Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Elena Tews

Transaktionskonzepte für Web Services

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Elena Tews

Transaktionskonzepte für Web Services

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. rer. nat. Christoph Klauck

Abgegeben am 19. März 2008

Elena Tews

Thema der Bachelorarbeit

Transaktionskonzepte für Web Services

Stichworte

Web Services, Transaktion, JBoss, WebSphere, Evaluierung

Kurzzusammenfassung

Transaktionsmanagement ist eine der Kernfunktionalitäten eines jeden Enterprise Informationssystems. In dieser Arbeit werden die derzeit gegebenen Möglichkeiten zur Realisierung der Transaktionen in Web Services-Architekturen untersucht. Es werden zwei Produkte vorgestellt, die die WS-Transactions Spezifikationen implementieren. Die Produkte werden hinsichtlich ihrer Funktionalität miteinander verglichen und auf Fehler und die Vollständigkeit der Implementierung geprüft. Abschließend wird zusammengefasst, inwieweit sich diese Implementierungen für die Anwendungserstellung mit Web Services Transaktionen eignen. Als Grundlage für die Evaluierung dient eine Beispielanwendung, die mit den beiden Transaction Frameworks implementiert wurde.

Elena Tews

Title of the bachelor thesis

Transaction Concepts for Web Services

Keywords

Web Services, Transaction, JBoss, WebSphere, Evaluation

Abstract

Transaction processing is at the core of any Enterprise Information System. This paper introduces two implementations of WS-Transactions - a set of specifications designed for transactions in Web services architectures. The tools have been tested with respect to their compliance with the specifications. Incomplete and erratic implementations have been detected. This evaluation is based upon a sample application that has been implemented with both transactional frameworks.

Inhaltsverzeichnis

Tabellenverzeichnis **7**

Abbildungsverzeichnis **8**

1 Einleitung **9**

2 Grundlagen **11**
2.1 Verteilte Transaktionen 11
2.1.1 Kurzlaufende verteilte Transaktionen 11
2.1.1.1 Das Zwei-Phasen-Commit-Protokoll 12
2.1.2 Langlaufende verteilte Transaktionen 13
2.1.2.1 ACID und langlaufende Transaktionen 13
2.1.2.2 Flache und verschachtelte Transaktionen 14
2.2 Web Services 15
2.2.1 Web Services Basiskomponenten 16
2.2.2 Web Services Eigenschaften 16
2.2.3 Die zweite Generation von Web Services 17
2.3 WS-Transactions Spezifikationen 19
2.3.1 WS-Coordination 19
2.3.1.1 Interposition 21
2.3.2 WS-AtomicTransaction 22
2.3.3 WS-BusinessActivity 24

3 Analyse und Entwurf **26**
3.1 Use Cases der Beispielanwendung 26
3.1.1 Autoteile bestellen 28
3.1.2 Lagerbestände kontrollieren 28
3.1.3 Bestellungen aufgeben (1) 28
3.1.4 Bestellungen aufgeben (2) 29
3.1.5 Auftragsbestätigung anfordern 29
3.2 Entwurf der Beispielanwendung 30
3.2.1 Koordinator 31
3.2.2 Transaktionsteilnehmer 31
3.2.3 Web Services Client 31
3.3 Testkriterien 34
3.3.1 Einhaltung der Spezifikationen 34

3.3.2 ACID-Transaktionen . . . 34
3.3.3 Business-Transaktionen . . . 34
3.3.4 Interposition . . . 36
3.3.5 (Protokoll-)Optimierungen . . . 36
3.3.6 Trennung von Koordination und Anwendung . . . 36

4 Implementierung 37
4.1 JBoss XTS . . . 37
4.1.1 Transaction Manager . . . 37
4.1.2 Servlets . . . 37
4.1.3 Transaktionale Web Services . . . 38
4.1.4 Participants . . . 38
4.1.5 Client . . . 39
4.1.6 Beispielanwendung mit XTS . . . 40
4.2 IBM WAS . . . 43
4.2.1 WS-AtomicTransaction . . . 44
4.2.2 WS-BusinessActivity . . . 44
4.2.2.1 WS-BusinessActivity Programmiermodel . . . 45
4.2.2.2 SDO . . . 46
4.2.3 Beispielanwendung mit WAS . . . 46

5 Evaluierung 49
5.1 Einhaltung der Spezifikationen . . . 49
5.2 ACID-Transaktionen . . . 49
5.3 Business-Transaktionen . . . 50
5.4 Interposition . . . 52
5.5 Optimierung vs Nachrichten-Overhead . . . 52
5.6 Trennung von Koordination und Anwendung . . . 52
5.7 Zusammenfassung . . . 53

6 Zusammenfassung und Ausblick 55

A Tests-Beschreibung 57
A.1 Test WAS 1 . . . 57
A.2 Test WAS 2 . . . 58
A.3 Test WAS 3 . . . 59
A.4 Test WAS 4 . . . 61
A.5 Test WAS 5 . . . 62
A.6 Test XTS 1 . . . 63
A.7 Test XTS 2 . . . 65
A.8 Test XTS 3 . . . 67

B Inhalt der begleitenden CD 68

Literaturverzeichnis **69**

Tabellenverzeichnis

4.1 XTS-API 2PC-Teilnehmer 38
4.2 XTS-API die `Vote`-Klassen 39
4.3 XTS-API WS-BA Teilnehmer_1 39
4.4 XTS-API WS-BA Teilnermer_2 40

5.1 Vergleich der WS-Transactions Implementierungen 53

B.1 SOAP-Nachrichten 68

Abbildungsverzeichnis

2.1 Verteilte Transaktionen aus [Coulouris u. a. (2005)] 15
2.2 Interposition aus [wscoor (2007)] 21
2.3 Completion-Protokoll aus [wsat (2007)] 23
2.4 WS-AtomicTransaction aus [wsat (2007)] 23
2.5 WS-BusinessActivity aus [wsba (2007)] 25

3.1 Komponenten für die Evaluierung 26
3.2 Supply Chain aus [Corsten und Gössinger (2001)] 27
3.3 Web Services Kommunikation 29
3.4 Bestellprozess 30
3.5 Koordinator 31
3.6 Die Business Activity-Teilnehmer 32
3.7 Der Atomic Transaction-Teilnehmer 32
3.8 UML Komponentendiagramm 33

4.1 XTS Autohersteller-Service UML 40
4.2 XTS Warehouse-Service UML 41
4.3 XTS Lager-Service UML 42
4.4 XTS Lieferant UML 42
4.5 WAS Lieferanten-Service UML 47
4.6 WAS Autohersteller-Service UML 47
4.7 WAS Warehouse-Service UML 48
4.8 WAS Lager-Service UML 48

A.1 Test WAS 1 57
A.2 Test WAS 4 61
A.3 Test WAS 5 62
A.4 Test XTS 1 64
A.5 Test XTS 2 66

Kapitel 1

Einleitung

Web Services erfüllen den Wunsch nach Interoperabilität in heterogenen Systemen – und das plattformneutral und herstellerunabhängig. Sie sind kostengünstig, da sie die Infrastruktur verwenden, die in der Regel in einem Unternehmen bereits vorhanden ist (Web Server, Netzwerk, Internet). Anders als DCOM oder Java RMI, sind Web Services nicht an eine bestimmte Technologie gebunden, sondern basieren auf bestehenden und weit verbreiteten Internet-Standards (HTTP, XML etc.). Prinzipiell können Anwendungen verschiedener Herkunft unabhängig von den verwendeten Plattformen und Programmiersprachen mit Web Services schnell miteinander verbunden werden.

In den Anfangszeiten des SOA-Booms wurde aber auch die Kritik laut, dass die erste Web Services Generation (SOAP, WSDL und UDDI) nicht für die Enterprise Anwendungen ausgerüstet seien. Für erweiterte Einsatzszenarien von Web Services - insbesondere im Umfeld von servicesorientierten Architekturen, die eine Vielzahl heterogener Komponenten integrieren - sind neben dem einfachen Schnittstellen- und Nachrichtenaustausch weitere Spezifikationen und Standards erforderlich. Zur Realisierung komplexer Anwendungen fehlten insbesondere die Standards für Sicherheit, Transaktionen und garantierte Nachrichtenzustellung.

Heute kann man nicht mehr von einem Mangel an Spezifikationen sprechen. Es existieren mittlerweile mindestens 70 Standards oder Entwürfe für Web Services, die teilweise miteinander konkurrieren (z.B. WS-Reliable Messaging und WS-Reliability).

Für Web Services Transaktionen existieren ebenfalls mehrere Standards:

- das Web Services Transactions Framework, eine Spezifikationsfamilie bestehend aus WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity
- das Business Transaction Protocol (BTP)
- das Web Services Composite Application Framework (WS-CAF)

WS-Transactions Framework scheint sich durchgesetzt zu haben, da die wichtigsten Software-Hersteller an dieser Spezifikation beteiligt sind. Die letzte Version (1.1, herausgegeben im April 2007) ist aus der Zusammenarbeit von über 20 Organisationen, darunter IBM, Microsoft, Oracle, Red Hat und Sun, entstanden. WS-Transactions sind OASIS-Standards,

d.h. es kann jede Firma ein Framework auf Basis dieser Spezifikationen auf den Markt bringen, ohne dafür Lizenzgebühren zu bezahlen.

Ziel dieser Arbeit ist die Untersuchung existierender Implementierungen der WS-Transactions Spezifikationen. Dabei soll experimentell ermittelt werden, ob die Implementierungen spezifikationskonform arbeiten, und inwieweit sich diese Implementierungen für die Anwendungserstellung mit Web Services Transaktionen eignen.

Das nachfolgende Kapitel liefert eine Einführung in den Bereich Web Services und in die für Web Services relevanten Transaktionskonzepte und stellt WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity vor. Im Kapitel 3 wird eine Beispielanwendung beschrieben, anhand derer die Tools für Web Services Transaktionen getestet werden sollen. Die Suche nach den WS-Transacitons Produkten hat folgendes ergeben:

- JBoss XTS: der XML Transaction Service (XTS) in JBoss ist ein Produkt der Firma Arjuna Technologies, das JBoss im Dezember 2005 gekauft hat und seit dem als Open Source weiterentwickelt. XTS unterstützt WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity.
- IBM WebSphere AS (WAS): mit dem WebSphere Application Server 6.1 hat die Firma IBM im Jahr 2006 ihre erste Implementierung des Transaktionsdienstes für Web Services herausgebracht. Es werden ebenfalls alle drei WS-Transactions Spezifikationen unterstützt.
- Microsoft WCF: die Windows Communication Foundation (WCF) im .NET Framework 3.0 implementiert WS-Coordination und WS-AtomicTransaction [Microsoft (2007)].
- Kandula: ein Apache-Projekt, das aus einer Diplomarbeit entstanden ist. Die Implementierung von WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity befindet sich noch in der Entwicklung [Erven und Hicker (2007)].

Somit sind JBoss und IBM derzeit die einzigen Software-Hersteller, die eine vollständige Implementierung von WS-Transactions anbieten. Die Produkte XTS und WAS werden in den Kapiteln 4 und 5 beschrieben. Abschließend folgt eine Evaluierung der Produkte, welche sich auf die Tests und Implementierung der Beispielanwendung stützt, sowie eine Zusammenfassung der gewonnenen Erkenntnisse (Kapitel 6).

Kapitel 2

Grundlagen

2.1 Verteilte Transaktionen

Transaktionen kommen in verschiedenen Bereichen des täglichen Lebens vor. Folglich kann der Transaktionsbegriff kontextabhängig unterschiedlich definiert werden. Im Kontext von Web Services werden die Transaktionen auf Informationssystemen betrachtet. Eine allgemeine Definition von Transaktionen findet man z.B. in [Dostal]: „Eine Transaktion ist eine Menge von Arbeitsschritten mit einem konsistenten Ergebnis."

Transaktionen in Informationssystemen stammen von Datenbankmanagementsystemen. Eine Transaktion in Datenbankmanagementsystemen ist die Ausführung eines Programms, das auf eine Datenbank zugreift.

Man spricht von einer verteilten Transaktion, wenn mehrere Ressourcen an einer Transaktion beteiligt sind. Transaktionen wurden in verteilten Systemen in der Form transaktionaler Datei-Server eingeführt. In diesem Kontext ist eine Transaktion die Ausführung einer Folge der vom Client angeforderten Datei-Operationen.

Eine Transaktion im Kontext von verteilten Objekten besteht aus Ausführung von Client-Anforderungen, welche einen einzigen Schritt darstellen, wobei die Serverdaten von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt werden.

Es sind zwei Arten von Subsystemen an einer verteilten Transaktion beteiligt: mehrere Teilnehmer und ein Transaktionskoordinator. Teilnehmer implementieren die Geschäftslogik und führen einen Teil der Operationen der Transaktion aus. Der Koordinator startet die Transaktion und ist nach deren Ausführung dafür verantwortlich, sie festzuschreiben oder abzubrechen. [Coulouris u. a. (2005)]

2.1.1 Kurzlaufende verteilte Transaktionen

Typischerweise haben kurzlaufende Transaktionen in enggekoppelten Systemen die ACID-Eigenschaften. [Härder und Reuter (1983)] haben die Mnemonik ACID vorgeschlagen, die die Eigenschaften einer Transaktion wie folgt beschreibt:

Atomicity Eine Transaktion wird entweder erfolgreich vollständig ausgeführt, und die Wirkung aller darin enthaltenen Operationen ist in den Objekten aufgezeichnet, oder sie hat überhaupt keine Wirkung (wenn sie fehlschlägt oder bewusst abgebrochen wurde). Diese Alles-oder-Nichts-Eigenschaft bleibt auch dann erhalten, wenn der Server abstürzt.

Consistency Transaktionen überführen das System von einem konsistenten Zustand in einen anderen konsistenten Zustand.

Isolation Jede Transaktion muss ohne Störungen durch andere Transaktionen ausgeführt werden. Zwischenergebnisse einer Transaktion sind für andere Transaktionen nicht sichtbar.

Durability Nachdem eine Transaktion erfolgreich ausgeführt wurde, werden alle ihre Wirkungen im permanenten Speicher festgeschrieben.

[Coulouris u. a. (2005)]

Die ACID-Eigenschaft in einer verteilten Transaktion wird durch das Atomic Commit Protocol (ACP) gewährleistet.

Das ACP ermöglicht es den an einer verteilten Transaktion beteiligten Servern eine gemeinsame Entscheidung darüber zu treffen, ob eine Transaktion festgeschrieben oder abgebrochen werden soll. Das Zwei-Phasen-Commit-Protokoll (2PC), beschrieben in [Gray (1978)], ist das am häufigsten verwendete ACP.

2.1.1.1 Das Zwei-Phasen-Commit-Protokoll

Der Transaktionskoordinator führt das 2PC Protokoll aus, nachdem er vom Client aufgefordert wird, die Transaktion festzuschreiben. Es erlaubt jedem Teilnehmer, seinen Teil der Transaktion abzubrechen. Die Atomarität von Transaktionen erfordert, dass die gesamte Transaktion abgebrochen wird, wenn ein Teil davon abgebrochen wird. Die zwei Phasen des 2PC sind:

Abstimmphase In der Abstimmphase schickt der Koordinator an alle Teilnehmer eine *prepare*-Nachricht. Jeder Teilnehmer stimmt darüber ab, ob die Transaktion festgeschrieben oder abgebrochen werden soll. Wenn der Teilnehmer dafür stimmt, eine Transaktion festzuschreiben, antwortet er mit *prepared* und darf die Transaktion nicht mehr abbrechen. Nach der *prepared*-Nachricht in der ersten Phase bleibt der Teilnehmer so lange blockiert, bis er eine Nachricht vom Koordinator in der zweiten Phase bekommt.

Abschlussphase Die Transaktion ist erfolgreich, wenn alle Teilnehmer, inklusive dem Koordinator mit *prepared* geantwortet haben. In diesem Fall versendet der Koordinator an alle Teilnehmer die *commit*-Nachricht, und die Transaktion wird festgeschrieben. Antwortet einer oder mehrere Teilnehmer mit *cancel*, wird die Transaktion abgebrochen.

Da das 2PC-Protokoll durch Sperren implementiert wird, ist es für die langlaufenden Transaktionen in lose gekoppelten Systemen meistens ungeeignet.

2.1.2 Langlaufende verteilte Transaktionen

Jede Transaktion, die für eine unbestimmte Zeit offen bleibt, kann als langlaufende Transaktion definiert werden, so [Chaudhari u. a. (2007)]. Dazu zählen Business Transaktionen, wie die Web Services Transaktionen, oder Transaktionen mit Abfragen über mehrere Datenbanken.

Folgendes muss beim Transaktionsmanagement in langlaufenden Transaktionen berücksichtigt werden:

- eine Transaktion muss für alle teilnehmenden Services eindeutig identifiziert werden
- Daten und Benachrichtigungen müssen garantiert ausgeliefert werden
- Ressourcen können nicht für die gesamte Dauer einer langlaufenden Transaktion gesperrt bleiben
- der Service muss die Ressourcen wieder freigeben, sobald seine Aufgabe beendet ist
- jeder Service kann prinzipiell während des Transaktionszyklus ausfallen
- Fehler in der Transaktion müssen kompensiert werden

2.1.2.1 ACID und langlaufende Transaktionen

Ausführung der Business Transaktionen ist oft ein lang andauernder Prozess, unter anderem wegen der Benutzerinteraktion, der üblichen Latenzzeiten im täglichen Geschäftsleben oder auch wegen eines Netzwerkfehlers [Pardon (2007)]. Dabei kann sich eine Transaktion über mehrere Minuten oder mehrere Tage erstrecken. In jedem Fall dürfen die Ressourcen der Transaktionsteilnehmer nicht für die gesamte Transaktionsdauer gesperrt bleiben. Im Unterschied zu lokalen Transaktionen ist es für eine verteilte Transaktion nicht immer möglich oder erwünscht, alle ACID-Eigenschaften zu erfüllen.

Atomicity Atomarität führt zu verlorener Arbeit: im Falle eines *rollback* müssen die von mehreren Teilnehmern erfolgreich ausgeführten Operationen wieder rückgängig gemacht werden. Durch Lockerung der Atomarität können einzelne Teilnehmergruppen die Transaktion festschreiben, noch vor dem Ende der gesamten langlaufenden Transaktion.

Consistency Um die Konsistenz zu wahren müssen viele Bestätigungsnachrichten und Statusanfragen verschickt und verarbeitet werden, und zwar auch nach dem Ende der Transaktion. Eine höhere Nachrichtenlast entsteht zum Beispiel dann, wenn ein Teilnehmer nach dem *commit* des Koordinators ausfällt, und der Server wiederholt Statusabfragen verschicken muss.

Isolation Isolation bedeutet hohe Sperr-Raten, was bei einem Netzwerkausfall zum Aushungern anderer Teilnehmer führen kann, die auf die gesperrte Ressource warten.

Durability Um die Durability-Eigenschaft zu erfüllen, muss jeder Teilnehmer selbst einen Mechanismus zur Wiederherstellung nach einem Systemausfall implementieren, da, anders als bei einem Datenbanksystem, werden die Daten in einer verteilten Umgebung nicht zentral gespeichert.

[Dostal u. a. (2005)]

2.1.2.2 Flache und verschachtelte Transaktionen

Zur Implementierung langlaufender Transaktionen existiert eine Reihe von erweiterten Transaktionsmodellen. [Elmargarmid (1992)] Ein häufig diskutiertes Modell ist das Modell der offenen verschachtelten Transaktionen.

Flache Transaktionen

Bei einer flachen Transaktion stellt ein Client Anforderungen an mehr als einen Server (Abbildung 2.1-a). Die Transaktion schließt jede ihrer Anforderungen ab, bevor sie die nächste absetzt. Die Transaktionen greifen also sequentiell auf die Objekte eines Servers zu. Verwendet der Server Sperren, kann eine Transaktion auf jeweils nur ein Objekt gleichzeitig warten. Die gesamte Arbeit geschieht auf derselben Ebene und es ist nicht möglich, Teile davon festzuschreiben oder abzubrechen.

Verschachtelte Transaktionen

Verschachtelte Transaktionen erlauben es, dass Transaktionen sich aus anderen Transaktionen zusammensetzen (Abbildung 2.1-b). Die Top-Level-Transaktion *T* öffnet untergeordnete Transaktionen *T1* und *T2*, und jede untergeordnete Transaktion kann wiederum weitere untergeordnete Transaktionen öffnen, sodass eine beliebige Verschachtelungstiefe entstehen kann.

Verschachtelte Transaktionen haben die folgenden Vorteile:

- Untergeordnete Transaktionen derselben Ebene können nebenläufig ausgeführt werden. In (2.1-b) rufen die Subtransaktionen *T1* und *T2* weitere Subtransaktionen, *T11, T12, T21* und *T22* auf. Da die Transaktionen auf Objekte auf verschiedenen Servern zugreifen, können sie parallel ausgeführt werden.

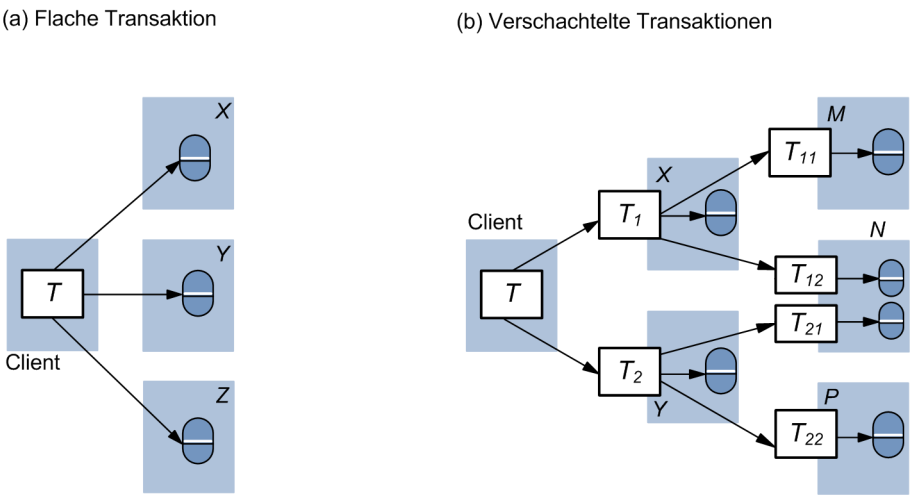

Abbildung 2.1: Verteilte Transaktionen aus [Coulouris u. a. (2005)]

- Untergeordnete Transaktionen können unabhängig voneinander festgeschrieben oder abgebrochen werden. Dadurch wird die gesamte Transaktion robuster, die übergeordnete Transaktion kann trotz des Abbrechens einer ihrer Subtransaktionen weiter ausgeführt werden. Die Ergebnisse bereits erfolgreich ausgeführter Subtransaktionen bleiben dadurch erhalten. Einzelne Fehler können isoliert behandelt werden, zum Beispiel indem eine abgebrochene Subtransaktion erneut gestartet wird.

[Coulouris u. a. (2005)]

Offene und geschlossene verschachtelte Transaktionen

In einer geschlossenen verschachtelten Transaktion warten Subtransaktionen auf das *commit* der Wurzel, bevor sie selbst ein *commit* ausführen und Ressourcensperren wieder freigeben. Im Modell der offenen verschachtelten Transaktionen dürfen die Subtransaktionen ihre Ergebnisse festschreiben, noch bevor das Ergebnis der gesamten Transaktion oder einer der übergeordneten Subtransaktionen bekannt ist. Sollte eine langlaufende Transaktion wieder rückgängig gemacht werden, wird dies durch Kompensation realisiert [Dostal u. a. (2005)].

2.2 Web Services

Es gibt keine verbindliche Definition des Begriffs Web Services. Web Services können unter verschiedenen Aspekten betrachtet werden. Einen konkreten Ansatz liefert die Sichtweise des W3C [W3C (2004)]. Sie nennt Technologien und Spezifikationen, durch die sich Web Services auszeichnen, und soll hier zitiert werden:

„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

2.2.1 Web Services Basiskomponenten

Web Services stützen sich auf die drei OASIS-Standards UDDI, WSDL und SOAP.

WSDL Die Web Service Description Language ist eine anbieter- und plattformunabhängige XML-basierte Sprache zu Beschreibung von Web Services. Ein WSDL-Dokument besteht aus einem abstrakten und einem konkreten Teil. Der abstrakte Teil beschreibt Methoden und Parameter, über die die Web Service-Dienste via SOAP-Nachrichten aufgerufen werden können. Der konkrete Teil beinhaltet Informationen darüber, wo sich die Schnittstellenimplementierung eines bestimmten Serviceanbieters befindet und mit welchen Protokollen die Operationen aufgerufen werden [WSDL (2001)].

UDDI (Universal Description, Discovery, and Integration) UDDI ist ein Registratur- und Verzeichnisdienst mit den dazugehörigen XML-Schnittstellen zum Veröffentlichen und Auffinden der Web Services. Die zentrale UDDI-Komponente ist das Business Register, an dem Web Services angemeldet werden können. Konzeptionell teilen sich die in einem UDDI-Register gespeicherten Informationen in White Pages (Firmenkontaktinformationen), Yellow Pages (Klassifikation nach Branchen) und Green Pages (Technische Daten) auf [UDDI (2004)].

SOAP Die meisten Webservices benutzen zum Nachrichtenaustausch das SOAP Protokoll, das über HTTP oder SMTP transportiert wird. SOAP beschreibt das XML-basierte Nachrichtenformat der Kommunikation und dessen Einbettung in ein Transportprotokoll. Ein SOAP-Dokument besteht aus einem optionalen Header-Element und einem obligatorischen Hauptteil, dem Body-Element. Im Header stehen Informationen für Nachrichtenrouting sowie Kontextinformationen der verwendeten Web Services-Erweiterungen. Der Hauptteil enthält die eigentlichen Nutzdaten der Nachricht [SOAP (2007)].

2.2.2 Web Services Eigenschaften

[Erl (2005)] hebt folgende Eigenschaften der Web Services hervor:

- **Lose Kopplung** Jeder Service ist eine unabhängige Entität. Es besteht nur eine minimale Abhängigkeit zwischen den Services: ein Service verfügt über die Adresse und die Schnittstellenbeschreibung seiner Kommunikationspartner. Diese Informationen sind in einem Service Vertrag enthalten.

- **Service Vertrag** Der Service Vertrag kann aus WSDL-Definitionen, XSD-Schema, Policies [WSPolicy (2006)] und anderen begleitenden Dokumenten bestehen.
- **Autonomie** Web Services haben zur Ausführungszeit eine vollständige Kontrolle über die gekapselte Geschäftslogik.
- **Abstraktion** Services verbergen die Implementierungsdetails der Geschäftslogik. Für die Außenwelt ist nur die Funktionalität sichtbar, die im Service Vertrag definiert wird.
- **Wiederverwendbarkeit** Als unabhängige Komponente kann ein Service prinzipiell in weiteren Anwendungen oder als Teil einer Servicekomposition eingesetzt werden. Dadurch kann die Geschäftslogik mit verschiedenem Granularitätsgrad abgebildet werden.
- **Zustandslosigkeit** Web Services sollen das Zustandsmanagement nicht selbst übernehmen, damit sie nach der Verarbeitung einer Anfrage schnell wieder anderen Clients zur Verfügung stehen.

2.2.3 Die zweite Generation von Web Services

Es wird kontinuierlich eine große Anzahl an Web Services Spezifikationen der zweiten Generation entwickelt, um den Einsatz der Web Services in Business Szenarien zu ermöglichen. Die Spezifikationen richten sich an folgende Aspekte:

- Authentifizierung und Autorisierung über die Organisationsgrenzen hinweg (WS-Federation)
- Signieren und Verschlüsseln von SOAP-Nachrichten (WS-Security)
- Garantierte Nachrichtenübertragung (WS-ReliableMessaging)
- Web Services Transaktionen (WS-Transactions)

und viele andere [Erl (2005)].

Die WS-Transactions Spezifikationsfamilie ist der Gegenstand des nachfolgenden Abschnitts.

Die Kernspezifikation der Web Services-Erweiterungen ist WS-Addressing, viele andere Web Services Spezifikationen setzen WS-Addressing implizit voraus. Die Spezifikation führt zwei neue Konzepte ein: die Endpoint-Referenzen sowie zusätzliche Nachrichten-Informationselemente. [WSADD (2004)]

Die Nachrichten-Informationselemente

Das HTTP-Kommunikationsmodell ist auf Request-Response-Szenarien beschränkt. Mit den WS-Addressing Nachrichtenelementen können komplexere Kommunikationsmodelle wie die asynchrone Kommunikation und das Publish-Subscribe-Pattern realisiert werden.

- **To** - Zieladresse in Form der HTTP-Request-URL
- **From** - Absenderadresse, sie wird als Ziel für die Antwortnachricht verwendet, wenn kein Antwort-Endpunkt (ReplyTo) angegeben wird
- **MessageId** - eindeutiger Nachrichtenbezeichner
- **ReplyTo** - Empfänger der Antwortnachricht
- **FaultTo** - Endpunkt, welcher Fehlernachrichten des Servers entgegen nimmt
- **Action** - die aufgerufene Service-Aktivität, z.B. *Register*
- **RelatesTo** - definiert den Kontext für die Nachricht, z.B. die MessageId der empfangenen Nachricht

Die Endpoint-Referenzen

Die Nachrichtenelemente From, ReplyTo und FaultTo sind vom Typ EndpointReferenceType. Die Referenz-Eigenschaft im EndpointReferenceType erweitert die Identität eines Endpunktes um zusätzliche Angaben. Somit können zwei Endpunkt-Referenzen mit gleicher URI und unterschiedlichen Referenz-Eigenschaften unterschiedliche Ressourcen repräsentieren. [Dostal u. a. (2005)]

Listing 2.1: Beispiel WS-Addressing in WS-Coordination

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <wsa:To
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
         http://192.1.1.10:8080/xts/soap/ActivationRequester
      </wsa:To>
      <wsa:Action
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
         http://.../ws/2004/10/wscoor/CreateCoordinationContextResponse
      </wsa:Action>
      <wsa:MessageID
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
         1e76c24e4ce7c26e:-7fa3a430:1186e693d59:-7ff2
      </wsa:MessageID>
      <wsa:From
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
         <wsa:Address>
            http://192.1.1.10:8180/xts/soap/ActivationCoordinator
         </wsa:Address>
         <wsa:ReferenceProperties>
            <example:runMode>Test</example:runMode>
         </wsa:ReferenceProperties>
         <wsa:ReferenceParameters>
            <example:VirtualHostName>
               default_host
            </example:VirtualHostName>
         </wsa:ReferenceParameters>
```

```
28        </wsa:From>
29        <wsa:RelatesTo
30           xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
31           -3ffefef6:4eb:47ca5739:66
32        </wsa:RelatesTo>
33     </soap:Header>
34     <soap:Body>
35        ...
36     </soap:Body>
37  </soap:Envelope>
```

2.3 WS-Transactions Spezifikationen

WS-Transactions Framework ist eine Spezifikationsfamilie. Es besteht aus den Spezifikationen WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity. Die drei Spezifikationen sind OASIS-Standards und liegen zurzeit in der Version 1.1 vor. Sie entstanden aus den früheren Versionen von WS-Coordination und WS-Transactions, welche im Jahr 2002 von einem Konsortium bestehend aus Arjuna Technologies, BEA Systems, Hitachi, IBM, IONA Technologies und Microsoft entwickelt wurden.

Die Spezifikationen definieren im Wesentlichen folgende Themen:

- wie wird eine Transaktion erzeugt, welche Services sind dafür zuständig
- wie werden verschiedene Services in eine gemeinsame Transaktion eingebunden
- wie wird eine Transaktion unter allen teilnehmenden Services in konsistenter Weise beendet

2.3.1 WS-Coordination

WS-Coordination ist ein generisches Plug-in-Framework für Protokolle zur Koordination der Aktivitäten in verteilten Anwendungen [wscoor (2007)]. Die Verwendung von WS-Coordination ist nicht auf das Transaktionsmanagement beschränkt. Sie kann generell immer dann eingesetzt werden, wenn alle Teilnehmer eine bestimmte Nachricht garantiert erhalten müssen, zum Beispiel, um das Security-, Workflow- oder Replikationskontext zu propagieren. Das Framework definiert folgende Aufgaben:

- Instantiierung bzw. Aktivierung eines Koordinatortyps
- Registrierung der Teilnehmer beim Koordinator
- Propagieren der Kontextinformation
- Ausführung des Koordinationsprotokolls

Der Koordinator stellt eine Aggregation von Activation Service, Registration Service und Protokollservices dar. Die Protokollservices werden in den Spezifikationen außerhalb dieser Spezifikation beschrieben. Implementierung des Registration Service ist nach WS-Coordination zwingend erforderlich. Optional kann ein WS-Coordination Koordinator den Activation Service bereitstellen.

ActivationService

Da der ActivationService asynchron aufgerufen wird, definiert WS-Coordination das Interface des ActivationService selbst und das des ActivationRequester.

CreateCoordinationContext
Mit dieser Anfrage fordert eine Anwendung den ActivationService auf, den CoordinationContext zu erzeugen. Die genaue Semantik des Aufrufs wird in der konkreten Koordinatorspezifikation (WS-AT oder WS-BA) definiert. Diese Anfrage hat folgende Struktur:

- CoordinationType - Koordinatortyp der neuen Transaktion
- CurrentContext - mit CurrentContext kann eine Beziehung zwischen der neuen und einer laufenden Activity hergestellt werden; mögliche Anwendungen sind Interposition (2.3.1.1), Protokoll Bridging oder Koordinator-Replikation
- Expires - optionale Angabe der Gültigkeitsdauer des CoordinationContext.
- weitere optionale Erweiterungselemente und -attribute

CreateCoordinationContextResponse
Der ActivationService erzeugt eine neue Activity und gibt den CoordinationContext zurück. Der CoordinationContext enthält:

- den eindeutigen Kontextbezeichner
- die Gültigkeitsdauer des CoordinationContext (optional)
- die Adresse des RegistrationService
- den Koordinatortyp
- weitere protokollbezogene Informationen

RegistrationService

Beim Erzeugen des CoordinationContext wird der RegistrationService kreiert, bzw. aktiviert. Der RegistrationService setzt ebenfalls eine asynchrone Kommunikation voraus. Mit Register/RegisterResponse werden die Endpunkt-Referenzen zwischen dem Koordinator und dem Teilnehmermanager ausgetauscht.

Register

Der Web Service registriert sich beim Koordinator unter Verwendung der Elemente:

- ProtokolIdentifier, z.B. wsba/ParticipantCompletion

- ParticipantProtocolService, z.B. BAParticipantCompletionParticipant

Die Antwortnachricht **RegisterResponse** enthält den

- CoordinatorProtocolService, z.B. BAParticipantCompletionCoordinator

2.3.1.1 Interposition

Erhält der Activation Service eine `CreateCoordinationContext`-Nachricht mit dem `CurrentContext`-Element, wird eine neue Activity als untergeordnete Activity erzeugt. Der importierte CoordinationContext enthält eine Referenz auf den übergeordneten Koordinator, für den der Koordinator der neuen Activity als ein Proxy fungiert.

Mit Interposition kann eine Koordinatorhierarchie beliebiger Tiefe gebildet werden. Jeder Koordinator ist dabei im Verhältnis zu dem übergeordneten Koordinator ein einfacher Teilnehmer; zur untergeordneten Ebene fungiert er als ein normaler Koordinator. Somit können Web Services in einzelne logische Activities gruppiert werden.

Abb. 2.2 zeigt zwei Anwendungen (App1 und App2) mit ihren eigenen Koordinatoren (CoordinatorA und CoordinatorB).

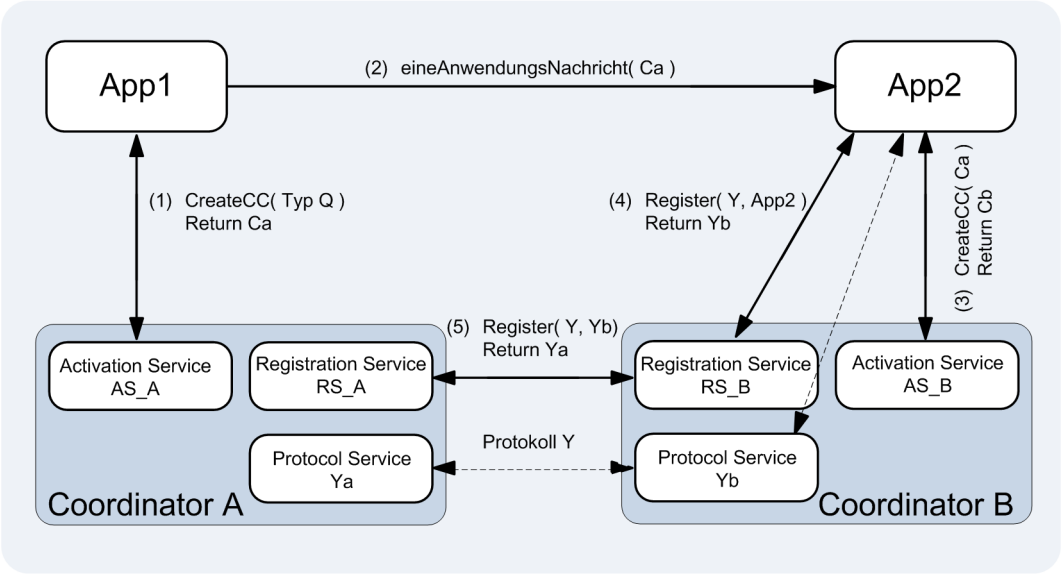

Abbildung 2.2: Interposition aus [wscoor (2007)]

1. App1 schickt eine `CreateCoordinationContext`-Nachricht an den Activation Service AS_A und erhält den Kontext Ca zurück. Ca enthält den Activity-Bezeichner A1, den Koordinatortyp Q und die Adresse des Registration Service RS_A. Der Koordinatortyp Q unterstützt das Protokoll Y.

2. App1 schickt den Kontext Ca mit der ersten Anwendungsnachricht an die Anwendung App2.
3. App2 bevorzugt ihren eigenen Koordinator und schickt `CreateCoordinationContext` mit dem `CurrentContext`-Element = Ca an CoordinatorB. Der vom Activation Service AS_B erzeugte Kontext Cb ist mit Ca identisch, außer, dass er die Adresse des Registration Service RS_B enthält.
4. App2 registriert sich bei CoordinatorB für das Koordinationsprotokoll Y. Die Endpoint-Referenzen zwischen App2 und dem Protokoll Service Yb werden ausgetauscht.
5. CoordinatorB leitet diese Registrierung an den Registration Service RS_A sofort weiter. Die Endpoint-Referenzen zwischen Ya und Yb werden ausgetauscht.

Die letzten beiden Schritte stellen eine logische Verbindung der Endpoint-Referenzen zwischen Ya, Yb und App2 her, die das Protokoll Y verwenden kann.

Mögliche Gründe für die Interposition sind:

- Erhöhte Performance - Es werden nur die Completion-Protokollnachrichten von dem zentralen Koordinator verschickt, der Proxy-Koordinator steuert den Protokollablauf innerhalb der Teilnehmergruppe.
- Sicherheitsüberlegungen - Höhere Vertrauenswürdigkeit des eigenen Koordinators.
- Modularer Aufbau eines Geschäftsprozesses.
- Das Koordinationsprotokoll kann flexibel gewählt werden. Der Proxy-Koordinator kann zum Beispiel Teilnehmer einer Business Activity sein, während er eine lokale Transaktion nach dem WS-AtomicTransaction Protokoll ausführt.

Nicht zu verwechseln ist die Interposition mit geschachtelten Transaktionen. Bei der Interpostion wird die neue Activity mit demselben Transaktionsbezeichner erzeugt wie die übergeordnete Activity. Es werden keine neuen Subtransaktionen erzeugt, folglich kann man nur die gesamte Transaktion abbrechen, bzw. kompensieren.

Eine geschachtelte Activity erhält einen neuen Kontext, der nach oben hin propagiert wird. Einzelne Scopes können unabhängig voneinander ihre Arbeit abbrechen oder beenden.

2.3.2 WS-AtomicTransaction

WS-AtomicTransaction ist ein 2PC-Protokoll für Web Services [wsat (2007)]. Die Spezifikation definiert die Protokolle:

- Completion
- Volatile 2PC
- Durable 2PC

Das Completion-Protokoll

Der WS-AT Initiator registriert sich für das Completion-Protokoll (Abbildung 2.3). Am Ende der Transaktion weist er den Koordinator an, die Transaktion festzuschreiben oder abzubrechen. Daraufhin führt der WS-AT Koordinator das 2PC-Protokoll aus (Abbildung 2.4) und gibt den Status (Committed oder Aborted) an die Anwendung zurück (Abbildung 2.3).

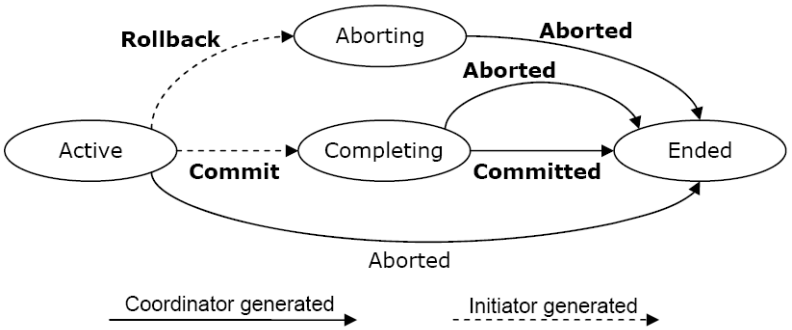

Abbildung 2.3: Completion-Protokoll aus [wsat (2007)]

Das Volatile 2PC-Protokoll

Teilnehmer, die flüchtige Ressourcen verwalten, zum Beispiel ein Cacheregister, registrieren sich für das Volatile 2PC-Protokoll. Erst wenn die Prepare-Phase des Volatile 2PC-Protokolls abgeschlossen ist, wird das Durable 2PC-Protokoll ausgeführt.

Das Durable 2PC-Protokoll

Teilnehmer, die dauerhaft gespeicherte Ressourcen verwalten, registrieren sich für das Durable 2PC-Protokoll.

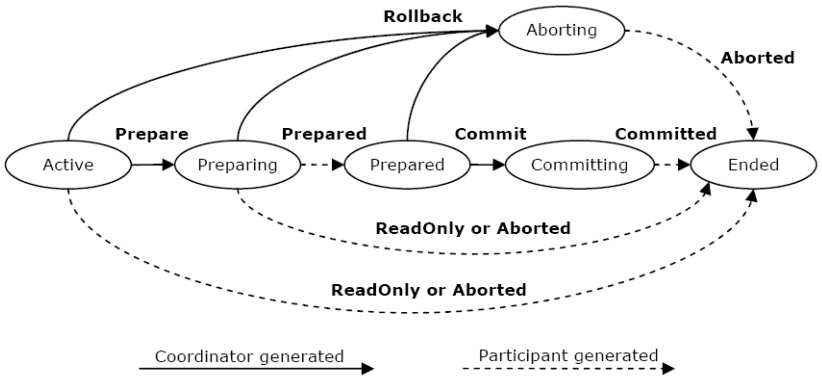

Abbildung 2.4: WS-AtomicTransaction aus [wsat (2007)]

Mit WS-AtomicTransaction können atomare Web Services Transaktionen realisiert werden. In der Spezifikation selbst aber gibt es keine Anforderung an die implementierende Transaktion, eine oder mehrere ACID-Eigenschaften zu erfüllen [Erl (2005)].

Die Autoren der WS-AtomicTransaction Spezifikation empfehlen, das Protokoll nur in solchen Szenarien einzusetzen, bei denen die 2PC-Sperren unproblematisch sind, zum Beispiel:

- in der firmeninternen SOA
- bei kurzlaufenden Transaktionen, die sich über mehrere Organisationen erstrecken, wenn davon ausgegangen werden kann, dass die aufrufende Anwendung die Sperren nicht länger als notwendig halten wird (dies ist zum Beispiel in einem Supply Chain mit einem hohem Maß an Vertrauen zwischen den Partnern der Fall)
- im Finanzsektor, wo traditionell immer ACID-Transaktionen eingesetzt werden

2.3.3 WS-BusinessActivity

WS-BusinessActivity wurde für die Langzeit-Transaktionen in lose gekoppelten Systemen entworfen [wsba (2007)]. Der Spezifikation liegt das Konzept der Kompensation zugrunde. Die WS-BA Protokolle erlauben keinen *rollback*: Teilnehmer behalten die Ressourcen-Sperren nur für die eigentliche Verarbeitungsdauer und schreiben die Ergebnisse sofort fest. Sollte der BA-Initiator beschließen, die Transaktion wieder rückgängig zu machen, werden die nicht beendeten Operationen abgebrochen (Cancel). Die bereits erfolgreich abgeschlossenen Operationen werden logisch wieder rückgängig gemacht, das heißt kompensiert (Compensate).

Anders als bei einer AtomicTransaction „wissen“ jetzt die Services, dass sie Teilnehmer einer Transaktion sind. Es muss zu jeder Web Service-Operation, die im Kontext der Business Activity ausgeführt wird, eine Kompensierungsoperation implementiert werden, da die Entscheidung darüber, wie eine bereits abgeschlossene Operation wieder rückgängig gemacht wird, von der Businesslogik abhängt.

WS-BusinessActivity definiert zwei sehr ähnliche Protokolle:

BusinessAgreementWithParticipantCompletion - Teilnehmer verlässt den Active-Status mit Completed sobald seine Aufgabe innerhalb der Business Activity beendet ist (Abb. 2.5).

BusinessAgreementWithCoordinatorCompletion - Der Koordinator benachrichtigt BACC-Teilnehmer, ihre Arbeit zu terminieren (Complete).

Ein WS-BA Protokoll kann optimiert werden, indem die Teilnehmer den Koordinator über ihren Status unaufgefordert informieren (z.B. mit Exit oder CannotComplete). Die Anwendung kann dann evtl. anhand dieser Zwischenergebnisse die Transaktion vorzeitig beenden.

Ferner unterstützt WS-BusinessActivity zwei Koordinatortypen: AtomicOutcome und MixedOutcome.

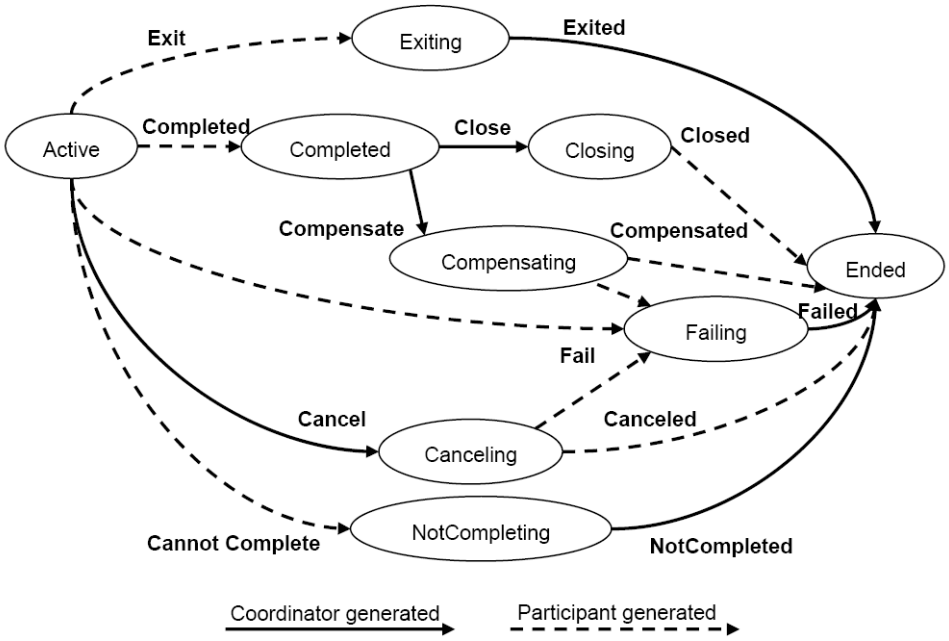

Abbildung 2.5: WS-BusinessActivity aus [wsba (2007)]

AtomicOutcome - Alle Teilnehmer im Status Completed erhalten entweder Close oder Compensate Nachricht vom AtomicOutcome-Koordinator. Jede WS-BA Implementierung muss diesen Koordinatortyp bereitstellen.

MixedOutcome - Optional kann der MixedOutcome-Koordinator implementiert werden, der einige Teilnehmer in den Status Closing während er die anderen in den Status Compensating überführt.

Kapitel 3

Analyse und Entwurf

Um die Tools-Unterstützung von Web Services Transaktionen zu bewerten, wird eine einfache Beispielanwendung mit einem Open Source Produkt, dem XTS Service im JBoss Server, und dem kommerziellen WebSphere AS 6.1 von IBM implementiert und getestet.

Es werden folgende Komponenten für die Evaluierung ausgewählt:

- eine WS-Transactions Implementierung
- eine Beispielanwendung
- eine Reihe von Testkriterien, beschrieben im Abschnitt 3.3
- eine Konfigurationskomponente zur Vorbereitung der Tests
- Logging der Transaktions-Protokollnachrichten

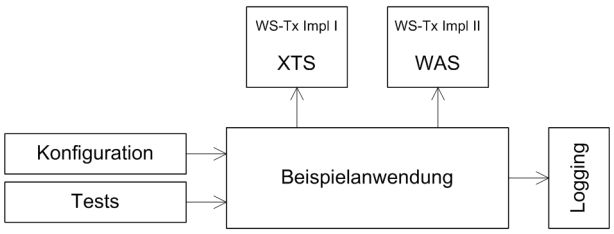

Abbildung 3.1: Komponenten für die Evaluierung

3.1 Use Cases der Beispielanwendung

Für das Szenario der Beispielanwendung soll der Einsatz von Web Services eine sinnvolle Lösung darstellen. Es wird daher ein Umfeld gewählt, das mehrere Unternehmen in einen Geschäftsprozess involviert. Ferner soll der Entwurf es ermöglichen, Teile des Prozesses als Transaktionen zu modellieren.

Das gewählte Referenzszenario beruht auf der Kooperation zwischen einem Einzelhandelsunternehmen, einem Großhändler und seinem Lieferanten, die gemeinsam eine Supply Chain zur Abwicklung der Bestellvorgänge bilden.

[Corsten und Gössinger (2001)] definiert Supply Chain als eine Lieferkette aus mehreren Unternehmen.

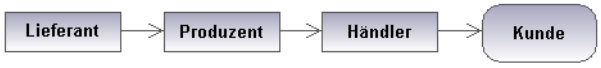

Abbildung 3.2: Supply Chain aus [Corsten und Gössinger (2001)]

Er hebt folgende Kerncharakteristiken der Supply Chain hervor:

- Supply Chain Management ist geschäftsprozessorientiert und zielt auf die optimale Gestaltung der Gesamtprozesse, und zwar unternehmungsübergreifend, ab.
- Es liegt eine kooperative Zusammenarbeit der Teilnehmer vor.
- Ausgangspunkt der Steuerung bildet der Bedarf der Endkunden, ermittelt auf Basis der Daten der Verkaufsstellen.
- Eine der Voraussetzungen für das Supply Chain Management ist die informationstechnische Verknüpfung der Teilnehmer, um so einen durchgängigen Informationsfluss realisieren zu können.

Die Beispielanwendung soll einen Bestellprozess in einer Lieferkette für Autoteile simulieren. Sie basiert auf dem Beispiel in [Ferguson u. a. (2003)]. Die beteiligten Partner sind Autohändler, Autohersteller und Autoteilelieferant, die in folgender Beziehung zu einander stehen:

- Der Autohändler bestellt Autoteile vom Autohersteller.
- Der Autohersteller wiederum erhält Autoteile von seinem Lieferanten, der über zwei eigene Warenlager verfügt und auch Geschäftsbeziehungen mit anderen Autoteileanbietern unterhält.

Ausgangspunkt des Prozessmodells ist eine Übereinkunft zwischen dem Autohändler und dem Autoteilelieferanten, nach der der Lieferant sich verpflichtet, die Herstellerlager rechtzeitig zu beliefern. Der Vorgang soll aus Sicht des Autoherstellers automatisch erfolgen.

Der bemessene Umfang der Bachelorarbeit macht eine reduzierte Darstellung der Beispielanwendung nötig. Die Beispielanwendung soll dazu dienen, die Produktunterstützung der Web Services Transaktionen nach WS-AtomicTransaction und WS-BusinessActivity zu untersuchen, und hat weder Anspruch auf fachliche Korrektheit, noch stellt sie eine einsatzfähige Lösung für automatisierte Bestellprozesse dar. Der Schwerpunkt bei der Implementierung und der Definition der Testfälle wird auf Operationen, die eine transaktionale Unterstützung erfordern, gelegt.

Zunächst folgt eine kurze Übersicht über die verwendeten Anwendungsfälle.

Der Autohändler kann auf die Anwendungen des Autoherstellers zugreifen, um neue Aufträge zu erteilen oder um den Status laufender Aufträge abzufragen.

Der Autoteilelieferant kontrolliert die Lagerbestände beim Autohersteller. Falls ein Minimumbestand unterschritten wird, gibt er im Auftrag des Autoherstellers Bestellungen auf. Dafür ruft sein eigenes Bestellprogramm ein hausinternes Warehouse System auf. Das Warehouse System gibt den Auftrag zunächst an die beiden eigenen Produktionsstätten weiter. Wenn die Bestellung dort nicht erfüllt werden kann, werden Vergleichsangebote von anderen Autoteileanbietern eingeholt.

3.1.1 Autoteile bestellen

Akteure: Auftragsannahme (Web Service), ein Mitarbeiter des Autohändlers.

Ablaufschritte: Der Autohersteller stellt einen Auftragsannahme-Service bereit. Ein Mitarbeiter des Autohändlers greift auf die Anwendungen des Autoherstellers über den Auftragserteilungs-Service zu. Er kann

- neue Aufträge erteilen
- Liste laufender Aufträge abrufen
- den Status eines Auftrages abfragen

3.1.2 Lagerbestände kontrollieren

Akteure: Bestellprogramm, Lagerverwaltung (Web Services)

Ablaufschritte: Der Autoteilelieferant kontrolliert die Lagerbestände beim Autohersteller. Dafür greift sein Bestellprogramm auf den Lagerverwaltung-Service des Autoherstellers zu. Falls ein Minimumbestand unterschritten wird, gibt das Bestellprogramm im Auftrag des Autoherstellers Bestellungen auf (Anwendungsfall 3.1.3). Zusätzlich fordert es eine Auftragsbestätigung vom Autohersteller an (Anwendungsfall 3.1.5).

3.1.3 Bestellungen aufgeben (1)

Akteure: Bestellprogramm, Warehouse-Service, Lagerprogramme I und II (Web Services)

Vorbedingung: Der Minimumbestand für ein Autoteil wurde unterschritten.

Ablaufschritte: Das Bestellprogramm gibt den Auftrag für ein Autoteil an den Warehouse-Service weiter. Der Warehouse-Service teilt den Auftrag gemäß der Firmenpolitik in zwei

Teilaufträge auf und leitet diese intern an die beiden eigenen Produktionslagern weiter. Jedes Lager wird von einem eigenen Lagerprogramm verwaltet. Im Erfolgsfall kann die gewünschte Menge geliefert werden.

Alternative Ablaufschritte: Sollte eines der Lager die nötige Menge nicht liefern können, bricht der Warehouse-Service den Bestellvorgang ab und gibt an den Aufrufer (in diesem Fall an das interne Bestellprogramm) eine Fehlermeldung.

3.1.4 Bestellungen aufgeben (2)

Akteure: Bestellprogramm, Warehouse-Service (Web Services)

Vorbedingung: Der Bestellvorgang aus 3.1.3 ist fehlgeschlagen.

Ablaufschritte: Nach der Benutzerbestätigung ruft das Bestellprogramm seinen Warehouse Service erneut auf. Dieser startet mehrere Anfragen an die externen Autoteileanbieter. Im Erfolgsfall kann mindestens ein Partner den Auftrag erfüllen.

Ausnahmen: Im Fehlerfall terminiert das Bestellprogramm den gesamten Prozess, da der Auftrag nicht weiter ausgeführt werden kann. Der Lagerverwaltung Service des Herstellers markiert das betreffende Autoteil als „nicht lieferbar“.

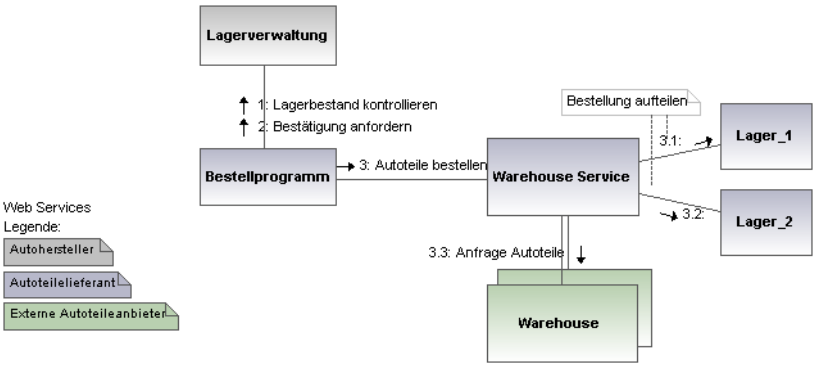

Abbildung 3.3: Web Services Kommunikation

3.1.5 Auftragsbestätigung anfordern

Akteure: Lagerverwaltung, Bestellprogramm, Warehouse-Service (Web Services)

Ablaufschritte: Nach der getroffenen Vereinbarung soll jede im Namen des Autoherstellers aufgegebene Bestellung von einem seiner Mitarbeiter explizit bestätigt (ggf. abgelehnt) werden. Um eine rechtzeitige Lieferung zu gewährleisten, gibt der Lieferant den Bestellauftrag auf, sobald der Lagerbestand unterhalb des Minimums liegt (siehe Anwendungsfall 3.1.2),

ohne die Bestätigung abzuwarten. Im Erfolgsfall bestätigt der Autohersteller die Autoteilebestellung.

Ausnahmen: Falls der Auftrag abgelehnt wird, wird der gesamte Bestellprozess abgebrochen. Die in den lokalen Produktionslagern bestätigten Mengen werden zurück gebucht.

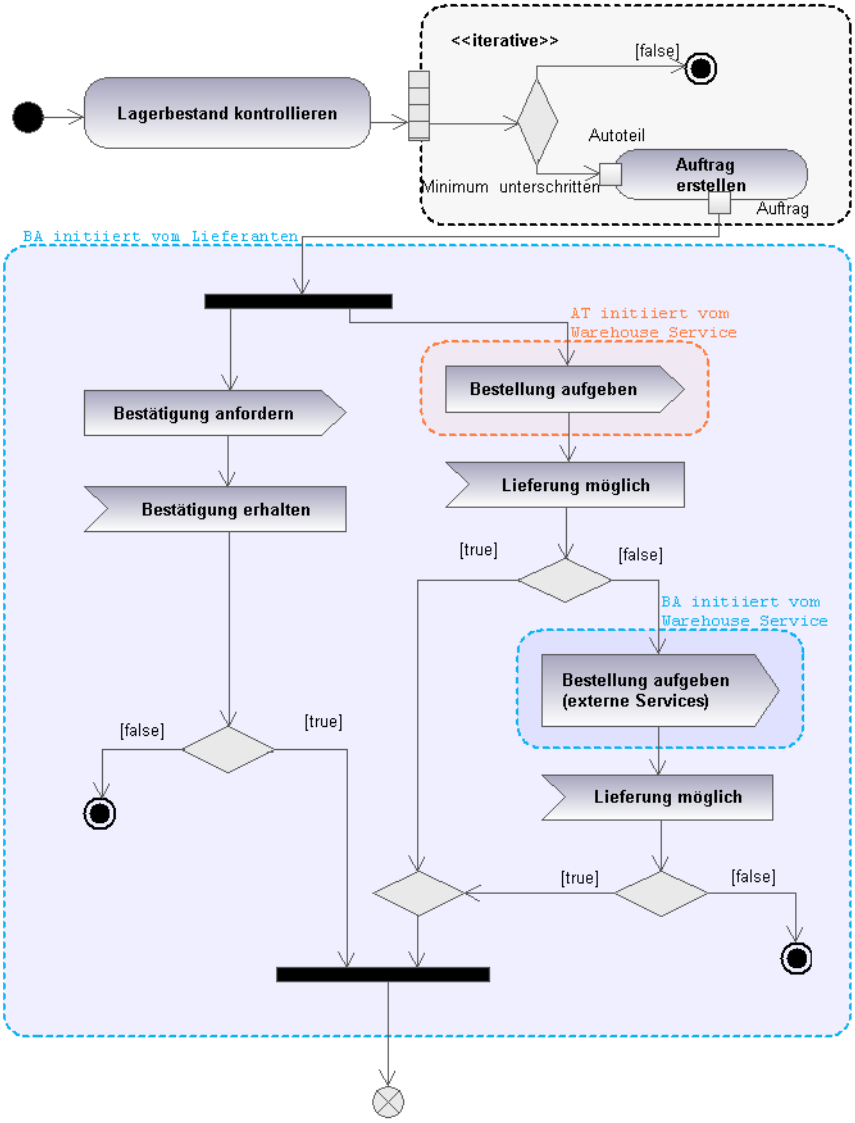

Abbildung 3.4: Bestellprozess

3.2 Entwurf der Beispielanwendung

Der nachfolgende Entwurf der Beispielanwendung ist nicht auf ein verwendetes Produkt bezogen und richtet sich konzeptionell ausschließlich an die Spezifikationen. Er ist daher mög-

licherweise unvollständig und wird bei der Implementierung an das gewählte Transactions Framework angepasst.

3.2.1 Koordinator

Die zentrale Transaktionskomponente ist der **Koordinator**. Der Koordinator ist eine externe Komponente und wird von einem gewählten Transaction Service bereit gestellt. Die Abbildung 3.5 zeigt mögliche Interfaces des Coordination Service. Die Operationsnamen orientiert sich an den WS-Transactions Spezifikationen.

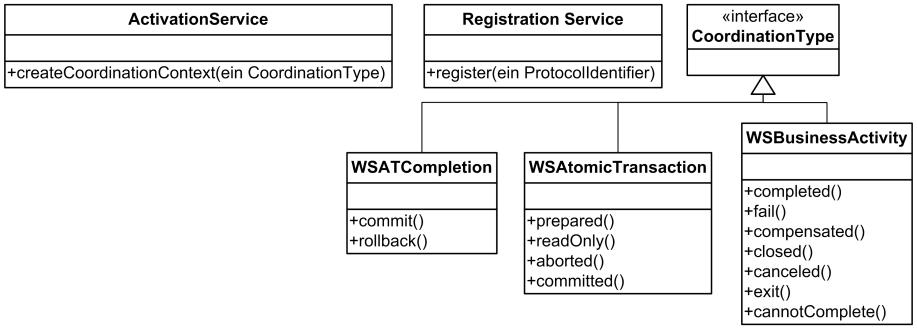

Abbildung 3.5: Koordinator

3.2.2 Transaktionsteilnehmer

Die transaktionalen Web Services sind die Lager-Services, der Autohersteller-Service und der Warehouse-Service.

Die Autohersteller Service und der Warehouse Service implementieren das Business Activity Participant-Interface. Außerdem initiiert der Warehouse Service während der Prozessausführung eine Atomic Transaction und muss daher das Completion-Protokoll unterstützen (Abbildung 3.6).

Die Lager-Services implementieren das Atomic Transaction Participant-Interface (Abbildung 3.7).

3.2.3 Web Services Client

Das Bestellprogramm des Autoteilelieferanten initiiert den Bestellprozess in der Abbildung 3.4 und steuert die Business Activity. Das Programm kann selbst ein Web Service sein oder in einer Prozessbeschreibungssprache, z.B. BPEL implementiert werden.

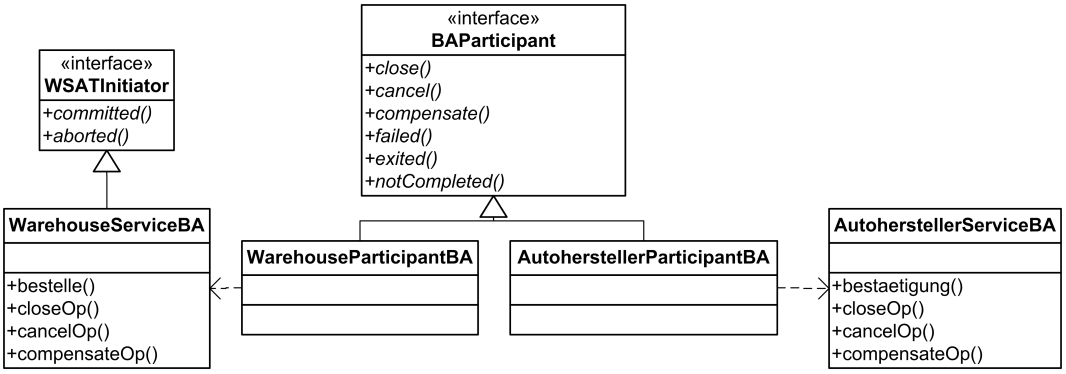

Abbildung 3.6: Die Business Activity-Teilnehmer

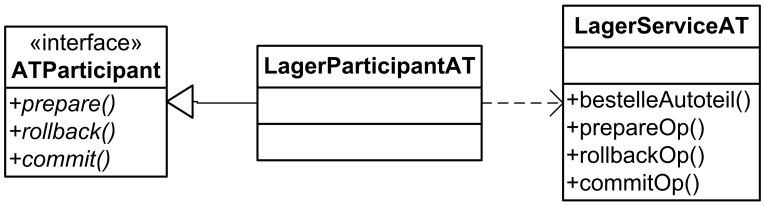

Abbildung 3.7: Der Atomic Transaction-Teilnehmer

Das nachfolgende Komponentendiagramm zeigt die oben beschriebenen Komponenten und ihre Abhängigkeiten untereinander zur Entwurfszeit.

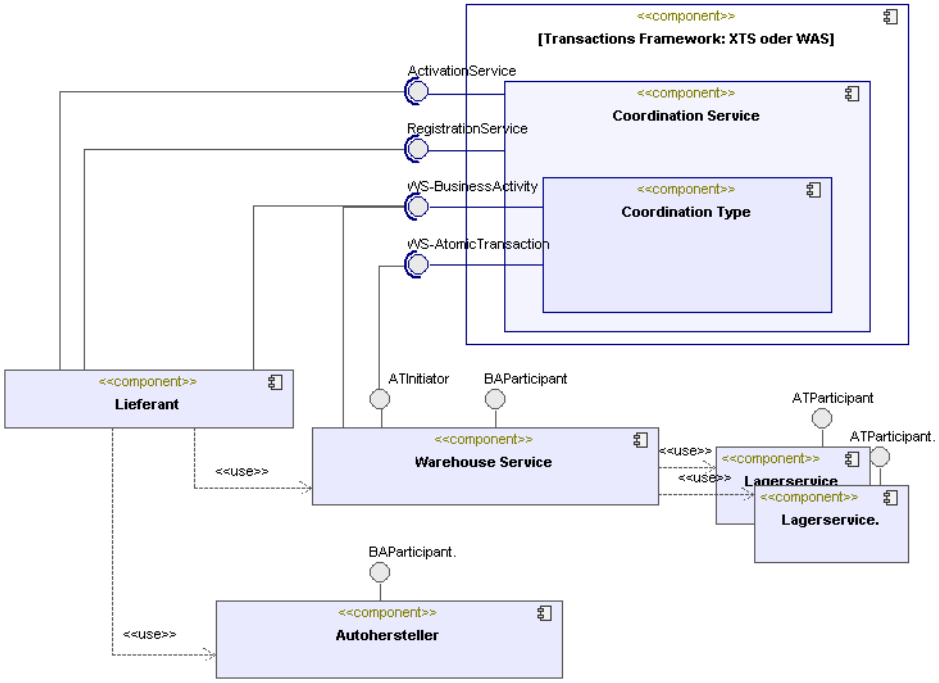

Abbildung 3.8: UML Komponentendiagramm

3.3 Testkriterien

3.3.1 Einhaltung der Spezifikationen

Die ausgetauschten SOAP-Nachrichten zur Kontexterzeugung und Teilnehmerregistrierung sowie die Protokollnachrichten zur Transaktionsabwicklung müssen die in WS-Transactions spezifizierten Signaturen (2.3.1, 2.3.2, 2.3.3) tragen. Der verwendete Koordinatortyp muss durch einen in der jeweiligen Spezifikation angegebenen XML-Namespace angegeben werden.

3.3.2 ACID-Transaktionen

Die Bestellung im Anwendungsfall 3.1.3 wird von einem WS-AtomicTransaction Koordinator gesteuert und soll atomar ausgeführt werden: wenn eines der beiden Lagerhäuser seinen Teilauftrag nicht erfüllen kann, muss der Koordinator die Transaktion mit `Rollback` abbrechen. In diesem Fall dürfen die Autoteilebestände in beiden Lager-Datenbanken durch die Transaktion nicht verändert werden.

3.3.3 Business-Transaktionen

Der Bestellprozess in Abbildung 3.4 wird als eine Business Activity ausgeführt. eine Bestellung soll nur dann erfolgreich abgeschlossen werden, wenn das Autoteil lieferbar ist und der Autohersteller den Auftrag bestätigt.

WS-BusinessActivity unterstützt zwei Koordinatortypen: den AtomicOutcome und - optional - den MixedOutcome, sowie zwei Koordinationsprotokolle: BusinessAgreementWithParticipantCompletion und BusinessAgreementWithCoordinatorCompletion. Jedes Protokoll kann mit einem der beiden Koordinatortypen verwendet werden. Es soll überprüft werden, welche dieser vier Kombinationen mit dem gewählten Transactions Framework implementiert werden können.

Koordinatortyp
Ein AtomicOutcome-Koordinator schickt dieselbe Nachricht - `Close` oder `Compensate` - an alle Teilnehmer, um die Transaktion zu beenden.

Im Falle des MixedOutcome kann die Lieferantenanwendung entscheiden, nur die eigentliche Bestellung zu kompensieren, nachdem beide Teilnehmer mit `Completed` geantwortet haben. Der Autohersteller Service soll dann eine `Close`-Nachricht und der Warehouse Service - eine `Compensate`-Nachricht vom Koordinator erhalten.

WS-BusinessActivity schreibt kein Schema vor, nach dem eine Anwendung dem Koordinator mitteilen kann, welcher BA-Teilnehmer welche Nachricht erhalten soll. Eine MixedOutcome-Implementierung ist also produktspezifisch.

Koordinationsprotokolle
Es wird das BusinessAgreementWithParticipantCompletion-Protokoll im Bestellprozess in Abbildung 3.4 verwendet, da beide Teilnehmer wissen, wann ihre Arbeit beendet ist und diese dann mit `Completed` abschließen.

Im BusinessAgreementWithCoordinatorCompletion-Protokoll wartet der Teilnehmer auf die `Complete`-Nachricht des Koordinators, um seine Arbeit zu beenden. Der weitere Protokollablauf ist derselbe wie in BusinessAgreementWithParticipantCompletion. Es soll leicht möglich sein, die Beispielanwendung anzupassen, um das BusinessAgreementWithCoordinatorCompletion-Protokoll zu verwenden, da die beiden Protokolle sich nur geringfügig unterscheiden.

Compensation
Alle Business Activity Teilnehmer in der Beispielanwendung müssen für jede im Kontext der Transaktion ausführende Operation eine Kompensierungsoperation implementieren. Da WS-BusinessActivity keinen Kompensationsmechanismus spezifiziert, bleibt es dem Produkthersteller überlassen, zu bestimmen, wie die Kompensierungsoperationen erzeugt und vom Koordinator gefunden werden sollen, zum Beispiel durch API-Aufrufe oder die Konfiguration in einer XML-Datei.

Teilnehmernachrichten
Anders als in einer Atomic Transaction, in der die gesamte Kommunikation zwischen den Teilnehmern und dem Koordinator stets vom Koordinator initiiert wird, kann ein Business Activity-Teilnehmer eine Nachricht an den Koordinator unaufgefordert übermitteln. Der Anwendungsfall 3.1.4 wird dazu verwendet, um zu testen, ob ein Teilnehmer die Business Activity mit `Exit`, `Fail` oder `CannotComplete` vorzeitig verlassen kann, und zwar ohne dass die Transaktion dadurch abbricht.

Beenden der Business Activity
Zum Beenden einer Transaktion fordert die Client-Anwendung den Koordinator auf, eine entsprechende Protokollnachricht an alle registrierten Transaktionsteilnehmer zu schicken. Dieser Vorgang ist in WS-AtomicTransaction durch das Completion-Protokoll (2.3.2) klar geregelt. Im Anwendungsfall 3.1.3 ist der Warehouse Service für das Completion-Protokoll registriert, da er eine Atomic Transaction initiiert. Er schickt `Commit` oder `Rollback` an den AT-Koordinator. Daraufhin führt der Koordinator das 2PC-Protokoll aus.

Ein ähnlicher Mechanismus für Transaktionsterminierung ist in WS-BusinessActivity nicht vorhanden. Die Lösung, wie ein BA-Initiator das Beenden der Business Activity einleitet, hängt von der WS-Transactions Implementierung ab.

Nested Activities
Es wird getestet, ob weitere Business Activities innerhalb einer Business Activity erzeugt werden können.

3.3.4 Interposition

Es soll untersucht werden, ob das Interpositions-Schema (2.3.1.1) von dem jeweiligen Produkt unterstützt wird, indem ein Transaktionsteilnehmer den eigenen Koordinator als Subkoordinator verwendet. Das Szenario der Beispielanwendung bietet folgende Möglichkeit, die Interposition zu testen. In den Anwendungsfällen 3.1.3 und 3.1.4 ruft der im Kontext einer Business Activity ausführende Web Service (Warehouse Service) weitere Web Services auf und kann den eigenen Koordinator zwischenschalten. Der Proxy-Koordinator stellt gegenüber dem übergeordneten Koordinator einen BA-Teilnehmer dar. Im Anwendungsfall 3.1.3 muss er aber eine WS-AtomicTransaction steuern.

Es stellt sich hierbei die Frage, wie die Operationen eines WS-BusinessActivity Protokolls auf die WS-AtomicTransaction Operationen abgebildet werden können. Der Subkoordinator muss also beide Protokolle beherrschen und er muss WS-AT-Nachrichten in WS-BA-Nachrichten übersetzen können und umgekehrt. Die Spezifikation definiert nicht, wie diese Umwandlung ablaufen soll.

Es soll überprüft werden, ob die Produkte die Interposition der Koordinatoren sowohl vom gleichen Typ als auch von verschiedenen Typen unterstützen.

3.3.5 (Protokoll-)Optimierungen

WS-AtomicTransaction definiert für kurze Transaktionen die Optimierung `ReadOnly`. Teilnehmer, die keine schreibende Operationen im Transaktionsverlauf ausführen, antworten mit `ReadOnly` in der ersten Phase des 2PC-Protokolls und werden von der zweiten Phase ausgeschlossen. Für langlaufende Transaktionen existiert in WS-BusinessActivity die Optimierung `Exit`. Es soll überprüft werden, ob die Produkte neben den hier genannten auch weitere eigene Protokolloptimierungen anbieten.

3.3.6 Trennung von Koordination und Anwendung

Die WS-Transactions Spezifikationen unterstützen den Ansatz von Trennung der Transaktionssteuerung von der Anwendungslogik. Es wird überprüft, ob die Transaktionskomponente eines Web Service getrennt von der Geschäftslogik mit dem ausgewählten Produkt implementiert werden kann, und inwieweit die Anwendung, die die Transaktion steuert, von der Koordination der Transaktion getrennt ist.

Kapitel 4

Implementierung

4.1 JBoss XTS

JBoss ist ein Open Source Middleware-Hersteller. Ab Dezember 2005 bietet JBoss den XML Transaction Service (XTS) für das Transaktionsmanagement von Web Services an [JBossTS (2005)]. Diese Technologie basiert auf der Entwicklung von Arjuna Technologies und HP.

Der XTS-Transaktionsmanager ist eine Komponente in JBossTS zur Koordination der Web Services Transaktionen. JBossTS ist die JTS-Implementierung in JBoss. Es wird im Abschnitt 4.2 auf JTS, den Java Transaction Service in J2EE, kurz eingegangen.

XTS kann als Web Service in den Application Servern JBoss, webMethods Glue 5 (www.webmethods.com) und BEA Web Logic (www.bea.com) installiert werden.

Die Hauptkomponenten einer Anwendung mit XTS sind der Transaktionsmanager, die Web Services, die Transaktionsteilnehmer der Web Services und der Web Services-Client.

4.1.1 Transaction Manager

Der Transaktionsmanager nimmt die zentrale Stelle in der JBossTS-Architektur ein [JBXTS:PG]. Diese Komponente wird nicht direkt vom Anwendungscode verwendet, sondern als ein request/response Web Service ausgeführt. Der Transaktionsmanager startet eine Transaktion und kreiert für jede neue Transaktion einen Koordinator. Der Transaktionskoordinator übernimmt die Kommunikation mit den registrierten Teilnehmern und steuert den Transaktionsablauf.

4.1.2 Servlets

WS-Transactions Protokolle basieren auf der asynchronen Kommunikation. Um eine Anfrage zu beantworten, ruft eine Entität, zum Beispiel ein Transaktionsteilnehmer, Operationen einer anderen Entität, zum Beispiel des Koordinators, auf. Bei diesem Programmiermodell

müssen alle Teilnehmer, Koordinatoren, Clients und Services eine aktive Komponente besitzen, um unaufgeforderte Nachrichten empfangen zu können. In der aktuellen XTS Version wird die aktive Komponente durch die Java Servlet Technologie realisiert. Das Einbinden von Servlets erfolgt für die Entwickler transparent.

4.1.3 Transaktionale Web Services

In WS-Transactions wird ein transaktionaler Web Service durch zwei Rollen realisiert: den Web Service, der die Geschäftslogik implementiert, und den Transaktionsteilnehmer, der an der Abwicklung eines WS-Transactions Protokolls beteiligt ist. XTS unterstützt das Konzept der Trennung von Web Services und Participants.

4.1.4 Participants

Transaktionsteilnehmer werden in XTS nicht direkt als Web Services veröffentlicht. Bei einer eingehenden SOAP-Nachricht des Koordinators (`Commit`, `Close`, `Cancel` etc.) ruft XTS eine entsprechende Teilnehmermethode auf. Zur Implementierung der transaktionalen Web Services bietet XTS eine umfangreiche API an. In diesem Abschnitt werden die für die Teilnehmerimplementierung relevanten Programmierschnittstellen vorgestellt. (Wegen einer besseren Lesbarkeit werden die Klassen ohne Package-Namen angegeben.)

Atomic Transaction-Teilnehmer

Ein transaktionaler Web Service kreiert einen Atomic Transaction-Teilnehmer und registriert ihn für das 2PC-Protokoll mit einer der beiden TransactionManager-Methoden:

- enlistForDurableTwoPhase(Durable2PCParticipant p, String id)
- enlistForVolatileTwoPhase(Volatile2PCParticipant p, String id)

Der registrierte Teilnehmer ist vom Typ

- Durable2PCParticipant oder
- Volatile2PCParticipant

und implementiert folgende Methoden des 2PC-Protokolls (Tabelle 4.1):

XTS API Methode	WS-AT Koordinator-Op
prepare()	Prepare
commit()	Commit
rollback()	Rollback

Tabelle 4.1: XTS-API 2PC-Teilnehmer

Die `prepare()`-Methode gibt eine `Vote`-Instanz zurück (Tabelle 4.2):

Vote	WS-AT Teilnehmer-Op
new ReadOnly()	ReadOnly
new Prepared()	Prepared
new Aborted()	Aborted

Tabelle 4.2: XTS-API die `Vote`-Klassen

Business Activity-Teilnehmer

Zum Registrieren der BA-Teilnehmer gibt es in der Klasse `BusinessActivityManager` die Methoden

- enlistForBusinessAgreementWithCoordinatorCompletion(BusinessAgreementWithCoordinatorCompletionParticipant bacc, String id)
- enlistForBusinessAgreementWithParticipantCompletion(BusinessAgreementWithParticipantCompletionParticipant bapc, String id)

Die registrierten Business Activity-Teilnehmer sind vom Typ

- BusinessAgreementWithParticipantCompletion oder
- BusinessAgreementWithCoordinatorCompletion

und implementieren folgende Methoden für die Call-Back Aufrufe des BA-Koordinators (Tabelle 4.3):

XTS API	WS-BA Koordinator-Op
close()	Close
cancel()	Cancel
compensate()	Compensate
complete()	Complete

Tabelle 4.3: XTS-API WS-BA Teilnehmer_1

Wenn ein Web Service seinen BA-Teilnehmer für die Business Activity registriert, erhält er ein Handle auf den Transaktionskoordinator - eine `BAParticipantManager`-Instanz. Mit den `BAParticipantManager`-Methoden kann eine Nachricht an den WS-BA Koordinator übermittelt werden (Tabelle 4.4).

4.1.5 Client

Mit der XTS Client-API kann ein Web Services-Client eine Transaktion starten und beenden. Im Hintergrund ruft XTS Transaktionsmanageroperationen via SOAP auf.

UserBusinessActivity

XTS API	WS-BA Teilnermer-Op
completed()	Completed
exit()	Exit
fault()	Fault
error()	Fault (V 1.0) CannotComplete (V 1.1)

Tabelle 4.4: XTS-API WS-BA Teilnermer_2

- **`begin()`** startet eine neue Business Activity
- **`cancel()`** die Business Activity-Transaktion wird rückgängig gemacht
- **`close()`** die Business Activity-Transaktion wird normal beendet, vorher wird das BusinessAgreementWithParticipantCompletion-Protokoll ausgeführt
- **`complete()`** die Business Activity-Transaktion wird normal beendet, vorher wird das BusinessAgreementWithCoordinatorCompletion-Protokoll ausgeführt

UserTransaction

- **`begin()`** startet eine neue Atomic Transaction
- **`commit()`** die Atomic Transaction wird normal beendet
- **`rollback()`** die Atomic Transaction wird abgebrochen

4.1.6 Beispielanwendung mit XTS

Die Implementierung der transaktionalen Web Services und des Client mit dem XTS-Framework sieht folgendermaßen aus.

Autohersteller-Service

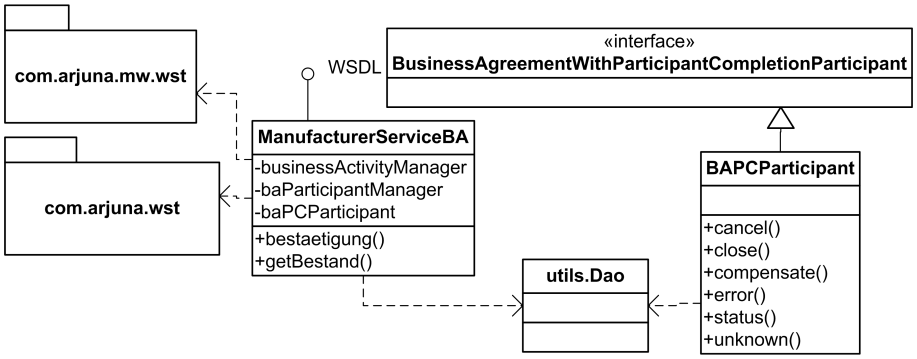

Abbildung 4.1: XTS Autohersteller-Service UML

Der Autohersteller-Web Service wird von der Klasse ManufacturerServiceBA.java implementiert. Der BAPCParticipant wird über eine BusinessActivityManager-Instanz beim BA-Koordinator für das BAPC-Protokoll registriert. Der Teilnehmer verlässt die Transaktion, wenn die BAParticipantManager-Methode completed() oder exit() aufgerufen wird. Der BAPCParticipant trägt den Transaktionsstatus des Autohersteller-Service in die Tabelle KRITISCHE_MENGE ein:

- AT_ID varchar - ID des bestellten Autoteils
- BESTELL_MENGE int - Menge des bestellten Autoteils
- STATUS varchar - bestaetigt, Exit, Cancel, Close, Compensate, Error, unknown
- TX_ID varchar - Es ist möglich in XTS, die Transaktions-ID über die API abzufragen

Warehouse-Service

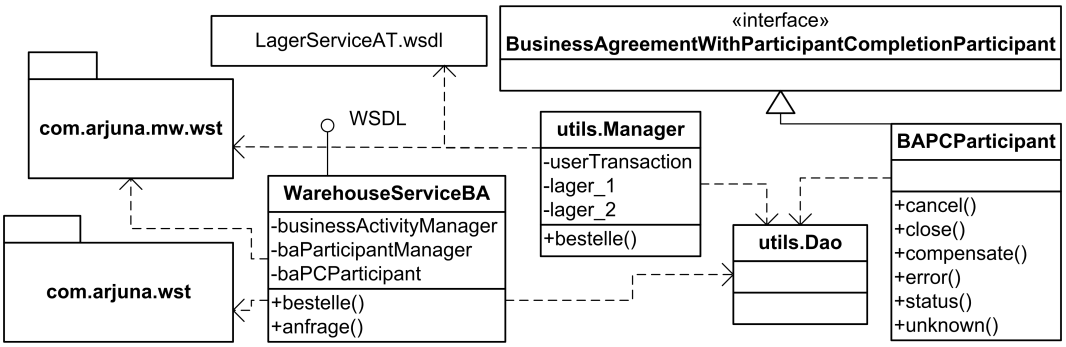

Abbildung 4.2: XTS Warehouse-Service UML

Hier wird der BAPCParticipant in gleicher Weise registriert wie in der Autohersteller-Komponente. Ferner wird eine WS-AtomicTransaction in der Hilfsklasse utils.Manager.java gestartet. Es wird eine UserTransaction-Instanz verwendet, um die Transaktionsgrenzen zu markieren. (Aufrufe begin() und commit()).

Tabelle BESTELLUNG_AT enthält die Einträge über den Transaktionsstatus des Warehouse-Service:

- P_ID varchar - Teilnehmer-ID: L_1001 (Lager_1) oder L_1002 (Lager_2)
- AT_ID varchar - ID des bestellten Autoteils
- MENGE_ALT int - Lagerbestand vor der Transaktion
- MENGE_NEU int - Lagerbestand nach der Transaktion
- STATUS varchar - bestellt, Exit, Cancel, Close, Compensate, unknown
- TX_ID varchar - Transaktions-ID

Lager-Services

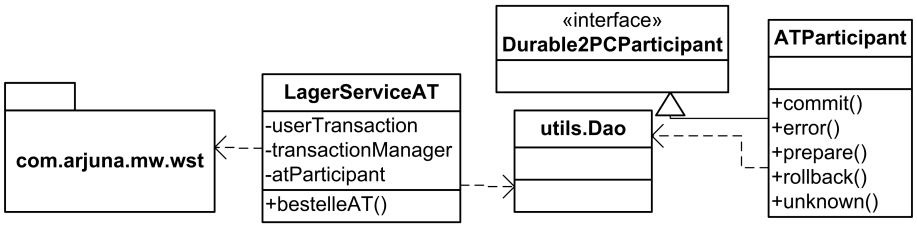

Abbildung 4.3: XTS Lager-Service UML

In der bestelleAT()-Methode wird ein WS-AtomicTransaction Teilnehmer über die TransactionManager-Instanz beim Koordinator registriert. Die Transaktions-ID wird von der UserTransaction abgefragt.

Lieferant

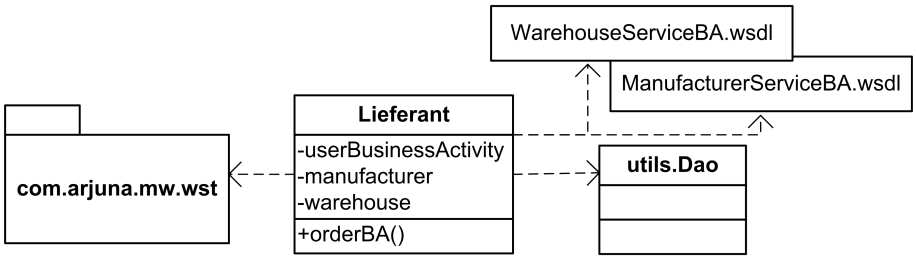

Abbildung 4.4: XTS Lieferant UML

Der Prozessablauf wird in der Methode orderBA() implementiert. Die Klasse UserBusinessActivity wird dazu verwendet, um die Business Activity zu initiieren und beenden. Die Transaktionsergebnisse werden in Tabelle CLOSE oder COMPENSATE eingetragen. Beide Tabellen haben folgende Spalten:

- AT_ID varchar - ID des bestellten Autoteils
- BESTELL_MENGE int - Menge des bestellten Autoteils
- MAN_ID varchar - ID des Autohersteller-Service
- STATUS_M varchar - bestaetigt, nicht bestaetigt
- WH_ID varchar - ID des Warehouse-Service
- STATUS_WH varchar - bestellt, nicht vorraetig
- TX_ID varchar - Transaktions-ID der abgeschlossenen Transaktion

4.2 IBM WAS

Die erste vollständige Implementierung von WS-Transactions im WebSphere Application Server (WAS) von der Firma IBM erschien mit der Version 6.1 im Mai 2006. [IBM-Infocenter (2006)]

WAS ist ein Applikationsserver nach der J2EE-Spezifikation [JSR-244 (2006)]. Transaktionen in J2EE werden vom Java Transaction Service (JTS) verwaltet. Transaktionale Komponenten kommunizieren mit dem JTS über das Java Transaction Interface (JTA). An den Transaktionen sind die Resource Manager beteiligt, die das JTA/XA-Interface implementieren, auch XAResources genannt. (Die XA Spezifikation [OpenGroup (1991)] standardisiert die Kommunikation zwischen dem Transaktionskoordinator und den Ressourcenmanagern in verteilten Transaktionen). J2EE unterscheidet drei Arten von Resource Managern: JDBC Datenbanken, JMS Message Queue Servers und „andere transaktionale Services“. Die letzteren werden über die JCA (J2EE Connector Architecture) angesprochen.

Der Container fungiert als Vermittlungsschicht zwischen einer Komponente (zum Beispiel einer Session Bean) und dem Resource Manager. Bei jeder Update-Operation, fragt der Container den Transaktionsmanager, ob der Aufruf im Kontext einer Transaktion ausgeführt wird. Ist dies der Fall, registriert er den Resource Manager für die Transaktion. Der Transaktionsmanager stellt die Konsistenz der registrierten XAResourcen am Ende einer Transaktion oder nach dem Serverausfall sicher.

Transaktionsdemarkation in J2EE kann entweder vom EJB-Container oder durch den Entwickler über die JTA-Programmierschnittstelle javax.transaction.UserTransaction gesteuert werden. Bei den container-verwalteten Transaktionen wird für jede EJB-Methode ein Transaktionsattribut im Implementierungsdescriptor angegeben:

- **Support** - die aufgerufene Methode läuft in der Transaktion, wenn sie mit einem Kontext aufgerufen wurde
- **Required** - die aufgerufene Methode läuft im Transaktionskontext des Aufrufers, oder sie startet eine neue Transaktion, falls keine existiert
- **RequiresNew** - die aufgerufene Methode läuft immer in der eigenen Transaktion
- **Mandatory** - die aufgerufene Methode muss in der Transaktion des Aufrufers ausgeführt werden und wirft eine Ausnahme, falls sie ohne Transaktionskontext aufgerufen wird
- **NotSupported** - die aufgerufene Methode läuft nie in einer Transaktion
- **Never** - die aufgerufene Methode läuft nie in einer Transaktion und wirft eine Ausnahme, falls sie mit einem Transaktionskontext aufgerufen wird

4.2.1 WS-AtomicTransaction

Die WS-AtomicTransaction Implementierung baut auf JTA auf und kann nur für Web Services, die als Web- oder EJB-Module implementiert werden, aktiviert werden.

Die Unterstützung von WS-AtomicTransaction wird im Implementierungsdescriptor deklariert.

Konfiguration einer Webkomponente

Listing 4.1: WS-AT Konfiguration einer Webkomponente

```
<webappext:WebAppExtension>
   <extendedServlets>
      <globalTransaction xmi:id="GlobalTransaction_1152654822927"
         sendWSAT="true" /> <!-- WS-AT Initiator -->
<!--  <webGlobalTransaction xmi:id="WebGlobalTransaction_1202306983984"
               supportsWSAT="true" />  WS-AT Web Service -->
   </extendedServlets>
</webappext:WebAppExtension>
```

Konfiguration einer Session Bean

Im EJB-Implementierungsdescriptor des WS-AT Initiators muss `sendWSAT=„true“` angegeben werden. Die aufgerufene Operation ist eine container-verwaltete EJB-Methode mit dem Transaktionsattribut „Mandatory“.

Listing 4.2: WS-AT Konfiguration einer EJB

```
<container-transaction>
   <method>
      <ejb-name>Lager_1</ejb-name>
      <method-name>bestelleAutoteil</method-name>
      <method-params>...</method-params>
   </method>
   <trans-attribute>Mandatory</trans-attribute>
</container-transaction>
```

Der WS-AtomicTransaction CoordinationContext wird implizit mit allen ausgehenden Nachrichten mitgeschickt. Auf der Empfängerseite registriert die WAS-Laufzeitumgebung den Web Service automatisch für die WS-AtomicTransaction. Bei der Registrierung und beim Transaktionsmanagement von WS-AT Teilnehmern und XAResources kommt derselbe Mechanismus zur Anwendung.

4.2.2 WS-BusinessActivity

Die Unterstützung von Business Activities in WAS ist nur für EJBs - und EJB-Web Services - mit container-verwalteten Transaktionen gegeben.

Der Scope einer Business Activity entspricht dem Scope einer UOW in WAS. (UOW = unit of work, zum Beispiel eine globale Transaction oder eine Activity Session). Für jeden WS-BA Teilnehmer muss eine CompensationHandler-Klasse im EJB-Implementierungsdescriptor angegeben werden.

Listing 4.3: WS-BA Konfiguration einer EJB

```
<compensationEJBJarExtension>
      <compensationBeanPolicies
         xmi:id="CompensationBeanPolicy_1202585788171"
         compensationKind="Required">
         <compensationHandler
            xmi:id="CompensationHandler_1202585788171"
            className="sc.compensation.Compensate" />
         <enterpriseBean xmi:type="ejb:Session"
            href="META-INF/ejb-jar.xml#Supplier" />
      </compensationBeanPolicies>
   </compensationEJBJarExtension>
```

4.2.2.1 WS-BusinessActivity Programmiermodel

Für WS-BusinessActivity stellt WAS im Package com.ibm.websphere.wsba zwei Interfaces zur Verfügung: UserBusinessActivity und CompensationHandler.

`UserBusinessActivity`

- **`setCompensationDataAtCommit(commonj.sdo.DataObject compensationData)`**
  Beim ersten Aufruf assoziiert diese Methode den im Implementierungsdescriptor konfigurierten CompensationHandler mit der BusinessActivity und übergibt ein DataObject-Objekt an den CompensationHandler. Wenn die BusinessActivity innerhalb einer globalen Transaktion ausgeführt wird, wird der CompensationHandler erst am Ende der Transaktion aktiviert, und zwar nur, wenn die globale Transaktion erfolgreich beendet wurde. Beim wiederholten Aufruf wird kein neuer CompensationHandler erzeugt, sondern das Compensation-Objekt wird überschrieben. Wenn die Methode mit `null` aufgerufen wird, wird der CompensationHandler aus der Business Activity entfernt, der WS-BA Teilnehmer verlässt die Transaktion mit `Exit`.

- **`setCompensationDataImmediate(commonj.sdo.DataObject compensationData)`** Mit diesem Methodenaufruf wird der CompensationHandler sofort aktiviert. Es wird gleich am Ende der Business Activity `close()` oder `compensate()` ausführt. Somit können geschachtelte Activities ihre Arbeit unabhängig von einander abbrechen oder beenden. Die gesamte Transaktion wird weiterhin normal ausgeführt. Es ist möglich, beide Methoden in einer Business Activity aufzurufen, um zwei CompensationHandler zu erzeugen.

- **`setCompensateOnly()`** Die Business Activity wird aufgefordert, alle erfolgreich ausgeführten Operationen zu kompensieren.

`CompensationHandler`

- **`close(commonj.sdo.DataObject compensationData)`**
  Die close()-Methode wird bei einer erfolgreich ausgeführten Business Activity aufgerufen.
- **`compensate(commonj.sdo.DataObject compensationData)`**
  Hier werden die Kompensierungsoperationen für die Business Activity programmiert.

Beide CompensationHandler-Methoden können folgende Ausnahmen auslösen:

- **`CompensationHandlerFailedException`**
  Diese Ausnahme tritt bei einem nicht zu behebenden Fehler auf.
- **`RetryCompensationHandlerException`**
  Der CompensationHandler signalisiert, dass die Operation erst bei einem späteren Aufruf erfolgreich ausgeführt werden kann.

Die Methoden `close()` und `compensate()` bekommen ein Service Data Object (SDO) über das UserBusinessActivity-Interface übergeben, welches alle nötigen Informationen zum Beenden der Business Activity enthält. Zum Beispiel die ID einer Tabellenzeile, die durch die Kompensierungsoperation wieder entfernt werden muss.

4.2.2.2 SDO

Die Service Data Objects-Spezifikation definiert ein herstellerunabhängiges Framework zum einheitlichen Datenzugriff [IBM und BEA (2003)]. Ein SDO ist ein Transportobjekt, das Daten für einen (Remote-)Methodenaufruf zu einem Datengraph bündelt. Die SDO-Daten können aus mehreren heterogenen Datenquellen stammen, wie RDBMS, XML Dokumenten, Web Services oder Enterprise Information Systems.

Der SDO-Zugriff erfolgt über einen Data Mediator Service, der die Daten aus der Datenquelle in einen Datengraph lädt, und sie wieder zurückschreibt. SDO verwendet das Prinzip entkoppelter Datengraphen (disconnected data graphs), bei dem die Daten offline manipuliert werden, eventuelle Kollisionen werden erst beim Update aufgedeckt.

4.2.3 Beispielanwendung mit WAS

Bei dem hier vorgestellten Programmiermodell für Business Activities muss auch der BA-Initiator (hier: der Lieferanten-Service), in die Business Activity als Teilnehmer eingebunden werden. Um die Operationen der drei Web Services – des Autohersteller-, des Warehouse-

und des Lieferanten-Service – in einer Business Activity ausführen zu können, sind folgende Schritte notwendig (Abbildungen 4.5, 4.6 und 4.7):

- es muss eine CompensationHandler-Klasse implementiert und im Implementierungsdescriptor (ejb-jar.xml) eingetragen werden
- die im WS-BA Kontext aufgerufene Methode erhält das Transaktionsattribut „Required“

Zur Laufzeit rufen die Services eine UserBusinessActivity-Instanz auf, um den CompensationHandler zu aktivieren, oder um die Kompensierung einzuleiten.

Der Warehouse-Service initiiert eine WS-AtomicTransaction, die zwischen den Lager-Transaktionsteilnehmern ausgeführt wird (Abbildungen 4.7 und 4.8). Es ist keine zusätzliche Implementierung für WS-AtomicTransactions in WAS notwendig.

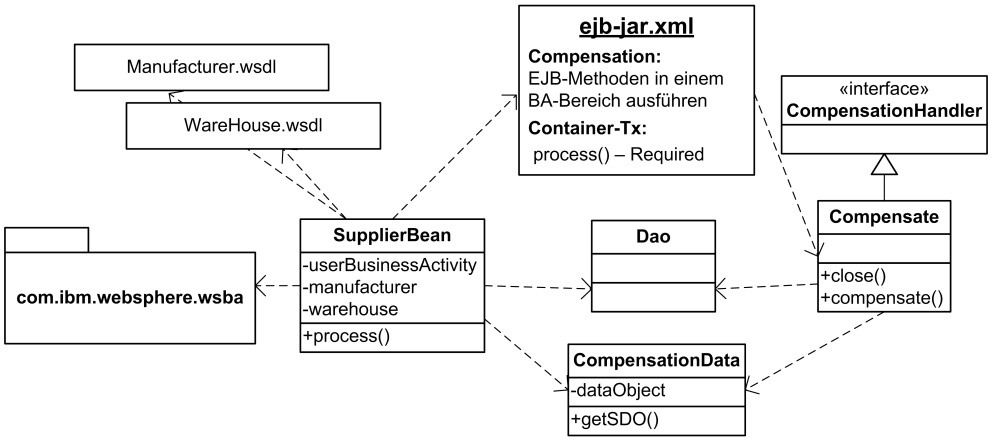

Abbildung 4.5: WAS Lieferanten-Service UML

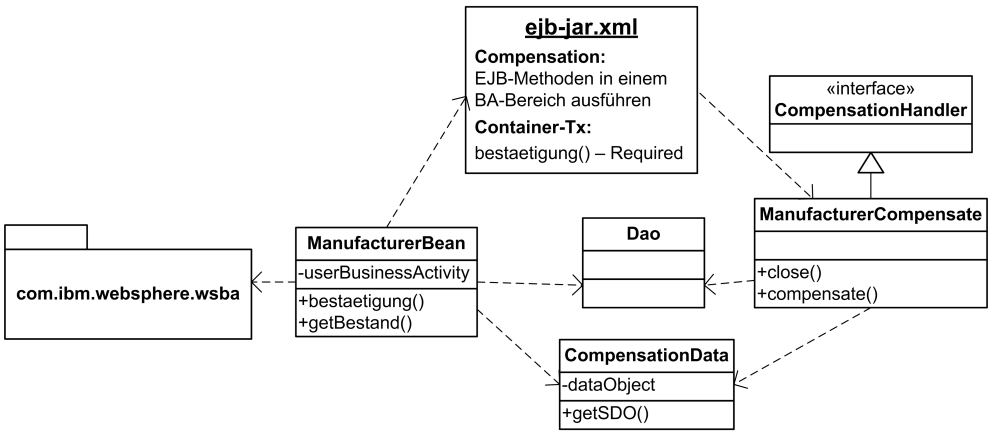

Abbildung 4.6: WAS Autohersteller-Service UML

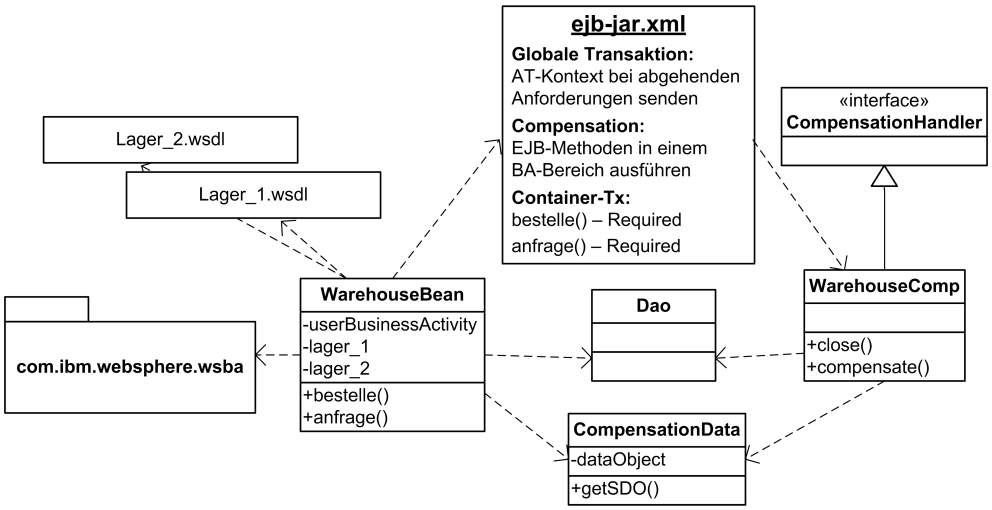

Abbildung 4.7: WAS Warehouse-Service UML

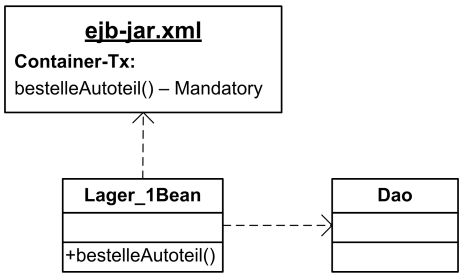

Abbildung 4.8: WAS Lager-Service UML

Kapitel 5

Evaluierung

Der nachfolgenden Produktevaluierung liegen die Testkriterien, die im Abschnitt 3.3 dargestellt wurden, sowie die im praktischen Teil der Arbeit durchgeführten Tests zu Grunde.

5.1 Einhaltung der Spezifikationen

Beide Produkte implementieren WS-Transactions Vesion 1.0

- die ausgetauschten Protokollnachrichten verwenden den geforderten Namespace http://schemas.xmlsoap.org/ws/2004/10
- der CoordinationContext hat die Elemente Identifier, CoordinationType (/wsat, bzw. /wsba/AtomicOutcome), RegistrationService
- die Register-Nachrichten enthalten die Elemente ProtocolIdentifier und Participant-ProtocolService

5.2 ACID-Transaktionen

Die CreateCoordinationContext-Anfrage sowie die Registrierungen in XTS erfolgen über die SOAP-Nachrichten, wie es in der WS-AT Spezifikation vorgeschlagen wird: der Transaktionsinitiator kreiert den CoordinationContext und registriert sich für das Completion-Protokoll beim ATCompletionCoordinator. Die WS-AT-Koordination findet zwischen dem ATCoordinator und den ATParticipant's statt. (Test A.6)

WAS nimmt an dieser Stelle eine Optimierung vor. Es findet kein separater Register-Aufruf statt, wenn zwei WAS-Server miteinander kommunizieren. Die Kontexterzeugung und die Registrierung erfolgt durch die WebSphere Laufzeitumgebung [IBM-Infocenter (2006)]. Es werden auch keine WS-AT Protokollnachrichten ausgetauscht, wenn die Teilnehmer auf einem WebSphere-Server installiert sind. (CD_01 B)

Um an einer WS-AT teilzunehmen, muss der Web Service in XTS einen [Durable|Volatile] 2PCParticipant registrieren, der auch die Methode `rollback()` implementiert. Somit kann in XTS - im Gegesatz zu WAS - auch eine nicht-XAResource am 2PC-Protokoll teilnehmen. (4.2.1) Des weiteren kann der WS-AT Teilnehmer in XTS selbst entscheiden, ob er seine Arbeit beim Rollback-Aufruf tatsächlich wieder rückgängig macht. Die Spezifikation fordert keine ACID-Eigenschaften von einer WS-AT.

5.3 Business-Transaktionen

Koordinatortyp
Die Beispielanwendung verwendet den AtomicOutcome-Koordinator. Der Koordinatortyp MixedOutcome ist nach der WS-BA Spezifikation optional und wird zum jetzigen Zeitpunkt von beiden Produkten nicht implementiert.

Koordinationsprotokolle
Eine WS-BA kann nach dem BusinessAgreementWithParticipantCompletion- (BAPC) oder BusinessAgreementWithCoordinatorCompletion- (BACC) Protokoll ablaufen. XTS stellt für jedes Protokoll ein Participant-Interface bereit [4.1.4]. Es ist auch möglich, BAPC- und BACC-Teilnehmer in eine gemeinsame WS-BA einzubinden, wie es im Test A.7 der Fall ist.

WAS implementiert ebenfalls beide Koordinationsprotokolle [IBM-Infocenter (2006)]. Allerdings bietet die Business Activity-API keine Möglichkeit, ein Protokoll auszuwählen. Die WAS-Laufzeitumgebung registriert alle BA-Teilnehmer stillschweigend für das BAPC-Protokoll.

Compensation Ein BA[P|C]C-Teilnehmer in XTS definiert alle notwendigen Kompensierungsoperationen in der vorgeschriebenen compensate()-Methode. Diese wird vom WS-BA Koordinator beim Beenden der BA aufgerufen, wenn der WS-BA-Initiator `Cancel` an den TerminationCoordinator sendet. (Test A.6)

In WAS hat man die Möglichkeit, einen CompensationHandler-*Immediate* und einen CompensationHandler-*AtCommit* für einen BA-Teilnehmer zu definieren (4.2.2.1).

Beide CompensationHandler haben unterschiedliche Semantik. Der letztere wird aufgerufen, wenn der BA-Teilnehmer seine Arbeit erfolgreich beendet hat (`Completed`) und diese beim Transaktionsende kompensiert werden muss. (Tests A.2 und A.4)

Der CompensationHandler-*Immediate* fungiert als Exception Handler für den Teilnehmer: er wird aktiv, sobald ein Fehler beim Ausführen einer Web Service-Operation auftritt und bevor der WS-BA Teilnehmer die Transaktion mit `Exit` verlässt.

Man kann allerdings pro EJB nur eine CompensationHandler-Klasse angeben. Wenn die Anwendung beide CompensationHandler-Typen verwendet, müssen in der `compensate()`-Methode zwei Abläufe implementiert werden, die sich semantisch von einander unterscheiden: es ist ein „undo“ einer erfolgreich abgeschlossenen Arbeitseinheit versus der „try-catch-Klammer“ um die aktuell ausführende Web Service-Operation.

Compensation Fehler

Ein Teilnehmer muss aus dem Zustand „Completed“ in einen der beiden Zustände übergehen:

- Compensating - Ended
- Compensating - Faulting - Ended

Fehler in Compensation werden in XTS durch die FaultedException in der `compensate()`-Methode eines BA-Teilnehmers ausgelöst (Test A.8).

In WAS teilt der CompensationHandler einen Fehler in der Kompensierung durch die CompensationHandlerFailedException mit (Test A.3). Unkorrekterweise übersetzt die Laufzeitumgebung diese Ausnahmen in `Compensated`, so dass an dieser Stelle ein inkonsistentes Ergebnis entsteht: der BAPC-Koordinator registriert eine erfolgreiche Kompensierung, die tatsächlich nicht ausgeführt wurde (CD_02 B).

WS-BA Teilnehmernachrichten
Nachrichten werden in WS-BA nicht nur vom Koordinator, sondern auch von den Teilnehmern initiiert: `Completed`, `Exit`, `Fail` (bzw. `Fault` in der Version 1.0) und `CannotComplete` (erst in der Version 1.1).

Ein Teilnehmer in XTS kann jede dieser Nachrichten durch eine entsprechende BAParticipantManager-Methode an den Koordinator übermitteln. (4.4)

In WAS verlässt der Transaktionsteilnehmer die Business Activity mit `Exit`, entweder in dem er eine java.rmi.RemoteException auslöst, oder durch den Aufruf setCompensationData[Immediate|AtCommit] mit dem null-Parameter. (Tests A.2 und A.4) Es konnte nicht erschlossen werden, wann die Nachricht `Fault` generiert wird.

Nested Activities

Business Activities in WAS entsprechen UOW's und können als solche verschachtelt werden. (Test A.5) Eine verschachtelte Transaktion erhält einen anderen Kontext (Im Test: Ctx_B) als die übergeordnete Transaktion (Ctx_A).

XTS unterstützt die Erzeugung von verschachtelten Business Activities nicht.

Beenden der BA-Transaktion

Zum Beenden der Business Activity registriert sich ein BA-Teilnehmer in XTS am Ende der Transaktion für das Termination-Protokoll und schickt `Close` oder `Cancel` an den TerminationCoordinator. Mit der letzteren Operation wird die Kompensierung angestoßen.

Das Termination-Protokoll baut ebenfalls wie WS-AT und WS-BA auf der WS-Coordination Spezifikation auf (Tests A.6 und A.8).

In WAS kann ebenfalls jeder Teilnehmer die Kompensierung einleiten, und zwar durch den Aufruf uba.setCompensateOnly(). Der Koordinator schickt dann an alle Teilnehmer die `Compensate`-Nachricht. `Close` wird nach dem return der Methode, aus der die WS-BA gestartet wurde, gesendet. (Test A.1)

5.4 Interposition

Interposition wird von beiden Produkten nicht unterstützt.

5.5 Optimierung vs Nachrichten-Overhead

Der 2PC-Protokollablauf mit atomarem Ergebnis ist klar definiert und wird vom WAS-Container komplett übernommen. Der Ansatz, transaktionales Verhalten im Implementierungsdescriptor zu definieren, entlastet die Anwendung vom transaktionsspezifischen Code, hat aber den Nachteil, dass durch das EJB-Attribut `sendWSAT=true` alle ausgehenden Web Service-Nachrichten den CoordinationContext enthalten, was den Umfang der SOAP-Pakete unnötigerweise vergrößert. (CD_03 B)

Bei WS-BA entsteht ebenfalls ein Nachrichten-Overhead. Hier erfolgt die Konfiguration auf der Methoden-Ebene mit dem Resultat, dass alle aus einer container-verwalteten Methode heraus gesendeten Nachrichten den WS-BA Kontext im Header mittragen. In der Beispielanwendung fragt der Lieferanten-Service den Lagerbestand beim Autohersteller ab, ehe der Bestellprozess gestartet wird. Bereits diese SOAP-Nachricht enthält den WS-BA CoordinationContext. (Auf der Empfängerseite wird der Kontext ignoriert.) (CD_04 B) Der Warehouse-Service bindet sogar beide Kontexte - WS-AT und WS-BA - in die ausgehenden Nachrichten ein (CD_05 B).

Ein solches Verhalten entsteht in XTS nicht. Sollte ein Teilnehmer während der WS-BA oder WS-AT einige Arbeitsschritte außerhalb des Transaktionskontextes ausführen, kann er sich mit `suspend()` von der Transaktion abmelden und mit `resume(TxContext)` seine Teilnahme an der Transaktion fortsetzen.

5.6 Trennung von Koordination und Anwendung

XTS unterstützt die Trennung von Web Services und Participants. Transaktionales Verhalten der Web Services wird getrennt von der Anwendungslogik in Participant-Klassen implementiert (4.1.4). Die XTS-API orientiert sich in ihrer Struktur und in der Methodenbenennung in

großem Maße an die WS-Transactions Spezifikationen. Diese Klarheit hilft zur Entwurfszeit, schnell den Bezug zwischen dem Ablauf der zu implementierenden Anwendung und den Spezifikationen herzustellen.

Der WAS-Container führt WS-AtomicTransactions transparent für die Anwendung aus. Für Business Activities werden nur die Operationen Close und Compensate im Compensation-Handler programmiert.

5.7 Zusammenfassung

In der Tabelle 5.1 werden die Eigenschaften der beiden Produkten zusammengefasst.

Kriterien / Produkt	**XTS**	**WAS 6.1**
Einhaltung der Spezifikationen	Ja	Ja
WS-AT	Ja	Ja
WS-BA AtomicOutcome	Ja	Ja
WS-BA MixedOutcome	Nein	Nein
WS-BA BAPC-Protokoll	Ja	Ja
WS-BA BACC-Protokoll	Ja	Ja, laut der Dokumentation
Compensation	Ja	Ja, Inkonsistenz bei Compensation Faults
WS-BA Teilnehmernachrichten	Ja	Ja, unvollständig
Nested Business Activities	Nein	Ja
Beenden der Business Activity	Ja, Termination-Protokoll	Ja
Interposition	Nein	Nein
Implementierung der transaktionalen Web Services	einfache Java-Klassen	WS-BA: EJB WS-AT: EJB und Servlets
Transactions API	Ja	Begrenzt auf WS-BA Close und Compensate
Unterstützte Plattformen	JBoss, WebLogic, webMethods	WebSphere
Open Source	Ja	Nein, kostenlose 60-Tage Trial-Version

Tabelle 5.1: Vergleich der WS-Transactions Implementierungen

Mit dem WebSphere Application Server 6.1 bietet die Firma IBM ihre erste Implementierung für die Web Services Transaktionen. Für die Produktion dürfte diese Implementierung gegenwärtig wenig geeignet sein, da sie noch unter einigen Fehlern leidet. So wird ein Fehler in der Kompensierung dem BA-Koordinator nicht mitgeteilt (5.3). Beim Ausführen des WS-BA Protokolls löst die WAS- Laufzeitumgebung an verschiedenen Stellen eine ClassNotFoundException aus, wie z.B. beim Aufruf der SOAP-Binding Klasse RegisterResponseTypeBinder während der Registrierung eines BA-Teilnehmers (CD_06 B). Der Fehler ist

nicht verlässlich reproduzierbar und lässt darauf schließen, dass die in WAS implementierten Hilfsklassen nicht immer richtig zur Laufzeit geladen werden.

Was das Konzept für WS-BusinessActivity betrifft, so wäre eine flexiblere Möglichkeit bei der Auswahl einer Web Services Implementierung zu wünschen. Die WS-BusinessActivity wird derzeit nur für EJB's unterstützt. Zum Vergleich: transaktionale Web Services in XTS können lediglich durch einfache Java-Klassen implementiert werden.

Ferner könnte eine Anwendung in WAS besser strukturiert werden, wenn man zwei logisch unterschiedliche CompensationHandler durch zwei verschiedene Klassen implementieren könnte. Derzeit muss man im SDO einen Flag zur Kennzeichnung des CompensationHandler-Types übergeben.

Der XML Transaction Service in JBossTS wurde von der Firma Arjuna Technologies für die Web Services Transaktionen entwickelt. Wie die Firma selbst betont, sind in die Entwicklung von XTS die Erfahrungen aus dem Produkt HP-WST (2002), dem ersten Produkt für Web Services Transaktionen, eingeflossen [Arjuna (2006)]. XTS scheint zurzeit die einzige stabile vollständige Implementierung der WS-Transactions Spezifikationen zu sein, wenn auch bis jetzt einige optionale Spezifikationsanforderungen noch nicht umgesetzt wurden.

Es fehlt die Implementierung des MixedOutcome-Koordinators für die Steuerung einer Business Activity. Eine WS-BA, die vom AtomicOutcome-Koordinator ausgeführt wird, ist im Grunde mit dem 2PC-Protokoll vergleichbar. Am Ende werden alle Teilnehmer, die mit `Completed` geantwortet haben, entweder über den Transaktionserfolg oder den Transaktionsfehler informiert. Mit dem MixedOutcome-Koordinator können erweiterte Anwendungsszenarien realisiert werden. Z.B. kann der BA-Initiator entscheiden, nur bestimmte Ergebnisse von Web Services anzunehmen (`Close`) während er die anderen ablehnt (`Compensate`).

Interposition, wie sie in WS-Coordination vorgeschlagen wird (2.3.1.1), steht ebenfalls auf der Aufgabenliste der XTS-Entwickler. Die Verwendung eines Proxy-Koordinators ist besonders dann von Interesse, wenn ein Transaktionsteilnehmer viele weitere Web Services in die Transaktion einbezieht.

Des Weiteren hat JBoss für die nächsten Releases geplant, neben JBoss, webMethods und WebLogic, auch für weitere SOAP-Plattformen den Transaktionsdienst für Web Services anzubieten.

Kapitel 6

Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit den Implementierungen der WS-Transactions Spezifikationen.

Kapitel 2 führte die für die Web Services-Architekturen relevanten Transaktionskonzepte ein: das ACID-Paradigma, das zwei-Phasen-Commit-Protokoll und die offenen verschachtelten Transaktionen. Es wurde diskutiert, dass es zur Ausführung der langlaufenden Transaktionen oft notwendig ist, die ACID-Eigenschaften zu lockern. Es folgte ein Überblick über die Basis-Spezifikationen für Web Services.

Im letzten Abschnitt des zweiten Kapitels wird das WS-Transactions Framework, bestehend aus WS-Coordination, WS-AtomicTransaction und WS-BusinessActivity, vorgestellt. Des weiteren wurden die Begriffe Interposition und Kompensierung erklärt.

Im Kapitel 3 wurde eine Beispielanwendung beschrieben, die dazu dienen soll, die Produkte XTS (JBoss) und WAS (IBM) hinsichtlich ihrer Unterstützung der Web Services Transaktionen zu testen. Der Entwurf der Beispielanwendung richtete sich konzeptionell zunächst an die WS-Transactions Spezifikationen. Die Vorgehensweisen bei der Implementierung mit WAS und mit XTS wurden in Kapitel 4 beschrieben. Sie unterschieden sich deutlich voneinander.

Anschließend wurde die Beispielanwendung nach den zuvor festgelegten Kriterien getestet. Wie die ausgeführten Tests zeigten, implementieren beide Produkte die WS-Transactions Spezifikationen nicht vollständig. Im Kapitel „Evaluierung“ werden die Testergebnisse dargestellt.

Die vollständige Konformität der Implementierungen zur Spezifikation konnte aus Zeitgründen nicht untersucht werden und bleibt einer Anschlussarbeit vorbehalten. Ferner dürfte es von Interesse sein, im Rahmen einer weiteren Arbeit, die Interoperabilität zwischen verschiedenen WS-Transactions Implementierungen zu testen. [W3C (2004)] beschreibt, wie eine WS-AtomicTransaction mit .NET WCF und IBM WAS realisiert werden kann. Für WS-BusinessActivity fehlt ein solches Beispiel an dieser Stelle, da .NET die Business Activity-Transaktionen nicht unterstützt.

In der hier vorgestellten Beispielanwendung wird der Ablauf der gesamten Business Activity in einer Client-Methode bestimmt. Es wäre denkbar, die Anwendungslogik als einen BPEL-Prozess zu modellieren. Die Business Process Execution Language (WS-BPEL) [bpel (2007)] ist eine XML-basierte Sprache zur Orchestrierung von Web Services. Für die langlaufenden Transaktionen definiert BPEL einen Fehlerbehandlungsmechanismus, den CompensationHandler, der sich allerdings nur lokal auf eine BPEL-Prozess-Instanz anwenden lässt. Kombiniert mit WS-BusinessActivity, können mit BPEL langlaufende Transaktionen in verteilten Web Services Anwendungen modelliert werden.

Anhang A

Tests-Beschreibung

A.1 Test WAS 1

Verwendete Anwendungsfälle sind 3.1.2 „Lagerbestände kontrollieren“, 3.1.3 „Bestellungen aufgeben (1)“ und 3.1.5 „Auftragsbestätigung anfordern“. Die Menge des Autoteils ’CA1054IR’ wird in der Autohersteller-Tabelle auf 3 reduziert:
MAN.LAGERBESTAND
[AT_ ID, BESTAND]
CA1054IR 3

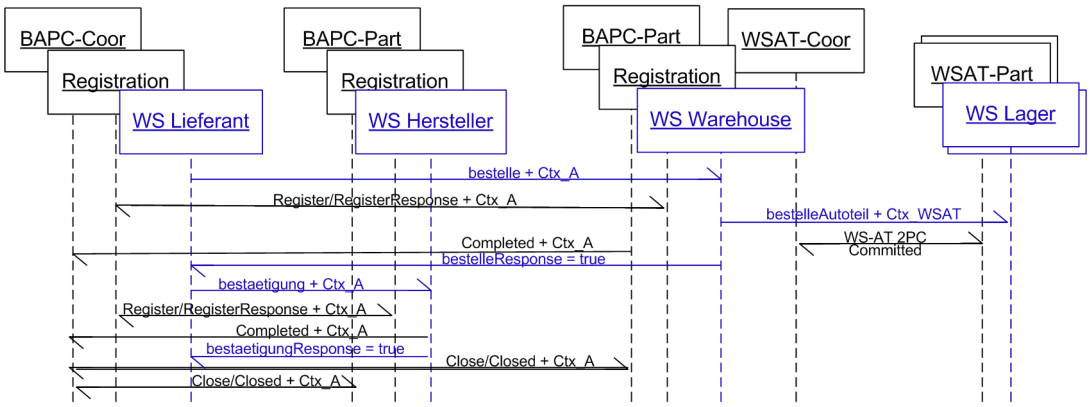

Abbildung A.1: Test WAS 1

CoordinationContext
Der Aufruf der Operationen `WarehouseService.bestelle()` und `Manufacturer.bestaetigung()` erfolgt im Kontext einer WS-BusinessActivity. Es wird derselbse Transaktionskontext unter allen Teilnehmern propagiert.

Der Warehouse-Service bindet den WS-AtomicTransaction Kontext in die ausgehenden `bestelleAutoteil`-Nachrichten ein und gibt den WS-AtomicTransaction Koordinator-

typ an. Somit werden die beiden Operationen in einer gemeinsamen WS-AtomicTransaction ausgeführt.

WS-BA Terminierung
Der Prozess wird ohne Fehler beendet. Der Warehouse Service und der Autohersteller Service schicken `Completed`-Nachricht an den WS-BusinessActivity Koordinator.

Das Protokoll wird durch den Austausch von `Close`- und `Closed`-Nachrichten beendet.

Teilnehmersicht
Das Ergebnis der Business Activity wird durch die `CompensationHandler.close()`-Operationen in die lokalen Tabellen eingetragen:

LIEF.CLOSED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]
CA1054IR manuf01 wh02 103 bestaetigt bestellt 2008-02-06 11:23:12.0

LIEF.COMPENSATED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]

MAN.KRITISCHE_MENGE
[AT_ID, BESTELL_MENGE, LIEF_ID, STATUS, TIME]
CA1054IR 103 lief01 geliefert 2008-02-06 11:22:54.0

WH.BESTELLUNG_AT
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]
CA1054IR lager_1 300 266 closed 2008-02-06 11:23:13.0
CA1054IR lager_2 300 231 closed 2008-02-06 11:23:13.0

A.2 Test WAS 2

Zusätzlich zur Einstellung im Test A.1 wird der Lagerbestand des Autoteils 'CA1054IR' beim Lager_1-Service reduziert.

WS-AtomicTransaction
Lager_1 antwortet mit `Aborted`, da die Menge des Teils nicht durch den Lagerbestand gedeckt werden kann. Die WS-AT wird abgebrochen, die Lagerbestände in den beiden Lagern bleiben unverändert.

Die WS-AtomicTransaction wird auch dann mit *rollback* beendet, wenn die Antwort für die asynchron aufgerufene `bestelleAutoteil`-Operation nicht innerhalb des angegebenen Zeitlimits (Standardwert = 30 s) eintritt.

WS-BA Terminierung
Der Warehouse Service bricht die `bestelle`-Operation mit der java.rmi.RemoteException ab. Sein Teilnehmermanager generiert eine `Exit`-Nachricht an den Koordinator.

WS-BA Compensation
Die Lieferanten-Anwendung ruft uba.setCompensateOnly() auf, woraufhin der WS-BA Koordinator die `Compensate`-Nachricht an den Teilnehmermanager des Autohersteller-Service schickt. Die `ManufacturerCompensate.compensate()`-Methode markiert das Autoteil 'CA1054IR' als „nicht lieferbar".

Teilnehmersicht
Der Warehouse Service hat die Transaktion mit `Exit` verlassen, sein CompensationHandler wird nicht aufgerufen und deshalb wird kein Eintrag in der Tabelle WH.BESTELLUNG_AT gemacht.

LIEF.CLOSED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]

LIEF.COMPENSATED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]
300624 manuf01 wh02 104 bestaetigt nicht vorraetig 2008-02-06 14:08:02.0

MAN.KRITISCHE_MENGE
[AT_ID, BESTELL_MENGE, LIEF_ID, STATUS, TIME]
300624 104 lief01 nicht lieferbar 2008-02-06 14:07:44.0

WH.BESTELLUNG_AT
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]

A.3 Test WAS 3

Der vorherige Test wird erneut ausgeführt.Beim ersten Aufruf wirft die `compensate()`-Methode des Autohersteller-Service eine RetryCompensationHandlerException.

Listing A.1: `ManufacturerCompensate.java`

```
1 public void compensate(DataObject sdo) throws RetryCompensationHandlerException,
2           CompensationHandlerFailedException {
3       if (txCount == 0) {
4           System.out.println("will throw RetryCompensationHandlerException");
5           txCount++;
6           throw new RetryCompensationHandlerException("Manuf: Retry. TxCount: " + txCount);
7       } //...
8 }
```

WS-BA Compensation Fehler
WAS gibt die Fehlermeldung an stdout aus:
[06.02.08 15:12:33:171 CET] 00000066 RetryAlarmCon I CSCP0020I: Der Aufruf von compensate in sc.compensation.ManufacturerCompensate ist fehlgeschlagen. Er wird in 30 Sekunden wiederholt.

Fehlerbeschreibung von [IBM-Infocenter (2006)]:
CSCP0020I: The call to {0} on {1} failed. It will be retried in {2} seconds.
Explanation A retry exception was caught by the compensation service triggering the retry process.
Action None. Informational Only.

Die Kompensierungsoperation wird beim nächsten Aufruf erfolgreich ausgeführt.

Die Standardeinstellung im WAS 6.1 lässt eine unbegrenzte Anzahl an Kompensierungsversuchen zu. Der Test wird erneut mit der Anzahl der Wiederholungsversuche gleich vier ausgeführt. Die `compensate()`-Methode wirft bei jedem Aufruf eine `RetryCompensationHandlerException`.

WAS-Fehlermeldungen:
[06.02.08 15:51:45:281 CET] 00000047 RetryAlarmCon I CSCP0020I: Der Aufruf von compensate in sc.compensation.ManufacturerCompensate ist fehlgeschlagen. Er wird in 15 Sekunden wiederholt.
...
[06.02.08 15:56:10:375 CET] 00000035 RetryAlarmCon I CSCP0021I: Die Methode compensate in sc.compensation.ManufacturerCompensate wurde 4 Mal wiederholt. Das Wiederholungslimit ist damit erreicht. Es findet keine weitere Wiederholung statt.

Fehlerbeschreibung von [IBM-Infocenter (2006)]:
CSCP0021I: The method {0} on {1} has been retried {2} times and has reached the retry limit. It will not be retried again.
Explanation The compensation service will not retry the compensation handler again as it has reached the retry limit.
Action None. Informational Only.

Trotz dieser Fehlermeldung antwortet der Teilnehmermanager mit `Compensated`.

Teilnehmersicht
Tatsächlich wurde diese Kompensierungsoperation nicht ausgeführt, daher fehlt der Eintrag des CompensationHandler in der Tabelle MAN.KRITISCHE_MENGE:

LIEF.CLOSED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]

LIEF.COMPENSATED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]
300624 manuf01 wh02 104 bestaetigt nicht vorraetig 2008-02-06 14:08:02.0

MAN.KRITISCHE_MENGE
[AT_ID, BESTELL_MENGE, LIEF_ID, STATUS, TIME]
300624 -1 lief01 null null

WH.BESTELLUNG_AT
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]

A.4 Test WAS 4

Dies ist ein weiterer Test für den Compensation Service. Der Autohersteller lehnt die Bestellung des Autoteils 'CA1054IR' ab (Anwendungsfall 3.1.5 „Auftragsbestätigung anfordern“.Ausnahmen).

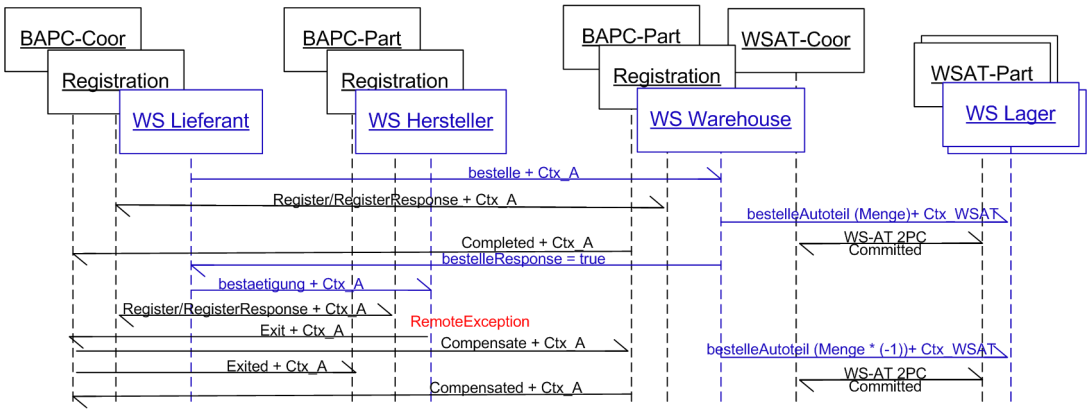

Abbildung A.2: Test WAS 4

WS-BA Compensation
Der WS-BA Teilnehmer „Autohersteller“ verlässt den `Active`-Status mit `Exit` und die `Warehouse.bestelle`-Operation wird kompensiert.

Teilnehmersicht
Der CompensationHandler des Warehouse Service bucht die zuvor in den beiden lokalen Lagern bestellten Mengen wieder zurückt und protokolliert diese Änderung in der Tabelle WH.BESTELLUNG_AT.

LIEF.CLOSED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]

LIEF.COMPENSATED
[AT_ID, MAN_ID, WH_ID, BESTELL_MENGE, STATUS_M, STATUS_WH, TIME]

CA1054IR manuf01 wh02 103 nicht bestaetigt bestellt 2008-02-06 14:49:57.0

MAN.KRITISCHE_MENGE
[AT_ID, BESTELL_MENGE, LIEF_ID, STATUS, TIME]
CA1054IR -1 lief01 null null

WH.BESTELLUNG_AT
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]
CA1054IR lager_1 266 300 compensated 2008-02-06 14:49:59.0
CA1054IR lager_2 231 300 compensated 2008-02-06 14:49:59.0

A.5 Test WAS 5

Der Anwendungsfall 3.1.4 „Bestellungen aufgeben (2)“ wird verwendet. Es soll der Compensation-Service in verschachtelten Business Activities getestet werden.

Zu Beginn wird die Menge des Autoteils ’CA1054IR’ beim Autohersteller sowie in einem der Warehouse Service-eigenen Lager reduziert, so dass der Warehouse-Service die `bestelle`-Anfrage mit false beantwortet.

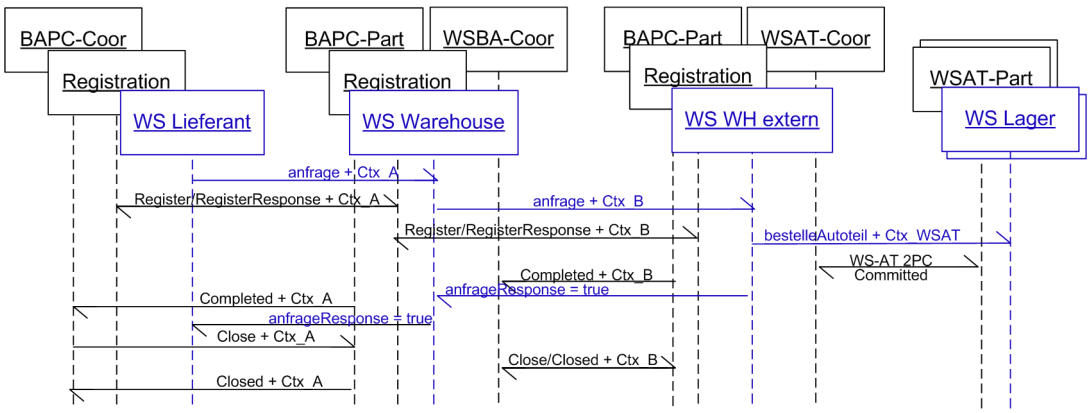

Abbildung A.3: Test WAS 5

WS-BA Close in Nested Scopes
Mit der `anfrage`-Operation wird eine neue UOW erzeugt. Die Kontrolle zu diesem Zeitpunkt liegt beim Warehouse-Service, der CompensationHandler des Lieferanten-Service bleibt so lange inaktiv bis die untergeordnete Business Activity ihre Arbeit abschließt: versucht man in der übergeordneten Activity den CompensationHandler

zu diesem Zeitpunkt aufzurufen, kann WAS diese Klasse nicht finden, und wirft eine `IllegalStateException`. Im Kontext der WS-BA weist diese Ausnahme darauf hin, dass es keine Business Activity auf dem Aufrufer-Thread gibt.

Teilnehmersicht

LIEF.CLOSED
[AT_ID, BESTELL_MENGE, MAN_ID, STATUS_M, WH_ID, STATUS_WH, TIME]
CA1054IR 103 manuf01 bestaetigt wh02 wh_2: EUR657.0 2008-02-06 22:08:21.0
CA1054IR 103 manuf01 bestaetigt wh02 nicht vorraetig 2008-02-06 22:08:21.0

LIEF.COMPENSATED
[AT_ID, BESTELL_MENGE, MAN_ID, STATUS_M, WH_ID, STATUS_WH, TIME]

MAN.KRITISCHE_MENGE
[AT_ID, NEUE_MENGE, LIEF_ID, STATUS, TIME]
CA1054IR 103 lief01 geliefert 2008-02-06 22:08:12.0

WH.BESTELLUNG_AT
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]
CA1054IR wh_1 -1 -1 closed 2008-02-06 22:08:54.0
CA1054IR wh_2 -1 -1 closed 2008-02-06 22:08:54.0

WH.BESTELLUNG_AT (Ext)
[AT_ID, L_ID, MENGE_ALT, MENGE_NEU, STATUS, TIME]
CA1054IR lager_1 300 231 closed 2008-02-06 22:08:55.0
CA1054IR lager_2 300 266 closed 2008-02-06 22:08:55.0

WS-BA Compensate in Nested Scopes
Der Testablauf wird um eine Operation erweitert: am Ende der Ausführung markiert der Lieferanten-Service den Prozess als „compensate only“. Alle Teilnehmer werden aufgefordert, ihre Arbeit zu kompensieren.

Der Kompensierungsprozess läuft ebenfalls „von unten nach oben“ ab: als erstes wird die Arbeit in der untergeordneten Business Activity kompensiert.

A.6 Test XTS 1

Anwendungsfälle: 3.1.2 „Lagerbestände kontrollieren“, 3.1.3 „Bestellungen aufgeben (1)“ und 3.1.5 „Auftragsbestätigung anfordern“. Der Bestand des Artikels ’00610’ wird sowohl beim Autohersteller als auch im Autoteilelager_1 auf 0 reduziert.

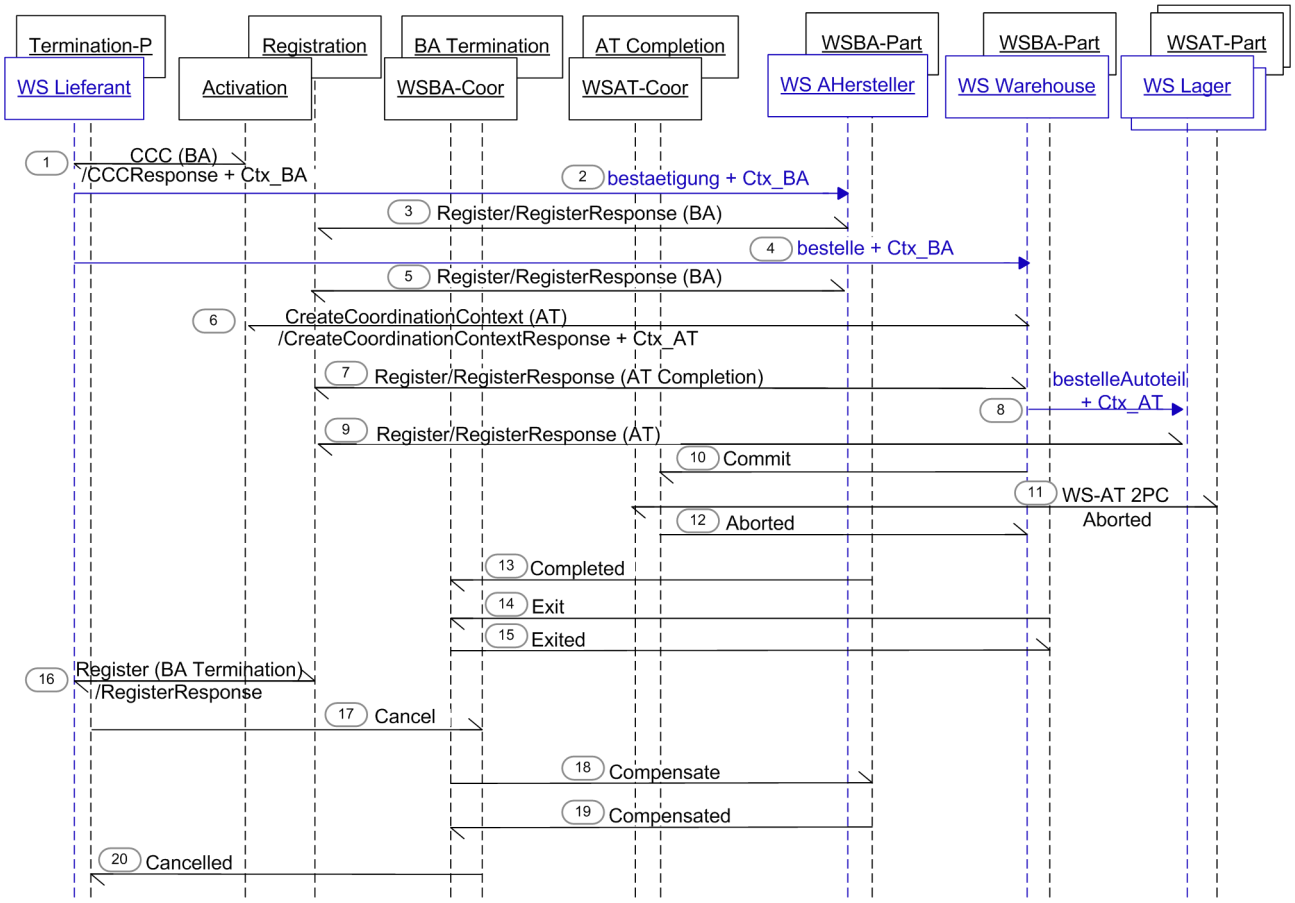

Abbildung A.4: Test XTS 1

CoordinationContext
Der WS-BA CoordinationContext wird bei den Aufrufen **(1)** erzeugt und in die ausgehenden Anwendungsnachrichten **(2)** und **(4)** eingebunden. Zu der WS-AT wird ein neuer Kontext in **(6)** erzeugt.

WS-AtomicTransaction
Der Warehouse-Service registriert sich für das WS-AT Completion-Protokoll **(7)** und bekommt die Aborted-Nachricht vom ATCompletionCoordinator **(12)**, da die WS-AT abgebrochen wird **(11)**.

Beenden der WS-BA und Compensation Der Lieferanten-Service registriert sich für das Termination-Protokoll **((16)** und schickt Cancel an den TerminationCoordinator **(17)**. Daraufhin leitet der BACoordinator die Kompensierung ein **(18)** und **(19)**. Der TerminationCoordinator schließt die Business Activity mit Cancelled ab **(20)**.

Teilnehmersicht

LIEF.COMPENSATED
[AT_ID BESTELL_MENGE, MAN_ID, STATUS_M, WH_ID varchar, STATUS_WH, TX_ID]
00610 107 M_1001 bestaetigt WH_1001 nicht vorraetig urn:-3ffefef6:501:47ca5862:99

MANUF.KRITISCHE_MENGE
[AT_ID BESTELL_MENGE STATUS IX_ID]
00610 107 bestaetigt urn:-3ffefef6:501:47ca5862:99
00610 107 Compensate urn:-3ffefef6:501:47ca5862:99

WH.BESTELLUNG_AT
[P_ID, AT_ID, MENGE_ALT, MENGE_NEU, STATUS, TX_ID]
L_1001 00610 1043 -1 bestellt urn:-3ffefef6:501:47ca5862:99
L_1002 00610 3 -1 bestellt urn:-3ffefef6:501:47ca5862:99
L_1001 00610 1043 1043 Exit urn:-3ffefef6:501:47ca5862:99
L_1002 00610 3 3 Exit urn:-3ffefef6:501:47ca5862:99

A.7 Test XTS 2

In diesem Testfall (Abbildung A.5) registriert der Autohersteller-Service einen BACC-Participant für die Business Activity **(3)**. Auf der Seite des BA-Initiators muss vor der uba.close() die uba.complete() API-Methode aufgerufen werden. Zunächst registriert sich der Lieferanten-Service für das Termintaion-Protokoll **(14)** und schickt Complete an den Termination-Koordinator **(15)**. Der BACC-Koordinator fordert den BACC-Teilnehmer des Autohersteller-Service auf, seine Arbeit zu beenden **(16)**. Der Lieferanten-Service registriert

sich erneut für das Termination-Protokoll **(18)**, um die Business Activity mit Close abzuschließen **(19)**-**(22)**.

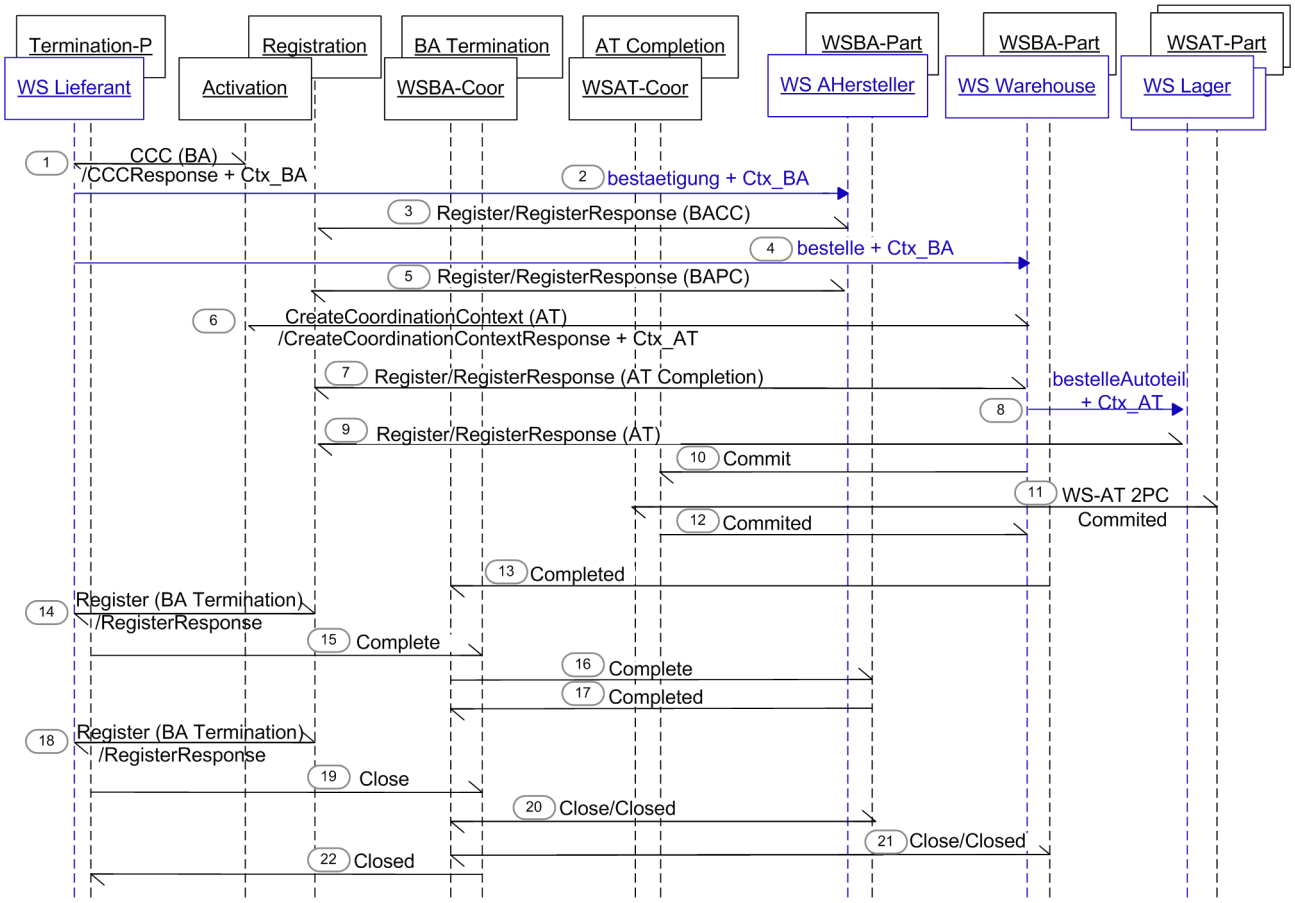

Abbildung A.5: Test XTS 2

Teilnehmersicht

MANUF.KRITISCHE_MENGE
[AT_ID BESTELL_MENGE STATUS TX_ID]
00610 104 bestaetigt urn:-3ffefef6:501:47ca5862:1f
00610 104 Complete urn:-3ffefef6:501:47ca5862:1f
00610 104 Close urn:-3ffefef6:501:47ca5862:1f

A.8 Test XTS 3

Der Test A.6 wird erneut ausgeführt. Der BAPC-Participant des Autoherstellers löst `FaultedException` während der Kompensierung aus. Der Protokollablauf unterscheidet sich zu A.4 dadurch, dass im Schritt **19** die Nachrichten Fault und Faulted ausgetauscht werden.

Anhang B

Inhalt der begleitenden CD

Die begleitende CD hat folgenden Inhalt:

- `CD-Inhalt.txt`: Inhalt der CD
- `Bachelorarbeit.pdf`: die vorliegende Arbeit
- Ordner `Implementierung_WAS`:
  - ear-Anwendungen des Beispielprogramms
  - README.pdf
- Ordner `Implementierung_XTS`:
  - das Beispielprogramm mit JBoss XTS
  - Ordner Tools: Lumbermill (Logger) und ant (Build-Tool)
  - README.txt
- Ordner `Trace_WAS`:
  - Protokollnachrichten der ausgeführten WAS-Tests
- Ordner `Trace_XTS`:
  - Protokollnachrichten der ausgeführten XTS-Tests

Es wird an einigen Stellen im Kapitel 5 auf die SOAP-Protokollnachrichten auf der beigefügten CD verwiesen. Die Pfadangaben der referenzierten Dateien werden nachfolgend aufgelistet.

Referenz	Pfad	Zeilen-Nr
CD_01	Trace_WAS\TEST_WAS_1\trace_AS01_2.pdf	
CD_02	Trace_WAS\TEST_WAS_3_2\trace_AS10_3.pdf	296, 399
CD_03	Trace_WAS\TEST_WAS_2\trace_AS01_2.pdf	69
CD_04	Trace_WAS\TEST_WAS_1\trace_AS01_1.pdf	2
CD_05	Trace_WAS\TEST_WAS_1\trace_AS01_2.pdf	77, 124
CD_06	Trace_WAS\TEST_WAS_faults\trace_AS01_4.pdf	135

Tabelle B.1: SOAP-Nachrichten

Literaturverzeichnis

[JBXTS:PG] JBoss Transactions 4.2.3. Web Service Transactions Programmers Guide. . – URL http://labs.jboss.com/jbosstm/docs/4.2.3/manuals/html/xts/ProgrammersGuide.html

[WSDL 2001] Web Services Description Language (WSDL) 1.1. In: *W3C* (2001). – URL http://www.w3.org/TR/wsdl

[UDDI 2004] UDDI Spec Technical Committee Draft. In: *OASIS* (2004). – URL http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[WSADD 2004] WS-Addressing. In: *W3C* (2004). – URL http://schemas.xmlsoap.org/ws/2004/08/addressing/

[WSPolicy 2006] WS-Addressing. In: *W3C* (2006). – URL http://www.w3.org/Submission/WS-Policy/

[SOAP 2007] SOAP Version 1.2. In: *W3C* (2007). – URL http://www.w3.org/TR/soap12-part0/

[wsat 2007] Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1. In: *OASIS* (2007). – URL http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html

[wsba 2007] Web Services Business Activity (WS-BusinessActivity) Version 1.1. In: *OASIS* (2007). – URL http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-os/wstx-wsba-1.1-spec-os.html

[bpel 2007] Web Services Business Process Execution Language Version 2.0. In: *OASIS* (2007). – URL http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf

[wscoor 2007] Web Services Coordination (WS-Coordination) Version 1.1. In: *OASIS* (2007). – URL http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-os/wstx-wscoor-1.1-spec-os.html

[Arjuna 2006] ARJUNA: 20 Year History. (2006). – URL http://www.arjuna.com/node/29

[Chaudhari u. a. 2007] CHAUDHARI, Anshuk P. ; MAJUMDAR, Bijoy ; SAXENA, Sunny: Long-Running Transactions in SOA. In: *SOA World Magazine* (2007). – URL http://soa.sys-con.com/read/318438.htm

[Corsten und Gössinger 2001] CORSTEN, Hans ; GÖSSINGER, Ralf: *Einführung in das Supply Chain Management*. Oldenbourg, 2001. – ISBN 3-4862-5819-2

[Coulouris u. a. 2005] COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim: *Distributed Systems. Concepts and Design*. Addison-Wesley, 2005. – ISBN 0-3212-6354-5

[Dostal u. a. 2005] DOSTAL, Wolfgang ; JECKLE, Mario ; MELZER, Ingo: *Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis*. Spektrum, Akad. Verl., 2005. – ISBN 3-8274-1457-1

[Elmargarmid 1992] ELMARGARMID, Ahmed K.: *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992. – ISBN 1-558-60214-3

[Erl 2005] ERL, Thomas: *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall International, 2005. – ISBN 0-1318-5858-0

[Erven und Hicker 2007] ERVEN, Hannes ; HICKER, Georg: Apache Kandula. (2007)

[Ferguson u. a. 2003] FERGUSON, Donald ; STOREY, Tony ; LOVERING, Brad ; SHEWCHUK, John: Secure, Reliable, Transacted Web Services. In: *IBM* (2003). – URL http://www.ibm.com/developerworks/webservices/library/ws-securtrans/

[Gray 1978] GRAY, Jim: *Operating Systems. An Advanced Course. Vol. 60, pp. 394-481*. Springer-Verlag, 1978. – ISBN 3-540-08755-9

[Härder und Reuter 1983] HÄRDER, T. ; REUTER, A.: Principles of Transaction-Oriented Database Recovery. In: *Computing Survers, Vol. 15, Nr. 4* (1983)

[IBM und BEA 2003] IBM ; BEA: SDO 2.1.0 Service Data Objects For Java. (2003). – URL http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf?version=1

[IBM-Infocenter 2006] IBM-INFOCENTER: (2006). – URL http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp

[JBossTS 2005] JBOSSTS: JBossTS. (2005). – URL http://labs.jboss.com/jbosstm/resources/product_overview/wst.html

[JSR-244 2006] JSR-244: (2006). – URL http://www.jcp.org/en/jsr/detail?id=244

[Microsoft 2007] MICROSOFT: WCF. (2007). – URL http://msdn2.microsoft.com/en-us/library/ms729784.aspx

[OpenGroup 1991] OPENGROUP: (1991). – URL http://www.opengroup.org/onlinepubs/009680699/toc.pdf

[Pardon 2007] PARDON, Guy: WS-BusinessActivity and WS-AtomicTransactions. In: *PARLEYS.com* (2007). – URL `http://www.parleys.com/display/PARLEYS/WS-BusinessActivity+and+WS-AtomicTransactions`

[W3C 2004] W3C: Web Services Glossary. (2004). – URL `http://www.w3.org/TR/ws-gloss/`

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw. §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19. März 2008	
Ort, Datum	Unterschrift (Elena Tews)