Technische Universität Kaiserslautern
FB Informatik
Lehrgebiet Datenverwaltungssysteme

Integriertes Seminar Datenbanken und Informationssysteme

Thema Nr. 5

Transaktionsmodelle

Betreuer: Prof. Dr. Dr. T. Härder

Christian Ziemann
c_zieman@informatik.uni-kl.de

7. Juli 2004

Inhaltsverzeichnis

1 Einleitung

1.1 ACID-Transaktionen

Transaktionen in homogenen Systemen unterliegen dem bereits bekannten ACID-Prinzip, das die Eigenschaften einer Transaktion widerspiegelt. Unter dem ACID-Prinzip verstehen wir:

- **A**tomicity
  Eine Transaktion ist die kleinste, nicht mehr weiter zerlegbare Einheit der Ablaufkontrolle. Dabei wird vorausgesetzt, dass entweder alle oder gar keine Änderungen der Transaktion festgeschrieben werden ("Alles-oder-Nichts"-Prinzip).

- **C**onsistency
  Eine Transaktion hinterlässt einen konsistenten Datenbank-Zustand, ansonsten wird sie komplett (Atomicity) zurückgesetzt. Dabei dürfen Zwischenzustände der Transaktion inkonsistent sein, allerdings muss der Endzustand die Integritätsbedingungen des Datenbankmodells erfüllen.

- **I**solation
  Nebenläufig (parallel, gleichzeitig) ausgeführte Transaktionen dürfen sich nicht gegenseitig beeinflussen.

- **D**urability
  Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank enthalten. Dabei kann die Wirkung einer erfolgreich abgeschlossenen Transaktion nur durch eine kompensierende Transaktion aufgehoben werden.

Um diese Eigenschaften sicherzustellen, existiert ein lokaler Mechanismus, verkörpert durch sog. TP-Monitore, die dafür sorgen, dass Transaktionen genau diese Eigenschaften erfüllen. Aber wie werden Transaktionen in einer heterogenen Umgebung wie dem Web abgewickelt, ohne das ACID-Prinzip zu verletzen? Welcher Mechanismus überwacht diese Transaktionen? Mit diesen Fragen wird sich meine Ausarbeitung auseinandersetzen; dabei werde ich wie folgt vorgehen:

1.2 Gliederung der Ausarbeitung

In dieser Ausarbeitung löse ich mich von den bisher bekannten "flachen Transaktionen" und werde das Verhalten sog. langlebiger Transaktionen in heterogenen Umgebungen beschreiben. Dabei gehe ich auf offen bzw. geschlossen geschachtelte Transaktionen ein und erörtere die jeweiligen Vor- bzw. Nachteile. Anschließend erläutere ich das Konzept der Geschäftstransaktionen sowie die dazugehörigen Besonderheiten. Schließlich füge ich diese

Prinzipien zusammen und zeige, wie diese Verfahren miteinander kooperieren.

2 Geschachtelte Transaktionen

2.1 Geschlossen geschachtelte Transaktionen

Wie oben bereits beschrieben geben die ACID-Eigenschaften die Merkmale von sog. "*flachen Transaktionen*" wieder. In einer heterogenen Umgebung werden allerdings keine flachen, sondern "*langlebige Transaktionen*" (siehe Kap. 3.1.) verwendet. Das Konzept der geschachtelten Transaktionen (nach [Hae] und [Wei])) erlaubt es nun, eine solch langlebige Transaktion intern in eine Hierarchie von Sub-Transaktionen zu zerlegen. Diese Zerlegung in Sub-Transaktionen erfolgt im Allgemeinen aufgrund der Modularisierung von Anwendungsfunktionen. So wird beim Aufruf einer Funktion eine Transaktion gestartet, die diese Funktion kapselt und nach Ausführung der Funktion wieder beendet wird. Es entsteht eine Hierarchie von Sub-Transaktionen. Die Funktionen können somit als Transaktionsprogramme realisiert werden (Begin- und Commit/Rollback-Operationen), welche als eigenständige ACID-Transaktion oder von unterschiedlichen Transaktionsprogrammen aus aufgerufen werden können.
Eine geschachtelte Transaktion kann, wie in Abbildung 1 gezeigt, durch einen Transaktionsbaum dargestellt werden. Dabei symbolisieren die Knoten eine Transaktion bzw. eine Sub-Transaktion. Die Kanten wiederum stellen die statische Aufrufbeziehung dar. Die Wurzel-Transaktion des Baumes wird als *Top-Level-Transaktion*, die inneren Transaktionen als Sub-Transaktionen, bezeichnet. Der Vorgänger einer Transaktion wird auch *Vater* genannt, die direkten Nachfolger *Kinder*. Jede Transaktion kann beliebig viele Kinder haben. Die Sub-Transaktionen eines Teilbaums mit Wurzel C bilden den so genannten Kontrollbereich von C (siehe Abbildung 1). Dies impliziert, dass der Kontrollbereich der Top-Level-Transaktion genau die gesamte geschachtelte Transaktion widerspiegelt.
Bei den geschachtelten Transaktionen gelten für die Top-Level-Transaktionen weiterhin die ACID-Eigenschaften. Für die Sub-Transaktionen müssen einige Kompromisse eingegangen werden, mehr dazu in Kapitel 2.1.2. und 2.2.
Geschachtelte Transaktionen haben nun einige Vorteile gegenüber den herkömmlichen flachen Transaktionen:

- Isolierte Rücksetzbarkeit von Sub-Transaktionen
  Tritt ein Fehler in Transaktion T ein, so müssen nicht alle Transaktionen zurückgesetzt werden, sondern zunächst nur T und dessen Nachfolger. Die Vater-Transaktion von T entscheidet was in diesem Falle zu tun ist.

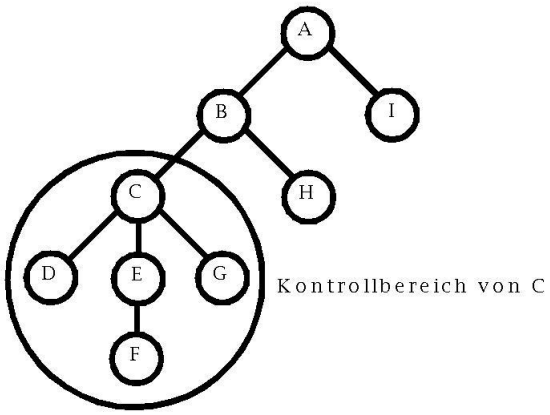

Abbildung 1: Hierarchie einer geschachtelten Transaktion

- Kontrollstruktur
  Eine Kontrollstruktur zum parallelen sowie verteilten Ausführen von Sub-Transaktionen auf unterschiedlichen Rechnern wird unterstützt.

- Kontrollierteres Zurücksetzen
  Wird eine Transaktion T abgebrochen, so müssen alle Sub-Transaktionen im Kontrollbereich von T ebenfalls abgebrochen werden. Wenn die Top-Level-Transaktion scheitert, bedeutet dies, dass der gesamte Baum zurückgesetzt werden muss.

2.1.1 Intra-Transaktionsparallelität

Bei der Ausführung der Sub-Transaktionen ist zwischen zwei verschiedenen Arten von *Intra-Transaktionsparallelität* (Parallelität innerhalb einer geschachtelten Transaktion) zu unterscheiden: *Vater/Kind-Parallelität* und *Geschwister-Parallelität*. Bei der Vater/Kind-Parallelität kann eine Transaktion parallel zu ihren Kindern ausgeführt werden. Während in der zweiten Variante die Geschwister nebenläufig ausgeführt werden können. Weiter wird zwischen verschiedenen Varianten dieser Parallelitäten unterschieden (siehe [VLDB]):

- *Weder Vater/Kind- noch Geschwister-Parallelität*
  Zu jedem Zeitpunkt in der geschachtelten Transaktion ist nur eine

Transaktion aktiv. Dies bedeutet, dass keine Art von Intra-Transaktionsparallelität vorhanden ist. Alle Transaktionen werden sequentiell ausgeführt.

- *Nur Geschwister-Parallelität*
  Nur Geschwister-Transaktionen können parallel ausgeführt werden. Eine Transaktion wird niemals mit einer ihrer Vorgänger-Transaktion zeitgleich ausgeführt. Diese Art von Parallelität ermöglicht es Transaktionen, gleiche Objekte mit ihren Vorfahren zu nutzen, ohne jegliche Parallelitätskontrolle dieser Objekte.

- *Nur Vater/Kind-Parallelität*
  Vater-Transaktionen können zeitgleich mit ihren Kinder-Transaktionen ausgeführt werden. Geschwister-Transaktionen werden in diesem Modell nicht parallel ausgeführt. Hier wird eine Parallelitätskontrolle in dem Sinne benötigt, dass Transaktionen im selben Pfad synchronisiert werden.

- *Vater/Kind-Parallelität und Geschwister-Parallelität*
  Bei dieser Variante wird jede Art von Intra-Transaktionsparallelität zugelassen. So können z.B. alle Transaktionen in einem Baum parallel aktiv sein. Dies hat allerdings zur Folge, dass hier die strengste Art der Parallelitätskontrolle benötigt wird.

2.1.2 ACID in geschlossen geschachtelten Transaktionen

Um Zusammenarbeit innerhalb geschachtelten Transaktionen zu realisieren (insbesondere im Hinblick auf die ACID-Eigenschaften), sind einige Regeln zu beachten, die im Folgenden erklärt werden. Wir gehen hier von einer Vater/Kind-Parallelität und Geschwister-Parallelität aus.

- *Rücksetzregel*
  Wird eine Transaktion zurückgesetzt, so werden alle ihre Sub-Transaktionen (nicht ihre Vater-Transaktion), unabhängig von ihrem Bearbeitungsstatus, zurückgesetzt. Diese Regel wird rekursiv angewendet.

- *Commit-Regel*
  Das (lokale) Commit einer Sub-Transaktion macht ihre Ergebnisse der Vater-Transaktion sichtbar. Das endgültige Commit einer Sub-Transaktion erfolgt nur, wenn alle ihre Vorfahren ihr Commit ausgeführt haben. Das bedeutet, dass eine geschachtelte Transaktion nur dann ihr Commit ausführt, wenn die Top-Level-Transaktion ihr Commit ausführt. Danach werden alle Änderungen dauerhaft gespeichert.

- *Sichtbarkeits-Regel*
  Alle Änderungen einer Sub-Transaktion werden nach ihrem Commit

der Vater-Transaktion sichtbar gemacht. Alle Objekte, die einer Vater-Transaktion sichtbar gemacht wurden, können den Sub-Transaktionen zugänglich gemacht werden. Änderungen einer Sub-Transaktion sind für gleichzeitig aktive Geschwister-Transaktionen nicht sichtbar.

Aus diesen Regeln sind die Bestandteile des ACID-Prinzips ableitbar, die in geschlossen geschachtelten Transaktionen gelten:
Aus den Commit- und Rücksetzregeln folgt die Atomarität der Sub-Transaktionen (vgl. [Tas]). Aufgrund der Sichtbarkeits-Regel ist die Eigenschaft I (Isolation) für parallel laufende Sub-Transaktionen gewährleistet. Allerdings besteht aufgrund der Abhängigkeit der Sub-Transaktionen zu ihrem Vorgänger in der Transaktionshierarchie keine Dauerhaftigkeit mehr (Änderungen können noch nach einem lokalen Commit zurückgesetzt werden). Die Eigenschaft C (Consistency) kann bezüglich der Gesamtaufgabe spätestens nach Abschluss der Top-Level-Transaktion erfüllt werden (bezüglich der zu realisierenden lokalen Funktion auch hinsichtlich Sub-Transaktionen).
Das bedeutet also, dass nur die Eigenschaften A und I für Sub-Transaktionen gelten müssen.

2.1.3 Synchronisation geschlossen geschachtelter Transaktionen

Um die oben beschriebenen Regeln zur Zusammenarbeit in geschlossen geschachtelten Transaktionen zu realisieren, beziehen wir uns auf ein Grundprinzip, das die *Vererbung von Sperren* (siehe [Hae]) zulässt. Hierbei erben Vater-Transaktionen die Sperren als *Platzhalter-* oder *Retained-Sperren.* Zusätzlich zu den bereits bekannten Lese- und Schreibsperren (RX-Verfahren) werden zwei weitere Arten von Retained-Sperren verwendet, die r-R- und r-X-Sperren.
Das Protokoll unterliegt dabei folgenden Regeln:

- Transaktion T kann eine X-Sperre anfordern, falls
  - keine andere Transaktion eine X- oder R-Sperre auf dem Objekt besitzt,
  - alle Transaktionen, die eine r-R- oder r-X-Sperre auf dem Objekt halten, Vorfahren von T sind.
- Transaktion T kann eine R-Sperre anfordern, falls
  - keine andere Transaktion eine X-Sperre auf dem Objekt besitzt,
  - alle Transaktionen, die eine r-X-Sperre auf dem Objekt halten, Vorfahren von T sind.
- Beim Commit von Transaktion T erbt die Vater-Transaktion sowohl die Platzhalter-Sperren als auch die regulären Sperren. Dabei werden reguläre Sperren als Platzhalter-Sperren vererbt (R→r-R; X→r-X).

- Wird Transaktion T abgebrochen, so werden alle von T gehaltenen Sperren wieder freigegeben. Die Sperren der Vater-Transaktion bleiben bestehen.

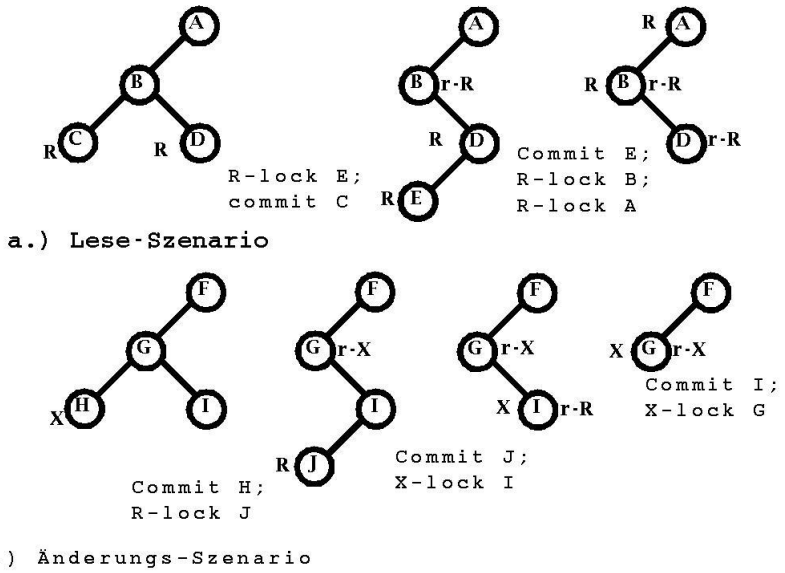

Abbildung 2: Beispiel von Vergabe und Vererben von Sperren

Somit impliziert eine r-R-Sperre in Transaktion T, dass nur in dem Kontrollbereich von T für das entsprechende Objekt eine Sperre erworben werden kann. Ebenso impliziert eine r-R Sperre, dass nur im Kontrollbereich von T eine X-Sperre bewilligt wird (siehe Abbildung 2). Somit ist der Kontrollbereich zur Recovery und Synchronisation eine maßgebende Einheit. Besitzt eine Transaktion bereits eine r-R- oder r-X-Sperre für ein Objekt und erbt erneut eine Sperre für dasselbe Objekt, wird nur die mächtigere behalten, d.h., falls eine r-X-Sperre bereits existiert und eine r-R-Sperre geerbt wird, bleibt die r-X-Sperre erhalten und die r-R-Sperre wird nicht beachtet. Sollte dagegen eine r-R-Sperre auf dem Objekt liegen und eine r-X-Sperre wird geerbt, so erfolgt ein "Upgrade" der r-R-Sperre auf eine r-X-Sperre.
In diesem Verfahren stellt sich allerdings jetzt das Problem, dass Sub-Transaktionen niemals auf Objekte der Vater-Transaktion zugreifen könnten. Um dieses Problem zu lösen, ist eine Erweiterung des Sperrverfahrens nötig: *Weiterreichen von Sperren.*
Lastet auf eine Transaktion T z.B. eine X-Sperre, so wird diese Sperre an die Sub-Transaktion, die dasselbe Objekt ändern möchte, weitergereicht. Die Sperre der Transaktion T ändert sich zu einer r-X-Sperre und verhindert so-

mit, dass das Objekt weiter von T verändert werden kann. Danach wird regulär, wie oben beschrieben, fortgefahren.
Das Verfahren der geschlossen geschachtelten Transaktionen unterstützt somit die Intra-Transaktionsparallelität auf gleiche, geänderte Objekte. Allerdings aufgrund der Tatsache, dass die gesamte geschlossen geschachtelte Transaktion vollkommen isoliert abläuft, erlaubt sie keine Inter-Transaktionsparallelität auf solche Objekte. Inter-Transaktionsparallelität wird neben Intra-Transaktionsparallelität von offen geschachtelten Transaktionen unterstützt.

2.2 Offen geschachtelte Transaktionen

Sub-Transaktionen in offen geschachtelte Transaktionen geben nun ihre isolierte Rücksetzbarkeit auf. Der Unterschied bei diesem Verfahren zum Vorherigen besteht darin, dass die Änderungen einer Sub-Transaktion augenblicklich nach ihrem Commit für die Außenwelt sichtbar werden. Die Ressourcen werden also frühzeitig für andere Transaktionen freigegeben; es lasten keine weiteren Sperren auf den Objekten (vgl [Hae]). Daraus resultiert neben einer Intra-Transaktionsparallelität auch eine Inter-Transaktionsparallelität auf Subtransaktionen gleichen Levels. Diese begründet sich in der Objekthierarchie geschachtelter Transaktionen. So liegen in so genannten *Mehrebenen-Transaktionen* eine feste Anzahl von Schichten mit bestimmten Operationen vor. Operationen der Ebene i werden jeweils durch Operationen der Ebene i-1 realisiert. Diese Zerlegung ist möglich, da z.B. eine mengenorientierte DB-Operation als Folge von Satzoperationen und jede Satzoperation durch mehrere Seitenoperationen realisiert werden kann. Hierbei ist zu beachten, dass die Ausführung jeder Operation atomar im Rahmen einer Sub-Transaktion ist. Für die Gesamt-Transaktion werden weiterhin die ACID-Eigenschaften angestrebt. Die Sperren werden jedoch nur für die Operationen (Sub-Transaktionen) auf der obersten Ebene bis zum Transaktionsende gehalten. Für Operationen darunterliegender Schichten dagegen erfolgt wie oben bereits beschrieben eine vorzeitige Sperrfreigabe am Ende der direkt übergeordneten Sub-Transaktion, unabhängig vom Ausgang der gesamten Transaktion.

2.2.1 Synchronisation

Aufgrund der frühzeitigen Freigabe von Sperren in offen geschachtelten Transaktionen kann es zu Synchronisationsproblemen kommen, daraus resultiert eine Verletzung der Serialisierbarkeit der Gesamttransaktion. Es muss somit eine geeignete Zusatzmaßnahme zur Synchronisation ergriffen werden (vgl. z.B. Contracts, bei denen die Synchronisation mittels Invarianten geregelt wird). Es besteht insbesondere das Problem der "schmutzigen Änderungen", woraus resultiert, dass ohne zusätzliche Synchronisationsmaßnahmen eine

vorzeitige Sperrfreigabe höchstens für solche Sub-Transaktionen vertretbar ist, welche im Anwendungskontext abgeschlossene Teilschritte verkörpern.

2.2.2 Recovery

Durch die frühzeitige Freigabe von Änderungen können andere Transaktionen gleichen Levels auf die freigegebenen Objekte zugreifen und eigene Änderungen vornehmen. Dies impliziert, dass es jetzt nicht mehr möglich ist, eine bereits abgeschlossene Sub-Transaktion mit Hilfe der zustandsorientierten Undo-Recovery zurückzusetzen. Die bereits festgeschriebenen Änderungen müssen mittels anwendungsspezifischen, *kompensierenden Sub-Transaktionen* rückgängig gemacht werden. Hierbei ist zu beachten, dass der neue DB-Zustand im Allgemeinen nicht mit dem ursprünglichen übereinstimmt. So werden beispielsweise bei Kontenbewegungen die Änderungen, die zwischen Transaktion und kompensierender Transaktion erfolgen, nicht beachtet. Zusätzlich ist zu beachten, dass die kompensierenden Transaktionen nicht zwingend den genauen Umkehrprozess einer Transaktion realisieren müssen (die Stornierung einer Platzreservierung beinhaltet neben der Platzfreigabe zusätzliche Gebühren).
Durch die kompensationsbasierte Fehlerbehandlung entstehen allerdings grundsätzliche Probleme:

- Hoher Aufwand aufgrund bereitzustellender Kompensationsprogramme
  Vom DB-Anwender muss ein Kompensationsprogramm für jede aufgerufene Sub-Transaktion bereitgestellt werden. Dies impliziert eine hohe Flexibilität, allerdings ebenfalls einen enormen Aufwand. Zusätzlich beinhalten die Kompensationsprogramme eine hohe Mitverantwortung des Programmierers an der korrekten Kompensation.

- Scheitern der Kompensationen
  Die Kompensationen dürfen nicht scheitern, da ansonsten die Rücksetzung einer Sub-Transaktion nicht erfolgen kann. Da auch nach mehrfacher Wiederholung von Kompensationen ein erfolgreicher Abschluss nicht garantiert werden kann, bleibt hier eine Sicherheitslücke. In solchen Fällen erfolgt im Allgemeinen eine manuelle Fehlerbehandlung.

- Nicht kompensierbare Operationen
  Es können nicht alle Operationen kompensiert werden, wie z.B. das Bohren eines Loches. Solche Operationen müssen immer bis zum Schluss der Gesamttransaktion verzögert werden.

Aus den Erkenntnissen der letzten Kapitel folgt nun, dass die Top-Level Transaktion serialiesierbar ist. Die einzelnen Sub-Transaktionen hingegen sind nur L2L-serialisierbar (Level-to-Level-serialisierbar), d.h., nur Sub-Transaktionen auf der gleichen Ebene können serialisiert werden.

2.2.3 Sagas

Eine besondere Form der offen geschachtelten Transaktionen sind die so genannten Sagas (nach [WfWS] und [Hae]). Bei diesem Verfahren werden die kompensierenden Transaktionen vorab definiert. Somit kann eine Folge von bereits erfolgreichen Transaktionen vom System rückgängig gemacht werden. Dabei ist zu beachten, dass wie bei offen geschachtelten Transaktionen die Kompensationen anwendungsspezifisch sind, d.h., es ist nicht immer möglich, eine Transaktion mittels eines genauen Umkehrprozesses zu kompensieren (siehe Kap. 2.2.2).
Ein Saga ist eine Sequenz von Transaktionen:

$$[(T_1, C_1), (T_2, C_2), ..., (T_n, C_n)]$$

Hierbei beschreiben die $T_1, T_2, ..., T_n$ die eigentlichen Transaktionen, und die $C_1, C_2, ..., C_n$ stellen die dazugehörigen kompensierenden Transaktionen (anwendungsspezifisch) dar. C_i ist somit für T_i die kompensierende Transaktion. Wird nun ein solches Saga ausgeführt, werden nur die Transaktionen T_1 bis T_n sequentiell ausgeführt, beruhen somit auf dem ACID-Prinzip und sind demnach serialisierbar. Erst wenn ein Abort ausgeführt wird, führt die aktuelle Transaktion ein Undo durch und die bereits abgeschlossenen Transaktionen werden mittels ihrer zugehörigen C_i in umgekehrter Reihenfolge kompensiert. Bsp.: $T_1, ...T_4$ wurden erfolgreich ausgeführt. In Transaktion T_5 tritt nun ein Fehler auf. Die Änderungen von T_5 werden mit Hilfe einer Undo-Operation rückgängig gemacht. Danach werden die kompensierenden Transaktionen $C_4, ..., C_1$ ausgeführt und somit alle Änderungen, die das Saga bis dato für die Umgebung freigegeben hat, kompensiert.

2.2.4 Contracts

Das Transaktionskonzept der Contracts ist eine Erweiterung des Sagas-Prinzips (ACID-Eigenschaften der Sub-Transaktionen bleiben erhalten). Hierbei wird zwischen der Ablaufkontrolle (*Skript*) und den eigentlichen Sub-Transaktionen (*Steps*) unterschieden (vgl. [WfWS]). Anders als bei Sagas existiert in dem Contract-Konzept eine reiche Kontrollstruktur, die es ermöglicht, Sequenzen, Verzweigungen, Parallelität etc. darzustellen (vgl. Abbildung 3). Da es sich bei diesem Verfahren um eine Art der offen geschachtelten Transaktionen handelt, werden kompensierende Transaktionen der einzelnen Steps benötigt, um eventuell einen Contract wieder zurückzusetzen. Die Synchronisation der einzelnen Steps erfolgt hierbei über Invarianten, die der Programmierer angibt. Diese Invarianten beschreiben Eingangsbedingungen, die von einzelnen Steps überprüft werden. Sollte eine dieser Invarianten für einen Step nicht mehr erfüllt sein, wird dieser kompensiert. Hier lässt sich auch der Zusammenhang der einzelnen kompensierenden Abhängigkeiten erkennen: Angenommen, bei der Kompensation von Step

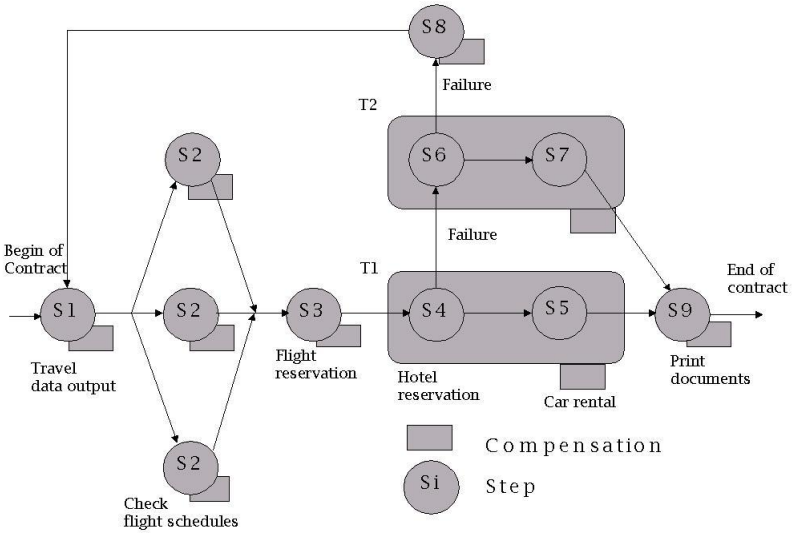

Abbildung 3: Beispiel eines Contracts

S_5 wird eine Invariante derart verändert, dass sie nicht mehr erfüllt ist. Setzt Step S_4 voraus, dass diese Invariante erfüllt ist, muss dieser Step ebenfalls kompensiert werden. Nach der Kompensation wird versucht, nach vorne weiterzuverfahren. Dies setzt allerdings ein persistentes Kontext-Management voraus, das die Informationen des alten Zustandes (im Bsp. der Zustand vor S_4) enthält. Dieser kann erst gelöscht werden, wenn der gesamte Contract ein Commit ausgeführt hat. Eine Kompensation ist, wie in Kapitel 2.2 bereits erörtert, anwendungsspezifisch, kann somit vom reinen Umkehrprozess abweichen. Dabei ist es möglich, eine Kompensation ebenfalls mit einem Abort zu beenden, sie muss allerdings danach wiederholt werden. Hierbei ist zu beachten, dass keine automatische Behandlung des Falles stattfindet, falls eine Kompensation auch nach k Wiederholungen nicht erfolgreich beendet worden ist. Entscheidet sich ein Benutzer, den gesamten Contract zurückzusetzen, müssen alle laufenden Steps abgebrochen werden und es muss sichergestellt werden, dass keine weiteren Steps gestartet werden.

2.2.5 Stratifizierte Transaktionen

Bei dem Prinzip der stratifizierten Transaktionen (nach [WfWS]) wird eine Transaktion T in Teiltransaktionen $T_1, ..., T_n$ zerlegt. Jede dieser T_i wird mit einer persistenten Warteschlange Q_i verbunden, aus der sie Anforderungen erhält, bestimmte Operationen auszuführen. Bei diesem Verfahren ist es

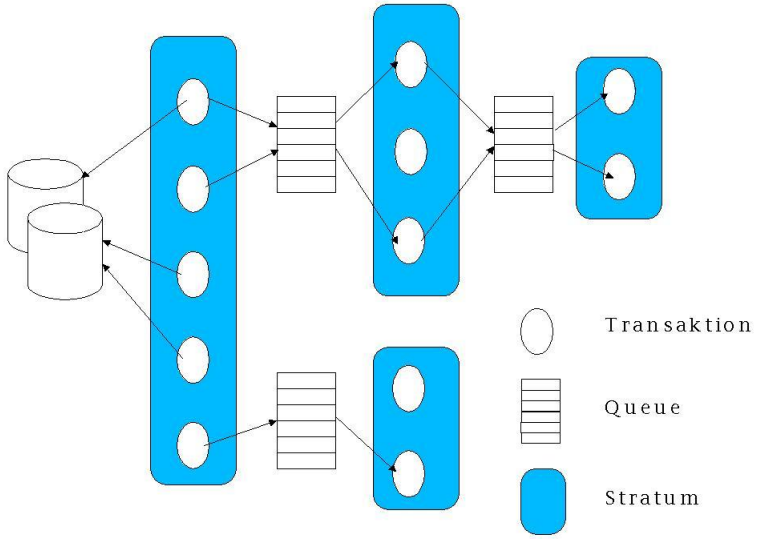

Abbildung 4: Stratifizierte Transaktion

wichtig, dass alle manipulierbaren Ressourcen (insbesondere die Nachrichten) wiederherstellbar sind. Das eigentliche Verfahren geht dann wie folgt vor:
Mit dem Ziel, eine erhöhte Nebenläufigkeit zu erreichen, werden die T_i, die zu einem gemeinsamen Ende kommen sollen, in so genannte Strata $(S_1, ...S_m)$ gruppiert (siehe Abbildung 4). Dabei ist darauf zu achten, dass eine disjunkte Zerlegung der T_i erfolgt, d.h., keine der Teiltransaktionen darf in einem oder mehreren Strata doppelt vorkommen. Die Summe der Teiltransaktionen in den einzelnen Strata muss wieder die Ausgangstransaktion ergeben.
Wird eine solche stratifizierte Transaktion ausgeführt, laufen die Transaktionen in einem Stratum parallel ab. Sind alle Teiltransaktionen erfolgreich abgeschlossen, führt das Stratum ein Commit durch, vorausgesetzt das Vater-Stratum hat bereits sein Commit ausgeführt. Scheitert ein solches Stratum, werden alle beinhaltenden Transaktionen mittels Undo-Operationen rückgängig gemacht und es wird erneut versucht, mittels der Informationen in der persistenten Eingangswarteschlange, das Stratum auszuführen. Dieses Verfahren ist nicht für ein Scheitern der gesamten Transaktion T ausgelegt, d.h., sollte ein Stratum selbst nach mehrfachem Aufruf nicht zu einem Commit kommen, müsste der Benutzer entscheiden, was zu tun ist.
Dieses Verfahren beruht auf dem 2-PC-Protokoll, weshalb auch alle Ressourcen in der globalen Transaktion eingebunden sind. Der Vorteil bei diesem Verfahren ist nun, dass gerade aufgrund des verwendeten 2-PC-Protokolls

ein geringeres Nachrichtenaufkommen als bei einer globalen Transaktion vorhanden ist. Ein weiterer positiver Effekt ist der hohe Grad an Nebenläufigkeit, Sperren werden früher freigegeben, und Ressourcen können so schneller genutzt werden.

3 Geschäftstransaktion

3.1 Geschäftsprozesse

Um Transaktionen in einer heterogenen Umgebung zu realisieren, ist von den Eigenschaften in einer homogenen Umgebung abzusehen. So sind Transaktionen im Web so genannte *langlebige Transaktionen*, d.h., sie können sich über Wochen, Monate oder sogar Jahre hinziehen (vgl. [WWW]). Dabei stellt der *Geschäftsprozess* eine Menge von logisch zusammenhängenden Aufgaben dar, um ein definiertes Ergebnis zu realisieren, vereinfacht ausgedrückt also eine Abfolge von Prozessen, die zu einem gewünschten Resultat führen. Wobei dieses Resultat sowohl eine Leistung (Überweisung, Buchung etc.) als auch ein physisches Produkt (Auto als Resultat des Prozesses "Fertigung") sein kann. Eine *Aktivität* in diesem System ist nun ein Element, das eine definierte Funktion realisiert. Dies wäre z.B. im Geschäftsprozess "Fertigung" die Aktivität "Scheibe einsetzen". Dabei können Aktivitäten simpler Natur sein (s.o. Scheibe einsetzen) oder sie können komplexerer Natur sein und so z.B. die Ausführung anderer Prozesse und Aktivitäten koordinieren. Um in einem Geschäftsprozess eine Logik, d.h. Bedingungen und Regeln einzubringen, werden so genannte *Geschäftsregeln* verwendet. Diese Geschäftsregeln sind eine kompakte Ansammlung von Statements, die in einer Anwendung gelten müssen.
Im Web ist eine Kommunikation zwischen verschiedenen Geschäftsprozessen notwendig, man denke nur einmal an das Fließen von Geld. Deswegen unterstützen *Geschäftsprozess-Management-Systeme* heutzutage die Definition (*definition*), Ausführung (*execution*) und Kontrolle (*monitoring*) von langlebigen Transaktionen. Dies ist notwendig, da diese Transaktionen mit verschiedenen Geschäftsprozessen interagieren, also nicht mehr nur von einem Geschäftsprozess überwacht werden müssen.
Aufgrund der Tatsache, dass das Web nicht wie herkömmliche Workflow-Modelle aufgebaut, sondern verteilter und lose gekoppelter Natur ist, sind globale Kontrollmechanismen (die die Aufgaben Koordination und Überwachung übernehmen) nicht anwendbar. Gründe hierfür lassen sich im Folgenden finden:

- Traditionelle Workflow-Modelle basieren auf einer kontrollierbaren, fest verbundenen Umgebung, was durch das verwendete Protokoll als gegeben vorausgesetzt wird.

- Traditionelle Workflow-Modelle unterscheiden nicht zwischen interner Implementierung und externer Protokoll-Beschreibung.
- In traditionellen Workflow-Modellen wird vorausgesetzt, dass die verschiedenen Teilnehmer vorab bekannt sind und nicht dynamisch hinzu kommen können.

In einer derartigen Umgebung gibt kein lokales Protokoll die Kontrolle oder Verwaltung der eigenen Ressourcen an ein übergeordnetes Protokoll ab. Daraus folgt, dass ein Mechanismus (ein Protokoll) benötigt wird, der die Koordination der teilnehmenden lokalen Protokolle organisiert. Dieses Protokoll muss mittels asynchroner Kommunikation die einzelnen Protokolle verbinden und sicherstellen, dass das Resultat der verschiedenen Prozesse (autonom von den lokalen Protokollen gestartet) korrekt ist. Weiterhin muss gewährleistet sein, dass durch das Warten auf einzelne Ergebnisse nicht das gesamte System gefährdet ist. Ein Protokoll, welches genau diese Eigenschaften erfüllt, ist beispielsweise CORBA oder DCOM.

3.2 Die Bedeutung eines Business Transaction Framework

An einen Geschäftsprozess werden Anforderungen wie asynchrone Interaktion, Flusskontrolle, Geschäftstransaktionsaktivität und Geschäftstransaktionsmanagement gestellt. Um diese zu gewährleisten, müssen einige Operationen im Web transaktionale Eigenschaften haben und wie eine eigenständige Einheit, als Teil der Geschäftstransaktion, behandelt werden. Eine *Geschäftstransaktion* bewirkt eine dauerhafte Veränderung im System, die von einer wohl definierten Geschäftsfunktion gesteuert wird. Wie bei einer gewöhnlichen Transaktion hinterlässt die Geschäftstransaktion einen konsistenten Zustand in den beteiligten Datenbanken. Geschäftsprozesse können auch von einer komplexeren Natur sein, wie z.B. die Koordination beim Umbuchen von Geld in verschiedenen heterogenen und autonomen Systemen. Bei einem derartigen Szenario sind mehrere selbstständige Workflows beteiligt, die miteinander interagieren müssen, um das gewünschte Resultat zu erzielen. Hier wird ein sog. *Business Transaction Framework (BTF)* benötigt, der die Zusammenarbeit der teilnehmenden Parteien koordiniert und dazu führt, dass die autonom handelnden Teilnehmer letzten Endes eine einheitliche Geschäftstransaktion realisieren. Weiterführend sollte ein BTF sich auf eine Partei dahingehend verlassen können, dass diese die Fähigkeit besitzt, komplexe Aktivitäten dynamisch zu erzeugen. Das bedeutet, ein Teilnehmer kann den Datenfluss verändern, Fehlermechanismen (exception / error handling) nutzen oder einen Prozess terminieren. Deswegen muss ein BTF einem Teilnehmer dahingehend vertrauen können, dass dieser keine unzulässigen Veränderungen im System herbeiführt. Ein BTF bietet die folgenden Komponenten an:

- Unterstützung von *Geschäftstransaktionsmodellen*,
  um langlebige Transaktionen, herkömmliche Transaktionen, Exceptionhandling-Mechanismen, kompensierende Aktionen und Atomarität anzubieten.

- Unterstützung von *Koordinationsprotokollen*,
  um verteilte Anwendungen in der Web-Umgebung zu koordinieren. Der BTF sollte dabei in der Lage sein, einer Aktivität den benötigten Kontext zu bieten, um mit anderen Services zu interagieren und an anderen Coordination Protocols teilzunehmen.

- Unterstützung von *Geschäftsprotokollen*,
  um Informationen zwischen beteiligten Parteien auszutauschen. Das bedeutet ein BTF (oder genauer: das unterstützte Geschäftsprotokoll) definiert die Folge, in welcher die Teilnehmer ihre Nachrichten an andere Parteien senden oder empfangen. Außerdem legt das Geschäftsprotokoll fest, welche Informationen übertragen werden müssen, also welche Informationen in der Nachricht enthalten sein müssen.

3.3 Geschäftstransaktionsmodell

Wie oben bereits erwähnt, basieren Transaktionen in homogenen Systemen auf den ACID-Eigenschaften. Allerdings sind diese Eigenschaften in einer Umgebung wie dem Web aufgrund ihrer flachen Struktur nicht anwendbar. Sind Transaktionen hier doch komplexerer Natur, so werden mehrere Parteien und verschiedene Unternehmen in eine Transaktion einbezogen, und eine Transaktion kann sich über einen längeren Zeitraum hinwegziehen. Genauer gesagt, beinhalten solche Geschäftstransaktionen Abläufe wie: Verhandlungen, Verpflichtungen, Verträge, Logistik, verschiedene Bezahlungsinstrumente und Exception Handling.
Geschäftstransaktionen haben deshalb folgende Eigenschaften:

- Normalerweise repräsentieren sie eine Funktion ähnlich dem Geschäft, z.B. Versorgungskette.

- Sie können mehrere Parteien / Unternehmen einbeziehen, die separate Ressourcen verwalten und einbringen.

- Sie definieren Kommunikationsprotokollverbindungen, die die Adresse des Web Service darstellen, behalten sich aber vor, Nachrichten auch über andere Wege zu verschicken.

- Sie sollten auf einem formalen Abkommen basieren (z.B. ebXML).

In Bezug auf diese Eigenschaften ist bereits festzustellen, warum bei dieser Art von Transaktionen die ACID-Eigenschaften nicht erfüllbar sind. So

können beispielsweise Transaktionen nicht isoliert ablaufen, weil die verschiedenen Parteien eine einzelne Transaktion realisieren und so Teile der Transaktion aufeinander aufbauen (siehe Kapitel 2). Bis jetzt wurde die Umgebung und der korrekte Ablauf einer Transaktion mit Hilfe eines BTF beschrieben. Jedoch kann bei einer Geschäftstransaktion nicht einheitlich vom selben Typ ausgegangen werden, es wird zwischen *atomaren Geschäftstransaktionen* und *langlebigen Geschäftstransaktionen* unterschieden: Die atomaren Geschäftstransaktionen zeichnen sich durch das Alles-oder-Nichts-Prinzip aus, d.h., alle teilnehmenden Parteien führen gemeinsam ein Commit aus und machen ihre Änderungen in ihren Datenbanken dauerhaft, oder keiner der Teilnehmer führt ein Commit aus. Falls ein Fehler auftritt, müssen die Operationen mittels einer Undo-Operation korrigiert (Rollback) werden; dabei kann die atomare Transaktion geschachtelt sein (geschlossen geschachtelt, siehe Kap. 2.1).
Die langlebigen Geschäftstransaktion hingegen basiert auf dem offen geschachtelten Modell, d.h., die teilnehmenden Parteien müssen nicht gemeinsam zu einem Commit kommen. Ein selektives Commit wird bei diesem Verfahren unterstützt, was bedeutet, dass die beteiligten Transaktionen kein gemeinsames Resultat liefern müssen. Einige dieser Transaktionen können erfolgreich abgeschlossen werden, wohingegen andere Transaktionen nicht zu einem Commit kommen.

3.4 Atomaritätsebenen einer Geschäftstransaktion

In einer Umgebung wie dem Web existieren mehrere, aufeinander aufbauende Ebenen der Atomarität (nach [WWW] und [Tas]) (siehe Abbildung 5), auf denen ein Geschäftsprozess arbeitet. Diese Ebenen seien im Folgenden erläutert:

1. *System-level Atomicity*
   Diese Kategorie beinhaltet die Default-Atomarität, die von jeder Transaktion im System eingehalten werden muss.

   - *Service Request Atomicity*
     besagt, dass jeder Service Provider Dienste und Operationen anbietet, die als atomare Einheit im System angesehen werden und so auch gemeinschaftlich zu einem Ergebnis kommen.

2. *Business Interaction-level Atomicity*
   Diese Art von Atomarität wird benötigt, wenn Unternehmen ihre Handelsmodalitäten diskutieren und verhandeln. Die Business Interaction-level Atomicity findet vor jeder Art von realem Geschäft statt, es schafft einen Rahmen, basiert im Regelfall auf Dokumenten, Nachrichtenaustausch und Geschäftsprotokollen.

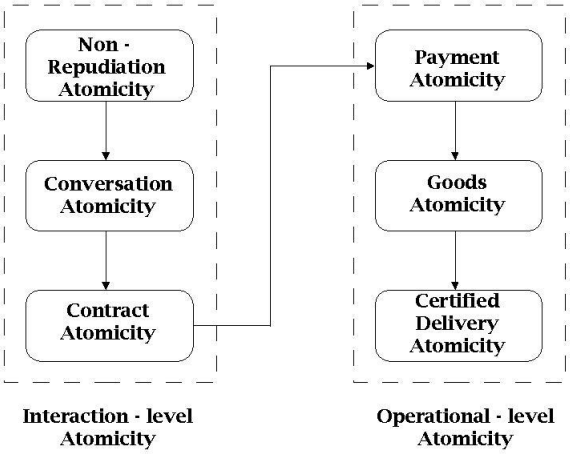

Abbildung 5: BT Atomaritäts-Typen

- *Non-repudiation Atomicity*
  Diese Atomarität sichert die Verbindlichkeit der Transaktion zu. Eine ausgeführte Transaktion ist dann nicht nur durch die Identität des Benutzers authentifiziert und verifiziert, sondern sie ist auch noch mit einer Unterschrift des Benutzers gekennzeichnet. Diese Informationen werden dann in einer persistenten Historie gesichert; damit sind Transaktionen immer wieder überprüfbar.
- *Conversation Atomicity*
  Diese Schicht beinhaltet die Korrelation von Folgen von Anforderungen der beteiligten Parteien. Dabei haben die teilnehmenden Parteien die Möglichkeit, die Konversation auf einen konsistenten Zustand zurückzusetzen. Dies geschieht mittels eines Undo der Operationen der jeweiligen Partei und Benachrichtigung der anderen teilnehmenden Parteien. Weiterhin kann sich eine Partei auf die transaktionalen Eigenschaften der anderen Partei verlassen, um die Konsistenz des globalen Zustands zu wahren.
- *Contract Atomicity*
  Die Grundelemente einer Geschäftstransaktion sind Verträge und Kontenbewegungen. Dabei kann Vertragsatomarität gesetzlich bis zu n Partner binden. Die elektronischen Verträge definieren dabei nicht nur die gesetzlichen Bedingungen, sondern auch die techni-

schen Spezifikationen, die benötigt werden, um miteinander auf elektronischem Wege Handel zu treiben.

3. *Operational-level Atomicity*
   Währen sich die Business Interaction-level Atomicity nur mit den Voraussetzungen (d.h. den erforderlichen Maßnahmen, damit überhaupt ein Handeln möglich ist) beschäftigt, definiert die Operational-level Atomicity nun die physischen Anforderungen. Das bedeutet, die Operational-level Atomicity steuert nun die Geschäftstransaktionen, die das Liefern von erstellter Ware (greifbar oder nicht greifbar) koordinieren.

   - *Payment Atomicity*
     Payment Atomicity ist die grundlegendste Eigenschaft, die jede Geschäftstransaktion erfüllen sollte. Dabei wird der Geldtransfer von einem zum anderen Konto absolviert. Hier muss das Erzeugen oder Vernichten von Geld unmöglich sein.
   - *Goods Atomicity*
     Diese wichtige Eigenschaft, die jede Geschäftstransaktion erfüllen sollte, gewährleistet den vollständigen Austausch von Geld und Ware. Das bedeutet, dass die Ware nur dann ausgeliefert wird, wenn auch das Geld gezahlt worden ist (Bsp.: Nachnahme). Dabei ist die grundlegende Idee, dass das Liefern der Güter Teil der Geschäftstransaktion ist, in der auch für die Güter bezahlt wird.
   - *Certified Delivery Atomicity*
     Die Certified Delivery Atomicity garantiert das Liefern der Ware an den Kunden, wobei zusätzlich auch das Liefern der richtigen Ware garantiert wird. Das bedeutet, die Eigenschaften der Certified Delivery Atomicity verlangen ein Protokoll, das für beide teilnehmenden Parteien den Waren- und Geldtransfer in einer Geschäftstransaktion abwickelt.

3.5 Geschachtelte Transaktionen in Geschäftsprozessen

Wir haben bis jetzt das Prinzip der geschachtelten Transaktionen kennengelernt und die Umgebung einer Geschäftstransaktion. Aber wie werden nun die geschachtelten Transaktionen in die Geschäftsprozesse eingebunden?
Hierbei wird, wie in Abbildung 6 gezeigt, eine Baumstruktur angelegt. Im oberen Teil dieser Struktur definieren die einzelnen Protokolle und Regeln (Koordinationsprotokoll, Geschäftsprotokoll etc.) mittels BTF die Grundlage der Geschäftstransaktion. Sie beherbergt also mit Hilfe der Agreement-Protokolle eine dialogorientierte Logik, d.h., in dieser Stufe werden alle Verhandlungsrelevanten Dinge geregelt. Die jeweiligen Protokolle werden durch Knoten, die Beziehungen durch Kanten dargestellt. Das bedeutet, dass

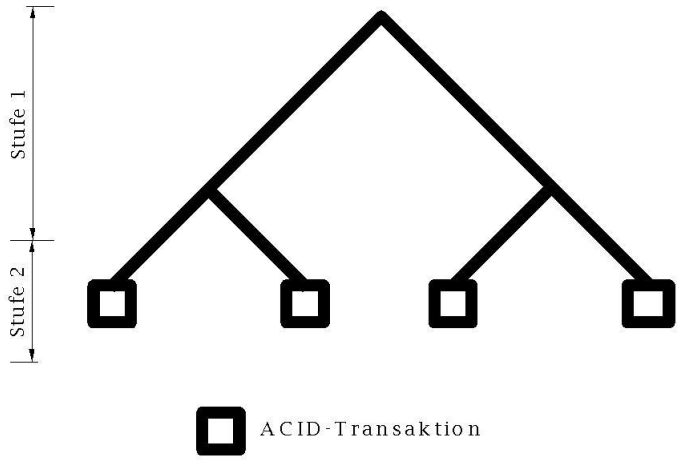

Abbildung 6: ACID-Transaktionen in einem Geschäftsprozess

die einzelnen Protokolle aufeinander aufbauen und voneinander abhängen können. Als Blätter (Stufe 2) stehen dann die einzelnen ACID-Transaktionen (flache oder geschachtelte Transaktionen), die die eigentlichen Aufgaben erfüllen. Diese ACID-Protokolle unterliegen einer Boolschen Logik, kennen als Resultat somit nur den Ausgang true oder false (was im transaktionalen Sinne beseutet, dass eine Transaktion entweder geglückt ist oder nicht).

4 Schluss

4.1 Resumé

In meiner Ausarbeitung bin ich auf die Besonderheiten von Transaktionen in einer heterogenen Umgebung eingegangen. Dabei habe ich langlebige Transaktionen in geschachtelten Transaktionen diskutiert. Hier bin ich zu dem Schluss gekommen, dass das ACID-Prinzip zu streng ist und nicht auf dieses Verfahren angewendet werden kann. Weiter habe ich festgestellt, dass geschlossen geschachtelte Transaktionen mittels Undo-Operationen rückgesetzt werden können, offen geschachtelte Transaktionen hingegen mittels kompensierenden Transaktionen rückgängig gemacht werden können. Aus dieser Tatsache folgerte ich, dass bei geschlossen geschachtelten Transaktionen die Sub-Transaktionen nur noch die AI-Eigenschaften des ACID-Prinzips beherbergen, wohingegen die Sub-Transaktionen offen geschach-

telter Transaktionen die AID-Eigenschaften besitzen. Danach ging ich auf die Intra/Inter-Transaktionsparallelität ein, und beschrieb die einzelnen Vor- bzw. Nachteile.
Im folgenden Kapitel erläuterte ich das Prinzip der sog. Geschäftsprozesse, definierte Komponenten (Aktivität, BTF etc.) und erklärte, was diese tun und was zu beachten ist.
Darauf aufbauend, wurden ACID-Transaktionen in diese Umgebung eingefügt und erörtert, wie alle Komponenten kooperieren.
Es sollte klar geworden sein, dass Transaktionen in einer heterogenen Umgebung wie dem Web grundsätzlich verschieden zu Transaktionen in homogenen Umgebungen sind.

Literatur

[IBM] M. Chessell et al, *IBM Systems Journal, No.4, 2002*, pp. 743-758

[WWW] Michael P. Papazoglou , *World Wide Web, No.6, 2003*, Kluwer Academic Publishers, pp. 49-92

[WfWS] S. Dessloch, Skript zur VL *Workflows und Webservice*, 2003,Universität Kaiserslautern

[Mid] S. Dessloch, Skript zur VL *Middleware für verteilte Informationssysteme*, 2003, Universität Kaiserslautern

[Tas] T. Härder, Skript zur VL *Transaktionssysteme*, 2002, Universität Kaiserslautern

[GI] Lexikon der Gesellschaft für Informatik *www.gi-ev.de*

[AK] A. Kruse, Diplomarbeit, *www.a-plus-a.de/armin/diplomarbeit,* 1997

[Hae] T. Härder, E. Rahm, *Datenbanksysteme*, 2001, Springer Verlag

[Wei] G. Weikum, *Transaktionen in Datenbanksystemen*, 1988, Addison-Wesley

[VLDB] T. Härder, *Concurrency Control Issues in Nested Transactions*, 1993, The VLDB Journal (Vol 2, Nr 1), pp. 39-74