Francis Gropengießer

Transaktionale Unterstützung kollaborativer Applikationen

Cuvillier Verlag Göttingen
Internationaler wissenschaftlicher Fachverlag

Transaktionale Unterstützung kollaborativer Applikationen

Transaktionale Unterstützung kollaborativer Applikationen

Dissertation
Zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der
Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von
Dipl.-Inf. Francis Gropengiesser

Tag der Einreichung: 28. November 2013
Tag der wissenschaftlichen Aussprache: 10. Juni 2014

Gutachter

1. Univ.-Prof. Dr.-Ing. habil. Kai-Uwe Sattler
2. Univ.-Prof. Dr. rer. nat. habil. Gunter Saake
3. Univ.-Prof. Dr.-Ing. Norbert Ritter

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

1. Aufl. - Göttingen: Cuvillier, 2014
Zugl.: (TU) Ilmenau, Univ., Diss., 2014

Umschlagabbildungen:

https://www.flickr.com/photos/torkildr/ (Torkild Retvedt)
https://www.flickr.com/photos/28581681@N04/ (leolintang)

Gutachter

1. Univ.-Prof. Dr.-Ing. habil. Kai-Uwe Sattler
   Technische Universität Ilmenau (Doktorvater)
2. Univ.-Prof. Dr. rer. nat. habil. Gunter Saake
   Universität Magdeburg
3. Univ.-Prof. Dr.-Ing. Norbert Ritter
   Universität Hamburg

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2014
Gedruckt auf umweltfreundlichem, säurefreiem Papier aus nachhaltiger Forstwirtschaft

ISBN 978-3-95404-740-6
eISBN 978-3-7369-4740-5

Kurzfassung

In einer Vielzahl von Anwendungsdomänen, beispielsweise im Design oder in der Medienproduktion, sind kollaborative Arbeitsprozesse kaum mehr wegzudenken. In u.U. weltweit verteilten Szenarien arbeiten mehrere gleichberechtigte Nutzer an einem gemeinsamen Projekt. Diese Arbeit stellt den vollständigen Entwurf einer Middleware vor, die es ermöglicht, eine Vielzahl von Anwendungen zum Einsatz in kollaborativen Arbeitsprozessen auf einem gemeinsamen monohierarchisch strukturierten Datenbestand zu befähigen. Hierbei kann wiederum aus einer Vielzahl unterschiedlicher Speicherlösungen zur persistenten Sicherung der Daten gewählt werden. Die dafür erforderlichen Konzepte und Techniken beruhen auf transaktionalen Semantiken.

Kollaborative Arbeitsprozesse sind durch spezifische Anforderungen gekennzeichnet. Daher stellt diese Arbeit ein maßgeschneidertes Transaktionsmodell vor, welches gezielt Eigenschaften der geschachtelten Transaktionen sowie der Multi-Level-Transaktionen, der Sagas, der Split&Join-Transaktionen und der dynamischen Aktionen kombiniert, um diese Anforderungen zu erfüllen. Es unterstützt die Spezifikation feingranularer atomarer Einheiten zur Zerlegung der lang andauernden Arbeitsprozesse und die Aufweichung der Isolation, um Ergebnisse eines Nutzers frühzeitig freizugeben und einen Informationsaustausch in beliebiger Richtung zwischen den Nutzern zu ermöglichen. Dabei wird stets die strikte Konsistenz des Datenbestands und die Dauerhaftigkeit der erfolgreich durchgeführten Änderungen gewahrt. Die weiterhin eingeführte neuartige AUTOCOMPLETE-Metrik unterstützt die dynamische Bildung von Transaktionen zur Laufzeit und ermöglicht eine kontrollierte und automatische Freigabe von Transaktionsergebnissen unter Einbeziehung der Kollaborationsbereitschaft der Nutzer.

Die Architektur der Middleware basiert auf dem Shared-Nothing-Ansatz und gestattet so auch den skalierbaren Umgang mit verteilten Szenarien. Unter Einsatz des aspektorientierten Paradigmas ist eine leichte Integration der entwickelten Transaktionssemantiken in die Applikation möglich, ohne deren Quellcode anpassen zu müssen.

Zur Durchsetzung der spezifizierten Isolations- und Sichtbarkeitseigenschaften des Transaktionsmodells beherbergt die Middleware geeignete Synchronisationsprotokolle. Hierbei liegt der Fokus dieser Arbeit auf der Ausnutzung der Semantiken des zugrunde liegenden Daten- und Operationsmodells. Dies erlaubt eine feingranulare Konfliktspezifikation, die sich positiv auf den erreichbaren Grad der Parallelität in den Arbeitsprozessen der Nutzer auswirkt. Die genaue Aufschlüsselung semantischer Abhängigkeiten zwischen den Operationen führt zu einem erweiterten Validierungskriterium, welches die Anzahl der Konflikte zwischen Transaktionen zusätzlich reduziert. Die positiven Auswirkungen semantischer Synchronisation auf eine Vielzahl transaktionaler Kennzahlen, z.B. die Transaktionsabbruchrate, werden in praktischen Experimenten nachgewiesen.

Zur Gewährleistung der spezifizierten Atomaritäts- und Dauerhaftigkeitseigenschaften des entwickelten Transaktionsmodells in Fehlersituationen diskutiert diese Arbeit Commit- und Recovery-Protokolle, adaptiert geeignete Vertreter auf das entwickelte Transaktionsmodell und gibt Hinweise zum Wiederanlauf einer möglichen Systemumsetzung der Middleware.

Abstract

In many application domains, for example in design or media production processes, collaborative working processes are essential. In up to world-wide distributed scenarios, several equally considered users work together on the same project. This thesis presents an entire concept of a middleware which enables a variety of applications to be integrated in collaborative working processes on tree-structured data. For storing the data, also a variety of different storage solutions can be applied. The necessary concepts and techniques are based on transactional semantics.

Collaborative working processes are characterized by specific requirements. Therefore, this thesis presents a tailored transaction model which combines specific properties of nested and multi-level transactions as well as Sagas, Split&Join transactions, and dynamic actions in order to fulfill these requirements. It supports the specification of fine-grained atomic units and weakens isolation. In this way, long lasting working processes can be split up, users' changes can be made visible very early, and arbitrary information exchange between different users is possible. Thereby, it always guarantees strict data consistency and the durability of successfully executed changes. A novel AUTOCOMPLETE metric supports dynamic transaction definition at runtime as well as controlled and automatic visibility of changes by considering a user's willingness to cooperate.

The architecture of the middleware is based on the shared-nothing approach and hence is able to deal with distributed scenarios in a scalable way. By exploiting the aspect-oriented paradigm, transactional semantics can be integrated into an application very easily and without touching the application code.

In order to enforce the specified isolation and visibility properties of the transaction model, the middleware contains appropriate synchronization protocols. Thereby, the focus of this work is on exploiting the semantics of the underlying data and operation model. Considering these semantic information supports a fine-grained conflict specification, which leads to positive effects on the degree of parallelism within the users' working processes. A detailed study of semantic dependencies between operations leads to an extended validation criterion, which additionally reduces conflicts between transactions. The positiv effects on a number of transaction measurements, for example the number of transaction aborts, by applying semantic synchronization techniques are proven during practical experiments.

In order to ensure the specific atomicity and durability properties of the developed transaction model in failure situations, several commit and recovery protocols are discussed. Appropriate protocols are adapted to the specified transaction model. Furthermore, this thesis provides some comments regarding recovery in a possible system implementation of the proposed middleware.

Danksagung

Diese Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Fachgebiet Datenbanken und Informationssysteme der Technischen Universität Ilmenau entstanden.

Ein besonderer Dank gilt zunächst meinem Doktorvater Univ.-Prof. Dr.-Ing. habil. Kai-Uwe Sattler. Er hatte stets ein offenes Ohr für meine Fragen. Die zahlreichen Diskussionen mit ihm und seine konstruktive Kritik haben in erheblichem Maße zum Gelingen dieser Arbeit beigetragen.

Weiterhin möchte ich mich bei Univ.-Prof. Dr. rer. nat. habil. Gunter Saake für die zahlreichen Hinweise und die Begutachtung dieser Arbeit bedanken. Ebenfalls ein Dank für die Begutachtung dieser Arbeit geht an Univ.-Prof. Dr.-Ing. Norbert Ritter.

Des Weiteren gilt ein besonderer Dank all meinen ehemaligen Kollegen, allen voran Christine Krause, Katja Hose, Liz Ribe-Baumann und Heiko Betz, die stets zu Diskussionen über Fragen im Kontext dieser Arbeit bereit waren und immer aufmunternde Worte gefunden haben. Viele dieser Kollegen, auch die an dieser Stelle nicht namentlich erwähnten, sind mir zu guten Freunden geworden.

Ein besonders herzlicher Dank gilt meiner Familie, meinen Freunden und meiner Partnerin Sabine. Ohne den Rückhalt all dieser mir sehr nahestehenden Menschen wäre die Erstellung dieser Arbeit nicht möglich gewesen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Gegenwart ist geprägt durch die Globalisierung und dem Streben nach der lückenlosen Vernetzung aller Menschen auf dem Planeten. Dem Zusammenwirken von Menschen im Rahmen von Projekten sind räumlich kaum noch Grenzen gesetzt. Die Gegenwart ist aber auch geprägt durch eine gewisse Schnelllebigkeit. Projekte haben häufig einen kurzfristigen Charakter und erfordern ein schnelles und unkompliziertes Zusammenarbeiten der beteiligten Akteure. Gerade in dieser Zeit sind daher sogenannte *kollaborative Arbeitsprozesse* kaum mehr wegzudenken – ob beim Produkt-Design, der Softwareentwicklung oder der Filmproduktion, um nur einige Beispiele zu nennen. Selbst Internetblogs stellen eine Art des kollaborativen Handelns dar.

An dieser Stelle muss zunächst geklärt werden, was sich hinter dem Begriff der Kollaboration verbirgt. Zunächst einmal bedeutet er die Zusammenarbeit von Individuen zur Erreichung eines gemeinsamen Zieles. Alle Akteure sind dabei gleichberechtigt und tragen gleichermaßen zur Erfüllung des Gesamtzieles bei. Im Gegensatz zum häufig synonym verwendeten Begriff der *Kooperation* gibt es keine strikte Arbeitsaufteilung und damit keine hierarchischen Weisungsstrukturen [Sch07].

Seit den 80er Jahren beschäftigen sich auch Datenbankforscher mit der Unterstützung kooperativer und kollaborativer Arbeitsprozesse durch den Einsatz klassischer Datenbankmethoden, wie beispielsweise Transaktionen. Anfänglich ging es dabei vorrangig um die Unterstützung von CAD-/CAM-Systemen [NH82, BKK85, NRZ92]. Doch durch die wachsende Verbreitung des Internets und das Aufkommen immer neuer Technologien, wie beispielsweise *Cloud-Computing* [AFG+09], kann das Forschungsgebiet der Unterstützung kollaborativer Arbeitsprozesse aus Sicht der Datenbankenwelt noch lange nicht als abgeschlossen bezeichnet werden.

Im weiteren Verlauf dieses Kapitels sollen zunächst anhand einiger Anwendungsbereiche wesentliche Charakteristika und Anforderungen kollaborativer Arbeitsprozesse abgeleitet werden. Aus diesen Anforderungen ergeben sich konkrete Ziele und Beiträge dieser Arbeit.

1.1 Anwendungsszenarien

Nachfolgend ist eine kleine Auswahl an Szenarien dargestellt, in denen Kollaboration erforderlich ist bzw. einen Mehrwert darstellen würde.

Medienproduktionsprozesse: Unter Medienproduktionsprozessen werden im Rahmen dieser Arbeit alle Prozesse verstanden, die mit der Verarbeitung von Bild- und Tonmaterial in Verbindung stehen. Ein Beispiel ist die Tonproduktion mit dem Raumklangverfahren IOSONO [IOS], welches das Fraunhofer Institut für Digitale Medientechnologie (IDMT) 2003 der Öffentlichkeit vorstellte. IOSONO beruht auf den Prinzipien der Wellenfeldsynthese [Ber88]. Es gestattet, virtuelle Schallquellen, wie z.B. einen Sprecher oder eine Explosion, im dreidimensionalen Raum zu positionieren und zu animieren. An der Tonproduktion eines Kinofilmes sind im Allgemeinen eine Vielzahl von Personen beteiligt. Diese wiederum sind meist in verschiedenen Teams, beispielsweise für Musik oder Effekte, organisiert. Innerhalb von Teams, aber auch teamübergreifend, ist eine kollaborative Arbeitsweise unerlässlich, da u.a. Bewegungspfade, Lautstärke oder auch die zeitliche Abfolge von Klangereignissen genau abgestimmt werden müssen.

Kollaboratives Schreiben: Darunter ist im Allgemeinen das Erstellen von Texten durch mehrere Autoren zu verstehen [RB06]. Anwendung findet dies z.B. beim Verfassen wissenschaftlicher Artikel. Ein sehr bekanntes Beispiel ist auch das freie Onlinelexikon Wikipedia [WIK]. Typischerweise verfasst hier ein Autor einen Beitrag. Dieser wird durch andere Autoren ergänzt oder korrigiert. Die Gemeinschaft aller Autoren trägt also zusammen dazu bei, dass ein Artikel der Wikipedia möglichst vollständig und fehlerfrei ist. Weitere Beispiele sind SubEthaEdit [SUB] oder Google Drive [GOOc] in Verbindung mit Google Docs [GOOb].

Kollaboratives Design: Hinter diesem Begriff stehen u.a. Anwendungen im Bereich des Produkt- und Softwaredesigns sowie des technischen Designs [Kva00]. Als konkrete Beispiele sind hier CAD- und UML-Applikationen zu nennen. Aber auch das Gebiet der Raumplanung stellt ein Anwendungsgebiet kollaborativer Konzepte dar. Gemein ist all diesen Applikationen, dass hier eine Gruppe von Designern gemeinsam ein Projekt, sei es der Entwurf einer Software oder die Ausstattung eines Bürokomplexes, bearbeitet. Gerade bei derartigen kreativen Aufgaben ist es wichtig, dass verschiedene unterschiedliche Ideen in das Projekt einfließen und verhandelt werden. Beispiele hier sind u.a. CollabCAD [COL] oder CoAutoCad [GZS07].

1.2 Anforderungen kollaborativer Prozesse

Ein kurzer Blick auf die im vorhergehenden Abschnitt beschriebenen Anwendungsszenarien lässt erahnen, dass es sich bei einem System zur Unterstützung kollaborativer

Arbeitsprozesse um ein eng gekoppeltes Mehrbenutzersystem handeln muss, welches das parallele Arbeiten der Nutzer an einem gemeinsamen Datenbestand unterstützt. Dabei stellt das Erzeugen bzw. Manipulieren von Daten einen festen Bestandteil eines kollaborativen Arbeitsprozesses dar. Um Datenverluste, und damit den Verlust von Arbeitsfortschritt, oder *Inkonsistenzen* im Datenbestand zu vermeiden bzw. soweit wie möglich einzuschränken, muss ein geeignetes System auch mit verschiedensten Fehlern, wie z.B. Systemausfällen, Benutzerfehlern oder Kommunikationsfehlern, umgehen können. Ein bewährter Ansatz zum Bau derartiger paralleler und robuster Mehrbenutzersysteme ist die Anwendung des Konzepts der *Transaktion* [GLP75], welche eine Folge von Operationen, für die die *ACID*-Eigenschaften garantiert werden [HR83], darstellt. *Atomarität* besagt dabei, dass die Transaktion entweder vollständig oder gar nicht ausgeführt wird. Konsistenz (*Consistency*) bedeutet, dass die Datenbasis vor und nach der Ausführung der Transaktion in einem konsistenten Zustand ist. *Isolation* ist die Garantie, dass jeder Nutzer den Eindruck hat, er arbeite allein auf dem Datenbestand. Die *Dauerhaftigkeit* schließlich besagt, dass die Ergebnisse einer Transaktion nach ihrem erfolgreichen Abschluss persistent gesichert sind. Dieses einfache Modell der Transaktion bildet die Basis für die in dieser Arbeit entwickelten Konzepte. Auf die besonderen Eigenschaften der Transaktion und ihrer Auszeichnung gegenüber alternativen Ansätzen wird in Abschnitt 2.1 ausführlich eingegangen.

Um nun konkrete transaktionale Konzepte zur Unterstützung kollaborativer Arbeitsprozesse zu erarbeiten, ist eine detaillierte Analyse deren Charakteristika und Anforderungen notwendig. Nachfolgend sind drei wesentliche Dimensionen dargestellt, hinsichtlich derer eine Einordnung kollaborativer Prozesse anhand der beschriebenen Anwendungsszenarien vorgenommen wird.

Lokalität der Daten: Allen beschriebenen Anwendungsbereichen ist gemein, dass eine Vielzahl von Nutzern auf einem gemeinsamen Datenbestand arbeitet. Die Nutzer können sich dabei innerhalb eines Gebäudes befinden, aber auch über den ganzen Globus verteilt sein. Ebenso kann auch der Datenbestand zentral oder verteilt vorliegen. Eine Systemlösung muss daher sowohl mit *zentralisierten Szenarien* als auch *verteilten Szenarien* möglichst transparent für die Applikation umgehen können.

Grad der ACID-Garantien: Kollaborative Prozesse haben strikte Konsistenzanforderungen. Jeder Nutzer sollte immer einen *aktuellen und widerspruchsfreien Blick auf das Projekt* haben, da dadurch frühzeitig Fehlentscheidungen bezüglich des Projektzieles erkannt werden können. Weiterhin lebt Kollaboration nur durch das Miteinander – die *enge Kopplung – aller Beteiligten*. Unmittelbare Reaktionen auf die Aktionen eines anderen Nutzers sind essentiell. Ebenso sollten auch alle erfolgreich durchgeführten Änderungen eines Nutzers am Projekt erhalten bleiben und daher persistent gespeichert werden. Strikte Atomarität und Isolation wirken sich jedoch hinderlich auf kollaborative Arbeitsprozesse aus. Typischerweise sind derartige Prozesse von langer Dauer. Tritt ein Fehler auf, so führt dies unter strikter Einhaltung der Atomarität zu einem untragbar hohen Arbeitsfortschrittsverlust. Ferner sind Ergebnisse eines Autors unter

strikter Einhaltung der Isolation erst am Ende des Arbeitsprozesses für andere Autoren sichtbar, was der Forderung, dass jeder Autor immer den aktuellen Stand des Projektes sieht, widerspricht. Ein weiterer Nachteil strikter Isolation ist, dass keine *zyklischen Abhängigkeiten bzw. Informationsflüsse zwischen den Nutzern* zulässig sind. So sind Operationsabfolgen, wie z.B. Nutzer A schreibt Objekt D, welches von Nutzer B gelesen, bearbeitet und anschließend wiederum von Nutzer A gelesen wird, innerhalb einer Transaktion pro Nutzer nicht gestattet. Jedoch spielen gerade derartige Informationsflüsse bei Verhandlungen über Lösungsansätze einzelner Kollaborationspartner eine zentrale Rolle. Ein geeignetes Transaktionsmodell muss daher *strikte Konsistenz und Dauerhaftigkeit* bei gleichzeitiger *Aufweichung der Atomarität und Isolation* ermöglichen.

Dynamik der Arbeitsabläufe Einerseits wird die Dynamik durch den Grad der Isolation beeinflusst. So ist bei voll isolierten Transaktionen lediglich eine sequentielle Abfolge derselbigen möglich. Je mehr die Isolation aufgeweicht werden kann, desto beliebigere Interaktionen zwischen Transaktionen werden möglich. Andererseits sind kollaborative Prozesse keine klassischen Anwendungsfelder von Transaktionen. Typische Anwendungsfelder, wie z.B. das Bankwesen, sind durch statische, systemnahe Arbeitsabläufe gekennzeichnet. Es gibt eine feste Menge von Operationsabfolgen, z.B. für eine Überweisung oder Umbuchung, die während der Applikationsentwicklung durch vordefinierte Transaktionen im System hinterlegt werden. Kollaborative Prozesse hingegen finden üblicherweise auf einer höheren Ebene statt. Es gibt Werkzeuge, z.B. grafische Editoren, die vom darunter liegenden Datenmodell und den physischen Datenoperationen stark abstrahieren. Auf Anwenderebene existieren logische Operationen, die durch Transaktionen im Voraus auf Systemebene hinterlegt werden können. Allerdings können diese logischen Operationen wiederum beliebig miteinander kombiniert werden, so dass sich dynamisch beliebige Operationsabfolgen und Interaktionen zwischen Nutzern ergeben. Auf Systemebene müssen somit *Transaktionen dynamisch zur Laufzeit gebildet* werden. Dabei muss die Bildung neuer Transaktionen nicht zwingend die Garantie strikter ACID-Eigenschaften bedeuten. So können, wie in dieser Arbeit noch beschrieben wird, diese Transaktionsgrenzen beispielsweise zum kontrollierten Freigeben von Transaktionsergebnissen genutzt werden, ohne strikte Atomarität zu fordern. Es existieren somit wohldefinierte Punkte in den Operationsabfolgen, an denen der Arbeitsfortschritt eines Nutzers anderen Nutzer zugängig gemacht wird. Die Definition dieser Punkte und somit die *Bestimmung der Transaktionsgrenzen* sollten jedoch *automatisch durch das System* erfolgen. Einerseits kann von einem Designer oder Autor nicht erwartetet werden, dass er Wissen über Transaktionen besitzt. Andererseits muss auch die Freigabe von Ergebnissen im Sinne eines kollaborativen Handelns erzwungen werden.

Abbildung 1.1 stellt die Anforderungen kollaborativer Arbeitsprozesse den Leistungen klassischer *relationaler Datenbankmanagementsysteme* (RDBMS), nach dem Vorbild des Systems *R* [ABC+76], gegenüber. Klassische relationale Datenbankmanagementsysteme sind als Mehrbenutzersysteme ausgelegt. Nutzer können sich weltweit verteilt über verschiedene Schnittstellen, z.B. SQL [SQL] oder JDBC [JDB], Zugriff

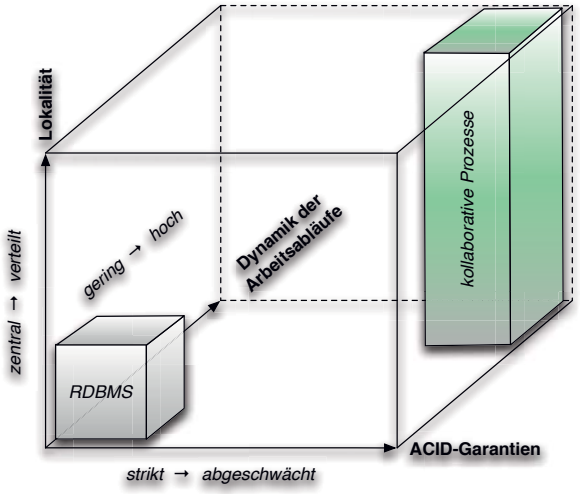

Abbildung 1.1: Einordnung kollaborativer Prozesse

auf die Daten verschaffen. Der eigentliche Datenbestand hingegen wird zentral verwaltet. Sollen verteilte Datenbestände verwaltet werden, erfolgt dies durch unabhängige Datenbankinstanzen. Um Transaktionen, die sich über mehrere Datenbestände erstrecken, auszuführen, ist daher zusätzliche externe Logik, z.B. das 2-Phasen-Commit-Protokoll [BHG87], erforderlich.

Relationale Datenbankmanagementsysteme garantieren strikte ACID-Eigenschaften. Lediglich die Isolation kann über verschiedene SQL-Isolationsstufen aufgeweicht werden [BBG+95]. Ein Herabsetzen des Grades an Isolation führt jedoch auch zu Einbußen bezüglich der Garantie strikter Konsistenz. So treten beispielsweise bei *READ UNCOMMITTED* sogenannte *Dirty Reads* und *Phantome* auf [WV01]. Eine flexible Wahl des Grades an garantierter Atomarität ist ebenfalls nicht möglich.

Transaktionen müssen stets durch den Nutzer im Voraus spezifiziert werden. Lediglich für einzelne Operationen wird ein sogenanntes AUTOCOMMIT, d.h. das automatische erfolgreiche Beenden einer Transaktion, durch das System ermöglicht. Die flexible und dynamische Zusammenfassung von Operationen zu Transaktionen zur Laufzeit ist dadurch nur sehr eingeschränkt möglich.

Zusammenfassend folgt aus dieser Einordnung, dass klassische relationale Datenbankmanagementsysteme die Anforderungen kollaborativer Arbeitsprozesse nur unzureichend erfüllen. Der Umgang mit verteilten Datenbeständen wird nicht transparent für die Applikation gewährleistet. Ferner wird keine Aufweichung der Isolation und Atomarität bei gleichzeitiger Wahrung strikter Konsistenz und Dauerhaftigkeit gestattet. Die automatische und dynamische Definition von Transaktionen zur Laufzeit ist ebenfalls nicht vorgesehen.

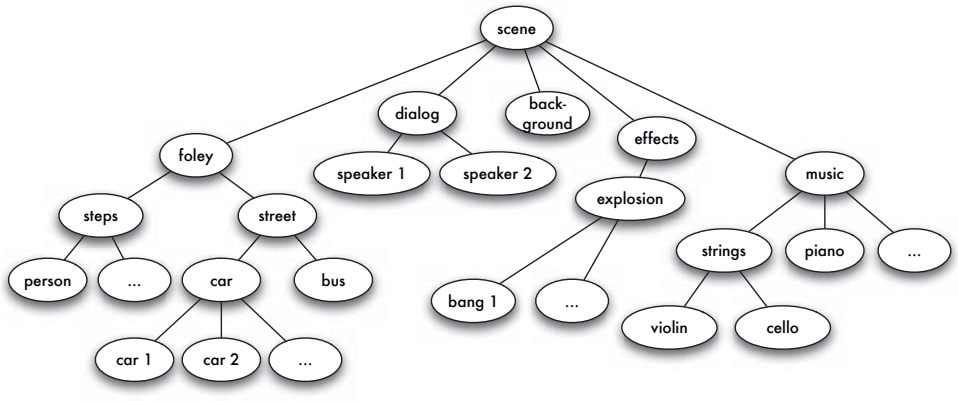

Abbildung 1.2: Szenegraph im IOSONO System

1.3 Beiträge der Arbeit

Aus den Diskrepanzen zwischen den im vorhergehenden Abschnitt genannten Anforderungen kollaborativer Arbeitsprozesse und den Leistungen klassischer relationaler Datenbankmanagementsysteme lassen sich unmittelbar konkrete Aufgaben ableiten:

1. Zur Unterstützung kollaborativer Arbeitsprozesse ist ein Transaktionsmodell erforderlich, welches die Spezifikation feingranularer atomarer Einheiten und die Aufweichung der Isolation bei gleichzeitiger Wahrung der Konsistenz und Dauerhaftigkeit ermöglicht.

2. Ferner ist ein Konzept notwendig, welches die automatische, dynamische und für den Nutzer transparente Definition von Transaktionen ermöglicht.

3. Weiterhin wird eine Systemarchitektur benötigt, die den Umgang mit zentralisierten und verteilten Szenarien ermöglicht, ohne zusätzliche Logik innerhalb der Applikation zu erfordern.

4. Schließlich sind zur Realisierung der spezifischen ACID-Eigenschaften geeignete *Commit-*, *Synchronisations-* und *Recovery-Protokolle* vonnöten [WV01].

Die Erfüllung dieser Aufgaben erfolgt unter besonderer Berücksichtigung des zugrunde liegenden Datenmodells. Der Fokus liegt dabei auf *monohierarchisch strukturierten Daten*, die folglich eine Baumstruktur aufweisen. Bäume bilden die grundlegende Datenorganisationsform in einer Vielzahl von Applikationsdomänen, in denen Informationen schrittweise in feingranulare Einheiten zerlegt werden [Sam84]. Ein Beispiel ist der in Abbildung 1.2 dargestellte Szenegraph im IOSONO System. Dieser umfasst die bei der Animation der Schallquellen innerhalb einer Klangszene entstehenden Metainformationen, wie z.B. die Lautstärke eine Dialoges oder Instrumentes.

Im Hinblick auf die oben genannten Aufgaben liefert diese Arbeit nun konkret folgende wissenschaftliche Beiträge:

1. Es wird ein allgemeines Daten- und Operationsmodell entwickelt und formalisiert, welches die Abbildung und Verarbeitung beliebiger monohierarchisch strukturierter Daten ermöglicht.

2. Darauf aufbauend wird ein maßgeschneidertes Transaktionsmodell entwickelt, welches die spezifischen ACID-Anforderungen kollaborativer Arbeitsprozesse erfüllt sowie deren hohe Dynamik abbildet. Dieses Transaktionskonzept kombiniert gezielt spezifische Eigenschaften der offen- und geschlossen geschachtelten Transaktionen sowie der Multi-Level-Transaktionen [Mos81, WS92], der Sagas [GMS87], der Split&Join-Transaktionen [PKH88] und der dynamischen Aktionen [NM95].

3. Die Eigenschaften dieses maßgeschneiderten Transaktionsmodells sowie die intra- und intertransaktionalen Beziehungen werden unter Anlehnung an generische Transaktionsmodelle, wie z.B. den Transaktionshüllen [Sch99], sowie den ihnen zugrunde liegenden formalen Modellen, im Speziellen ACTA [CR90], in formaler Weise spezifiziert.

4. Ferner wird eine AUTOCOMPLETE-Metrik vorgestellt, welche die automatische und dynamische Definition von Transaktionen des entwickelten Transaktionsmodells ermöglicht. Diese Metrik beschränkt die Transaktionslänge unter Berücksichtigung der Kollaborationsbereitschaft der Nutzer. Insgesamt wird dadurch eine kontrollierte und automatische Freigabe von Transaktionsergebnissen ermöglicht.

5. Der transparente Umgang mit zentralen und verteilten Szenarien wird durch eine Middelware realisiert, die durch Einsatz des *aspektorientierten Paradigmas* [KLM+97] eine leichte Integration der gewünschten transaktionalen Semantiken in die Applikation ermöglicht. Somit kann eine Vielzahl von Anwendungen zum Einsatz in kollaborativen Arbeitsprozessen auf einem gemeinsamen monohierarchisch strukturierten Datenbestand befähigt werden. Durch die Definition einer allgemeinen Schnittstelle wird von der konkreten Speicherung der Daten abstrahiert, so dass eine Vielzahl auf dem Markt verfügbarer Speicherlösungen zur persistenten Datenhaltung verwendet werden kann.

6. Demonstrativ wird die Systemumsetzung der entwickelten Middleware auf dem *Storage-as-a-Service*-Dienst [AFG+09] Amazon S3 [DHJ+07, Amac] skizziert. Hierbei werden konkret Fragen nach der Wahl des physischen Schreib-/Lesegranulats sowie der durch Amazon S3 verursachten monetären Kosten behandelt.

7. Unter Ausnutzung der *semantischen Informationen* des definierten Daten- und Operationsmodells erfolgt eine feingranulare Konfliktspezifikation. Unter Einsatz dieser semantischen Konfliktspezifikation werden das 2-Phasen-Sperrprotokoll [WV01] sowie das *Backward Oriented Concurrency Control* (BOCC) -Protokoll [KR81] auf das entwickelte Transaktionsmodell angepasst.

Ferner wird für letzteres Protokoll ein *erweitertes Validierungskriterium* vorgestellt, welches *semantische Abhängigkeiten* zwischen Operationen berücksichtigt. Im Rahmen einer Evaluierung wird gezeigt, dass sich die Berücksichtigung semantischer Informationen positiv auf eine Vielzahl von transaktionalen Kennzahlen, wie z.B. die Anzahl der während der Validierung abgebrochenen Transaktionen, auswirkt.

Neben diesen Schwerpunkten behandelt diese Arbeit selbstverständlich auch die Themen *Umsetzung der Atomrität in verteilten Umgebungen* und *Recovery*, um den Entwurf der Middleware zu vervollständigen. Dabei diskutiert diese Arbeit verteilte Commit- und Recovery-Protokolle, adaptiert geeignete auf das entwickelte Transaktionsmodell und gibt Hinweise zum Wiederanlauf einer möglichen Systemumsetzung der Middleware.

1.4 Grundlegende Begriffe

Dieser Abschnitt fasst grundlegende Begriffe aus dem Bereich transaktionaler Informationssysteme in knapper Weise zusammen, um ein besseres Verständnis für die folgenden Ausführungen beim Leser zu erzielen. Für detaillierte Informationen sei der interessierte Leser jedoch auf die einschlägige Literatur, wie z.B. [WV01] oder [BHG87], verwiesen.

Die flache Transaktion: Eine Transaktion ist eine partiell geordnete Folge von Operationen mit einem Anfangsereignis und verschiedenen Terminierungsereignissen. Für diese Operationsfolge werden die bereits erläuterten ACID-Eigenschaften garantiert. Im Datenbankbereich entstammen diese Operationen traditionell dem einfachen *Schreib-Lese-Modell* (*Page-Model*), welches nur aus einfachem Lesen *read(x)* und Schreiben *write(x)* einer Seite x von der Festplatte bzw. auf dieselbige besteht. Durch das Erzwingen einer lediglich partiellen Ordnung wird die Möglichkeit eröffnet, Operationen innerhalb einer Transaktion parallel auszuführen. Eine Transaktion beginnt mit einem *Begin Of Transaction* (BOT). Wurde sie erfolgreich ausgeführt, so wird sie mit einem *Commit* erfolgreich abgeschlossen. Ein Abbruch einer Transaktion erfolgt durch ein *Abort*. In diesem Fall müssen aufgrund der Forderung nach Atomrität alle durchgeführten Operationen rückgängig gemacht werden. Abbildung 1.3 zeigt nun das Zustandsmodell einer Transaktion in einer leicht von der klassischen Lehrbuchdarstellung abgewandelten Form. Die Besonderheit im Bild ist der üblicherweise nicht dargestellte Übergang vom Zustand *Committed* in den Zustand *Aborted*. Grundsätzlich wird für eine Transaktion die Dauerhaftigkeit gefordert. Ein nachträglicher Abbruch widerspricht dieser Forderung. Bedingt durch eine verzahnt geschachtelte Ausführung mehrerer Transaktionen kann es jedoch trotzdem notwendig sein, eine Transaktion nach dem Commit zurückzusetzen. Dies ist genau dann der Fall, wenn die verzahnt geschachtelte Ausführung nicht Element der Fehlersicherheitsklasse *Recoverability* ist, die nachfolgend eingeführt wird. Hierzu werden typischerweise *Kompensationstransaktionen* eingesetzt,

welche die Schreiboperationen der ursprünglichen Transaktion mit Hilfe inverser Schreiboperationen zurücksetzen. Der Einsatz von Kompensationstransaktionen kann jedoch zu applikationsspezifischen Problemen führen [GFJK03].

Die verteilte Transaktion: Setzt sich eine flache Transaktion aus Operationen zusammen, die auf mehreren getrennten Datenbeständen ausgeführt werden, wird diese Transaktion verteilt ausgeführt. Die ursprüngliche flache Transaktion bildet nun eine sogenannte *globale Transaktion*. Alle Operationen, die einen bestimmten Datenbestand betreffen, werden in einer sogenannten *lokalen Transaktion* zusammengefasst, die von ihrem Aufbau her einer flachen Transaktion entspricht. Sowohl für die globale als auch die lokalen Transaktionen gelten die ACID-Eigenschaften.

Die Historie: Die verschränkte Ausführung einer Menge von Transaktionen wird im weiteren Verlauf der Arbeit als *Historie* oder *vollständiger Schedule* bezeichnet. Sie umfasst sowohl alle Operationen aller Transaktionen als auch die Anfangs- und Terminierungsereignisse aller Transaktionen und Operationen. Auch für eine Historie wird lediglich eine partielle Ordnung gefordert, um die parallele Ausführung von Transaktionen bzw. deren Operationen zu ermöglichen. Hierbei wird jedoch stets die für eine Transaktion spezifizierte Ordnung eingehalten. Wird lediglich ein beliebiger Präfix einer Historie betrachtet, bezeichnet man diesen auch als *Schedule*.

Korrektheit in Mehrbenutzersystemen: Werden Transaktionen beliebig verschachtelt ausgeführt, treten Inkonsistenzen im Datenbestand und sogenannte *Nebenläufigkeitsanomalien*, wie z.B. die bereits erwähnten Dirty Reads und Phantome sowie *inkonsistentes Lesen* oder *Lost Updates*, auf. Das einfachste Mittel, um Isolation, und damit die Konsistenz der persistenten Datenbasis in Mehrbenutzersystemen zu garantieren, ist, alle Transaktionen nacheinander, d.h. *seriell*, auszuführen. Da jede Transaktion für sich die Konsistenz wahrt, ist somit auch die beliebige serielle Ausführung einer Menge von Transaktionen konsistenzerhaltend. Da sich eine derartige Ausführung jedoch negativ auf den *Transaktionsdurchsatz*, d.h. die Menge erfolgreich abgeschlossener Transaktionen pro Zeiteinheit, auswirkt, muss eine andere Strategie bzw. ein

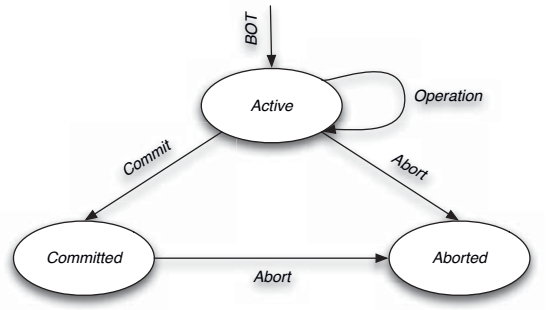

Abbildung 1.3: Abgewandeltes Zustandsmodell einer flachen Transaktion

anderes Korrektheitskriterium gefunden werden, welches unter Wahrung der Isolation mehr Parallelität ermöglicht. Dies führt zum Begriff der *Serialisierbarkeit*. Umgangssprachlich ausgedrückt ist ein Schedule *serialisierbar*, wenn er den gleichen Zustand der persistenten Datenbank erzeugt, wie ein serieller Schedule. Allerdings lässt sich Serialisierbarkeit nicht zur Laufzeit überprüfen, da der durch einen Schedule erzeugte Zustand eines Objektes erst nach Ausführung desselbigen besteht. Eine Verfeinerung dieses Korrektheitskriteriums ist die *Sichtserialisierbarkeit*. Ein Schedule für eine Menge von Transaktionen T ist *sichtserialisierbar*, wenn er *sichtäquivalent* zu einem seriellen Schedule ist. Sichtäquivalent bedeutet dabei, dass er grundsätzlich dieselbe Menge von Operationen enthält und dieselbe *Liest-von-Relation* besitzt. Die Liest-von-Relation ist eine Menge von Ordnungspaaren. Jedes Ordnungspaar beschreibt, dass eine Transaktion t das von einer Transaktion s geschriebene Objekt liest. Sichtserialisierbarkeit ist für den praktischen Einsatz ungeeignet. Einerseits fehlt ein effizienter Algorithmus zur Überprüfung einer Historie von Transaktionen bezüglich dieses Kriteriums. Andererseits ist eine Überprüfung eines unvollständigen Schedules hinsichtlich dieses Kriteriums nicht zweifelsfrei möglich (fehlende *Präfix-Commit-Abgeschlossenheit*). Daher existiert ein weiteres verfeinertes Kriterium der Serialisierbarkeit – die *Konfliktserialisierbarkeit*. Hierfür muss zunächst der Begriff *Konflikt* geklärt werden. Operationen können auf verschiedene Weise miteinander in Konflikt stehen. Ein einfaches Modell lautet wie folgt: Zwei Operationen stehen in Konflikt, sofern beide Operationen auf demselben Objekt arbeiten und der Zustand des Objektes nach der Ausführung beider Operationen von der Ausführungsreihenfolge der Operationen abhängt. Besitzt eine Operation einen Rückgabewert, so existiert dann ein Konflikt, wenn der Rückgabewert davon abhängig ist, ob die andere Operation ausgeführt wurde oder nicht. Für in Konflikt stehende Operationen wird eine Ordnung erzwungen. Sie müssen stets hintereinander ausgeführt werden, um das Auftreten von Nebenläufigkeitseffekten zu verhindern. Die Menge der geordneten Konfliktpaare innerhalb eines Schedules wird als *Konfliktrelation* bezeichnet. Ist nun ein beliebiger Schedule *konfliktäquivalent* zu einem seriellen Schedule, d.h. beide besitzen dieselbe Menge von Operationen und dieselbe Konfliktrelation, so wird dieser Schedule als konfliktserialisierbar bezeichnet. Anders formuliert ist ein Schedule genau dann konfliktserialisierbar, wenn die dazugehörige Konfliktrelation zyklenfrei ist. Dies kann auch mittels eines *Konfliktgraphen*, in dem die Transaktionen die Knoten und die Konfliktpaare die gerichteten Kanten bilden, überprüft werden. Aus diesem kann auch mittels topologischer Sortierung die Serialisierungsreihenfolge der Transaktionen abgeleitet werden kann.

Fehlerklassen: Serialisierbarkeit garantiert im Mehrbenutzerbetrieb die Isolation der Transaktionen bzw. die Konsistenz der Daten. Allerdings beeinflusst die verschränkte Ausführung von Transaktionen auch die Atomarität und Dauerhaftigkeit einer Transaktion. Betrachtet wird dazu folgender Beispiel-Schedule: $read_1(x)$ $write_1(x)$ $read_2(x)$ $Abort_1$. Die Transaktion t_2 liest den von Transaktion t_1 geschriebenen Wert x. Aufgrund des Abbruchs von t_1 muss nun auch t_2 abgebrochen werden, da diese Transaktion mit einem nun nicht mehr gültigen Wert von x arbeitet, was die Konsistenz der Daten

gefährdet. Aus den Liest-von-Beziehungen zwischen Transaktionen ergeben sich daher Abbruchbeziehungen. Damit jedoch ein Abbruch möglich ist, darf eine Transaktion nicht bereits ein Commit ausgeführt haben, da ansonsten die Dauerhaftigkeit verletzt wird. Liest eine Transaktion t daher einen Wert von einer Transaktion s, so muss das Commit der Transaktion s vor dem Commit der Transaktion t erfolgen. Schedules, in denen alle Transaktionen dieser Bedingung folgen, gehören zur Klasse der *rücksetzbaren Schedules* (Recoverability). Problematisch an dieser Klasse von Schedules ist jedoch, dass sich Ketten von *abbruchabhängigen* Transaktionen bilden und sich im Falle des Abbruchs der Transaktion am Kopf dieser Kette eine Reihe sogenannter *kaskadierender Abbrüche* ereignen. Um dies zu vermeiden, kann eine verschränkte Ausführung von Transaktionen so eingeschränkt werden, dass nur das Lesen von Ergebnissen solcher Transaktionen erlaubt ist, die bereits ein Commit ausgeführt haben. Dies führt zur Klasse *Avoiding Cascading Aborts* (ACA). Bei dieser Fehlerklasse können jedoch trotz dieser Restriktion noch Probleme auftrete, z.B. beim Recovery mit *Before-Images*, da das Auftreten „blinder“ Schreiboperationen nicht berücksichtigt wird. Unter „blindem Schreiben“ versteht man, dass eine Operation einen neuen Objektzustand erzeugt, ohne den vorhergehenden Zustand gelesen zu haben. Es kommt daher nicht zur Ausbildung einer Liest-von-Relation. Daher muss die Klasse ACA dahingehend eingeschränkt werden, dass auch das Überschreiben der Ergebnisse solcher Transaktionen, die noch kein Commit ausgeführt haben, verboten wird. Derartige Schedules gehören dann zur Klasse der *strikten Schedules* (*Striktheit*).

Verfahren zur Garantie der Konfliktserialisierbarkeit: Um die Konfliktserialisierbarkeit in einem realen System zur Laufzeit garantieren zu können, existieren sogenannte Synchronisationsprotokolle. Im Laufe der Zeit sind zahlreiche Protokolle entstanden, die sich im Wesentlichen in die 4 Klassen *Sperrprotokolle* (auch *pessimistische Protokolle*), *Zeitstempelverfahren*, *Serialisierbarkeitsgraphtester* und *optimistische Protokolle* einordnen lassen. Sperrprotokolle bestehen im Kern immer aus zwei Teilen – der Sperrdisziplin sowie dem eigentlichen Protokoll. Ersteres beschreibt, welche Arten von Sperren existieren, wie diese zueinander kompatibel sind und welche Operationen welche Sperren auf welchen Objekten setzen müssen, um gleichzeitige konfliktäre Zugriffe auf Objekte zu verhindern. Das Protokoll spezifiziert, wie das eigentliche Setzen und Freigeben der Sperren im Rahmen einer jeden Transaktion erfolgen muss, damit konfliktserialisierbare Schedules entstehen. Durch den Einsatz eines Sperrprotokolls werden daher konfliktserialisierbare Schedules zur Laufzeit von vornherein erzwungen. Das klassische Beispiel für ein Sperrprotokoll ist das 2-Phasen-Sperrprotokoll, welches die Konfliktserialisierbarkeit dadurch erzwingt, dass nach dem Freigeben einer Sperre durch eine Transaktion, dieser nicht gestattet ist, erneut eine Sperre zu setzen. Die Grundidee der Zeitstempelverfahren besteht darin, Transaktionen mit eindeutigen, fortlaufenden Kennzeichnungen zu versehen. Anhand dieser Kennzeichnung kann für in Konflikt stehende Operationen abgeleitet werden, in welcher Reihenfolge sie ausgeführt werden müssen, damit die Konfliktserialisierbarkeit der Historie gegeben ist. Die Idee hinter den Serialisierbarkeitsgraphtestern beruht auf dem Aufbau und der Pflege eines be-

reits erwähnten Konfliktgraphen. Ist während der Arbeit auf einem Datenbestand mit sehr wenigen Konflikten zwischen den Tätigkeiten der einzelnen Beteiligten zu rechnen, bedeutet der Einsatz eines Sperrprotokolls unnötigen *Overhead* durch das Setzen und Verwalten von Sperren. Optimistische Protokolle versprechen unter derartigen Bedingungen einen erhöhten Transaktionsdurchsatz. Einer der bekanntesten Ansätze ist in [KR81] beschrieben. Jede Transaktion arbeitet in einem privaten Universum. Die durch die Transaktion benötigten Daten werden in einer sogenannten Arbeitsphase in das private Universum geladen und bearbeitet. Anschließend wird die Transaktion validiert. Wurden keine Konflikte mit anderen parallel laufenden Transaktionen festgestellt, werden die durchgeführten Änderungen in der Schreibphase in die persistente Datenbasis eingebracht. Um den Zeitraum für das Auftreten von Konflikten zwischen den Transaktionen auf deren Arbeitsphasen zu beschränken, werden die Validierungs- und Schreibphase als eine atomare Einheit betrachtet. Schreibvorgänge in die persistente Datenbasis erfolgen in diesem Fall immer seriell. Ist diese Atomaritätsforderung zu strikt, muss die Schreibphase durch entsprechende Sperranforderungen serialisiert werden.

Verfahren zur Garantie von Atomarität und Dauerhaftigkeit: Auch die Sicherung der Atomarität und Dauerhaftigkeit, was gleichzeitig den Umgang mit Fehlern, wie z.B. Systemfehlern oder Transaktionsfehlern bedeutet, erfordert entsprechende Verfahren, die sich im Wesentlichen in Recovery-Protokolle und Commit-Protokolle einordnen lassen. Konkrete Recovery-Protokolle realisieren jeweils eine der vier Strategien UNDO/REDO, No-UNDO/REDO, UNDO/No-REDO und No-UNDO/No-REDO. UNDO bedeutet das Zurücksetzen einzelner oder mehrerer Transaktionen und No-UNDO entsprechend die Negation. REDO wiederum bedeutet das Wiederholen einzelner oder mehrerer Transaktionen und No-UNDO, dass entsprechend kein Wiederholen von Transaktionen erforderlich ist. Grundlegende Verfahren im Bereich der Recovery-Protokolle sind das *Shadow Paging* und das *Logging*. Das grundlegende Prinzip des Shadow Paging [Lor77, EN10] besteht darin, dass laufende Transaktionen ihre Änderungen nur auf Kopien der Objekte durchführen. Die Adressen von Kopien und Original der Objekte werden in einem Verzeichnis verwaltet. Erst mit dem Commit einer Transaktion wird das originale Objekt durch die Kopie ersetzt. Dies verhindert, dass schmutzige Änderungen in die persistente Datenbasis gelangen. Das Shadow Paging realisiert auf diese Weise eine No-REDO/No-UNDO Recovery-Strategie. Grundvoraussetzung, um die Konsistenz der persistenten Datenbasis unter Anwendung dieses Verfahrens gewährleisten zu können, ist, dass das Ersetzen der originalen Seite atomar durchgeführt wird. Dies ist durch die Verwendung eines Verzeichnisses sehr leicht möglich. Bevor eine Transaktion Änderungen ausführt, wurde das originale Verzeichnis bereits auf den stabilen Speicher geschrieben. Führt die Transaktion anschließend Änderungen durch, werden neue Kopien der Objekte angelegt und deren Adressen in dem sich im Hauptspeicher befindenden Verzeichnis hinterlegt. Beim Commit einer Transaktion muss das veränderte Verzeichnis auf den stabilen Speicher geschrieben werden. Um die Atomarität zu Gewährleisten, wird zwischen alter und neuer Version des Verzeichnisses durch

eine Bitoperation umgeschaltet. Eine bekannte Schwäche des Shadow Paging in einem klassischen seitenbasierten Datenbanksystem ist, dass es nur in Kombination mit einem Sperrgranulat auf Seitenebene anwendbar ist. Ansonsten ist der Abbruch einzelner Transaktionen nicht möglich. Ferner kann im Falle des Auftretens konkurrierender Transaktionen ein korrekter Wiederanlauf ohne zusätzliches Loggen nicht gewährleistet werden [EN10]. Ein weiteres Recovery-Protokoll, das Logging, ist eine der am häufigsten eingesetzten Recovery-Technik [MHL+92, BHG87]. Sie beruht auf dem Führen eines Logbuches, welches für jede am Datenbestand durch eine Transaktion durchgeführte Änderung einen entsprechenden Eintrag enthält. Dieser beinhaltet, je nach dem welche Recovery-Strategie umgesetzt werden soll, entweder die UNDO-Informationen, die REDO-Informationen oder beides. Diese Informationen können in *physischer*, *logischer* oder *physisch-logischer* Form enthalten sein. Unter physischem Loggen versteht man in einem klassischen seitenbasierten Datenbanksystem typischerweise das Speichern der Before- und After-Images ganzer Seiten als UNDO- bzw. REDO-Informationen. Es ist nur anwendbar, wenn das Sperrgranulat auf Seitenebene angesiedelt ist. Andernfalls ist das Rücksetzen einzelner Transaktionen nicht möglich. Allerdings existieren auch Spezialformen, wie das Zustandslogging, bei dem die Wertausprägungen der Objekte einer Seite vor und nach der Änderung protokolliert werden. Hier ist wiederum eine feingranulare Synchronisation möglich. Beim logischen Logging werden die durchgeführten semantischen Änderungsoperationen als REDO- und die dazugehörigen Umkehroperationen als UNDO-Informationen protokolliert. Der Vorteil dieser Vorgehensweise ist, dass feingranulare Synchronisationsprotokolle angewendet werden können. Die Wiederherstellung eines konsistenten Datenbankzustands nach einem Fehlerfall gestaltet sich mit rein logischem Logging u.U. sehr schwierig. Logische Log-Einträge erstrecken sich in einem klassischen seitenbasierten Datenbanksystem typischerweise über mehrere Seiten. Kommt es während des Schreibens der geänderten Seiten in die persistente Datenbasis zu einem Systemausfall, befindet sich die Datenbank nicht in einem *aktionskonsistenten Zustand*, d.h., es ist für eine logische Operation nicht sichergestellt, dass sie entweder ganz oder gar nicht in die persistente Datenbank eingebracht wurde. Die Aktionskonsistenz ist jedoch die Grundvoraussetzung für die Durchführung eines REDO oder UNDO mit Hilfe eines logischen Logs. Um dies zu umgehen, werden beim physisch-logischem Logging komplexe logische Operationen, welche sich über mehrere Seiten erstrecken, in mehrere logische Operationen zerlegt. Diese beziehen sich jeweils genau auf eine Seite. Die Gewährleistung eines aktionskonsistenten Zustands ist im Vergleich zum rein logischen Logging einfacher, da nur die Atomarität für das Schreiben einer einzelnen Seite garantiert werden muss. Da in verteilten Systemen einzelne Komponenten ausfallen können, ist bei der verteilten Verarbeitung von Transaktionen zur Durchsetzung der Atomarität und Dauerhaftigkeit zusätzlich zu Recovery-Protokollen der Einsatz von Commit-Protokollen erforderlich. Commit-Protokolle stellen sicher, dass für eine globale Transaktion, die aus einer Menge von lokalen Transaktionen besteht, entweder alle lokalen Transaktionen in den Zustand Committed versetzt werden oder alle abgebrochen werden. Es existieren zahlreiche Varianten, wie beispielsweise das 2-Phasen-Commit-Protokoll. Dabei schickt der Koordinator, als Ausführender einer globalen Transaktion eine PREPARE-Nachricht an alle Teilnehmer, die lokale Trans-

aktionen der betrachteten globalen Transaktion ausführen. Diese antworten entweder mit einem VOTE_COMMIT (READY), wenn sie die lokale Transaktion erfolgreich ausgeführt haben, oder einem ABORT, sofern die lokale Transaktion gescheitert ist. Haben alle Teilnehmer mit VOTE_COMMIT geantwortet, leitet der Koordinator das globale COMMIT ein. Hat auch nur einer mit ABORT geantwortet, wird die gesamte globale Transaktion und damit auch alle lokalen Transaktionen abgebrochen. Das Problem des 2-Phasen-Commit-Protokolls ist, dass es blockiert, sofern der Koordinator ausfällt und nicht wieder anläuft (*Fail-Stop*-Fehler) [WV01]. In diesem Fall können die Teilnehmer ihre lokale Transaktion weder in den Zustand Committed noch Aborted versetzen. Das 3-Phasen-Commit-Protokoll verhindert diese Situation durch eine zusätzliche PRE-COMMIT-Phase. Diese ermöglicht im Falle eines Koordinatorausfalls die Wahl eines neuen Koordinators, der das Protokoll fortsetzt und beendet. Problematisch ist, dass nicht in jedem Falle die Existenz mehrerer Koordinatoren ausgeschlossen ist [GL06]. Dann kann es zu widersprüchlichen Entscheidungen kommen, wenn beispielsweise ein Teilnehmer zwei unterschiedliche Nachrichten von zwei verschiedenen Koordinatoren empfängt. Dieses Problem wird, neben dem Problem einer Blockierung des Commit-Protokolls, durch das Paxos-Commit-Protokoll gelöst [GL06].

1.5 Aufbau der Arbeit

Der weitere Verlauf der Arbeit gliedert sich wie folgt. Im anschließenden Kapitel werden zunächst die existierenden Technologien zur Unterstützung kooperativer und kollaborativer Arbeitsprozesse dargestellt. Vertiefend wird dabei auf verschiedene Transaktionsmodelle und Synchronisationskonzepte eingegangen, da diese für die in dieser Arbeit entwickelten Lösungen von besonderer Bedeutung sind. Kapitel 3 stellt die *kollaborative Transaktionsschicht* als Vorstufe einer Middleware-Architektur vor, auf deren Details in den nachfolgenden Kapiteln vertiefend eingegangen wird. Kapitel 4 umfasst die formale Darstellung des entwickelten Daten- und Operationsmodells zur Abbildung und Verarbeitung monohierarchisch strukturierter Daten. In Kapitel 5 wird das entwickelte Transaktionsmodell zunächst grundlegend skizziert und anschließend formal spezifiziert. Weiterhin beleuchtet dieses Kapitel das Problem der automatischen Transaktionsdefinition im Detail und stellt eine entsprechende Lösung in Form der bereits erwähnten AUTOCOMPLETE-Metrik vor. Der konzeptionelle Teil dieser Arbeit schließt mit einer qualitativen Evaluierung der entwickelten transaktionalen Konzepte hinsichtlich der Anforderungen kollaborativer Arbeitsprozesse.

Der zweite Teil der Arbeit beginnt in Kapitel 6 mit der Darstellung der Anforderungen an eine Middleware zur Unterstützung beliebig verteilter kollaborativer Szenarien. Daran anschließend werden in Kapitel 7 verwandte Arbeiten zu diesem Schwerpunkt hinsichtlich dieser Anforderungen diskutiert. Aus der Anforderungsanalyse und den aus der Diskussion gewonnen Erkenntnissen wird in Kapitel 8 eine allgemeine Middleware-Architektur abgeleitet. Dabei bilden die Beschreibung der einzelnen Komponenten der Middleware sowie die Definition einer allgemeinen Schnittstelle zur Speicherschicht die

Schwerpunkte. Gleichzeitig wird beispielhaft ein Systementwurf über der Speicherlösung Amazon S3 skizziert. Kapitel 9 gibt anschließend detaillierte Einblicke in die Abbildung des entwickelten Transaktionskonzepts auf den Systementwurf. Dabei wird u.a. auf die Integration der transaktionalen Semantiken in die Applikation mittels des aspektorientierten Paradigmas eingegangen. Im darauffolgenden Kapitel 10 wird die Problematik der Synchronisation kollaborativer Arbeitsprozesse aufgegriffen. Hier erfolgt eine detaillierte Konfliktanalyse des definierten Operationsmodells unter Berücksichtigung semantischer Informationen sowie die Anpassung des 2-Phasen-Sperrprotokolls und des BOCC-Protokolls an das entwickelte Transaktionsmodell. Ebenfalls in diesem Kapitel wird das erweiterte Validierungskriterium vorgestellt. Kapitel 11 beginnt mit der Identifikation von Fehlerklassen innerhalb eines kollaborativen Gesamtsystems. Anschließend werden mögliche Commit- und Recovery-Protokolle zum Umgang mit diesen Fehlerklassen diskutiert und geeignete Vertreter auf das entwickelte Transaktionsmodell adaptiert. Ferner werden Hinweise zum Wiederanlauf einer möglichen Systemumsetzung der Middleware gegeben. In Kapitel 12 erfolgt die Evaluierung der entwickelten Synchronisationsprotokollerweiterungen. Die Tests werden mittels einer prototypischen Implementierung der Middleware durchgeführt. Die zentrale Hypothese dieser Tests ist, dass sich die Ausnutzung der semantischen Informationen des entwickelten Daten- und Operationsmodells sowie der Kenntnisse über semantische Abhängigkeiten zwischen Operationen positiv auf eine Vielzahl von Kennzahlen, wie z.B. die Anzahl der während der Validierung abgebrochenen Transaktionen, auswirkt. Weiterhin werden am Beispiel der Speicherlösung Amazon S3 Fragen nach der Wahl des physischen Schreib-/Lesegranulats sowie entstehender monetärer Kosten behandelt. Die Arbeit schließt mit einer Zusammenfassung und einer Darlegung offener Forschungsfragen.