <vsis:participantId>
    16103ECC-3AA1-42B2-8218-DF716763E94C
```

```
10    </vsis:participantId>
11  </coord:CoordinationContext>
```

Andere Teilnehmer können so erkennen, von wem sie aufgerufen wurden und teilen dies dem Koordinator bei ihrer Registrierung als Element `callerId` mit (vgl. Listing 5.5). Der Koordinator verwendet diese Information, um einen Aufrufbaum der Teilnehmer zu erzeugen.

Listing 5.5: Übermittlung der `callerId` an den Koordinator.

```
1  <coord:CoordinationContext>
2    ...
3    <vsis:callerId>
4      2BAE044F-E034-4B25-B4DA-4ADC4E3C2CF1
5    </vsis:CallerId>
6  </coord:CoordinationContext>
```

Wenn Teilnehmer beabsichtigen, weitere Dienste aufzurufen, müssen sie dies dem Koordinator ebenfalls mitteilen, damit der Koordinator abschätzen kann, ob noch weitere Teilnehmer der Transaktion beitreten werden. Die Registrierungsnachricht muss in diesem Fall das Element `subsequentCalls` enthalten:

Listing 5.6: Übermittlung der Anzahl nachfolgender Aufrufe an den Koordinator.

```
1  <coord:CoordinationContext>
2    ...
3    <vsis:subsequentCalls>
4      2
5    </vsis:subsequentCalls>
6  </coord:CoordinationContext>
```

Die letzte Erweiterung betrifft die Auswahl von Teilnehmern einer Choice-Menge mit Hilfe der neu eingeführten `Choose`-Nachricht:

Listing 5.7: Auswahl eines Teilnehmers mit der `Choose`-Nachricht.

```
1  <soapenv:Body>
2    <ns2:Choose ChoiceSet="516D8452...">
3      <ns2:ParticipantId>
4        8A895863-CE72-4F2C-BBE8-96BD215801AA
5      </ns2:ParticipantId>
6    </ns2:Choose>
7  </soapenv:Body>
```

Diese Erweiterungen der Nachrichten reichen aus, um das beschriebene Konzept des Koordinators umzusetzen und seine Autonomie hinsichtlich einer Entscheidung über den Transaktionsabschluss zu gewährleisten.

5.5 Zusammenfassung und Fazit

In diesem Kapitel konnte nur auszugsweise auf einige interessante Aspekte der Implementierung eingegangen werden. Zusätzlich wurden beispielsweise Hilfsklassen implementiert, die die Verarbeitung der SOAP-Header erleichtern oder Teilnehmer-Ids erzeugen.

Außerdem wurden Teilnehmer für die Protokolle WS-AT[3] und WS-BA umgesetzt, die eine einfache Anwendungslogik implementieren und zu Testzwecken eingesetzt wurden. Verschiedene Demo-Applikationen für die unterschiedlichen Protokolle demonstrieren eine Transaktionskoordinierung und ermöglichen auf diese Weise die Überprüfung des Nachrichtenverkehrs zwischen dem Koordinator und den Teilnehmern. Für die Sichtung der Nachrichten kam der in Axis2 integrierte SOAPMonitor zum Einsatz, der das Monitoring von ein- und ausgehenden Nachrichten an einem bestimmten Port erlaubt. JUnit-Tests konnten ausschließlich für die Erzeugung des Koordinationskontextes und die Registrierung von Teilnehmern umgesetzt werden.

Der entwickelte Prototyp basiert auf einer im Arbeitsbereich VSIS bestehenden Umsetzung von WS-Coordination und WS-AtomicTransaction und erweitert das Projekt um die komplette Umsetzung von WS-BusinessActivity. Darauf aufbauend konnte das entwickelte Konzept der aggregierten Koordinatorzustände sowie die Entscheidungsfindung des Koordinators mit Hilfe der Inferenzregeln vollständig umgesetzt werden. Insgesamt umfasst der Quellcode drei Eclipse-Projekte mit ca. 400 Klassen und etwa 80000 Zeilen Java-Quellcode. Hinzu kommen ferner die WSDL-Dokumente, mit denen die Client-Stubs und Server-Skeletons erzeugt wurden. Der Prototyp ist mit dieser Arbeit ungefähr auf das Doppelte gewachsen.

[3] Dieser Quellcode war bereits zuvor im TracG-Projekt realisiert worden.

6 Zusammenfassung und Ausblick

Dieses abschließende Kapitel liefert eine Zusammenfassung der vorangegangenen Arbeit und präsentiert die erhaltenen Ergebnisse. Desweiteren wird ein Ausblick auf weiterführende Themen gegeben.

6.1 Zusammenfassung und Ergebnisse

Als wesentliche technische Grundlage wurden im zweiten Kapitel zunächst serviceorientierte Architekturen einschließlich der erforderlichen Spezifikationen und Protokolle vorgestellt. Ein wichtiges Merkmal dieses Konzeptes ist die angestrebte Interoperabilität in heterogenen Umgebungen, die eine Grundvoraussetzung für die Kopplung von Diensten darstellt. Mit weiterführenden Betrachtungen zur Komposition von Web-Services zu geschäftlichen Anwendungen aus der Perspektive von Orchestrierungen und Choreographien konnte ferner die Problematik verdeutlicht werden, die der Aufbau von Geschäftsprozessen im organisationsübergreifenden Umfeld birgt: Unabhängige Teilnehmer, die im Kontext dieser Arbeit durch Web-Services repräsentiert werden, müssen kollaborativ zusammenarbeiten, um eine gemeinsame Übereinkunft hinsichtlich geschäftlicher Ziele zu erreichen.

Das darauffolgend eingeführte Konzept der Transaktionen dient als Rahmen, um eine Folge von Operationen konsistent in einen neuen Zustand zu überführen. Für die Ausweitung dieses Modells auf eine verteilte Transaktionsverarbeitung wurde X/Open-DTP mit dem 2PC-Protokoll betrachtet, das mit Hilfe eines Transaktionsmanagers den geregelten Abschluss der Teilnehmer erreicht. Es zeigte sich, dass verfügbare Spezifikationen für die transaktionale Unterstützung von Web-Service-Prozessen im wesentlichen nach demselben Prinzip arbeiten. Verschiedene Spezifikationen wurden vorgestellt, wobei sich WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity als aussichtsreichste Standards herausstellten.

Im dritten Kapitel konnte nun schließlich die Frage erörtert werden, wie eine transaktionale Zusammenarbeit von unabhängigen Teilnehmern zu bewerkstelligen ist. Bereits im Vorwege einer späteren Implementierung erwiesen sich hier die untersuchten Spezifikationen WS-Coordination und WS-BusinessActivity für den Bereich langlebiger Transaktionen als lückenhaft. Folgende Defizite wurden identifiziert:

- In WS-BA existiert kein Initiator-Koordinator-Protokoll, wie dies bei WS-AT der Fall ist. Der Initiator hat somit keine Möglichkeit, dem Koordinator mitzuteilen, wie die Transaktion abzuschließen ist.
- Das Privileg zum Transaktionsabschluss liegt stets beim Initiator einer Transaktion, dieser kann es wegen des fehlenden Completion-Protokolls jedoch nicht ausüben.

- Vor dem Hintergrund begrenzter lokaler Sichten besitzen der Initiator sowie andere Teilnehmer unter Umständen keine Qualifikation, die Demarkation wahrzunehmen.

Auf Grund der unvollkommenen Spezifikation, muss folglich jeder eigene Versuch einer Lösung zu Inkompatibilitäten mit anderen Umsetzungen führen. Der naheliegende Ansatz einer Integration von Initiator und Koordinator als Ausgleich für das nicht existente Steuerungsprotokoll widerspricht zusätzlich dem Verständnis einer SOA als dienstbasierte Architektur und kann dennoch das Problem der lokalen Sichtbeschränkung nicht beheben.

Der in dieser Arbeit realisierte Ansatz zeigt einen Ausweg auf, indem das Demarkationsrecht an den Koordinator übertragen wird und er zusätzlich mit folgenden Informationen versorgt wird:

- Der Anzahl der Teilnehmer zur Bestimmung des Zeitpunktes für die Einleitung des Prozessabschlusses,
- strukturellen Informationen zur Erstellung einer globalen Sicht auf den Prozess und
- Choice-Mengen zur Realisierung von Transaktionen mit nicht-atomarem Ergebnis.

Hierdurch erübrigt sich ein Completion-Protokoll nach Vorbild von WS-AT und es wird sichergestellt, dass der Koordinator das Privileg zur Demarkation stets wahrnehmen kann. Den Kern dieser Lösung bildet ein im Rahmen des TracG-Projekts entwickeltes Koordinations-Framework, das auf Basis von Inferenzregeln arbeitet, und in Verbindung mit der an den Koordinator weitergegebenen Information, die autonome Prozessbeendigung einleiten kann.

Der Vorschlag wurde in der vorliegenden Arbeit dargestellt und hinsichtlich einer Realisierung konzeptionell präzisiert. Es wurden konkrete Erweiterungen der Registrierungsnachricht, des Koordinationskontextes sowie eine zusätzliche `Choose`-Nachricht für WS-Coordination und WS-BusinessActivity eingeführt. Beim Entwurf des Koordinators wurde eine Überführung der lokalen Teilnehmerzustände in aggregierte Koordinatorzustände entwickelt, die den globalen Prozesszustand ausdrücken und so das weitere Vorgehen des Koordinators beschreiben. Die Ausarbeitung eines Algorithmus für die Koordinierung auf Basis der Protokollnachrichten lieferte schließlich die Grundlage für eine Implementierung.

Mit der vollständigen Umsetzung des Koordinators für langlebige Transaktionen sowie der dafür notwendigen Protokollerweiterungen konnte die Realisierbarkeit des Konzeptes dokumentiert werden. Die Hauptaufgabe der vorliegenden Arbeit wurde damit erfolgreich verwirklicht.

Dabei zeigte sich, dass auch WS-Coordination unzureichend spezifiziert ist. So werden beispielsweise keine Vorgaben zur Identifizierung von Teilnehmern an einer Transaktion gemacht – diese sind jedoch eine essentielle Voraussetzung für die Steuerung eines

koordinierten Prozesses. Entsprechende Mängel mussten daher durch eigene Entwurfsentscheidungen ausgeglichen werden, um einen funktionsfähigen Koordinator zu erhalten. Hierdurch kann es zu Inkompatibilitäten zu anderen Implementierungen kommen, beiden denen die eine andere Entwurfsentscheidung getroffen wurde.

Im Kontext des entwickelten Koordinatorentwurfs wurden außerdem erweiterte Anforderungen von Transaktionen diskutiert, welche insbesondere die Fehlertoleranz und Flexibilität transaktionaler Abläufe adressieren. Die Vitalitätseigenschaft von Teilnehmern stellte sich hierbei als geeignete Ergänzung langlebiger Business-Transaktionen heraus, da mit der Einbindung von nicht-vitalen Teilnehmern die Abbruchwahrscheinlichkeit des Prozesses gesenkt werden kann. Die Regelmenge des Koordinators, deren Auswertung über den Prozessabschluss entscheidet, konnte für eine Umsetzung entsprechend erweitert werden. Mit der Implementierung der Ausgangsregeln konnte zudem gezeigt werden, dass das regelbasierte Konzept tragfähig ist, so dass die Integration der Vitalitätsregeln als unproblematisch angesehen werden kann.

Abhängigkeiten zwischen den Dienstaufrufen von Teilnehmern ließen sich dagegen nicht vollständig auf das Konzept eines dienstbasierten Koordinators abbilden. Das vorrangige ist hierbei die Unfähigkeit des Koordinators selbst Dienste aufzurufen. Eine ähnliche Problematik besteht bei der Realisierung von Implementationsmustern. Um die Transaktionssicherheit zu gewährleisten, müssen auch hier negative Abhängigkeiten, also Alternativdienste, herangezogen werden können. Falls dies möglich wird, lassen sich die Implementationsmuster zusätzlich als Optimierungskriterium für die erste Phase des Prozesses nutzen.

Die Umsetzung von Fristen und Verfallsdaten erfordert eine zeitbasierte Komponente, um das Verstreichen von Terminen zu signalisieren. Der Abbruch von Transaktionen bei Erhalt eines Zeitsignals stellt dagegen kein Problem dar. Leider bietet das verwendete Axis2-Framework keine derartige Unterstützung an, da es selbst vollkommen nachrichtenbasiert arbeitet.

6.2 Ausblick

Das in dieser Arbeit vorgestellte Konzept eines dienstbasierten Koordinators konzentriert sich ausschließlich auf die technische Umsetzung der transaktionalen Steuerung von Web-Service-Prozessen. Daher ergeben sich mit den eingeführten Erweiterungen Fragestellungen im „Umfeld“ der Transaktionssteuerung. Wenn die vorgestellten Erweitungen beispielsweise bei der Modellierung von choreographierten Dienstkompositionen eingesetzt werden sollen, ist zu überprüfen, welche Auswirkungen sich für verfügbare Beschreibungssprachen ergeben (z.B. WS-CDL [FJ07]) können. Wünschenswert wäre eine Einbindung der erweiterten Koordinationskonzepte in bestehende Beschreibungsmodelle.

Bei Transaktionen mit nicht-atomaren Ergebnissen zeigte sich außerdem, dass insbe-

sondere die `Fail`-Nachricht eines Teilnehmers unterschiedliche Interpretationen und damit unterschiedliche Verhaltensweisen des Koordinators zulässt. Unter der Prämisse, ein möglichst robustes Transaktionsverhalten zu realisieren, kann die Bedeutung der `Fail`-Nachricht beispielsweise als „geringfügig“ eingestuft werden, wenn der Teilnehmer nicht erforderlich für einen Prozessabschluss ist. Dies hätte zur Folge, dass der Koordinator das Scheitern des Teilnehmers weitestgehend ignorieren kann und die Transaktion mit den verbleibenden Teilnehmern beendet. Bei einer entgegengesetzten Interpretation, führt eine `Fail`-Nachricht dagegen in jedem Fall zu einem Prozessabbruch, da sie nun ein schwerwiegendes Ereignis darstellt. In diesem Kontext ist eine sinnvolle Parametrisierung von Koordinatoren zu untersuchen. Da in den Protokollen von WS-BA bis zu vier unterschiedliche `Fail`-Nachrichten vorkommen, kann eine Konfiguration des Koordinators sogar noch weiter heruntergebrochen werden. Eine Aufgabe für zukünftige Arbeiten ist darin zu sehen, Konfigurationen für unterschiedliche Einsatzgebiete der Transaktionssteuerung zu finden. Vorkonfigurierte Koordinatoren könnten auf diese Weise schließlich in Dienstverzeichnissen bereitgestellt werden.

Zusätzlicher Forschungsbedarf besteht ferner in der Integration der vorgestellten erweiterten Aspekte der Transaktionssteuerung zu einem ganzheitlichen Konzept. Einerseits sind in diesem Bereich noch Fragestellungen hinsichtlich der Kommunikation der Teilnehmer untereinander zu klären und zum anderen konnten wechselseitige Effekte der diskutierten Erweiterungen nicht erschöpfend betrachtet werden. So muss beispielsweise genauer analysiert werden, wie Teilnehmerabhängigkeiten und Implementationsmuster in Verbindung mit Choice-Mengen oder der Vitalitätseigenschaft umzusetzen sind.

6.3 Fazit

Die Realisierung der Transaktionkoordinatoren zeigt, dass mit der regelbasierten Transaktionssteuerung durch einen autonomen Koordinationsdienst ein tragfähiges Konzept für die transaktionale Absicherung in einer organisationsübergreifenden Dienstkomposition besteht.

Gleichzeitig offenbart die vorliegende Arbeit spezifikationsinhärente Kompatibilitätsprobleme in zweierlei Hinsicht. Einerseits bleiben essentielle Voraussetzungen für eine Transaktionssteuerung, wie beispielsweise Teilnehmer-Ids, unspezifizert und andererseits werden implizite Annahmen über die Rollen- und Rechteverteilung der Teilnehmer getroffen. Beide Aspekte müssen durch individuelle Entwurfsentscheidungen kompensiert werden, die den Einsatz der Protokolle in heterogenen Umgebungen erschweren. Strenggenommen werden die Zielvorgaben einer serviceorientierten Architektur bezüglich der Interoprabilität von Diensten durch die unzureichenden Spezfikationen verletzt.

Literaturverzeichnis

[AAA+96] ALONSO, G.; AGRAWAL, D.; ABBADI, A. E.; KAMATH, M.; GÜNTHÖR, R.; MOHAN, C.: Advanced Transaction Models in Workflow Contexts. In: *Proceedings of the Twelfth International Conference on Data Engineering*, 1996, S. 574–581

[AAA+07] ALVES, Alexandre; ARKIN, Assaf; ASKARY, Sid; BARRETO, Charlton; BLOCH, Ben; CURBERA, Francisco; FORD, Mark; GOLAND, Yaron; GUÍZAR, Alejandro; KARTHA, Neelakantan; LIU, Canyang K.; KHALAF, Rania; KÖNIG, Dieter; MARIN, Mike; MEHTA, Vinkesh; THATTE, Satish; VAN DER RIJN, Danny; YENDLURI, Prasd; YIU, Alex: *Web Services Business Process Execution Language Version 2.0*. OASIS, April 2007. – http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

[ACKM04] ALONSO, Gustavo; CASATI, Fabio; KUNO, Harumi; MACHIRAJU, Vijay: *Web Services: Concepts, Architectures and Applications*. Springer, 2004

[BBF+05] BARRETO, Charlton; BURDETT, David; FLETCHER, Tony; LAFON, Yves; KAVANTZAS, Nickolas; RITZINGER, Gregory: *Web Services Choreography Description Language Version 1.0*. W3C, November 2005. – http://www.w3.org/TR/ws-cdl-10/

[BCH+03] BUNTING, Doug; CHAPMAN, Martin; HURLEY, Oisin; LITTLE, Mark; MISCHKINSKY, Jeff; NEWCOMER, Eric; WEBBER, Jim; SWENSON, Keith: *Web Services Composite Application Framework (WS-CAF) Ver1.0*, Juli 2003. – http://www.oasis-open.org/committees/download.php/4343/WS-CAF%20Primer.pdf

[BDO05] BARROS, Alistair; DUMAS, Marlon; OAKS, Phillipa: *A Critical Overview of the Web Services Choreography Description Language (WS-CDL)*. Online. März 2005. – http://www.bptrends.com/publicationfiles/03-05%20WP%20WS-CDL%20Barros%20et%20al.pdf

[BHLT06] BRAY, Tim; HOLLANDER, Dave; LAYMAN, Andrew; TOBIN, Richard: *Namespaces in XML 1.1*. Second Edition. W3C, August 2006. – http://www.w3.org/TR/xml-names11/

[BHM+04] BOOTH, David; HAAS, Hugo; MCCABE, Francis; NEWCOMER, Eric; CHAMPION, Michael; FERRIS, Chris; ORCHARD, David: *Web Services Architecture*. W3C, Februar 2004. – http://www.w3.org/TR/ws-arch/

[BL07] BOOTH, David; LIU, Canyang K.: *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. W3C, März 2007. – http://www.w3.org/TR/wsdl20-primer/

[BPSM+06] BRAY, Tim; PAOLI, Jean; SPERBERG-MCQUEEN, C. M.; MALER, Eve; YERGEAU, François; CROWAN, John: *Extensible Markup Language (XML) 1.1*. Second Edition. W3C, September 2006. – http://www.w3.org/TR/xml11/

[CCJL04] CABRERA, Luis F.; COPELAND, George; JOHNSON, Jim; LANGWORTHY, David: *Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity*. Online. Januar 2004. – http://msdn.microsoft.com/en-us/library/ms996526.aspx

[CCMW01] CHRISTENSEN, Erik; CURBERA, Francisco; MEREDITH, Greg; WEERAWARANA, Sanjiva: *Web Services Description Language (WSDL) 1.1*, März 2001. – http://www.w3.org/TR/wsdl

[CHRR04] CLEMENT, Luc; HATELY, Andrew; VON RIEGEN, Claus; ROGERS, Tony: *Universal Description, Discovery and Integration v3.0.2 (UDDI)*. OASIS, Oktober 2004. – http://uddi.org/pubs/uddi_v3.htm

[CT04] COWAN, John; TOBIN, Richard: *XML Information Set*. Second Edition. W3C, Februar 2004. – http://www.w3.org/TR/xml-infoset/

[DH01] DIETRICH, Jens; HILLER, Jochen: Mandarax – Ein OpenSource Ansatz zur Verwaltung und Verarbeitung von Geschäftsregeln. In: *Tagungsband Net.ObjectDays 2001*, 2001

[Die05] DIETRICH, Jens: *The Mandarax Manual*. Massey University, Palmerston North, New Zealand: Institute of Information Sciences & Technology, März 2005. – http://mandarax.sourceforge.net

[DJMZ05] DOSTAL, Wolfgang; JECKLE, Mario; MELZER, Ingo; ZENGLER, Barbara: *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis*. Elsevier, Spektrum Akademischer Verlag, 2005

[DKLW07] DECKER, Gero; KOPP, Oliver; LEYMANN, Frank; WESKE, Mathias: *BPEL4Chor: Extending BPEL for Modeling Choreographies*. Internet. Mai 2007. – https://bpt.hpi.uni-potsdam.de/twiki/pub/Public/GeroDecker/icws2007-BPEL4Chor.pdf

[DWSM05] DUERST, M.; W3C; SUIGNARD, M.; MICROSOFT: *Internationalized Resource Identifiers (IRIs)*. Internet Engineering Task Force, Januar 2005. – http://www.ietf.org/rfc/rfc3987.txt

[EHHZ07] ERVEN, Hannes; HICKER, Gerorg; HUEMER, Christian; ZAPLETAL, Marco: The Web Services-BusinessActivity-Initiator (WS-BA-I) Protocol: an Extension to the Web Services BusinessActivity Specification. In: *IEEE International Conference on Web Services 2007 (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, 2007, S. 216–224

[ELLR90] ELMAGARMID, Ahmed K.; LEU, Yungho; LITWIN, Witold; RUSINKIEWICZ, Marek: A Multidatabase Transaction Model for InterBase. In: *Proceedings of the 16th International Conference on Very Large Data Bases, Brisbane, Australia 1990*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1990, S. 507–518

[Elm92] ELMARGARMID, Ahmed K.; SPATZ, Bruce M. (Hrsg.): *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, San Mateo, 1992

[FDF+04] FURNISS, Peter; DALAL, Sanjay; FLETCHER, Tony; GREEN, Alastair; HAUGEN, Bob; CEPONKUS, Alex; POPE, Bill: *Business Transaction Protocol (BTP 1.1)*. OASIS, Oktober 2004. – `http://www.oasis-open.org/committees/download.php/9836/business_transaction-btp-1.1-spec-wd-04.pdf`

[FG05] FURNISS, Peter; GREEN, Alastair: *Choreology Ltd. Contribution to the OASIS WS-TX Technical Committee relating to WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity*. Online. November 2005. – `http://www.oasis-open.org/committees/download.php/15808/Choreology.WS-TX.TC.Contribution.2005-11-16.doc`

[FJ07] FEINGOLD, Max; JEYARAMAN, Ram: *Web Services Coordination (WS-Coordination) Version 1.1*. OASIS, Juli 2007. – `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec/wstx-wscoor-1.1-spec.html`

[FL07] FREUND, Tom; LITTLE, Mark: *Web Services Business Activity (WS-BusinessActivity) Version 1.1*. OASIS, Juli 2007. – `http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec/wstx-wsba-1.1-spec.html`

[FTW07] FROTSCHER, Thilo; TEUFEL, Marc; WANG, Dapeng: *Java Web Services mit Apache Axis2*. entwickler.press, 2007

[Fur04] FURNISS, Peter: *OASIS BTP Scope, Status and Directions*. Online. April 2004. – `http://www.oasis-open.org/events/symposium/2004/slides/furniss.ppt`

[FW04] FALLSIDE, David C.; WALMSLEY, Priscilla: *XML Schema Part 0: Primer*. Second Edition. W3C, Oktober 2004. – http://www.w3.org/TR/xmlschema-0/

[GHM+03a] GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean-Jacques; NIELSEN, Henrik F.: *SOAP Version 1.2*. W3C, Juni 2003. – http://www.w3.org/TR/soap/

[GHM+03b] GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean-Jacques; NIELSEN, Henrik F.: *SOAP Version 1.2 Part 1: Messaging Framework*. W3C, Juni 2003. – http://www.w3.org/TR/soap12-part1/

[GHM+03c] GUDGIN, Martin; HADLEY, Marc; MENDELSOHN, Noah; MOREAU, Jean-Jacques; NIELSEN, Henrik F.: *SOAP Version 1.2 Part 2: Adjuncts*. W3C, Juni 2003. – http://www.w3.org/TR/soap12-part2/

[GHR06a] GUDGIN, Martin; HADLEY, Marc; ROGERS, Tony: *Web Services Addressing 1.0 – Core*. W3C, Mai 2006. – http://www.w3.org/TR/ws-addr-core/

[GHR06b] GUDGIN, Martin; HADLEY, Marc; ROGERS, Tony: *Web Services Addressing 1.0 – SOAP Binding*. W3C, Mai 2006. – http://www.w3.org/TR/ws-addr-soap/

[GMGK+90] GARCIA-MOLINA, Hector; GAWLICK, Dieter; KLEIN, Johannes; KLEISSNER, Karl; SALEM, Kenneth: Coordinating Multi-Transaction Activities / Department of Computer Science, Princeton University. 1990 (CS-TR-297-90). – Technical Report

[GMUW02] GARCIA-MOLINA, Hector; ULLMAN, Jeffrey D.; WIDOM, Jennifer: *Database Systems: The Complete Book*. International Edition. Prentice Hall, 2002

[GR93] GRAY, Jim; REUTER, Andreas: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993

[GSLV07] GOETHALS, Frank G.; SNOECK, Monique; LEMAHIEU, Wilfried; VANDENBULCKE, Jacques: Considering (de)centralization in a Web Services World. In: *The Second International Conference on Internet and Web Applications and Services 2007 (ICIW 2007)* IEEE, 2007, S. 22

[HB04] HAAS, Hugo; BROWN, Allen: *Web Services Glossary*. W3C, Februar 2004. – http://www.w3.org/TR/ws-gloss/

[Hom06] HOMANN, Jorge: *Spezifikation und Ausführung transaktionaler Abläufe in Grid-Umgebungen*, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Arbeitsbereich Verteilte Systeme und Informationssysteme, Diplomarbeit, 2006

[HRR07] HUSEMANN, Martin; VON RIEGEN, Michael; RITTER, Norbert: Transactional Coordination of Dynamic Processes in Service-Oriented Environments. In: *IEEE International Conference on Web Services 2007 (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, IEEE Computer Society, 7 2007, S. 1024–1031

[HRS05] HALLER, Harald; RICHTER, Jan-Peter; SCHREY, Peter: Serviceorientierte Architektur. In: *Informatik-Spektrum* 28 (2005), Oktober, Nr. 5, S. 413–416

[HTKR05] HÖPFNER, Hagen; TÜRKER, Can; KÖNIG-RIES, Birgitta: *Mobile Datenbanken und Informationssysteme – Konzepte und Techniken*. dpunkt.verlag GmbH, 2005

[KL04] KOSSMANN, Donald; LEYMANN, Frank: Web Services. In: *Informatik-Spektrum* 27 (2004), April, Nr. 2, S. 117–128

[Kra04] KRATZ, Benedikt: *Protocols for Long Running Business Transactions*. Infolab Technical Report Series, Nr. 17. Februar 2004. – `http://infolab.uvt.nl/research/itrs/itrs017.pdf`

[LF03] LITTLE, Mark; FREUND, Thomas: *A comparison of Web services transaction protocols (A comparative analysis of WS-C/WS-Tx and OASIS BTP)*. Oktober 2003. – `http://www.ibm.com/developerworks/webservices/library/ws-comproto/`

[LNP05] LITTLE, Mark; NEWCOMER, Eric; PAVLIK, Greg: *Web Services Coordination Framework Specification (WS-CF)*. OASIS, Oktober 2005. – `http://www.oasis-open.org/committees/download.php/19658/WS-CF.zip`

[LNP06a] LITTLE, Mark; NEWCOMER, Eric; PAVLIK, Greg: *Web Services ACID Specification (WS-ACID)*. OASIS, August 2006. – `http://www.oasis-open.org/committees/download.php/19474/WS-ACID.zip`

[LNP06b] LITTLE, Mark; NEWCOMER, Eric; PAVLIK, Greg: *Web Services Business Process Specification (WS-BP)*. OASIS, August 2006. – `http://www.oasis-open.org/committees/download.php/19475/WS-BP.zip`

[LNP06c] LITTLE, Mark; NEWCOMER, Eric; PAVLIK, Greg: *Web Services Long Running Action Specification (WS-LRA)*. OASIS, August 2006. – `http://www.oasis-open.org/committees/download.php/19473/WS-LRA.zip`

[LNP07] LITTLE, Mark; NEWCOMER, Eric; PAVLIK, Greg: *Web Services Context Specification (WS-Context) Version 1.0*. OASIS, April 2007. – `http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.pdf`

[LW07] LITTLE, Mark; WILKINSON, Andrew: *Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1*. OASIS, Juli 2007. – http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec/wstx-wsat-1.1-spec.html

[ML07] MITRA, Nilo; LAFON, Yves: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C, April 2007. – http://www.w3.org/TR/soap12-part0/

[MRSK92] MEHROTRA, Shared; RASTOGI, Rajeev; SILBERSCHATZ, Abraham; KORTH, Henry F.: A Transaction Model for Multidatabase Systems. In: *Proceedings of the 12th International Conference on Distributed Computing Systems*, IEEE, Juni 1992, S. 56–63

[Ope96] The Open Group: *Distributed Transaction Processing: Reference Model, Version 3*. Ferbuar 1996. – http://www.opengroup.org/bookstore/catalog/g504.htm

[Pel03] PELTZ, Chris: Web Services Orchestration and Choreography. In: *Computer* 36 (2003), Oktober, Nr. 10, S. 46–52

[Rat04] RATHIG, Denis: *Transaktionskontrolle für Grid Data Services*, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, Diplomarbeit, Juni 2004

[RHR08] VON RIEGEN, Michael; HUSEMANN, Martin; RITTER, Norbert: Providing Decision Capabilities to Coordinators in Distributed Processes. In: *The Third International Conference on Internet and Web Applications and Services 2008 (ICIW 2008)*, 2008, S. 500–505

[Rit05] RITTER, Norbert: *Vorlesung: Workflows und Web Services*. Internet. Juli 2005. – http://vsis-www.informatik.uni-hamburg.de/teaching/ss-05/wfws/K3.pdf

[RS04] REICHERT, Manfred; STOLL, Dietmar: Komposition, Choreographie und Orchestrierung von Web Services – ein Überblick. In: *EMISA Forum* 2 (2004), Nr. 24, S. 21–32. – http://www.informatik.uni-ulm.de/dbis/01/dbis/downloads/ReSt04.pdf

[RZ07] VON RIEGEN, Michael; ZAPLATA, Sonja: Supervising Remote Task Execution in Collaborative Workflow Environments. In: BRAUN, Torsten (Hrsg.); CARLE, Georg (Hrsg.); STILLER, Burkhard (Hrsg.): *Konferenzband zur KiVS 2007 für Industrie-, Kurz- und Workshopbeiträge* Gesellschaft für Informatik, VDE Verlag, 2 2007, S. 337–358

[SR93] SHETH, Amit; RUSINKIEWICZ, Marek: On Transactional Workflows. In: *Bulletin of the IEEE Technical Committee Data Engineering* 16 (1993), Nr. 2, S. 37–40

[VGZ+05] VOGT, Friedrich H.; GRUSCHKO, Boris; ZAMBROVSKI, Simon; FURNISS, Peter; GREEN, Alastair: Implementing Web Service Protocols in SOA: WS-Coordination and WS-BusinessActivity. In: *7th IEEE International Conference on E-Commerce Technology Workshops (CECW'05)*, 2005, S. 21–28

[WV02] WEIKUM, Gerhard; VOSSEN, Gottfried: *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002

Eidesstattliche 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.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____