SERVICE ORIENTED ARCHITECTURES: TRANSAKTIONSMANAGEMENT MIT SERVICES UND GESCHÄFTSPROZESSEN

von

Martin Gerlach

Vortrag im Rahmen der Veranstaltung
Anwendungen 1 im Studiengang

MSc Informatik

Hochschule für Angewandte
Wissenschaften, Hamburg

Mai 2005

HAW HAMBURG

ABSTRAKT

SERVICE ORIENTED ARCHITECTURES: TRANSAKTIONS-MANAGEMENT MIT SERVICES UND GESCHÄFTSPROZESSEN

von Martin Gerlach

Professor Kai v. Luck
Fachbereich Informatik

Die Grundlagen des Paradigmas „Service Oriented Architecture“ (SOA) wurden im Rahmen der Veranstaltung „Anwendungen 1“ von Sven Stegelmeier vorgestellt. Diese Arbeit und der zugehörige Vortrag schließen daran an.

Die in SOAs verwendeten Dienste sind zumeist stark verteilt. Prozesse, die mehrere Dienste in einer wohldefinierten Art und Weise aufrufen (d.h. benutzen) und auf diese Weise Geschäftsprozesse abbilden, können unter Umständen lange Laufzeiten haben. Damit ein SOA-basiertes IT-System zuverlässig und stabil funktionieren kann, müssen die beteiligten, sich gegenseitig aufrufenden Prozesse und Dienste entsprechend koordiniert werden. Das System muss jederzeit in einem konsistenten Zustand sein, d.h. es wird ein Transaktionskonzept benötigt, welches die besonderen Eigenschaften Verteilung und lange Laufzeit von Prozessen und Diensten berücksichtigt. Im Vortrag und in dieser Arbeit stellt der Autor drei Ansätze zum Transaktionsmanagement für Services vor. Dabei wird speziell auf Web Services eingegangen. Für Web Services wird weiterhin ein Ansatz zur Kombination der Geschäftsprozessmodellierungssprache BPEL4WS und der Koordinierungsmanagement-Spezifikation WS-Coordination vorgestellt.

INHALT

Kapitel 1

EINLEITUNG UND MOTIVATION

Zur Zeit der Veranstaltung „Anwendungen 1“ im zweiten Semester des Studiengangs MSc Informatik an der HAW Hamburg (1. HJ 2005, im folgenden nur „aktuell“ oder „zur Zeit“) sind große IT-Lösungen zunehmend auf der Basis von „Service Oriented Architectures“ (SOAs) entworfen worden ([5]), in der IT-Systeme mittels lose gekoppelter Dienste (im Folgenden nur noch mit Services bezeichnet) realisiert werden, die sich gegenseitig aufrufen (benutzen). Sven Stegelmeier hat in seinem Vortrag ([20]) dieses Paradigma vorgestellt sowie bereits einen Ansatz gezeigt, um durch einen „(Enterprise-)Service-Bus“ eine gewisse Ordnung in ein Service-basiertes IT-System zu bringen.

In dieser zweiten Arbeit über SOAs in der Veranstaltung werden zwei eng zusammenhängende, bereits existierende Konzepte, Transaktionen und Geschäftsprozesse, nun speziell für die Anwendung in SOAs betrachtet. Geschäftsprozesse werden dabei durch technische Prozesse („Workflows“, [13]) abgebildet, die wiederum selbst Services sind und andere Services in wohldefinierter Art und Weise aufrufen. Es werden Transaktionskonzepte für Services und Prozesse vorgestellt, die die besonderen Eigenschaften von SOAs, wie z. B. hochgradige Verteilung der an Prozessen beteiligten Services sowie lange Laufzeiten von Prozessen, berücksichtigen. Das der Veranstaltung zugrunde liegende Ferienclub-Szenario wird dabei so betrachtet, als solle es auf Basis einer SOA realisiert werden und dementsprechend für Beispiele herangezogen. Abb. 1 zeigt ein beispielhaftes (unvollständiges) Modell. Zum Beispiel soll sichergestellt sein, dass die kombinierte Buchung von Leistungen (z.B. Mietwagen sowie Theater, Restaurant, Hotel in einem weiter entfernten, sehenswerten Ort) zuverlässig funktioniert bzw. das System nicht in einem inkonsistenten Zustand

hinterlassen wird, wenn sie einmal nicht funktioniert. Die Betrachtungen gelten aber auch allgemein für jedes SOA-basierte IT-System.

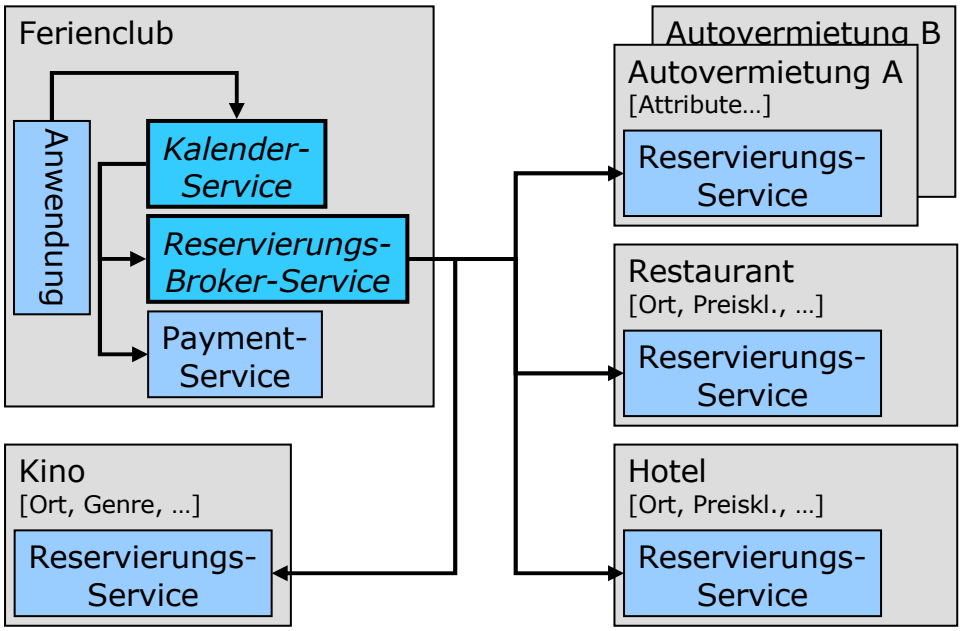

Abb. 1: Verteilte Services im Ferienclub-Szenario

In Kapitel 2 wird als Einführung und Gedächtnisstütze kurz der klassische Transaktionsbegriff der Informatik erklärt. Weiterhin werden einige Eigenschaften klassischer Transaktionen und einige Methoden klassischen Transaktionsmanagements betrachtet, die im weiteren Verlauf wichtig sind.

In Kapitel 3 wird erklärt, wie Services klassische Transaktionen zum Zugriff auf Ressourcen nutzen können und welche Probleme daraus speziell für Services und Prozesse mit langer Laufzeit entstehen können.

In Kapitel 4 werden aus den in Kapitel 3 aufgezeigten Problemen Anforderungen für Transaktionen in SOAs abgeleitet sowie drei Ansätze zur Koordinierung von Services und Ressourcen vorgestellt. Dabei wird der Fokus, entsprechend des aktuell in der Industrie vorherrschenden Trends im

B2B-Bereich, auf Web Services gelegt. Die Spezifikationen WS-Coordination und WS-Transaction werden daher besonders ausführlich behandelt.

In Kapitel 5 wird kurz auf die Implementierung von Geschäftsprozessen durch technische Prozesse eingegangen, bevor ein Ansatz vorgestellt wird, um die Prozessmodellierungssprache BPEL4WS mit der WS-Coordination Spezifikation zu kombinieren, um BPEL4WS-Prozesse transaktionsfähig zu machen, ohne dass die Semantik der Prozessbeschreibung geändert werden muss.

In Kapitel 6 wird ein Ausblick auf die vom Autor geplanten Aktivitäten für die im folgenden Semester stattfindenden Veranstaltungen „Anwendungen 2“ sowie „Projekt“ gegeben.

Die Kapitel entsprechen, beginnend mit der Einleitung, in der Reihenfolge den Punkten der Agenda des Vortrags: Motivation, Klassische Transaktionen, Services und Transaktionen, Long Running Distributed Transactions, Geschäftsprozesse, sowie Ausblick auf die Veranstaltungen des dritten Semesters.

K a p i t e l 2

KLASSISCHE TRANSAKTIONEN

2.1 Transaktionsbegriff

Der klassische Transaktionsbegriff leitet sich aus dem englischen Wort für Geschäftsvorfall, „Business Transaction", ab und bezieht sich auf manuelle und elektronische Vorgänge im Handel und hier insbesondere auf Finanzen ([10]).

In der Informatik wurde der Begriff erstmals im Zusammenhang mit Datenbanken gebraucht. Hier ging es darum, allgemein zugängliche Daten beim Zugriff durch mehrere Clients konsistent zu halten ([10], [11]). Der Begriff wird mittlerweile auch für den konsistenten Zugriff auf andere Ressourcen, z.B. Files, Message Queues und Verzeichnisdienste, gebraucht. Jeder Client greift dabei auf Ressourcen nur innerhalb von Transaktionen zu, die die folgenden – auch als *ACID* bekannten – Eigenschaften haben:

- *Atomicity:* Alle Operationen einer Transaktion erscheinen außerhalb der Transaktion als eine atomare Operation.

- *Consistency:* Alle Änderungen sind außerhalb der Transaktion erst nach erfolgreichem Abschluss (COMMIT) einer Transaktion sichtbar. Endet die Transaktion nicht erfolgreich (ROLLBACK), so wird außerhalb der Transaktion keine Änderung sichtbar.

- *Isolation:* Keine Transaktion sieht die Ergebnisse anderer Transaktionen.

- *Durability:* Alle Änderungen werden nach erfolgreichem Abschluss (COMMIT) einer Transaktion persistent gemacht, d.h. die Änderungen überleben einen Ausfall des Ressourcenmanagers (z.B. DB-Server).

2.2 Verteilte Transaktionen

Verteilte Transaktionen beziehen Ressourcen auf verteilten Systemen in eine Transaktion ein. Hier wird dafür gesorgt, dass der Zustand aller an der Transaktion beteiligten (logischen) Systemknoten konsistent bleibt.

Es existieren verschiedene erweiterte Konzepte, die in [2] näher erklärt werden. In [2] wird auch auf weiterführende Literatur dazu verwiesen, hier soll eine kurze Auflistung genügen:

- *Nested Transactions:* Eine Transaktion kann beliebig viele Kind-Transaktionen starten. Die Eltern-Transaktion kann erst dann beendet werden, wenn alle Kind-Transaktionen beendet sind. Wenn die Eltern-Transaktion abbricht, so werden alle Kind-Transaktionen abgebrochen. Drei Vorteile dieses Konzepts sind: Mehr Modularität bei voller globaler Isolation, feinere Granularität der Operationen, sowie mögliche parallele Ausführung von Kind-Transaktionen.

- *Open Nested Transactions:* Wenn eine Kind-Transaktion erfolgreich abgeschlossen wird (COMMIT), so werden die Ergebnisse auch außerhalb der Eltern-Transaktion sichtbar (gelockerte Isolation.) Bei einem ROLLBACK der Eltern-Transaktion werden Kompensierungs-Transaktionen für die abgeschlossenen Kind-Transaktionen notwendig.

- *Saga:* Eine möglicherweise lang andauernde Transaktion wird in *N* Einzel-Transaktionen und *N* jeweils passenden kompensierenden Transaktionen aufgeteilt. Die Einzeltransaktionen werden der Reihe nach ausgeführt. Schlägt Transaktion *n* fehl, so werden die Kompensierungs-Transaktionen *n-1* bis *1* in der Reihenfolge *n-1, n-2, …, 1* (also rückwärts) ausgeführt.

- *SplitJoin:* Ein Konzept zur Aufteilung einer Transaktion in mehrere, parallel laufende „Stränge“, die später wieder zusammengeführt werden.

- *ConTracts:* Ein Konzept zur Gruppierung von Einzel-Transaktionen nach einem Ablaufplan (Skript).

- *Long Running Activity:* Ein Prozess wird als Menge von Ausführungseinheiten aufgefasst, die aus Aktivitäten und Transaktionen bestehen. Eine Ausführungseinheit kann selbst eine Aktivität sein (rekursiv). Daten und Kontrollflüsse können statisch (Skript) oder dynamisch (über Ereignisse) spezifiziert werden.

Es wird sich zeigen, dass sich Ansätze einiger dieser erweiterten Konzepte in den in Kapitel 4 vorgestellten Konzepten wiederfinden.

2.3 Koordination klassischer, verteilter Transaktionen

Klassische, verteilte Transaktionen werden üblicherweise von einem Prozess koordiniert, der bei Abschluss jeder verteilten Transaktion diesen Abschluss (COMMIT oder ROLLBACK) auf jedem an der Transaktion beteiligten (logischen) Knoten bzw. Ressourcenmanager ausführt.

Die beteiligten Knoten haben unter Umständen lokale Änderungen durchgeführt, die aber erst nach Abschluss der (verteilten) Gesamttransaktion sichtbar werden sollen. Man sagt, diese Änderungen wurden im Transaktionskontext der Gesamttransaktion (als lokale Transaktionen) durchgeführt.

Das Ausführen von COMMIT oder ROLLBACK auf den beteiligten Knoten geschieht gemäß eines sogenannten „Completion Protocols“. Ein Beispiel für ein solches Protokoll ist das 2-Phase-Commit-Protokoll (2PC) (Abb. 2, [1]). Hier fragt der Koordinator in der ersten Phase des Protokolls alle Knoten, ob sie bereit zum COMMIT sind (PREPARE). Ist die Antwort von allen Knoten

positiv, so wird auf allen Knoten das COMMIT ausgeführt, ansonsten ROLLBACK.

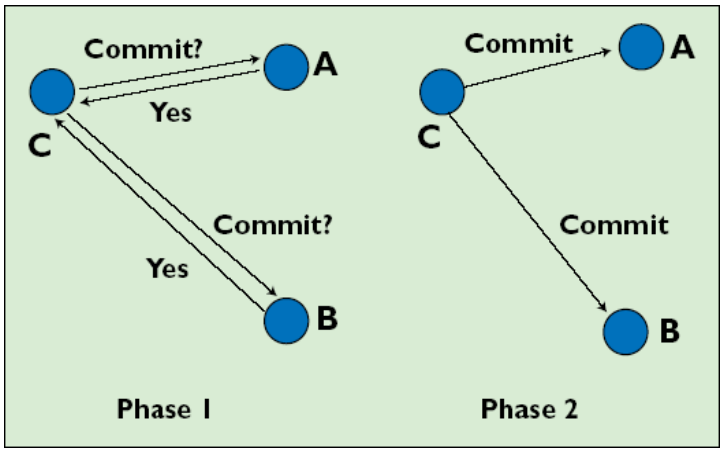

Abb. 2: 2-Phase-Commit-Protokoll (aus [1])

Komplexere Protokolle wie z.B. 3-Phase-Commit (siehe [10]) berücksichtigen auch den Ausfall des Koordinators. Bei 2PC würde dies zu lang blockierten Ressourcen auf den Knoten bzw. Timeouts auf den Knoten führen.

Oft wird jedoch ein einfaches Protokoll benutzt und der Koordinator-Knoten besonders ausfallsicher gemacht.

2.4 Zeitbetrachtung klassischer Transaktionen

Klassische Transaktionen, z.B. Datenbank-Updates, beinhalten in der Regel nur automatische, d.h. von Rechnern ausführbare, Schritte sowie Kommunikation zwischen Applikationen (ohne Einbeziehung von Benutzern). Sie laufen daher „schnell“, d.h. der typische Abstand zwischen „PREPARE“ und „COMMIT“ im 2-Phase-Commit-Protocol liegt nach [1] im Millisekundenbereich, maximal im Bereich weniger Sekunden. Bei einem Timeout erfolgt ein ROLLBACK der Gesamttransaktion.

Kapitel 3

SERVICES UND TRANSAKTIONEN

3.1 Nutzung klassischer Transaktionen durch Services

Services[1] können zur Erfüllung ihrer Aufgaben klassische lokale oder verteilte Transaktionen benutzen, z.B. für Datenbank-Zugriffe. Alle von einem Service genutzten Ressourcen sollten in einem Transaktionskontext angesprochen werden.

3.2 Nutzung weiterer Services durch Services

Wenn von einem Service weitere Services aufgerufen (benutzt) werden sollen, kann es – wie im Fall von einfachen Web Services – ein Problem sein, dass keine Konzepte zur Koordinierung, zur Übergabe eines Transaktionskontexts, bzw. zur Abschlusssteuerung durch ein Completion Protocol existieren.

Im Ferienclub sind durchaus Services vorstellbar, die eine Reihe weiterer interner Services (z.B. Kalender, Bezahlung, Buchungen) und externer Services (z.B. Autovermietungen, Restaurants, Kinos, Theater) aufrufen.

Es muss daher für SOA-basierte Systeme ein Ziel sein, mehrere, auch verschachtelte, Service-Aufrufe so zusammenfassen und koordinieren zu können, dass die Konsistenz der beteiligten Teilsysteme (und damit des Gesamtsystems) gewährleistet bleibt.

Intuitiv erscheint hierfür der erste (oberste) Service-Provider in der Aufrufhierarchie als der geeignete Koordinator (Abb. 3, mitte).

[1] Definition. genauere Erklärung und Referenzen siehe Vortrag von Sven Stegelmeier ([20]).

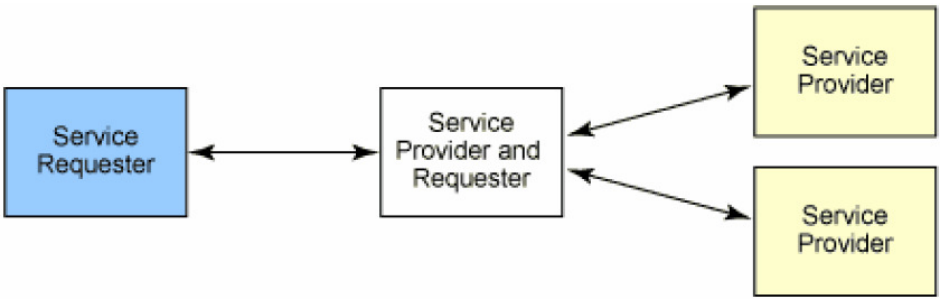

Abb. 3: Verschachtelte Service-Aufrufe

3.3 Verteilung und Zeit

Im Ferienclub muss von einer hochgradigen Verteilung der Services ausgegangen werden. Dabei sollen wie oben bereits angedeutet auch externe Services mit eingebunden werden.

Die Abarbeitung einer Operation durch einen Service ist nun unter Umständen nicht vollautomatisch. Man stelle sich eine Autovermietung vor, die an den Ferienclub durch einen Service-Proxy angebunden ist, welcher Service-Aufrufe des Ferienclub-Buchungsservices in Emails umwandelt, die dann an die Mitarbeiter der Autovermietung geschickt werden. Die Mitarbeiter bearbeiten dann den Mietauftrag und füllen ein (elektronisches) Formular aus, welches wiederum eine Antwort-Email mit dem Ergebnis an den Service-Proxy schickt. Dies wird zumindest einige Minuten, eher aber einige Stunden oder sogar Tage dauern, was im Vergleich zum üblichen Zeitverständnis für Transaktionen (siehe 2.4) „lang“ ist.

D.h. über den Aufruf von Services, deren Implementierung nicht bekannt ist, kann man keine Zeitaussage machen, wenn nicht vorher gewisse nichtfunktionale Anforderungen dazu vereinbart wurden.

D.h. weiterhin, dass Serviceaufrufe „lange“ dauern können.

Verzögerungen und erfolglose Aufrufe können schließlich auch dadurch zustande kommen, dass die Aufrufe „entfernt“ stattfinden, d.h. zwischen unterschiedlichen physikalischen Knoten. Gründe für Verzögerungen können hier z.B. Netzwerk-Überlastung durch Bandbreiten-Engpässe oder Kollisionen bei der Funkübertragung sein.

Würde man nun ein Transaktionskonzept implementieren, welches die klassischen *ACID*-Eigenschaften aufweist und ein übliches Completion-Protocol (z.B. 2-Phase-Commit) benutzt, kann es passieren, dass Ressourcen, die von Services oder Knoten benutzt werden, die an einer Transaktion beteiligt sind, verhältnismäßig lange gesperrt bleiben müssen, wenn auch nur ein beteiligter Serviceaufruf „lange“ dauert.

„Lange“ bezeichnet in diesem Zusammenhang eine Dauer der Sperrung von Ressourcen, die die Funktionalität und die nicht-funktionalen Eigenschaften (insb. Antwortzeiten) eines Systems oder Teilsystems derart verschlechtert, dass das System oder Teilsystem nicht mehr benutzbar im Sinne des Endanwenders (Ferienclub: Gast, Animateur, Manager, …) ist.

K a p i t e l 4

LONG RUNNING DISTRIBUTED TRANSACTIONS[2]

4.1 Anforderungen

Aus den Betrachtungen in Kapitel 3 ergeben sich folgende Anforderungen an Transaktionen in SOA-basierten Systemen:

Verteilung: Die beteiligten Services sind hochgradig verteilt. Prozesse müssen in einer Transaktion Services entfernt aufrufen können, die dann ihre lokalen Aktionen innerhalb der selben Transaktion ausführen.

Zeit: Services können eine lange Laufzeit haben.

Sperren von Ressourcen: Lokale Ressourcen sollen von den Services nicht zu lange gesperrt werden.

ACID-Transaktionen und die dazugehörigen Protokolle wie 2PC erweisen sich hier zwar für die hohe Verteilung geeignet, haben aber Schwächen bei langlaufenden Prozessen, da (veränderte) Ressourcen bis zum endgültigen COMMIT oder ROLLBACK gesperrt bleiben.

Damit – wie bei Open Nested Transactions ([2]) – Ressourcen schnell wieder freigegeben werden können, müssen die *ACID*-Anforderungen etwas gelockert werden, insbesondere Atomicity, Consistency und Isolation.

Lässt man nun aber zu, dass die Änderungen einer geschachtelten Kind-Transaktion vor Abschluss der Long Running Transaction (Eltern-Transaktion) schon über letztere hinaus sichtbar werden, so ergeben sich Probleme, die das folgende Beispiel verdeutlichen soll:

---

[2] Der Autor zieht hier den englischen Begriff der deutschen Übersetzung vor. Eine andere, häufig in der Literatur zu findende englische Bezeichnung ist „Long Lived (Distributed) Transaction"

Ein Gast des Ferienclubs bucht eine auswärtige Aktivität, etwa den Besuch des besten Restaurants der Region. Der Gast entscheidet sich, mit einem noch zu buchenden Mietwagen dorthin zu fahren. Das System soll die Buchung als einen Geschäftsprozess in einer Transaktion durchführen. Es reserviert zunächst den Mietwagen. Bei der anschließenden Reservierung des Restauranttisches läuft das System in einen Fehler (oder es gibt einfach keinen Tisch mehr), so dass die Reservierung nicht durchgeführt werden kann.

Hat nun der Mietwagen-Service die Reservierung bereits abgeschlossen (lokales COMMIT), so kann diese nicht einfach rückgängig gemacht werden, wenn das System nicht weiß wie. Es müsste eine kompensierende Transaktion durchgeführt werden (Storno).

Kompensierende Aktionen müssen von jedem Service als eigene Operation angeboten werden. Es sollte aber vermieden werden, dass insgesamt zu viele Stornos auftreten, um die Wahrscheinlichkeit von „Dirty Reads“ gering zu halten. Z.B. könnte im obigen Beispiel eine andere Anfrage für den Mietwagen eintreffen und mit „Nicht verfügbar“ beantwortet werden, bevor die Reservierung storniert wird. Mit *ACID*-Transaktionen hätte die anfragende Transaktion warten müssen, hätte dann aber aufgrund des ROLLBACKS der Mietwagen-Restaurant-Transaktion die Antwort „Verfügbar“ erhalten.

Prozesse, die mehrere Services aufrufen, sowie die beteiligten Services selbst, sollten also im Falle von möglichen langen Laufzeiten so ausgelegt sein, dass man in der Regel davon ausgehen kann, dass keine Kompensierung nötig wird. Dieses Prinzip ist in anderen Zusammenhängen (Betriebssysteme, verteilte Systeme) als „Optimistische Nebenläufigkeitskontrolle“ bekannt.

4.2 Kompensation

Das Prinzip der Kompensation soll anhand eines weiteren Beispiels verdeutlicht werden, welches ursprünglich aus [1] stammt und hier an das Ferienclub-Szenario angepasst wurde:

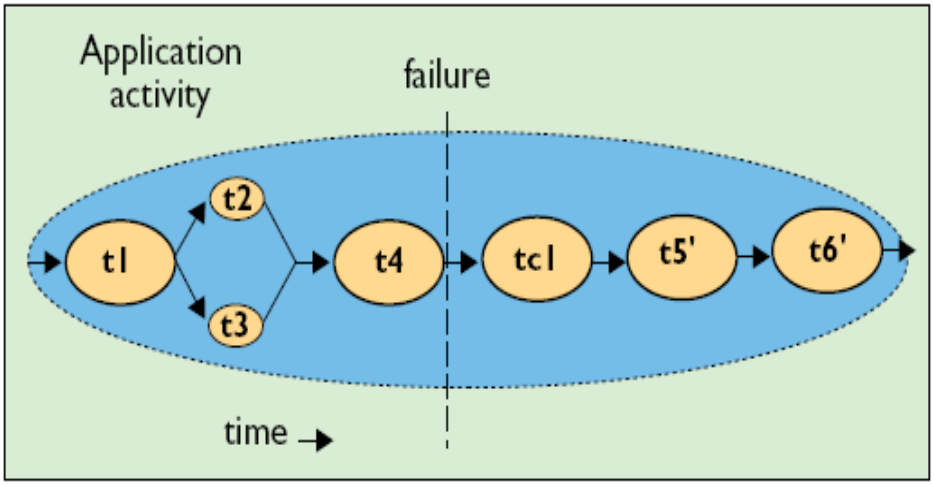

Abb. 4: Beispiel für Kompensation aus [1]

In Abb. 4 wird ein Prozess dargestellt, der aus folgenden Aktivitäten besteht, die als lokale, atomare Transaktionen ablaufen, deren Ergebnisse bei erfolgreichem Abschluss sofort nach außen hin sichtbar werden.

t1: Reservierung eines Mietwagens

t2: Reservierung eines Restaurants

t3: Bestellung von Theaterkarten (wird parallel zu t2 ausgeführt)

t4: Buchen eines Hotels. Dies schlägt fehl, etwa weil das Hotel ausgebucht ist. Alle vorherigen Buchungen müssen nun rückgängig gemacht werden.

tc1: Kompensation von t2 und t3 (auch als zwei getrennte Kompensationen denkbar, je nach Aufteilung der Buchungs-Services.)

t5' und t6': Durchführen von Alternativen, etwa Bestellung von Kinokarten (t5') und anschließender Reservierung eines Karaoke-Raums (t6')

Zu t5' und t6' sowie zur Tatsache, dass t1 nicht storniert wird, ist anzumerken, dass das automatische Durchführen von alternativen Aktivitäten sowie die automatische Auswahl der zu kompensierenden Aktivitäten beim Scheitern einer Aktivität nicht Gegenstand dieser Arbeit sind. Denkbar wären z.B. die Angabe von Alternativen bei der Buchung oder in der Geschäftsprozessdefinition (siehe 4.4.2), oder auch die Ableitung möglicher Alternativen aus einem Benutzerprofil, aus einer Wunschliste, oder aus erhobenen Daten über das Verhalten der Ferienclub-Gäste.

4.3 SOA Systementwurf

Beim Systementwurf ist es wichtig, die in den vorigen Abschnitten entwickelten Anforderungen von Anfang an zu berücksichtigen.

D.h. insbesondere, dass man nicht zunächst alle Services als lokal, schnell, zuverlässig (korrekt arbeitend, hochverfügbar) betrachten kann und unter dieser Annahme Prozesse implementiert. Unter Umständen müssen anfängliche Dummy-Implementierungen lange Antwortzeiten, Fehler, etc. auf Zufallsbasis simulieren, dabei aber wiederum nicht die Wertebereiche der vorher spezifizierten Systemanforderungen bzw. Anforderungen an einzubindende (externe) Services verlassen.

Die Architektur des zu entwickelnden Systems muss also für verteilte Services entworfen werden und nicht umgekehrt. Dies ist die Grundlage einer Service Oriented Architecture.

4.4 Konzepte für Long Running Distributed Transactions

4.4.1 Business Transaction Protocol

Das Business Transaction Protocol (BTP, [2], [17]) wird aktuell von der „Organization for Advance Structured Information Systems“ (OASIS, http://www.oasis-open.org/) entwickelt, die schon seit 1996 im B2B-Bereich aktiv ist. Das BTP liegt seit November 2004 in der Version 1.1.0 vor. Es handelt sich dabei um ein Koordinator-basiertes Transaktionsprotokoll wie 2PC, jedoch mit wesentlich mehr Freiheiten für die koordinierende Applikation. Es wurde mit dem Ziel entwickelt, Zuverlässigkeit über unzuverlässige Kanäle (z.B. das Internet) zu erreichen und dabei explizit Long Running Transactions zu unterstützen. Aufgrund seiner Flexibilität eignet es sich auch für Workflow-Steuerungen.

Durch das BTP hat der Koordinator die Kontrolle darüber, wann eine Transaktion für ein COMMIT vorbereitet wird (PREPARE). Mittels Business-Logik kann dann für jeden Teilnehmer (Service) entschieden werden, ob die dortige Transaktion bestätigt (CONFIRM) oder abgebrochen (CANCEL) wird. Anschließend kann dann auf Basis der von jedem Teilnehmer erhaltenen Antworten (CONFIRM oder CANCEL) entschieden werden, ob die Gesamttransaktion erfolgreich (COMMIT) oder nicht erfolgreich (ROLLBACK) beendet wird.

Im BTP gibt es zwei Arten von Transaktionen:

Atom: Atoms verhalten sich wie klassische *ACID*-Transaktionen. Sie sind nur dann erfolgreich, wenn alle Teilnehmer auch erfolgreich sind. Ergebnisse der Teilnehmer werden erst sichtbar, wenn die Gesamt-Transaktion erfolgreich abgeschlossen wird.

Cohesion: In einer Cohesion können die Teilnehmer unterschiedliche Ergebnisse haben und trotzdem kann die Gesamttransaktion erfolgreich abschließen. Das 2-Phasen-Protokoll, das zur Anwendung kommt, erlaubt

eine genaue Parametrisierung dafür, welche Teilnehmer für ein COMMIT bestätigt (CONFIRM) oder abgebrochen (CANCEL) werden. Wie bei Open Nested Transactions ist das Ergebnis eines Teilnehmers (der die Kind-Transaktion ausführt) bei Erfolg bereits vor dem Gesamt-COMMIT außerhalb der Cohesion sichtbar. Schlägt die Cohesion fehl (ROLLBACK), so müssen alle Teilnehmer, die ein CONFIRM erhalten haben, im Protokoll zu definierende kompensierende Aktionen ausführen.

Aufgrund der teilweisen Aufhebung von „*A*tomicity“ und *I*solation (und damit auch *C*onsistency) in Cohesions wird das BTP auch als „Open Top COMMIT Protocol“ bezeichnet.

Beispiele für die Anwendung des BTP finden sich in [2].

Das BTP benötigt entsprechende Implementierungen (Framework und APIs) auf Koordinator- und Teilnehmer-Seite. Die Protokollnachrichten sollen innerhalb der Nachrichten des verwendeten Kommunikationsprotokolls für Services übertragen werden.

Logisch ist das BTP in die Quality of Service (QoS) Schicht einzuordnen.

Für Web Services wird dies in Abb. 5 und 6 verdeutlicht.

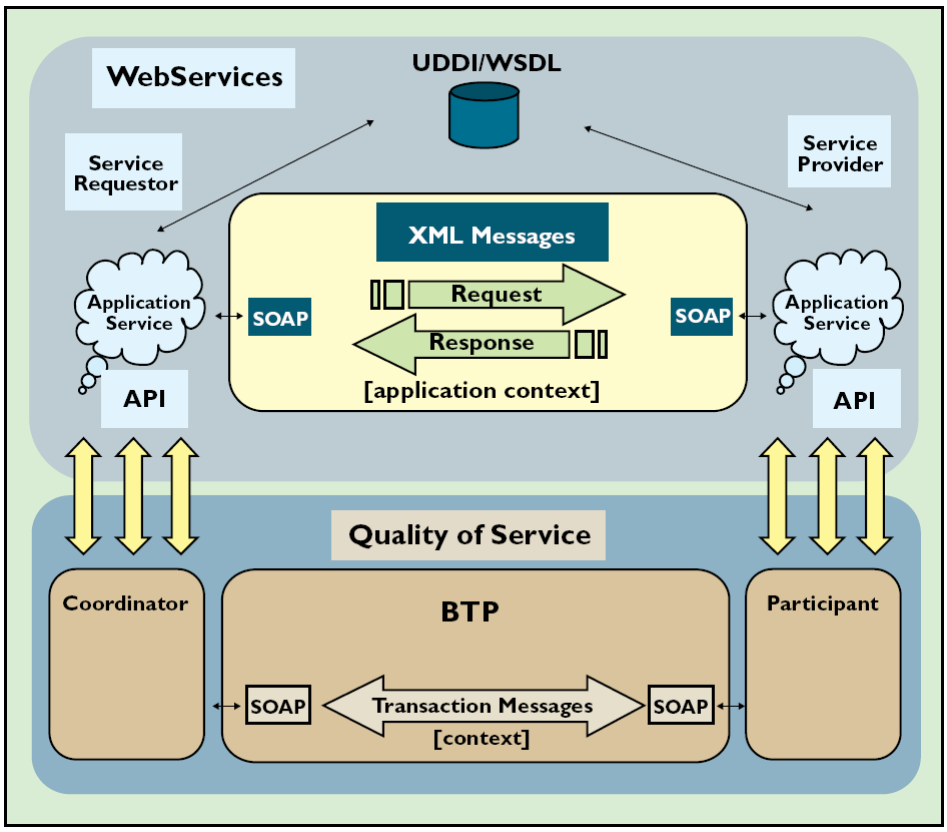

Abb. 5: BTP in der QoS-Schicht für Web Services (aus [1])

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP:Envelope SOAP:encodingStyle=
   "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
   <btp:begin transaction-type="atom"
    xmlns:btp="urn:oasis:names:tc:BTP:1.0:core"/>
  </SOAP:Body>
</SOAP:Envelope>
```

Abb. 6: BTP-Nachricht zum Start einer „Atom“-Transaktion innerhalb einer SOAP-Nachricht (aus [1])

4.4.2 Tentative Hold Protocol

Das Tentative Hold Protocol (THP, [2], [18]) wurde bereits 2001 vom W3C entwickelt. Das THP entstammt dem Online-Shopping-Paradigma und beschreibt einen Informationsaustausch zwischen lose gekoppelten Geschäftspartnern vor der eigentlichen Transaktion, in dem sog. nicht-blockierende („tentative“) Sperren („holds“) für Ressourcen angefordert werden. Dies geschieht im Sinne der optimistischen Nebenläufigkeits-kontrolle: Jeder Teilnehmer (Requestor), der eine solche Sperre angefordert hat und diese auch von der Gegenseite (Provider) erhält, kann zunächst so tun, als hätte er die Ressource „sicher“ für sich, obwohl durchaus mehrere Teilnehmer eine nicht-blockierende Sperre für die selbe Ressource erhalten können.

„Ressourcen“ sind im Falle vom BTP nicht als technische Ressourcen, wie etwa eine Datenbank-Verbindung oder eine Queue, zu sehen, sondern als geschäftliche Ressourcen, wie etwa ein Artikel, der vor dem Kauf in einem Online-Shop in den Warenkorb gelegt wird, oder ein Mietwagen, der zunächst ausgewählt wird, bevor er endgültig reserviert wird. Eine nicht-blockierende Sperre für eine solche Ressource wird in diesem Zusammenhang auch als „Lightweight Reservation“ bezeichnet.

Bezogen auf das Warenkorb-Paradigma ist folgendes in einem THP-basierten System möglich: Obwohl ein Artikel nur N mal im Lager ist, können mehr als N Kunden den Artikel in den Warenkorb legen. Erst nach erfolgreicher Bezahlung wird die Anzahl im Lager heruntergezählt. Ist das Lager irgendwann erschöpft, wird an die restlichen Kunden, die den Artikel im Warenkorb haben, eine Nachricht gesendet und der Artikel aus den Warenkörben entfernt.

Eine Implementierung des THP muss also nachrichtenbasiert sein, insbesondere muss die Requestor-Seite imstande sein, asynchrone Nach-richten zu verarbeiten, wenn eine zuvor erhaltene Sperre ungültig wird. Beim

COMMIT der aktuellen Transaktion werden alle angeforderten Sperren dann nochmals daraufhin überprüft, ob sie noch gültig sind. Falls nicht, scheitert das COMMIT und der Teilnehmer muss diese Ausnahme behandeln. Auf Provider- und Requestor-Seite müssen außerdem APIs für das Anfordern, Überprüfen, Wieder-Abgeben und Verwalten der nicht-blockierenden Sperren zur Verfügung stehen.

In THP-basierten Systemen sind zwar keine Kompensierungs-Aktivitäten aufgrund nicht-verfügbarer geschäftlicher Ressourcen notwendig, weil keine blockierenden Sperren vergeben werden und daher keine schnellen COMMITs nötig sind. Die Anwendungen bzw. Services in einem solchen System werden jedoch komplexer sein als z.B. unter Verwendung des BTP, da eine asynchrone Nachrichtenbehandlung nötig ist und genau der Fall, dass eine eigentlich reservierte Ressource auf einmal nicht mehr verfügbar ist, ja auch von der Geschäftslogik behandelt werden muss.

Durch die Auslegung des THP als nachrichtenbasiertes teilweise asynchrones Protokoll können Requestor und Provider lose gekoppelt sein und trotzdem ist ein Requestor jederzeit über den Zustand seiner erhaltenen Sperren informiert.

In [2] finden sich genauere Beschreibungen darüber, wie die nicht-blockierenden Sperren verwaltet werden. Es werden außerdem einige Anwendungsbeispiele gegeben.

Speziell für Web Services haben die Autoren von [2] eine Erweiterung erdacht, das „Web Service Tentative Hold and Compensation Composition Transaction Model“. Dieses Modell sieht neben Kompensierungs-Aktivitäten auch dynamische Service Choreography (siehe Kapitel 5) mittels Constraints vor. Das Zustandsdiagramm für eine Aktivität ist in Abb. 7 gezeigt, weitere Details sind in [2] nachzulesen.

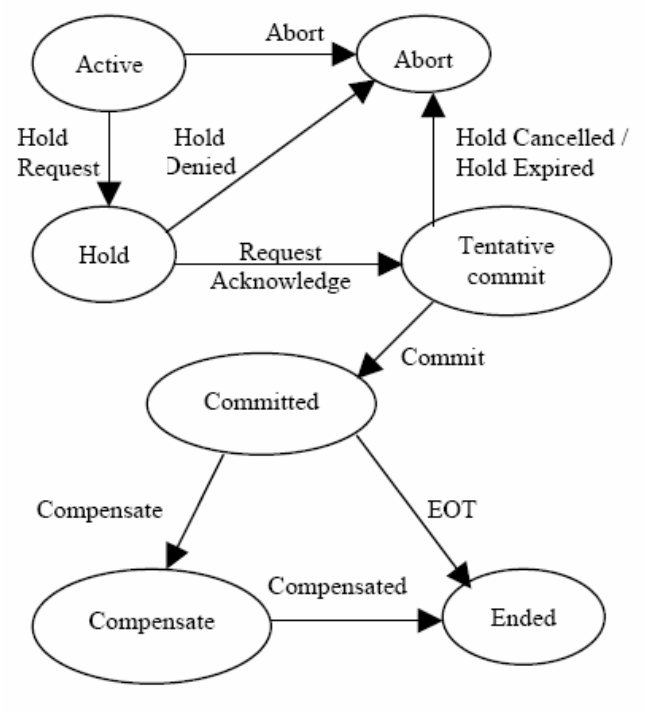

Abb. 7: Zustände einer Aktivität im „Web Service Tentative Hold and Compensation Composition Transaction Model“ (aus [2])

4.4.3 WS-Coordination und WS-Transaction

*4.4.3.1 WS-**

Bereits seit 2002 entwickeln verschiedene Unternehmen und Organisationen, wie z.B. BEA, Computer Associates, IBM, Microsoft und VeriSign, Spezifikationen und Vorschläge zur Standardisierung für Vorgänge und Protokolle rund um das Web Services Referenzmodell.

Dabei geht es vor allem darum, Web Services für Geschäftsprozesse und B2B-Anwendungen zu benutzen und Nachteile der ursprünglichen Web Service Spezifikationen zu kompensieren. Insbesondere sind einfache Web Services nicht sicher, zustandslos und nicht transaktional. Desweiteren sind die Transportmechanismen (HTTP, JMS, SMTP) nicht grundlegend zuverlässig und nicht geeignet für Service Level Agreements.

Für verschiedene Bereiche des Web Services Referenzmodells werden zur Zeit Spezifikationen entwickelt, einen Überblick gibt Abb. 8.

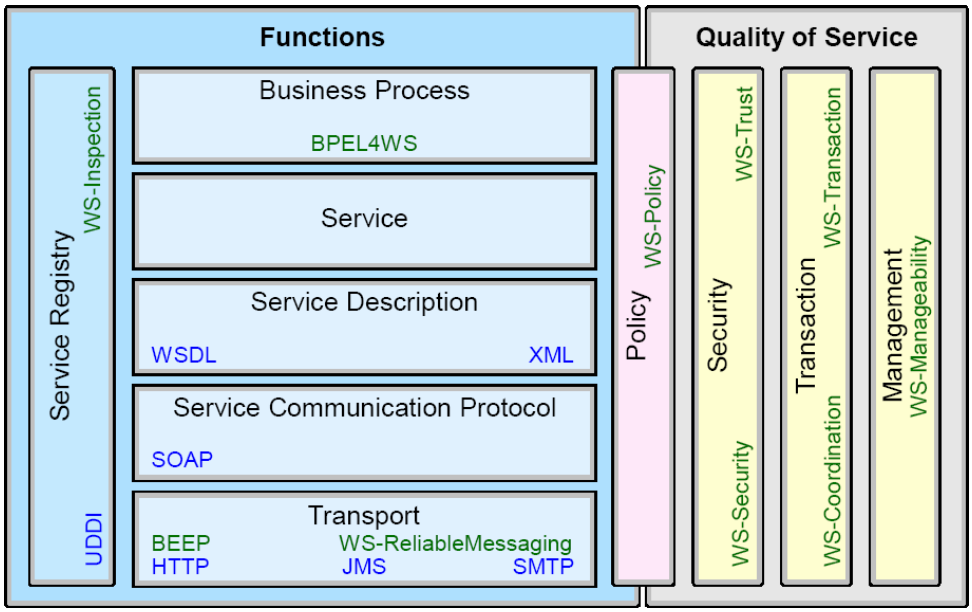

Abb. 8: Web Services Referenzmodell, Standards und Spezifikationen (Blau: Existierende Standards, grün: Aktuelle Entwicklungen, Stand April 2004, siehe [5])

Für Transaktionen sind insbesondere WS-Coordination und WS-Transaction von Interesse.

4.4.3.2 WS-Coordination

WS-Coordination spezifiziert ein Framework zur Koordinierung von Web Services, das selbst aus drei Services besteht, die es Applikationen bzw. Services erlaubt, sogenannte Aktivitäten, bestehend aus Logik und insbesondere Service-Aufrufen, innerhalb eines Kontexts auszuführen.

Der *Activation Service* wird aufgerufen, um eine neue Aktivität zu starten. Gleichzeitig wird das Completion Protocol für die Aktivität festgelegt (bisher möglich: *AtomicTransaction* oder *BusinessActivity,* siehe unten). Optional kann

die neue Aktivität in Relation zu einer bereits existierenden Aktivität gesetzt werden, was geschachtelte Aktivitäten ermöglicht. Wie weiter unten erklärt wird, ist dies notwendig um Long Running Distributed Transactions zu realisieren. Der Activation Service gibt der aufrufenden Applikation einen *Coordination Context* zurück, der u. a. eine Referenz auf den Registration Service enthält (Abb. 9). Anm.: Die Referenz wird gemäß WS-Addressing Spezifikation [7] angegeben.

```
<?xml version="1.0" encoding="utf-8"?>
<S:Envelope
   xmlns:S="http://www.w3.org/2001/12/soap-envelope">
 <S:Header>
  . . .
  <wscoor:CoordinationContext
     xmlns:wsme="http://www.w3.org/2002/06/msgext"
     xmlns:wscoor="http://www.w3.org/2002/06/Coordination"
     xmlns:myApp="http://www.w3.org/2002/06/myApp">
    <wsme:Identifier>
     http://foobaz.com/SS/1234
    </wsme:Identifier>
    <wsme:Expires>
     2005-06-30T13:20:00.000-05:00
    </wsme:Expires>
    <wscoor:CoordinationType>
      http://xml-soap.org/2002/06/AtomicTransaction
    </wscoor:CoordinationType>
    <wscoor:RegistrationService>
     <Address>
http://myservice.com/mycoordinationservice/registration
     </Address>
     <myApp:BetaMark> ... </myApp:BetaMark>
     <myApp:EBDCode> ... </myApp:EBDCode>
    </wscoor:RegistrationService>
    <myApp:IsolationLevel>
     RepeatableRead
    </myApp:IsolationLevel>
  </wscoor:CoordinationContext>
  . . .
 </S:Header>
 . . .
</S:Envelope>
```

Abb. 9.: Coordination Context in einer SOAP Nachricht. Der Service-Aufruf wird innerhalb einer atomaren Transaktion ausgeführt. <myApp> Elemente sind proprietäre Erweiterungen.

Der *Registration Service* dient dazu, für eine Aktivität den teilnehmenden Applikationen und Services Rollen wie etwa „Koordinator“ und „Teilnehmer“ zuzuordnen. Weiterhin müssen alle Teilnehmer sich selbst sowie sämtliche verwendeten transaktionalen Ressourcen (z.B. DB-Verbindung) beim Registration Service anmelden, damit später das Completion-Protocol korrekt abgewickelt werden kann. Als Rückgabewert für die Registrierung erhalten die Aufrufer eine Referenz auf den Coordination Service. Ein Service in der Rolle „Koordinator“ kann diese z.B. benutzen, um die Aktivität abzuschließen.

Der *Coordination Service* wickelt schließlich den Abschluss der Aktivität gemäß dem gewählten Completion-Protocol ab. Das Protokoll legt das Verhalten und die zu unterstützenden Operationen der Teilnehmer fest. Für atomare Transaktionen mit 2PC muss ein „Koordinator“ zum Beispiel ein Interface für COMMIT und ROLLBACK anbieten, während ein „Teilnehmer“ die Operationen PREPARE (2PC Phase 1), COMMIT und ROLLBACK unterstützen muss.

D.h. Services, die mit WS-Coordination koordiniert werden, müssen APIs gemäß der verwendeten Completion-Protocols anbieten.

4.4.3.3 WS-Transaction

Zur Zeit existieren Spezifikationen für zwei Completion-Protocols: WS-AtomicTransaction und WS-BusinessActivity, zusammengefasst auch oft als WS-Transaction bezeichnet.

WS-AtomicTransaction wird für kurzlebige Aktivitäten verwendet. Das Protokoll ist analog zum 2-Phase-Commit aufgebaut, d.h. die Aktivität ist nur dann erfolgreich, wenn sie vom Koordinator mittels COMMIT beendet wurde und alle Operationen und geschachtelten Aktivitäten erfolgreich waren.

WS-BusinessActivity unterstützt schließlich langlebige Aktivitäten, d.h. Long Running (Distributed) Transactions, in der Spezifikation etwas uneinheitlich auch Business Transactions (Geschäftsprozesse) oder Business Activities genannt. Diese haben Ähnlichkeiten zu den Cohesions im BTP (siehe 4.4.1) und Open Nested Transactions (siehe 2.2). Sie bestehen aus einer Menge kurzlebiger Teilaktivitäten, die über den Registration Service als „Kinder" zu der Long Running Transaction registriert wurden und das WS-AtomicTransaction Protokoll verwenden. Die Ergebnisse dieser kurzlebigen Teilaktivitäten werden vor dem Ende der Long Running Transaction sichtbar, eventuell benutzte transaktionale Ressourcen werden jeweils bei Ende der Teilaktivitäten freigegeben. WS-BusinessActivity legt daher fest, dass ein Service-Aufruf innerhalb einer Long Running Transaction innerhalb eines sog. Business Tasks stattfinden muss, für den bei der Registrierung eine Kompensierungs-Aktion angegeben werden muss (in der Regel auch wieder ein Service-Aufruf, z.B. Storno einer Buchung). Bei Abbruch der Long Running Transaction kann der Coordination Service dann mit Hilfe der im Registration Service vorliegenden Informationen alle bereits abgeschlossenen Teilaktivitäten kompensieren.

In welcher Reihenfolge die Teilaktivitäten ausgeführt werden, ist nicht festgelegt, sondern hängt von der Business-Logik ab. Sequentielle Ausführung, parallele Ausführung oder Mischformen sind möglich.

4.4.3.4 Zusammenspiel der Spezifikationen

Eine genaue Beschreibung der Interaktionen von Applikationen, Coordination Services und Protokollen mit beispielhaften XML-Nachrichten findet sich in [8].

Es sind über den Registration Service theoretisch beliebige Schachtelungen von Aktivitäten mit den beiden Protokollen denkbar. Ein Beispiel zeigt Abb. 10. Eine Vorstellung darüber, wie sich das Coordination-Framework in das Web Services Referenzmodell einfügt, gibt Abb. 11.

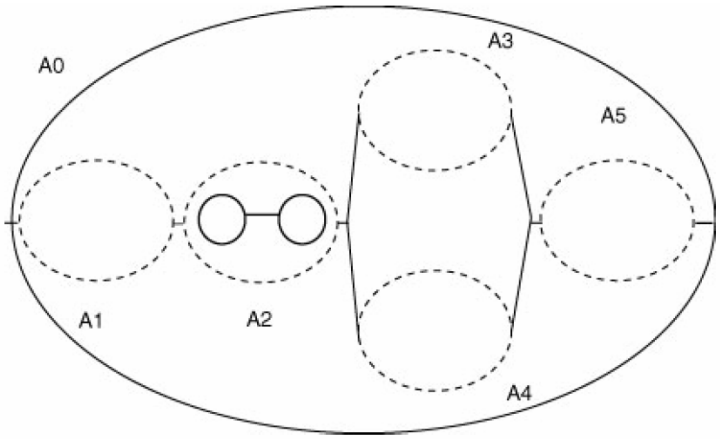

Abb. 10.: Kurzlebige Aktivitäten (A1-A5) innerhalb einer Long Running Transaction A0 (aus [8]). A2 benutzt verschachtelte Aktivtiäten, deren Ergebnisse vor Abschluss von A0 sichtbar werden und ggf. kompensiert werden müssen.

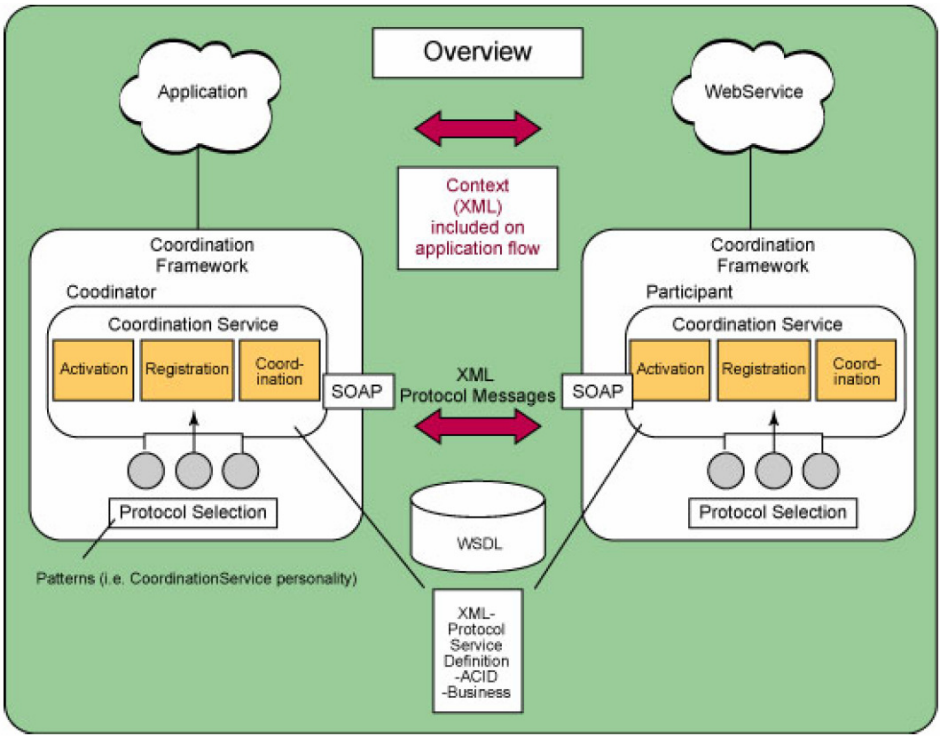

Abb. 11.: WS-Coordination und WS-Transaction im Web Services Referenzmodell (aus [8])

4.4.3.5 Offene Fragen

Die WS-Coordination und WS-Transaction Spezifikationen sind von 2002 bis November 2004 entwickelt worden. Trotzdem bleiben noch einige Fragen ungeklärt:

- Wie wird die Einhaltung der Protokolle sichergestellt? Hier existieren Ansätze unter Verwendung von WS-Policy ([7],[21]).

- Entsprechen die Spezifikationen, insbesondere die Protokolle, aktuellen Sicherheitsstandards? In diesem Punkt wird in den referenzierten Quellen – wenn überhaupt – lediglich auf WS-Security, WS-Trust, etc. ([7],[21]) verwiesen. Es ist also zu prüfen, ob diese Spezifikationen zusammenpassen.

- Wie vollständig sind die Spezifikationen bzw. gibt es funktionierende Referenzimplementierungen? Dies müsste geprüft werden.

Kapitel 5

GESCHÄFTSPROZESSE

5.1 Begriffsklärung und Einordnung

Geschäftsprozesse sind die Prozesse in einem Unternehmen. Ausgehend von diesen Prozessen, die z.B. im Rahmen von Business Process Reengineering aus reiner Geschäftssicht definiert werden, müssen für die Teile der Prozesse, die automatisiert werden sollen, entsprechende technische Prozesse (auch als Workflows bezeichnet) definiert werden. Die Definition von Geschäftsprozessen und das Umsetzen in technische Prozesse ist nicht Gegenstand dieser Arbeit. Für Geschäftsprozessmodellierung, Business Process Reengineering sowie Workflow-Management sei auf einschlägige Literatur verwiesen. Für die Abbildung von Geschäftsprozessen auf technische Prozesse (Workflows) finden sich Anregungen in [13].

Im Zusammenhang mit Transaktionsmanagement ist ein technischer Prozess (im Folgenden als Prozess bezeichnet) zunächst eine Abfolge von Aktivitäten, die in einem gemeinsamen Transaktionskontext ausgeführt werden. Im Zusammenhang mit Service Oriented Architectures werden diese Aktivitäten durch Services ausgeführt und der Prozess steuert die Aufrufe der Services. Der Prozess tritt dabei selbst als Service auf, kann also von anderen Services, Prozessen und Applikationen aufgerufen (gestartet, benutzt) werden.

Prozesse werden in SOAs auch als Process Centric Services und finden sich folglich im Process Layer (Abb. 12). Näheres dazu in [20].

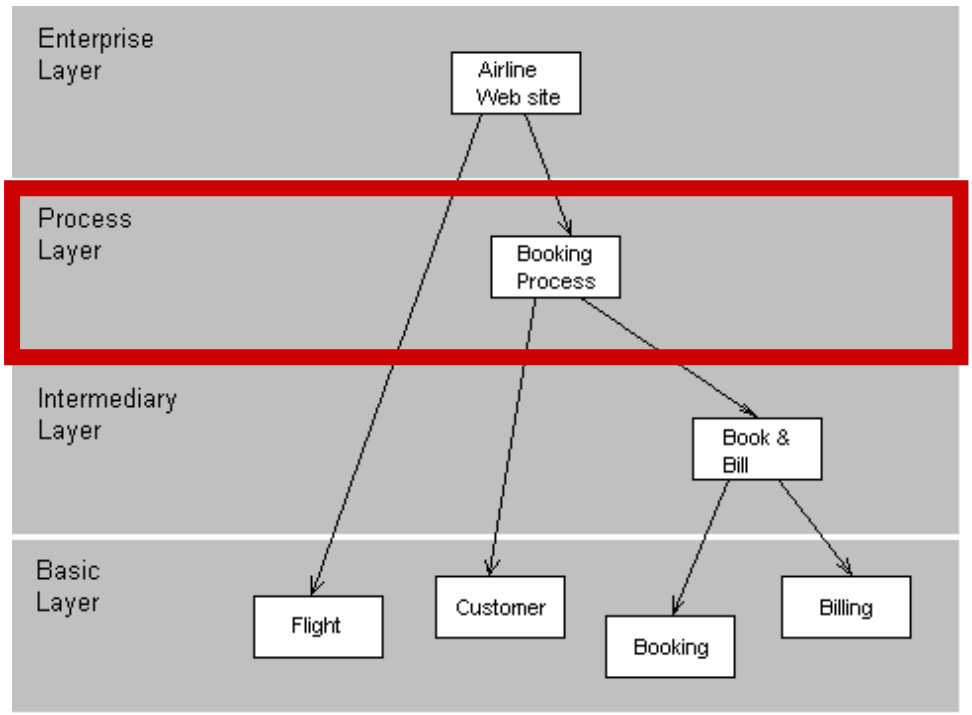

Abb. 12: Prozess-Services in den SOA-Schichten (aus [20])

Die Definition der Interaktionen zwischen Service und der hierzu verwendeten Protokolle in der Gesamtsicht wird als *Service Choreography* bezeichnet. Die Steuerung der Service-Aufrufe in einer Applikation oder innerhalb eines Prozesses wird als *Service Orchestration* bezeichnet. Informationen hierzu, insbesondere im Hinblick auf Verteilung, finden sich in [3] und [4].

Für das Transaktionsmanagement ist vor allem die Service Orchestration interessant, da durch sie die Business-Logik definiert wird, die über Erfolg oder Misserfolg von Aktivitäten entscheidet. Ausgehend vom Ausgang der Aktivitäten müssen dann, wie in Kapitel 4 beschrieben, Transaktionen mit COMMIT oder ROLLBACK abgeschlossen werden und im zweiten Fall bereits abgeschlossene Aktivitäten ggf. kompensiert werden (Storno).

5.2 Service Choreography

5.2.1 Prozess-Integration

Jeder Service, den ein Prozess aufruft, wird durch den Aufruf in den Prozess *integriert*. Hinter einem Service stehen in vielen Fällen bereits existierende Anwendungen, für die ein sog. Wrapper-Service implementiert wird, um die Anwendung in einem SOA-basierten System zu benutzen. Es existieren verschiedene Muster (Patterns) zur Integration existierender Anwendungen, die ihren Ursprung im Enterprise Application Integration (EAI) Paradigma haben.

IBM hat eine ganze Reihe solcher Muster entworfen und erfolgreich benutzt. Sven Stegelmeier geht im Rahmen der Veranstaltung „Anwendungen 1" auf die IBM-Muster ein [20], eine gute Dokumentation der Muster findet sich in [5] und [6].

Beispielsweise können Aktivitäten eines Prozesses sequentiell, parallel, oder in Mischform organisiert werden. Je nach Kombination ergeben sich immer wieder die Muster in Abb. 13, für die in [6] ausführlich Vorgehensweisen beschrieben werden.

Parallele Abläufe? → Serielle Abläufe?	NEIN	JA
JA	Serieller Prozess	Paralleler Prozess Work Flow
NEIN	Einzelne Nachricht	Broker Router

Abb. 13: Muster der Prozess-Integration (nach [6])

5.2.2 BPEL4WS und Transaktionen

Die Business Process Execution Language for Web Services (BPEL4WS, [9]) dient zur Modellierung der Service Orchestration für Prozesse. BPEL4WS ist eine XML-basierte Sprache, mit der Kontrollsemantik auf Mengen von

Interaktionen mit (anderen) Services, die Bestandteil des Prozesses sind, definiert und implementiert werden kann. Zur Ausführung von BPEL4WS-Prozessen wird eine spezielle Runtime Engine benötigt, z.B. BPWS4J ([19]) für die Programmiersprache Java. BPEL4WS-Prozesse sind selbst wieder Web Services, d.h. sie bieten WSDL-Interface Definitionen an und können von anderen Prozessen, inkl. anderer BPEL4WS-Prozesse, aufgerufen werden. Details zu BPEL4WS finden sich in [9].

Es existieren bereits etliche Werkzeuge, um BPEL4WS-Prozesse zu entwerfen und zu implementieren. Dies kann wie z.B. im IBM WebSphere Studio Application Developer 5.1 Integration Edition (WSAD 5.1 IE) mittels Flussdiagrammen geschehen, wie in Abb. 14 anhand eines Dummy-Beispiels gezeigt wird. Links ist der generierte XML-Code zu sehen, der dann in eine für die Runtime Engine ausführbare Sprache kompiliert wird, rechts der grafische Entwurf des Programmierers.

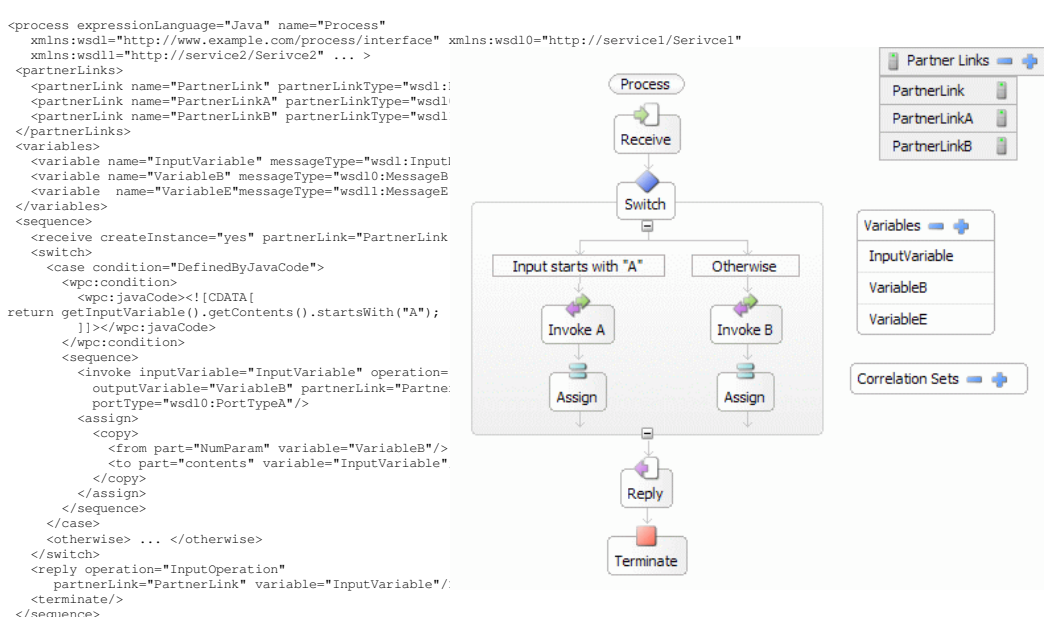

Abb. 14: BPEL4WS-Programmierung im IBM WSAD 5.1 IE

BPEL4WS arbeitet dabei direkt mit den WSDL-Definitionen der beteiligten Services.

Aufgrund der Orientierung an Standards (XML, WSDL, Java) und der breiten Unterstützung durch Werkzeuge wird BPEL4WS in der IT-Industrie mittlerweile verbreitet eingesetzt, weshalb es wichtig ist, BPEL4WS darauf hin zu untersuchen, ob es für Transaktionsmanagement geeignet ist bzw. ob es kompatibel zu existierenden Ansätzen ist.

Eine solche Untersuchung würde den Rahmen dieser Arbeit jedoch sprengen. Der Autor verweist diesbezüglich auf den Ausblick in Kapitel 6 und auf einen interessanten Ansatz in [3] zur Koordinierung von BPEL-Service-Aufrufen oder auch ganzen BPEL-Scopes (zu dem Begriff Scope siehe auch [9]) durch Anbindung von WS-Coordination und WS-Transaction mittels WS-Policy ([7]).

Kapitel 6

FAZIT UND AUSBLICK

Im dritten Semester des Studiengangs MSc Informatik an der HAW Hamburg soll (im Laufe des Wintersemesters 2005/2006) in den Veranstaltungen „Anwendungen 2“ und „Projekt“ das IT-System eines Ferienclubs im Hinblick auf neue bzw. interessante Entwicklungen in den Bereichen Mobilität und verteilte Systeme entworfen und (soweit möglich, prototypisch) implementiert werden.

Angesichts der aktuellen Entwicklungen im Bereich Systemarchitekturen, schlägt der Autor zusammen mit Sven Stegelmeier ([20]), Thies Rubarth ([21]) und Tobias Krause ([22]), die sich ebenfalls mit dem Thema SOA auseinandergesetzt haben, als Basis für die Systemarchitektur den Ansatz Service Oriented Architecture mit Web Services vor.

Für das Transaktionsmanagement wird die Kombination WS-Coordination und WS-Transaction vorgeschlagen, da hier schon Ansätze im Zusammenhang mit Web Services Security und der Anbindung an den Web Services „Protokoll-Stack“ mittels WS-Policy existieren (siehe 4.4.3.1, 5.2.2, [21], [7]). Eine geeignete Implementierung muss allerdings noch gefunden oder selber durchgeführt werden (siehe 4.4.3.5).

Das Business Transaction Protocol (BTP, siehe 4.4.1) weist mit Atoms und Cohesions ähnliche Konzepte auf wie WS-Transaction mit WS-AtomicTransaction und WS-BusinessActivity. Es wäre zu prüfen, ob das BTP Merkmale oder Ideen enthält, die auch für eine Implementierung von WS-* nützlich sein könnten. Immerhin ist die letzte Version des BTP genauso aktuell wie die letzte Version von WS-Coordination und WS-Transaction.

Das Tentative Hold Protocol (THP, siehe 4.4.2) wird wegen des hohen Kommunikationsaufwands als nicht geeignet für eine mobile, verteilte Anwendungslandschaft beurteilt.

Bezüglich Geschäftsprozessmodellierung und -implementierung (Service Choreography und Service Orchestration) mittels BPEL4WS oder anderen Modellierungssprachen sollte zunächst geprüft werden, ob dies im gegebenen Szenario sinnvoll und gewünscht ist und im Rahmen der zur Verfügung stehenden Zeit überhaupt machbar ist. Falls ja, würde der Autor vorschlagen, entweder den in [3] dargestellten Ansatz zur Kombination von BPEL4WS und WS-Coordination mittels WS-Policy zu prüfen oder gleich eine Sprache und Umgebung zu wählen, die Transaktionen (atomar und langlebig) berücksichtigt.

Der Autor plant, im dritten Semester folgende Aufgaben zu übernehmen:

- Ausarbeitung der Gesamtarchitektur zusammen mit den drei oben genannten Kommilitonen

- Evaluierung der Ansätze zum Transaktionsmanagement und Bereitstellung eines Frameworks inkl. Templates für transaktionale Services und Prozesse

- Falls (zeitl.) möglich, geeignet, und gewünscht: Versuch der Service Orchestration (und ggf. Choreography) mittels einer geeigneten Sprache unter der Bedingung, dass das Transaktionsmanagement damit funktioniert

LITERATUR, QUELLEN, REFERENZEN

1. Mark Little. *Service-oriented Computing: Transactions and Web Services.* Communications of the ACM Vol. 46 Issue 10, Oktober 2003
2. Benchaphon Limthanmaphon, Yanchun Zhang, *Web Service Composition transaction management.* Proceedings of the fifteenth conference on Australasian database – Vol. 27, Januar 2004
3. Stefan Tai, Ranja Khalaf, Thomas Mikalsen. *Composition of Coordinated Web Services.* Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware, Oktober 2004
4. Girish B. Chafle, Sunil Chandra, Vijay Mann, Mangala Gowri Nanda. *Decentralized orchestration of composite web services.* Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, Mai 2004
5. Mark Endrei et. al. *Patterns: Service Oriented Architecture and Web Services.* IBM ITSO Redbook SG246303, April 2004.
6. Martin Keen et. al. *Patterns: Serial and Parallel Processes for Process Choreography and Workflow.* IBM ITSO Redbook SG246306, April 2004
7. *Überblick und Quellensammlung für WS-* Spezifikationen.* http://www.ibm.com/developerworks/views/webservices/standards.jsp (25.05.2005)
8. Tom Freund, Tony Storey. *Transactions in the world of Web services.* IBM developerWorks, August 2002. http://www.ibm.com/developerworks/webservices/library/ws-wstx1/ und http://www.ibm.com/developerworks/webservices/library/ws-wstx2/ (25.05.2005)
9. Tony Andrews et. al. *Business Process Execution Language for Web Services (BPELAWS) Version 1.1 Mai 2003.* ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf (25.05.2005)
10. Tanenbaum, van Steen. *Distributed Systems.* Prentice Hall, 2002
11. Couloris, Dolimore, Kindberg. *Distributed Systems.* Pearson Education, 2001
12. Martin Gudgin. *Secure, Reliable, Transacted: Innovation in Web Services Architecture.* Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Juni 2004 (Kurze, gute Übersicht.)
13. Martin Henkel, Jelena Zdravkovic, Paul Johannesson. *Service-based Processes: Design for Business and Technology.* Proceedings of the 2nd international conference on Service oriented computing, November 2004 (Abbildung Geschäftsprozesse auf technische Prozesse, Workflows)

14. Anis Charfi, Mira Mezini. *Service Composition: Hybrid Web Service Composition: Business Processes Meet Business Rules.* Proceedings of the 2nd international conference on Service oriented computing, November 2004 (Verbindung von Prozesse mit Business Rules, Inferenz-Maschinen)
15. Fabien Baligand, Valérie Monfort. *Service security: A Concrete Solution for Web Services Adaptability Using Policies and Aspects.* Proceedings of the 2nd international conference on Service oriented computing, November 2004 (Beschreibung eines Modifizierten Java Class Loaders, der über Aspects zur Ladezeit Code zu Realisierung der WS-Policy und WS-Security Mechanismen in Web Services Klassen einfügt.)
16. *IBM developerWorks Web Services und SOA Ressourcen.* http://www.ibm.com/developerworks/webservices (25.05.2005)
17. *Business Transaction Protocol (BTP) bei OASIS*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=business-transaction (25.05.2005)
18. Jerry Roberts, Krishnamurthy Srinivasan et al. *Tentative Hold Protocol (THP).* W3C Note, November 2001 http://www.w3.org/TR/tenthold-1/ und http://www.w3.org/TR/tenthold-2/ (25.05.2005)
19. *BPWS4J (BPEL4WS Java Runtime Implementierung).* http://www.alphaworks.ibm.com/tech/bpws4j (25.05.2005)
20. Sven Stegelmeier. *Service Oriented Architecture und Service Bus.* Vortrag im Rahmen der Veranstaltung Anwendungen 1, 2. Semester MSc Informatik, HAW Hamburg, Mai 2005
21. Thies Rubarth. *Web Service Security.* Vortrag im Rahmen der Veranstaltung Anwendungen 1, 2. Semester MSc Informatik, HAW Hamburg, Juni 2005
22. Tobias Krause. *Service Repository und Service Lookup.* Vortrag im Rahmen der Veranstaltung Anwendungen 1, 2. Semester MSc Informatik, HAW Hamburg, Juni 2005