Fachbereich Mathematik/Informatik an der Universität Osnabrück

Verwaltung geowissenschaftlicher 3D Daten in mobilen Datenbanksystemen

Dissertation

zur
Erlangung des Grades
Doktor der Naturwissenschaften (Dr. rer. nat.),

eingereicht am Fachbereich Mathematik/Informatik
der Universität Osnabrück
im Mai 2007

Dipl.-Umweltwiss., Dipl.-Ing. (FH) Wolfgang Bär

Danksagung des Autors

Als erstes möchte ich mich bei meinem Mentor und Betreuer dieser Arbeit Prof. Dr. Martin Breunig bedanken, der mir für die Erstellung dieser Arbeit mit konstruktiver Kritik und vielen Anregungen zur Seite stand und mir stets die nötigen Freiräume gewährte. Mein weiterer Dank gilt Prof. Dr. Thomas Brinkhoff für die Übernahme des Koreferats.

Große Bereiche dieser Arbeit sind aus dem BMBF-Projekt „Weiterentwicklung von Geodiensten“ hervorgegangen. Ich möchte an dieser Stelle deshalb den Personen danken, die mit mir in diesem Forschungsprojekt zusammengearbeitet und meine Arbeit durch Diskussionen bereichert haben.

Spezieller Dank geht an Andreas Thomsen und Dag Hammerich, die mit mir über die letzten Jahre im Projekt gearbeitet und mir immer mit Rat und Tat zur Verfügung standen.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Einbettung der Arbeit 2

1.2 Mobile Informationssysteme 3

1.2.1 Kapazitätsbeschränkungen 4

1.2.2 Verbindung zu Netzwerken 4

1.3 Verwaltung geowissenschaftlicher 3D Daten 6

1.3.1 Geowissenschaftliche 3D Daten und Modelle 6

1.3.2 Anwendungsfelder 8

1.3.3 Anforderungen an 3D Datenbanksysteme 9

1.4 Ziele und Aufbau der Arbeit 11

2 Grundlagen **13**

2.1 3D Datenbanksysteme 13

2.1.1 Geometrische und topologische Datenmodelle 13

2.1.1.1 Geometrische Modelle 14

2.1.1.2 Topologische Modelle 18

2.1.2 Verhalten von 3D Datenbanksystemen 20

2.1.2.1 Zugriffsmethoden (Indexe) 20

2.1.2.2 Geometrische Algorithmen 24

2.1.3 Interoperabilität 26

2.1.4 Relevanz für die Arbeit 29

2.2 Mobile Datenbanken 30

2.2.1 Anwendungsklassen 30

2.2.2 Verbindungszustände und Trennungsmodi 30
2.2.3 Architekturen mobiler Datenbanksysteme 31
2.2.4 Modelle zur Transaktionsunterstützung in mobilen Umgebungen . . 33
2.2.4.1 Transaktionsmodelle 34
2.2.4.2 Replikationsverfahren 35
2.2.5 Relevanz für die Arbeit 36
2.3 Versionsmodelle 38
2.3.1 Übersicht 38
2.3.2 Begriffe und Definitionen 39
2.3.3 Unterstützung in räumlichen Datenbanksystemen 42
2.4 Verwandte Forschungsgebiete und -arbeiten 42

3 Geodatenverwaltung in mobilen Datenbanken 47
3.1 Anforderungen 47
3.2 Probleme 48
3.3 Geowissenschaftliche Beispielanwendungen 51
3.3.1 Geologische Modellierung der Schaumburg-Lippeschen Kreidemulde 51
3.3.2 Hangrutschungen am Albtrauf in Balingen 52
3.4 Konzept 54

4 Datenbanksystem für geowissenschaftliche 3D Anwendungen 57
4.1 Ausgangspunkt und Architektur 57
4.2 3D-Datenbankkern 59
4.2.1 Abstraktion 59
4.2.2 Organisation 60
4.2.3 Datenmodell 61
4.2.3.1 Geometriebibliothek 62
4.2.3.2 3D-Datenbankobjekte 63
4.2.4 Elementare Operationen 67
4.2.5 XML-Modell 68
4.3 Externe Operationen 68

4.3.1 Ein-/Ausgabeoperationen 68
4.3.2 Geometrieoperationen 70
4.4 Dienst-Framework 70
4.4.1 Architektur 71
4.4.2 Atomare Dienste 73
4.4.3 Verkettung von Diensten 74

5 Erweiterung um ein Versionsmanagement zur Nutzung mit mobilen Datenbanken 77

5.1 Definitionen 77
5.2 Integration des Versionsmanagements 78
5.3 Komplexe versus einfache DesignObjekte 80
5.3.1 Repräsentation als komplexes DesignObjekt 80
5.3.2 Repräsentation als einfaches DesignObjekt 83
5.4 Versionsmodell 84
5.4.1 History 84
5.4.2 Versionsmanagement 85
5.4.3 Operationen 89
5.4.3.1 Workspace-Operationen 89
5.4.3.2 History-Operationen 90
5.5 Umsetzung für 3D-Geoobjekte 93
5.5.1 Speicherung der Versionen 93
5.5.2 Repräsentation von Object3D 95
5.5.3 Repräsentation von Space3D 96
5.5.4 Konvertierung zwischen Datenmodellen 98
5.5.5 Operationen im Versionsmanagement 105
5.5.6 Konfliktverarbeitung 107
5.6 Mobile 3D Datenbank 109
5.6.1 Konzeption einer mobilen 3D Datenbank 109
5.6.1.1 Unterschiede zum Server-Datenbanksystem 110
5.6.1.2 3D-Funktionalität und Versionenverwaltung 111
5.6.2 Integration mit dem Server-Datenbanksystem 112

6 Evaluation **117**

6.1 Laufzeitverhalten 117

6.2 Ergebnisse mit einer realen Anwendung 121

6.2.1 Testapplikation 121

6.2.2 Aufnahme von Spaltenbewegungen 122

7 Diskussion **127**

8 Zusammenfassung und Ausblick **133**

8.1 Zusammenfassung 133

8.2 Ausblick 136

9 Summary **137**

Anhang **152**

A DB3D-XML Schema Definition **153**

Abbildungsverzeichnis

1.1 Schematische Darstellung mobiler Informationssysteme 5
1.2 Ausschnitt aus einem komplexen 3D Untergrundmodell 8

2.1 Geologische Störung im Vektor-Randmodell 15
2.2 Geologische Störung repräsentiert mittels Zellzerlegungsverfahren 15
2.3 Geologische Störung dargestellt im Enumerationsverfahren 16
2.4 Darstellung der geologischen Störung in einem Octree 16
2.5 Geologische Störung durch Translationssweep dargestellt 17
2.6 Geologische Störung repräsentiert durch Konstruktion mit Raumprimitiven 18
2.7 2-simplizialer Komplex 19
2.8 Beispiele für simpliziale d-Komplexe 20
2.9 Aufbau eines R-Baumes im 2D 23
2.10 Octree 24
2.11 Schnittüberlappungsbereich zweier großer Flächenobjekte im 3D-Raum . . . 25
2.12 Beispiel für Versionshistorie 40
2.13 Beispiel für ein komplexes DesignObjekt 41

3.1 Untergrundmodell der Schaumburg-Lippeschen Kreidemulde 52
3.2 Felsspalte am Winkelgrat in Balingen 53
3.3 In einer Felsspalte installiertes Extensometer 53

4.1 Schichten der Softwarearchitektur von DB3D 58
4.2 Verteilung der Softwarearchitektur auf Rechnerknoten 59
4.3 Organisationsstruktur einer Datenbank im 3D-Datenbankkern 60

4.4 Ausgewählte Klassen der Geometriebibliothek (vereinfacht) 62
4.5 Schnitt zweier Tetraeder: Ergebnis als Wireframe dargestellt 63
4.6 Schnittstellen der Datenbankobjekte 63
4.7 Geometriemodell der Datenbankobjekte in UML-Notation 65
4.8 Thematisches Modell der Datenbankobjekte in UML-Notation 67
4.9 Operationenmodell mit Beispiel-Operationen in UML-Notation 69
4.10 Sequenzdiagramm der Benutzung des Dienst-Frameworks 71
4.11 Aufbau eines Dienstes nach der Dienststruktur 73
4.12 Prinzipieller Ablauf des 3Dto2D-Dienstes 75
4.13 Porta Datensatz mit Lageangabe der berechneten Profilschnitte 76
4.14 Ausschnitt eines berechneten Profilschnittes durch den Porta Datensatz . . 76

5.1 Beziehungen zwischen DesignObjekt, Version und VersionWorkspace 79
5.2 Aufbau von Object3D für simplizialen 2-Komplex (vereinfacht) 81
5.3 Bildung eines komplexen DesignObjektes am Beispiel Object3D 81
5.4 Auswirkungen einer Änderungsoperation auf die Versionierung 82
5.5 Bildung eines einfachen DesignObjektes am Beispiel Object3D 83
5.6 History-Modell zur Verwaltung der Versionen von DesignObjekten 85
5.7 Versionsmanagement-Modell mit dem Bezug zum History-Modell 86
5.8 Beziehung zwischen Historie von DesignObjekt und VersionWorkspace . . . 88
5.9 Sequenzdiagramm des Ablaufs der Registrierung einer Revisionsversion . . . 91
5.10 Existierende Versionsbeziehungen für die Versionslöschung 92
5.11 Änderungsspeicherung für Versionen im Versionsmodell 95
5.12 Modell der Änderungsspeicherung für Object3D-Datenbankobjekte 96
5.13 Konfigurationen im Versionsmodell 97
5.14 Integration neuer DesignObjekte in das Versionsmanagement 106
5.15 API-Erweiterungen der mobilen Datenbank 111
5.16 Struktur der Integration von Clients in das Versionsmanagement 114

6.1 Aufbau bzw. Synchronisation der mobilen Datenbank 121
6.2 Vergleichende Visualisierung zwischen lokalem Objekt und Version 122
6.3 Albtrauf mit Spalten und wandernder Spalte in GOCAD 123

6.4 Visualisierung und Update von Objekten in der mobilen Datenbank 124
6.5 Vergleich mit Versionen im Versionsmanagement 124

Tabellenverzeichnis

1.1 Leistungsfähigkeit mobiler Endgeräte 4

2.1 Transaktionsmodelle 34

6.1 Durchschnittliche Laufzeiten verschiedener Basis-Dienste im 3Dto2D-Dienst 118

6.2 Laufzeiten pro Simplex der Ergebnismenge für verschiedene Basis-Dienste . 118

6.3 Aufbauzeiten für 3D-Objekte in der mobilen Datenbank auf einem PDA . . 120

6.4 Aufbauzeiten für 3D-Objekt in der mobilen Datenbank auf Subnotebook . . 120

Pseudocodeverzeichnis

5.1 Konvertierung: Versionsmodell → Datenbank-Objektmodell 99
5.2 Funktion: Erzeugung der Versionslinie aus einer Version v 100
5.3 Konvertierung: Datenbank-Objektmodell → Versionsmodell 103
5.4 Berechnung des ChangeSet zwischen Simplex-Mengen 104
5.5 Berechnung geometrisch äquivalenter Simplexe 105

Abkürzungsverzeichnis

0D	0. Dimension
1D	1. Dimension
2D	2. Dimension
3D	3. Dimension
CAD/CAM	Computer-Aided-Design/Manufacturing
CORBA	Common Object Request Broker Architekture
DBMS	Datenbankmanagementsystem
DO	DesignObjekt
GML	Geography Markup Language
GPRS	General Packet Radio Service
HTTP	Hyper Text Transfer Protokoll
LAN	Local Area Network
OGC	Open Geospatial Consortium
OMG	Object Management Group
OODBMS	Objektorientiertes Datenbankmanagementsystem
PAM	Point Access Method
PDA	Personal Digital Assistant
RAM	Random Access Memory
RMI	Remote Method Invocation
SAM	Spatial Access Method
SOAP	Simple Object Access Protokoll
UMTS	Universal Mobile Telecommunications System
VWS	VersionWorkspace
WLAN	Wireless Local Area Network
WS	Workspace
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation

Kapitel 1

Einleitung

Die Entwicklungen der letzten Jahre im Bereich der Übertragungstechnologien wie UMTS und die ständig verbesserte Leistungsfähigkeit mobiler Endgeräte führt zu neuen Möglichkeiten im Bereich der mobilen Datennutzung und -management. Hersteller von Datenbanksystemen bieten bereits seit einiger Zeit spezielle, für mobile Endgeräte konzipierte, mobile Datenbanken an [42, 64, 82].

Die hohen Kosten der Mobilfunkverbindungen sowie die Leistungsfähigkeit der neuen mobilen Endgeräte legen es nahe auch auf mobilen Endgeräten Datenbanken zu etablieren, um längerfristig Daten eines Dienstes vorhalten zu können. Hiermit können viele Datenbankanfragen bereits durch die mobile Datenbank beantwortet werden. Vorteile einer solchen Lösung sind geringere Verbindungskosten, eine reduzierte Anfälligkeit durch Verbindungsstörungen und eine schnellere Auskunftsmöglichkeit, wenn benötigte Daten bereits in der mobilen Datenbank vorliegen.

Im Bereich der Geoinformatik sind Geo-Dienste aktueller Gegenstand der Forschung (siehe z.B. [136, 24]). Zu diesen Diensten, oft unter dem Stichwort Location Based Services zusammengefaßt, zählen mobil nutzbare Dienste wie Navigations-, Routenplanungs- und Tourismusdienste. Während in diesem Bereich in den letzten Jahren zahlreiche Arbeiten erfolgten (siehe z.B. [111, 112]), steht die Beschäftigung mit mobilen räumlichen Datenbanken zur Unterstützung mobiler Dienste noch am Anfang. Weiterhin hat man es in den Geowissenschaften mit inhärent dreidimensionalen Objekten zu tun, die beispielsweise ein 3D Untergrundmodell – etwa von Niedersachsen – beschreiben. Eine Datenverwaltung für derart komplexe Objekte in mobilen Datenbanken existiert bisher nicht. Für diesen Bereich geowissenschaftlicher Anwendungen im 3D werden in der vorliegenden Arbeit Konzepte und eine prototypische Beispielimplementierung für ein System zur Verwaltung von 3D Daten mit mobilen Datenbanken vorgestellt.

1.1 Einbettung der Arbeit

Die vorliegende Arbeit ist in das BMBF-Projekt „Weiterentwicklung von Geodiensten“ [24, 25] eingebettet. Es handelt sich um ein Verbundprojekt innerhalb des geowissenschaftlichen Forschungs- und Entwicklungsprogramms GEOTECHNOLOGIEN [40, 41] des Bundesministerium für Bildung und Forschung (BMBF) und der Deutschen Forschungsgemeinschaft (DFG). Es ist dort innerhalb des Projektbereiches „Informationssysteme im Erdmanagement - von Geodaten zu Geodiensten“ angesiedelt.

Das Verbundprojekt wurde unter der Leitung von Prof. Martin Breunig (Universität Osnabrück) in Zusammenarbeit mit dem European Media Laboratory GmbH (EML) in Heidelberg, der Arbeitsgemeinschaft GIS (AGIS) der Universität der Bundeswehr München und dem Institut für Photogrammetrie und Fernerkundung (IPF) der Universität Karlsruhe durchgeführt. Ziel des Verbundes war es, ein Gesamtkonzept für die Erfassung, Verwaltung, Nutzung und Visualisierung von Geodaten für moderne mobile Endgeräte zu erarbeiten.

Dabei wurde die mobile Erfassung, Aktualisierung, Nutzung, Analyse und Visualisierung von Geodaten auf mobilen Clients von der Arbeitsgruppe der Universität der Bundeswehr München (AGIS) und dem EML Heidelberg behandelt. Die Entwicklung eines Augmented Reality 3D GIS-Clients fand an der Universität Karlsruhe statt.

Gegenstand der Arbeiten der Universität Osnabrück war die Entwicklung von Softwarekomponenten für den internetbasierten Zugriff auf Geodatenbank-Dienste [17, 18]. Im Mittelpunkt des Interesses stand die Datenbankunterstützung für raum- und zeitbezogene Objekte (3D/4D) und der effiziente Zugriff auf Geodatenbank-Dienste über das Internet, auch von mobilen Clients aus. Geodatenbank-Dienste sind dabei nicht nur als Ein- und Ausgabedienste für Geodaten des Datenbanksystems zu verstehen. Die Dienste sollen neben Standardoperationen von Datenbanksystemen auch weitergehende Operationen wie mengenorientierte Operationen (z.B. räumliche Verschneidung zwischen Objektmengen), sowie Operationen zur Konstruktion neuer Geoobjekte aus bestehenden gespeicherten Daten bieten. Hierzu sollen die Dienste einzelne Datenbankoperationen für 3D/4D Geodaten bereitstellen und die Möglichkeit der Komposition zu höherwertigen Diensten bieten. Des Weiteren soll die Möglichkeit bestehen jederzeit neue Dienste in die bestehende Architektur einzufügen und eine Verkettung von Diensten zu einer Art Arbeitsablauf auszuführen.

Ausgehend von der Konzeption und Implementierung des Server-Datenbanksystemes für 3D/4D Geodaten erfolgt in der vorliegenden Arbeit die Einbeziehung von mobilen Datenbanken als Clients innerhalb eines mobilen, verteilten Datenbanksystems. Gegenüber *normalen* mobilen Clients, die über die angebotenen Geodatenbank-Dienste auf das Server-Datenbanksystem zugreifen und die Ergebnisse der Dienste verarbeiten, bietet eine mobile Datenbank die Möglichkeit auch ohne eine Verbindung zu dem Server-Datenbanksystem mit den lokal vorhandenen Daten weiterzuarbeiten.

1.2 Mobile Informationssysteme

Die Einführung von mobilen Geräten in Informationssystemen führt zu neuen Herausforderungen. Es müssen zusätzliche Variablen, wie der sich dynamisch ändernde Standort mobiler Geräte, variierende Qualität der Verbindung, häufige Verbindungsunterbrechungen und eingeschränkte Kapazitäten der mobilen Geräte berücksichtigt werden. Neben Herausforderungen entstehen dadurch jedoch auch neue Möglichkeiten, wie zum Beispiel die Einbeziehung des Standortes des mobilen Nutzers in Anfragen der Informationsverarbeitung. Für Übersichten über Herausforderungen, Probleme und derzeitige Forschungsprojekte im Umfeld mobiler Informationssysteme sei auf die Literatur verwiesen [99, 37, 62].

Im folgenden wird nur auf die zwei Kernprobleme, die Kapazitätsbeschränkungen und die Verbindungsprobleme bei mobilen Informationssystemen vertiefend eingegangen, die einen besonderen Bezug zur vorliegenden Arbeit haben. Die Einbeziehung mobiler Datenbanken in ein mobiles Informationssystem und damit der Aufbau einer mobilen Datenbankverwaltung wird in Kapitel 2.2 behandelt.

Die Beschränkungen in den Kapazitäten der mobilen Geräte ist in ihrer Größe vom Typ des mobilen Endgerätes abhängig. Derzeit können mobile Endgeräte, anhand ihrer Leistungsfähigkeit, grob in die folgenden drei Klassen eingeordnet werden:

- Laptops: Diese können heutzutage als relativ gleichwertig zu Desktop-Systemen angesehen werden. Auch wenn sie nicht die Leistungsfähigkeit von Desktop- bzw. Server-Systemen erreichen, so ist doch jedes Datenbanksystem auf dieser Geräteklasse ohne Einschränkungen oder Modifizierungen lauffähig.

- PDAs: Personal Digital Assistants unterscheiden sich dagegen durch deutliche Kapazitätsbeschränkungen. Diese Beschränkungen bedingen derzeit noch eine spezielle Entwicklung von Datenbanksystemen, die auf diesen Geräten lauffähig sind. Diese für PDA's optimierten Datenbanksysteme sind in ihrer Funktionalität gegenüber den Desktop/Server-Pendants meist deutlich eingeschränkt.

- Mobiltelefone: Die Mobiltelefone liegen am untersten Bereich möglicher mobiler Endgeräte. Ihre Leistungsfähigkeit ist gegenüber den PDAs nochmals deutlich reduziert. Mobiltelefone sind hier der Vollständigkeit halber aufgeführt, werden im Rahmen der vorliegenden Arbeit jedoch nicht als Zielsysteme in Betracht gezogen. Weiterhin ist derzeit ein verstärkter Übergang hin zu sogenannten Smartphones zu beobachten, die die Funktionalitäten von Mobiltelefon und PDA vereinigen.

1.2.1 Kapazitätsbeschränkungen

Die folgende Tabelle 1.1 (Quelle: Internet [1] – Stand August 2006) soll einen groben Überblick über die Klassen mobiler Endgeräte geben. Die wichtigsten Kenngrößen sind hierbei die Prozessorleistung, die Größe des Arbeitsspeichers, sowie die Größe und der Typ des Sekundärspeichers der Geräte. In der Gegenüberstellung sind Mobiltelefone ausgenommen.

Ausstattung	Laptop	PDA (Standard)	PDA (HighEnd)
RAM	bis 2 GB	64 MB	128 MB
Gewicht	1,6 bis 4 kg	um 200 g	um 200 g
Auflösung	bis 1600 X 1200	240 x 320	480 x 640
Farben	bis 16 Millionen	65536	65536
Größe	bis 16 ¨	57 x 76 mm	57 x 76 mm
Schnittstellen	Ethernet, WLAN IrDA, Bluetooth	WLAN, Bluetooth	WLAN, Bluetooth

Tabelle 1.1: Leistungsfähigkeit mobiler Endgeräte.

Die Tabelle zeigt, daß in der Laptop-Klasse keine Beschränkungen für die Ausführung derzeitiger Datenbanksysteme mehr existieren. Zielgruppe für die Entwicklung und Nutzung spezieller Datenbanksysteme ist derzeit die Gruppe der PDA-Geräte. Für diese Klasse existieren derzeit von den meisten Herstellern spezielle mobile Datenbanken, auf die in Kapitel 2.4 noch weiter eingegangen wird. Allgemein ist festzustellen, daß die Entwicklung mobiler Endgeräte einer hohen Geschwindigkeit unterliegt und somit viele Hardwareeinschränkungen in naher Zukunft bereits nicht mehr zutreffen werden. Einschränkungen, wie Displaygröße oder Energieverbrauch, werden jedoch auch in Zukunft wichtige zu berücksichtigende Faktoren bleiben.

1.2.2 Verbindung zu Netzwerken

In einem mobilen Informationssystem können Rechner anhand ihrer Netzwerkverbindung unterschieden werden. Zum einen gibt es über ein festes Netzwerk mit einer stabilen Verbindung angebundene Rechner (Fixed Hosts) und zum anderen Rechner, die über verschiedene mobile Funknetzwerktechniken mit dem festen Netzwerk verbunden sind (Mobile Hosts). In Abbildung 1.1 ist eine schematische Übersicht über ein mobiles Informationssystem gegeben. Mobile Rechner sind hierbei über die für ihre Funknetzwerk-Zelle zuständige Basisstation mit dem festen Netzwerk verbunden. Wechselt ein mobiler Rechner aufgrund seiner Mobilität in eine andere Funknetzwerk-Zelle, so muß die Verbindung (unterbrechungsfrei) auf die neue Basisstation übertragen werden.

[1] URL: www.zdnet.de

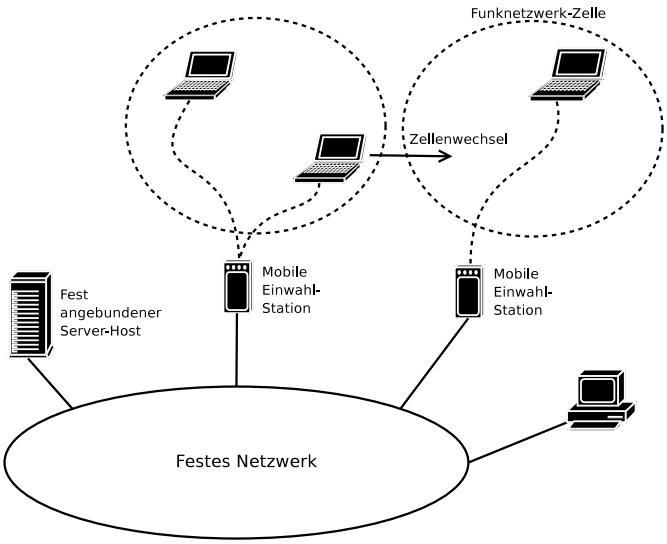

Abbildung 1.1: Schematische Darstellung mobiler Informationssysteme

Im folgenden wird ein kurzer Überblick über die Technologien mobiler Verbindungen unter dem Aspekt der Bandbreite gegeben. Als Techniken kommen hierbei die Wireless Local Area Network Technologie (W-LAN / IEEE 802.11) bzw. die Mobilfunktechnologien General Packet Radio Service (GPRS) und Universal Mobile Telecommunications System (UMTS) in Frage. Während W-LAN mit Bandbreiten von derzeit bis zu 54 MBit/s (für alle Teilnehmer eines Accesspoints) nur innerhalb räumlich begrenzter Bereiche (sogenannter Hotspots) in Ballungsgebieten verfügbar ist, bietet die Mobilfunktechnologie eine Möglichkeit der flächendeckenden Übertragung von Daten. Die hier zur Verfügung stehenden Bandbreiten liegen derzeit zwischen theoretischen 100 kbit/s bei GPRS und in Zukunft bei 2 Mbit/s mit UMTS. In der Praxis werden derzeit etwa 33 kbit/s bei GPRS erreicht und bis ca. 384 kbit/s bei UMTS realistisch sein [70]. Die hier gezeigten Übertragungsbandbreiten liegen jedoch trotzdem weit unter denen derzeitiger LAN Verbindungen von 100 MBit bis 1 GBit Netzwerken bzw. von WAN-Bandbreiten im mehreren MBit-Bereich zwischen Internet-Backbones. Es ist davon auszugehen, daß auch in Zukunft eine deutliche Diskrepanz zwischen den Bandbreiten für mobile Zugänge und netzgebundenen Zugängen bestehen bleibt.

Zusätzlich zur eingeschränkten Bandbreite bei mobilen Verbindungen kommt gerade im Bereich des Mobilfunks die Möglichkeit eines *Verbindungsverlustes* durch unvollständige Netzabdeckung oder Empfangsprobleme hinzu. Ein Verbindungsverlust kann durch eine ungenügend starke Netzabdeckung im Aufenthaltsbereich oder durch einen Standort innerhalb von Funkschatten (z.B. Wald, Gebäudeschatten) auftreten.

Je nach Verbindungsart zwischen dem mobilen Gerät und dem festen Netzwerk können

drei Fälle unterschieden werden:

- **fest verbunden:** Das mobile Gerät ist über eine feste Netzwerkverbindung z.B. Modem oder LAN Verbindung verbunden.
- **drahtlos verbunden:** Das mobile Gerät ist über eine drahtlose Technologie mit dem festen Netzwerk verbunden. Hier treten die oben beschriebenen geringeren Bandbreiten sowie die Möglichkeit des Verbindungsverlustes auf.
- **unverbunden:** Das mobile Gerät ist nicht mit dem festen Netzwerk verbunden.

Ist ein mobiles Gerät nicht mit dem festen Netzwerk verbunden, kann dieser Zustand durch zwei Fälle hervorgerufen sein. Einerseits kann es sich dabei um eine ungewollte Verbindungsunterbrechung durch Probleme in der drahtlosen Verbindung handeln, andererseits auch um eine gewollte, d.h. vom Benutzer initiierte Unterbrechung der Verbindung.

1.3 Verwaltung geowissenschaftlicher 3D Daten

Wie bereits angesprochen wird bei der Bearbeitung geowissenschaftlicher Projekte größtenteils mit 3D Daten gearbeitet. Im folgenden wird deshalb auf die Unterschiede zwischen 2D Daten und *echten* 3D Daten in den Geowissenschaften eingegangen. Anschließend werden Anwendungsfelder für geowissenschaftliche Datenbanksysteme, sowie die daraus resultierenden Anforderungen, aufgezeigt. Weiterführende Informationen zur Datenbankunterstützung in geologischen Anwendungen sind in [125] zu finden.

1.3.1 Geowissenschaftliche 3D Daten und Modelle

Während klassische geographische Informationssysteme (GIS) die Erdoberfläche beschreibende Daten verarbeiten und dazu 2D bzw. 2.5D geometrische Modelle verwenden, beschäftigen sich geowissenschaftliche Anwendungen vornehmlich mit dem Untergrund. Hierfür werden jedoch 2.5D bzw. *echte* 3D-Modelle des Untergrundes benötigt, die eine weitere Dimension in die Datenverarbeitung mit einbringen. Beim Aufbau von Modellen unterscheiden sich geowissenschaftliche Anwendungen dadurch, daß der Erduntergrund nicht für Beobachtungen vollständig zugänglich ist, wie es zum Beispiel bei 2D Daten der Landnutzung möglich ist. Modelle des Untergrundes basieren deshalb immer auf Schätzungen, geologischer Interpretation und fachlichem Hintergrundwissen. Bei der Verbindung zwischen 2D und 3D GI-Systemen ist weiterhin zu beachten, daß die in geowissenschaftlichen Modellen betrachteten geometrischen 3D Objekte sich im allgemeinen nicht ohne Informationsverlust auf die 2- oder 2.5-dimensionalen Darstellungen klassischer GIS reduzieren lassen.

Im folgenden wird die Besonderheit geowissenschaftlicher 3D-Modelle anhand des Aufbaus eines Untergrundmodelles in der Geologie beschrieben. Bereits die geologischen Basisinformationen, aus denen das spätere 3D-Modell aufgebaut werden soll, weisen hier eine beachtliche Vielfalt auf. Diese reicht von an Aufschlüssen oder im Bergbau entnommenen Proben und photographischen Aufnahmen über Bohrkerne und Bohrlochmessungen bis hin zu 2D- und 3D-Seismik. In jedem Fall müssen die so gewonnenen Daten jedoch erst interpretiert werden, bevor sie in ein geologisches Modell eingebracht werden können. Weiterhin ist - mit Ausnahme von 3D-Seismik - das von den Basisdaten abgedeckte Gebiet verschwindend klein im Vergleich zum Modellgebiet, so daß Schätz- und Simulationsverfahren angewendet werden, um ein vollständiges Bild zu erlangen. Dieses erhaltene Bild weist wiederum weit größere Ungenauigkeiten auf, als z.B. ein auf Satellitenaufnahmen beruhendes Modell der Bodennutzung. Andererseits können seismische Daten erst nach geowissenschaftlicher Interpretation in ein geologisches Modell eingebracht werden - auch hier liegt eine Quelle für Unsicherheiten und Ungenauigkeiten des erzeugten Modells vor. Dazu kommt noch die Unsicherheit der Ansprache bzw. Klassifizierung der aufgenommenen Daten, die teilweise selbst erst durch Rückgriff auf das in der Entstehung begriffene Modell gelöst werden kann. Diese Unsicherheiten und Ungenauigkeiten sind bei geologischen Modellen unvermeidlich und führen dazu, daß man in einem geologischen Informationssystem nicht einfach einen bekannten, höchstens noch zu verbessernden Tatbestand speichert, sondern ein Modell bzw. Versionen von Modellen, die das Ergebnis komplexer Überlegungen, Konstruktionen und Berechnungen sind.

Abbildung 1.2 zeigt einen Ausschnitt aus einem geologischen 3D-Modell des Untergrundes. Der Ausschnitt aus dem Modell der „Schaumburg-Lippeschen Kreidemulde“ [2] in Niedersachsen gibt ein gutes Beispiel für die Komplexität solcher Modelle (eine nähere Beschreibung des Forschungsprojektes folgt in Kapitel 3.3.1). Vor allem die in der Mitte des Modells zu sehenden Störungsflächen mit Überlappungen und die Begrenzungsflächen eines Salzstockes (rechts im Bild) können nicht mehr mit 2.5D Darstellungen verarbeitet werden.

Neben der geometrischen 3D-Modellierung zeichnen sich geometrische Modelle im 3D ebenfalls durch eine größere Vielfalt und Komplexität der geometrischen und topologischen Beziehungen und Operationen aus. Man vergleiche etwa die möglichen Fälle der Lage zweier Dreiecke in der Ebene (2D-Raum) zueinander mit den möglichen Konfigurationen im 3D-Raum. Entsprechend vielfältig sind die möglichen Ergebnisse einer Schnittoperation zwischen zwei Dreiecken im 3D-Raum. Dies führt im Allgemeinen zu einer deutlich höheren Komplexität in den Algorithmen zur Verarbeitung von 3D-Geoobjekten unter Berücksichtigung vieler Sonderfälle.

[2] Quelle: Siehl & Cremers, Universität Bonn – DFG-Projekt 3D-Karte

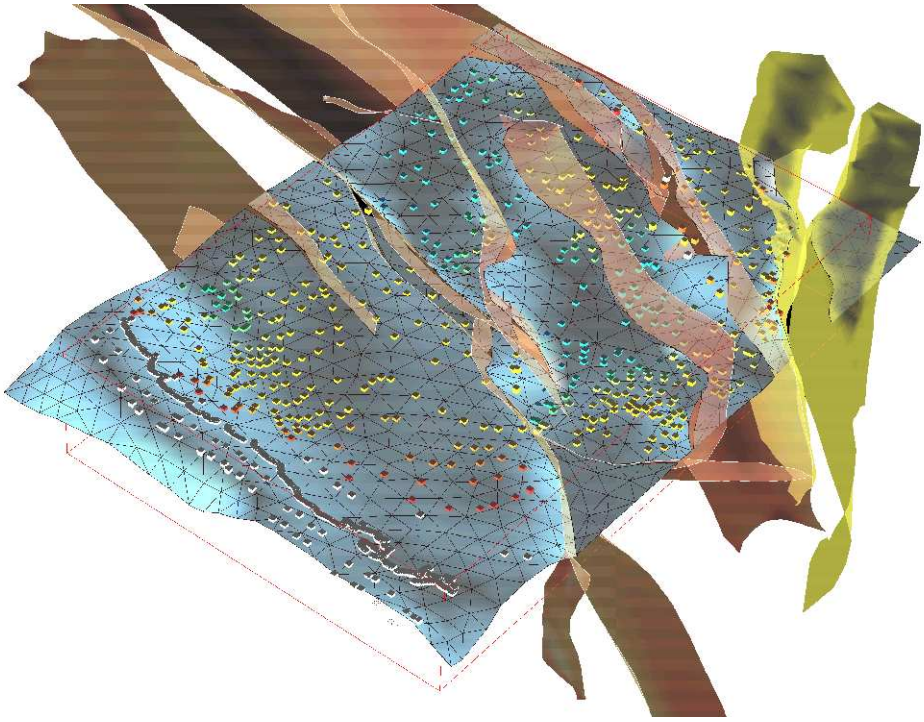

Abbildung 1.2: Ausschnitt aus einem komplexen 3D Untergrundmodell (Quelle: [22])

1.3.2 Anwendungsfelder

3D Datenbanksysteme können eine verbesserte Arbeitsunterstützung gegenüber bisherigen Formen der Modellverwaltungen (z.B. Dateisystem) in den Geowissenschaften bieten. Für grundlegende Vorteile einer Datenverwaltung mit Datenbankmanagementsystemen (z.B. Mehrbenutzerzugriff, Datenintegrität) sei auf die Literatur zu Datenbanksystemen in [35, 129, 54] verwiesen. Weitere Informationen zu allgemeinen Anwendungsfeldern für 3D Geoinformationssysteme, unabhängig der Art der Datenverwaltung, finden sich in [100, 101, 127].

Grundlegende Anwendung finden Datenbanken in der Geologie und Ingenieurgeologie in der effizienten Verwaltung von 3D/4D-Modellen [15]. Zusätzlich zu den allgemeinen Vorteilen von Datenbanksystemen wie Mehrbenutzerbetrieb bzw. erhöhte Datensicherheit ermöglicht ein 3D-Datenbanksystem – gegenüber derzeitigen Dateisystem-basierten Lösungen – eine kompakte übersichtliche Verwaltung der 3D-Modelldaten mit der Möglichkeit von Suchanfragen auf den Daten. Suchanfragen in Datenbanksystemen können dabei gesamte Modelle, aber auch nur definierte Teilbereiche und auch Teilobjekte von Modellen zur Bearbeitung oder Visualisierung zurückliefern. Durch die Unabhängigkeit von einem bestimmten Datenformat einer spezifischen Applikation kann ein Datenbanksystem bei

Anfragen verschiedene Datenformate aus dem internen Datenmodell liefern bzw. konvertieren.

Neben der Bereitstellung existierender Modelle kann ein Datenbanksystem auch Unterstützung bei der geologischen Modellbildung geben. Durch die Möglichkeit verschiedene Zustände eines Modells im Datenbanksystemen zu verwalten (z.B. über Versionsverwaltung), besteht die Möglichkeit mit einer externen Modellierungssoftware den gesamten Entstehungsprozeß eines 3D-Modells ohne eine Überschreibung älterer Modellzustände zu begleiten. Neben der reinen Verwaltung dieser Modellzustände bietet ein solches Datenbanksystem auch die Möglichkeit zu älteren Modellzuständen zurückzugehen bzw. Änderungen zwischen verschiedenen Modellzuständen zu analysieren.

Als Kommunikationshilfe unter Geowissenschaftlern können in Datenbanksystemen vorgehaltene Untergrundmodelle und Modellzustände mit Techniken der Virtuellen Realität (Virtual Reality – VR) gekoppelt werden. Hiermit kann es prinzipiell Wissenschaftlern ermöglicht werden, in einem speziellen virtuellen Realitätsraum gemeinsam durch ein 3D-Modell virtuell zu laufen und über das Modell und die dahinterstehenden Prozesse zu diskutieren. Innerhalb der Ingenieurgeologie, die entgegen der Geologie auch viele oberirdische Anwendungsbereiche abdeckt, können auch Techniken der Erweiterten Realität (Augmented Reality – AR) angewandt werden, um die Ingenieure bei der Interpretation und Auswertung von 3D Daten im Gelände zu unterstützen [75, 19]. Im Bereich der unterirdischen Anwendungsbereiche, wie z.B. dem Tunnelbau kann dagegen, wie bei der Geologie, die Technik der Virtuellen Realität benutzt werden.

An konkreten Anwendungsfeldern für 3D-Datenbanksysteme kann in der Geologie z.B. der Bergbau, der Tagebau. die Erdölexploration und wissenschaftliche Untersuchungen geowissenschaftlicher Prozesse genannt werden. Im Berg- und Tagebau können Datenbanksysteme ebenfalls in der Abbauplanung durch Bilanzierungen der 3D-Modelle und der Einbringung aktueller Abbaudaten verwendet werden. In der Ingenieurgeologie sind mögliche Anwendungsfelder beispielsweise der Tunnelbau, Untersuchungen zur Standsicherheit des Untergrundes, Steinbruchberatungen und Talsperrensanierungen. Mit mobilen Datenbanken besteht darüberhinaus auch die Möglichkeit, Anfragen auf der Datenbasis in einem verbindungslosen Zustand zu stellen. Anwendungsfelder mobiler Datenbanken sind damit jegliche Anwendungsfelder, bei denen keine Netzverbindung zu festen Server-Datenbanksystemen existiert oder garantiert werden kann. Hierunter fallen der Untertage-Bergbau in der Geologie und der Tunnelbau in der Ingenieurgeologie.

1.3.3 Anforderungen an 3D Datenbanksysteme

Die Verwaltung von 3D-Modellen inklusive ihrer thematischen Sachdaten kann als die Grundanforderung an 3D-Datenbanksysteme angesehen werden. Da geowissenschaftliche Anwendungen oftmals eine echte 3D Darstellung, z.B. bei überhängenden Störungsflächen,

benötigen, muß ein vollständiges 3D Datenmodell unterstützt werden. Das 3D Datenmodell darf keine Beschränkungen – z.B. nur konvexe Körper – beinhalten, um alle natürlich vorkommenden Phänomene modellieren zu können. Die thematischen Sachdaten müssen einerseits pro Objekt unterstützt werden, andererseits ist in vielen Anwendungsbereichen der Simulation auch eine Unterstützung von thematischen Sachdaten auf geometrischen Teilbereichen eines 3D-Objektes des Modells nötig. Hiermit können beispielsweise innerhalb eines 3D-Körpers Stoffverteilungen und andere physikalische Größen modelliert werden.

Anfragen auf den Datenbeständen müssen die Möglichkeit bieten komplette 3D-Modelle, einzelne Objekte von 3D-Modellen, aber auch Teilobjekte in Bereichen eines 3D-Modelles zu verarbeiten. Die Erzeugung von neuen berechneten Objekten in anderer dimensionaler Darstellung kann dabei in die Anfragen integriert werden. So sind beispielsweise Anfragen wie – „Liefere den Teilbereich im räumlichen Bereich $(x_{min}, y_{min}, z_{min}; x_{max}, y_{max}, z_{max})$ aus dem 4D-Modell zum Zeitpunkt t als 3D-Modell“ – möglich. Diese Anfrage erzeugt dabei neue 3D Objekte. Die Komplexität größerer 3D-Modelle stellt weitere Anforderungen an die Anfrageverarbeitung. Je nach Leistungsfähigkeit des Clients treten unterschiedliche Anforderungen an die Auflösung und Detaillierung der angefragten Daten auf. So kann z.B. ein Client zuerst eine Übersicht mit geringer Auflösung anfordern, um anschließend in einem damit ausgewählten Ausschnitt einen höheren Detaillierungsgrad abzufragen.

Verwaltung raumzeitlicher Modelle Zeitabhängigkeit in den Geowissenschaften tritt in zwei verschiedenen Formen auf. Zum einen befindet sich jedes 3D-Modell in einem jeweils aktuellen Bearbeitungszustand. Dieser Zustand wird durch Änderungen am Modell in Form von Update-Operationen fortgeschrieben. Allerdings kann die Dauer der Bearbeitung eines 3D-Modells die Unterstützung sehr langer Transaktionen erforderlich machen. Hier erweisen sich die klassischen pessimistischen Transaktionsverfahren (siehe z.B. [35, 54, 132], vgl. Kapitel 2.2) als ungeeignet. Auf die Notwendigkeit der Unterstützung langer Transaktionen für die geologische Modellierung geht der nächste Abschnitt ein.

Zum anderen existieren explizit zeitabhängige 3D-Modelle, im folgenden als 4D-Modelle bezeichnet. Hier bildet die Zeit t neben den drei räumlichen Dimensionen einen unabhängigen Parameter, welcher ein veränderliches 3D-Modell kontrolliert. Zeitabhängige 3D-Modelle können als Ergebnisse einer numerischen Prozeßmodellierung oder aus der geologischen Rückformung gestörter und gefalteter Strukturen entstehen, aber auch z.B. aus in kurzen Zeitabständen gemessenen Bewegungen und Verformungen eines geologischen 3D-Körpers. Für die Unterstützung von 4D-Modellen innerhalb eines Datenbanksystems wird, neben neuen erweiterten Datenmodellen mit Zeitbezug, auch ein effizienter raumzeitlicher Zugriff auf die 4D-Modelle anhand von räumlichen, zeitlichen und thematischen Selektionskritieren benötigt. Zusätzlich müssen neue Konstruktionsalgorithmen zum Übergang zwischen 4D- und 3D-Modellen im Datenbanksystem bereitgestellt werden. Beispiel für

einen solchen Übergang ist die Abfrage eines 3D-Modells zu einem bestimmten Zeitpunkt t aus dem 4D-Modell.

Unterstützung bei der geologischen Modellierung Der Modellierungsprozeß geologischer 3D-Modelle basiert auf einem kontinuierlichen Prozeß der Interpretation der geologischen Meßdaten und deren Integration in das vorläufige Modell. Dieser iterative Prozeß wird über einen langen Zeitraum fortgeführt, bis ein erstes stabiles Modell existiert. Ausgehend von diesem Modell erfolgt dauerhaft eine weitere Bearbeitung und Verfeinerung des Modells durch neue Erkenntnisse.

In diesem Anwendungsfall arbeiten mehrere Personen bzw. Gruppen getrennt am gleichen geologischen Modell. Änderungen werden meist jedoch nicht individuell, sondern bei einem gemeinsamen Abgleich in Besprechungen bzw. nach Rücksprache, vollzogen. Die Bearbeitungszeiten solcher Modelle können dabei sehr lange (u.U. mehrere Wochen oder Monate) dauern. Wie bereits angesprochen, sind solch lange Bearbeitungszeiten nicht mit klassischen, sperrorientierten Transaktionsmodellen möglich. Aufgrund des kooperativen Ansatzes der Modellentwicklung in der Geologie können jedoch andere Ansätze für eine mehrbenutzerorientierte Bearbeitung verfolgt werden.

Ein solcher Ansatz für kooperative Bearbeitungen ist das Versionsmanagement (zu Versionsmodellen siehe Kapitel 2.3). In der Literatur finden sich auch die Begriffe Workspacemanagement [96] und Alternativenmanagement [1]. Dieser Ansatz geht von einer kooperativen Zusammenarbeit der Nutzer des zentralen Datenbanksystems aus. Im Gegensatz zu den klassischen Transaktionsverfahren resultieren hier Änderungen an den Datenbankobjekten nicht in einem Update der Datenbank, sondern in der Speicherung des neuen Objektzustandes als eine Version. Die Datenbanksysteme stellen Methoden zur Verfügung, um solche Änderungen an den Datenobjekten, den Versionen, zu verwalten. Unter Verwaltung fällt hierbei die Möglichkeit zur Gruppierung von Versionen in spezielle Bereiche des Datenbanksystems mit spezifischen Zugriffsrechten. Über den gesamten Zeitraum der Modellentwicklung mit allen entstandenen Modellzuständen (Versionen) existiert somit eine Historie der Änderungen.

1.4 Ziele und Aufbau der Arbeit

Ziele der vorliegenden Arbeit sind, einerseits die Möglichkeiten von 3D Datenbanksystemen für mobile Endgeräte zu untersuchen und andererseits sich mit der Unterstützung von geowissenschaftlichen Anwendungen durch Datenbanken, insbesondere auf mobilen Endgeräten, auseinanderzusetzen. Hauptziel ist damit die Konzeption einer mobilen 3D Datenbank und deren Kopplung bzw. Integration mit einem Server-Datenbanksystem. Die

hierfür verwendeten Konzepte sollen anhand realer Anwendungsszenarien evaluiert werden. Die weitere Arbeit ist wie folgt untergliedert:

Kapitel 2 stellt die benötigten Grundlagen vor. Hierbei wird auf räumliche 3D Datenbanksysteme und mobile Datenbanken eingegangen. In den Grundlagen über mobile Datenbanken werden derzeitige Architekturen, sowie Konzepte für die Transaktionsverwaltung und Synchronisation im mobilen Umfeld vorgestellt. Das Kapitel schließt mit einem Überblick über verwandte und aktuelle Forschungsarbeiten und -gebiete sowie existierende Systeme ab.

Ein Konzept zur Verwaltung geowissenschaftlicher 3D Daten mit mobilen Datenbanken und deren Integration mit Server-Datenbanksystemen wird in Kapitel 3 vorgestellt. Es wird auf wesentliche Problemfelder eingegangen und es werden die grundlegenden Entscheidungen diskutiert.

In Kapitel 4 erfolgt die Beschreibung des Datenbanksystems DB3D. Dieses Datenbanksystem bildet die Grundlage für die prototypische Implementierung in der vorliegenden Arbeit. Neben der Architektur wird auf das entwickelte Datenmodell und das Dienst-Modell für den externen Zugriff eingegangen.

Die Erweiterung des Server-Datenbanksystems um ein Versionsmanagement zur Nutzung mit mobilen Datenbanken wird in Kapitel 5 eingehend beschrieben. Das Versionsmanagement stellt die Grundlage für die Unterstützung des Modellierungsprozesses in den Geowissenschaften und für die Integration von mobilen Datenbanken mit dem Server-Datenbanksystem dar.

Im Kapitel 6 wird auf die Evaluation des Prototypen anhand künstlicher Testdaten und realer geowissenschaftlicher Anwendungen eingegangen. Anschließend erfolgt in Kapitel 7 eine Diskussion der Konzepte, des Prototypen sowie der Optimierungsmöglichkeiten und Probleme.

Eine Zusammenfassung der vorliegenden Arbeit und ein Ausblick auf weitere Forschungsmöglichkeiten in diesem Bereich werden abschließend im Kapitel 8 gegeben.

Kapitel 2

Grundlagen

2.1 3D Datenbanksysteme

Einen entscheidenden Einfluß auf die Verwendbarkeit eines räumlichen Datenbanksystems für eine bestimmte Anwendung hat das Datenmodell zur Repräsentation der räumlichen Objekte. Daher werden im folgenden verschiedene geometrische Datenmodelle aus dem Bereich des *Solid-Modelling* vorgestellt. Anschließend wird auf verschiedene in räumlichen Datenbanksystemen verwendete Techniken für Zugriffsoptimierung und Algorithmen eingegangen. Der Abschnitt Interoperabilität beschäftigt sich mit dem Austausch von Geodaten. Gerade dieser Bereich hat in den letzten Jahren stark an Bedeutung gewonnen. Der Schwerpunkt dieses Kapitels liegt bei 3D-Datenbanksystemen, wobei Teilbereiche (z.B. die Übersicht über Indexstrukturen) auch für 2D-Datenbanksysteme gelten. Für eine generelle Übersicht über räumliche Datenbanksysteme sei auf [50, 104, 119, 26] verwiesen.

2.1.1 Geometrische und topologische Datenmodelle

Die geometrische Modellierung im 3D erweitert die geometrischen Repräsentationen im 2D – *Punkt*, *Linie* und *Fläche* – um die dritte Dimension des *Volumenkörpers*. Im 2D existieren mittlerweile einheitliche Datenmodelle (z.B. Simple Feature Specification - siehe [94]), die in den letzten Jahren an Akzeptanz stark gewonnen haben und vermehrt auch von kommerziellen Systemen verwendet bzw. unterstützt werden. Im 3D-Bereich existiert dagegen kein einheitliches geometrisches Datenmodell. Unterschiede treten hier vor allem bedingt durch verschiedene Anwendungsbereiche wie z.B. CAD/CAM für Gebäudearchitektur, Geologie oder 3D-Stadtmodelle auf. Ansätze für eine einheitliche Verwendung eines Datenmodelles sind nicht vorhanden und auch nicht unbedingt sinnvoll. Verschiedene Modelle sind unterschiedlich geeignet zur Darstellung eher vom Menschen konstruierter

Objekte oder natürlicher Objekte, was sich auch in den unterschiedlichen Anwendungsbereichen (CAD vs. Geologie) ausdrückt. Allerdings sind Ansätze vorhanden, um den Austausch von 3D-Daten bestimmter Modelle zu standardisieren. Hierauf wird in Kapitel 2.1.3 näher eingegangen.

Die folgende Übersicht zu geometrischen Datenmodellen orientiert sich an der vielfältigen Literatur zu diesem Gebiet z.B. [102, 103, 84, 56, 14]. Für eine vertiefte Auseinandersetzung sei beispielsweise auf Mäntylä [84] und Hoffmann [56] verwiesen.

2.1.1.1 Geometrische Modelle

Im CAD-Bereich werden 3D-Modelle grob in Kantenmodelle, Flächenmodelle und Volumenmodelle unterteilt. Ein Würfel beispielsweise wird in den drei Modellen aus seinen 12 Kanten (Gerüst), seinen 6 Flächen (Oberfläche) oder seinem Volumen (Körper) gebildet. Angelehnt an die Begrifflichkeiten in [14] werden im folgenden die geometrischen Modelle in vektorbasierte Modelle, Zerlegungsmodelle und analytische bzw. hybride Modelle unterteilt.

Vektorbasierte Modelle Als erstes Modell ist hier das *Kantenmodell* (Drahtmodell, Wireframe) zu nennen, bei dem die Geometrie von Objekten durch Liniensegmente oder Kurvenstücke repräsentiert wird. Es ist mehrdeutig, d.h. ein Kantenmodell kann als mehrere verschiedene Körper interpretiert werden. Das Fehlen expliziter Flächeninformation macht dieses Modell für räumliche Datenbanksysteme unbrauchbar, da darauf keine Algorithmen, beispielsweise für Schnittberechnungen, möglich sind.

Beim *Randmodell* (Boundary model) erfolgt die geometrische Repräsentation von Körpern durch ihre Begrenzungsflächen. Diese können Polygone, Rechtecke, Dreiecke, aber auch analytische Funktionen oder durch Interpolations- und Approximationsverfahren definierte Flächenstücke (Bezier- oder B-Spline-Flächen) sein. Im ersten Fall kann von einem *Vektor-Randmodell* gesprochen werden. Bei der Nutzung von analytischen Funktionen zur Flächenbeschreibung kann man dann vom *Funktions-Randmodell* sprechen.

Beim Vektor-Randmodell können natürliche Objekte in beliebiger Feinheit durch ihre Begrenzungsflächen approximiert werden. Eine gute Approximation gekrümmter Oberflächen zu erreichen bedeutet jedoch auch eine Steigerung der Komplexität durch eine sehr hohe Anzahl an Begrenzungsflächen. Abbildung 2.1 zeigt exemplarisch eine geologische Störung (*Abschiebung* des rechten Blocks entlang einer geneigten Ebene) im Vektor-Randmodell.

Zerlegungsmodelle Zerlegungsmodelle (space decomposition models) unterteilen den Raum, den ein Körper einnimmt, in eine Menge nicht überlappender, primitiver geometrischer Objekte. Dabei können diese primitiven geometrischen Objekte auch von unter-

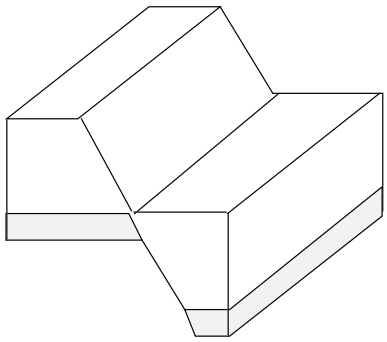

Abbildung 2.1: Geologische Störung im Vektor-Randmodell (Quelle: [14])

schiedlichem Typ sein. Man unterscheidet die *Zellzerlegungsverfahren* (cell decomposition) und das *Enumerationsverfahren* (spatial occupancy enumeration).

Bei den *Zellzerlegungsverfahren* werden die Körper durch beliebig im Raum liegende einfachere Zellobjekte zusammengesetzt. Das Zusammensetzen entspricht einer Vereinigung, bei der keine Überschneidungen erlaubt sind. Hierbei werden verschiedene Formen und Größen der Zellen in der Komposition eines Objektes zugelassen. Die Zellen müssen also nicht notwendigerweise identisch sein. Abbildung 2.2 zeigt eine Zellzerlegung anhand einer geologischen Störung.

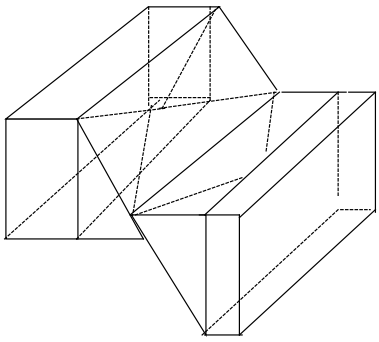

Abbildung 2.2: Geologische Störung dargestellt im Zellzerlegungsverfahren (Quelle: [14])

Beim *Enumerationverfahren* erfolgt die Darstellung eines Körpers durch die Zerlegung seiner Geometrie in identische Zellen. Diese sind in einem festen, regulären Raumgitter angeordnet. Die Zellen sind adjazent, verbunden und nicht überlappend. Im zwei dimensionalen Fall entspricht das Enumerationsverfahren der Raster-Repräsentation. Im drei dimensionalen Fall spricht man von einer Voxel-Repräsentation. Abbildung 2.3 zeigt die geologische Störung repräsentiert nach dem Enumerationsverfahren.

Um den hohen Speicherverbrauch einer solchen Darstellung zu reduzieren, existieren beispielsweise hierarchische Enumerationsverfahren wie der Octree (vgl. Samet [108, 107]).

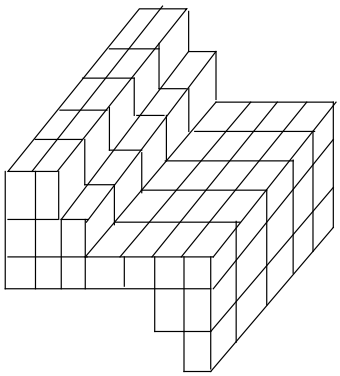

Abbildung 2.3: Geologische Störung dargestellt im Enumerationsverfahren (Quelle: [14])

Beim Octree wird die Geometrie nach dem divide-and-conquer-Ansatz binär unterteilt. Ein Oktand kann voll, teilweise gefüllt oder leer sein, je nachdem wie stark die Geometrie den Oktanden schneidet (siehe Abbildung 2.4). Ein teilweise gefüllter Oktant wird rekursiv in Unteroktanden aufgeteilt. Die Unterteilung endet, wenn alle Oktanden entweder leer sind, voll sind oder wenn eine festgesetzte Tiefe des Octrees erreicht wird. Die Unterteilungen können als ein 8 Wege-Baum dargestellt werden.

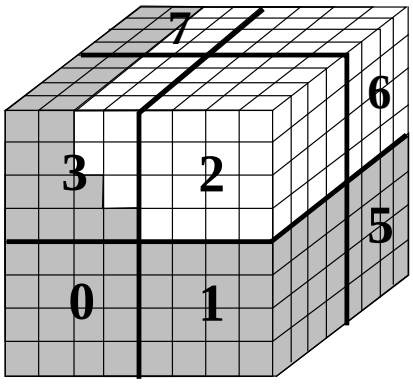

Abbildung 2.4: Darstellung der geologischen Störung in einem Octree (Quelle: [14])

Analytische Modelle Unter analytischen Modellen werden diejenigen Verfahren verstanden, bei denen anhand von Funktionen und Parametern aus Freiformflächen bestehende Geometrien beschrieben werden. Hierunter fallen das *Funktions-Randmodell*, das *Sweep-Modell* und das *parametrisierte Modell* (primitive instancing).

Beim Funktions-Randmodell werden die Begrenzungen des Körpers durch analytische Funktionen oder durch Interpolations- und Approximationsverfahren definierte Flächen gebildet. Das Modell hat den großen Nachteil, daß geometrische Eigenschaften schwierig

zu überprüfen sind und oft aufwendige Algorithmen verlangen.

Beim Sweep-Modell erfolgt die Definition von Körpern durch das Entlangführen einer Kontour an einer Raumkurve (Translationssweep), durch deren Rotation (Rotationssweep), bzw. durch eine Kombination dieser Sweep-Methoden. Sowohl die Form der Kontour als auch die Raumkurve sind durch Parameter definiert. Zum Beispiel kann eine Scheibe mit Radius r entlang eines Segmentes geführt werden und somit einen Zylinder erzeugen. In Abbildung 2.5 wird schematisch ein Beispiel für eine durch einen Translationssweep repräsentierte Geometrie einer geologischen Störung gezeigt. Die Teilstücke A, B und C sind dabei durch einen Translationssweep der dick umrandeten Geometrien entlang der y-Achse entstanden.

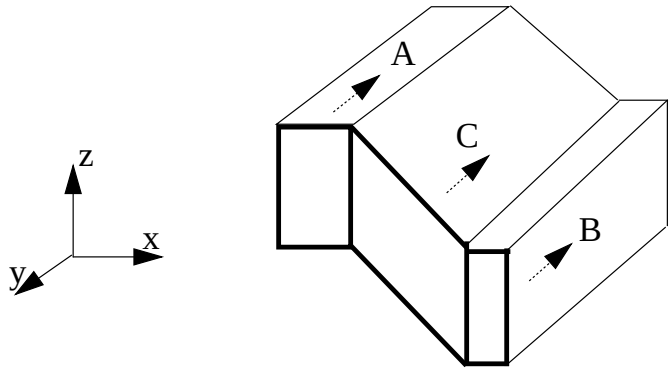

Abbildung 2.5: Geologische Störung durch Translationssweep dargestellt (Quelle: [14])

Beim parametrisierten Modell werden die Geometrien der Objekte (Primitive) durch eine feste Anzahl von Parametern beschrieben. Die Primitive werden hierbei durch Gleichungen definiert. Eine konkrete Ausprägung eines Primitives ist dann durch eine Menge numerischer Werte dieser Gleichung beschrieben.

Hybrides Modell - Konstruktion mit Raumprimitiven In der Konstruktion mit Raumprimitiven (Constructive Solid Geometry - CSG) ist ein Körper durch die mengentheoretische Kombination von primitiven Standardkörpern, von einfacherer Struktur, repräsentiert. Der Aufbau eines CSG-Objektes erfolgt aus einfachen Standard-Körpern (z.B. Quader, Kegel, Zylinder), durch Mengenoperationen (Vereinigung, Schnitt und Differenz) und Transformationen (Translation, Rotation und Skalierung). Die Konstruktion eines CSG-Objektes kann als binärer Baum (Konstruktionsbaum - vgl. Abbildung 2.6) dargestellt werden. Dabei repräsentieren die Blätter die primitiven Standardkörper und die Knoten die Operationen, also entweder eine Mengenoperation oder eine Transformation. Die primitiven Standardkörper selbst können in einer beliebigen Repräsentation, beispielsweise der parametrisierten Repräsentation oder der Rand-Repräsentation dargestellt sein, weshalb von einer hybriden Repräsentation gesprochen wird. Das Enumerationsverfahren

ist damit ein Sonderfall der Konstruktion mit Raumprimitiven, bei dem der einzig erlaubte Operator das Zusammensetzen von Geometrien ist. Abbildung 2.6 zeigt eine geologische Störung repräsentiert durch Konstruktion mit Raumprimitiven.

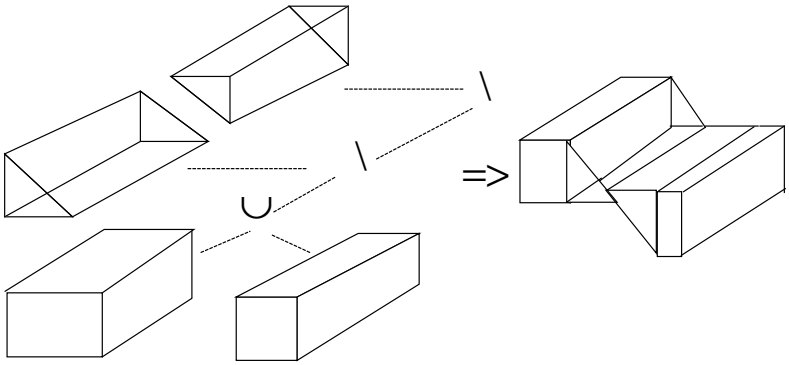

Abbildung 2.6: Geologische Störung repräsentiert durch Konstruktion mit Raumprimitiven (Quelle: [14])

Eignung für 3D Datenbanksysteme Die hier vorgestellten geometrischen 3D-Modelle werden von Breunig [14] ausführlich unter dem Aspekt der Eignung für 3D-Geoinformationssysteme diskutiert. Der Vergleich der verschiedenen Modelle erfolgt anhand von sieben Kriterien - Definitionsbereich, Gültigkeit, Nicht-Mehrdeutigkeit und Eindeutigkeit, Abgeschlossenheit der Operationen, Effizienz der geometrischen Algorithmen, Genauigkeit und Speicherplatzbedarf. Als Ergebnis der Untersuchungen werden für geowissenschaftliche Anwendungsbereiche das Vektor-Randmodell, das Enumerationsverfahren und die Zellzerlegung als am besten geeignet beurteilt.

2.1.1.2 Topologische Modelle

Für die Modellierung naturgeformter Objekte in geowissenschaftlichen Anwendungen existiert mit dem Modell der *simplizialen Komplexe* eine besonders geeignete Sonderform der Zellzerlegung, bei der auch die Topologie (Nachbarschaften zwischen Teilgeometrien) berücksichtigt wird. Hier erfolgt die Zellzerlegung durch Zellen gleichen Typs, z.B. bei der Flächenmodellierung durch Dreieck-Zellen und bei der Volumenmodellierung durch Tetraeder-Zellen.

Die Zellen (sog. Simplexe), aus denen die simplizialen Komplexe aufgebaut werden, sind als die jeweils einfachste geometrische Struktur der jeweiligen Dimension definiert:

0-Simplex Knoten

1-Simplex Kante

2-Simplex Dreieck

3-Simplex Tetraeder

Ein k-simplizialer Komplex ist die Vereinigung aller k-dimensionalen und niedriger dimensionalen Simplexe. Die Dimension k eines k-simplizialen Komplexes ist definiert als die maximale Dimension der Simplexe im Komplex. Alle Simplexe sind selbst wiederum aus Simplexen der jeweils niedrigeren Dimension aufgebaut. Man spricht in Analogie zur Bezeichnung der Seiten eines Dreiecks/Tetraeders von Seitensimplexen eines Simplex. Ein 2-Simplex (Dreieck) ist also aus 1-Simplexen (Kanten) aufgebaut und ein 1-Simplex wiederum aus 0-Simplexen (Ecken der Kante). Abbildung 2.7 zeigt einen 2-simplizialen Komplex aufgebaut aus mehreren 2-Simplexen, 1-Simplexen und einem 0-Simplex.

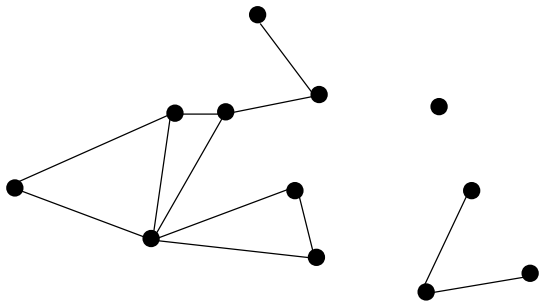

Abbildung 2.7: 2-simplizialer Komplex (Quelle: [14])

Für die Darstellung natürlicher Objekte in den Geowissenschaften sind vor allem die simplizialen Komplexe interessant, bei denen alle Simplexe eines Komplexes die gleiche Dimension d haben. Durch diese Einschränkung entsteht eine weitere Form der simplizialen Komplexe, die *simplizialen d-Komplexe*. Ein simplizialer d-Komplex ist definiert als eine Menge miteinander verbundener Simplexe der Dimension d. Abbildung 2.8 zeigt einen simplizialen 2-Komplex bestehend aus einer Menge miteinander verbundener 2-Simplexe und einen simplizialen 3-Komplex aufgebaut aus einer Menge von 3-Simplexen.

Mit der Hilfe solcher simplizialen d-Komplexe lassen sich in den Geowissenschaften beispielsweise gute Approximationen für Freiformflächen (2-Komplexe) und Volumina (3-Komplexe) darstellen. Die Eigenschaft einheitlicher Zelltypen der gleichen Dimension und die definierte Verbundenheit der Simplexe führt zu relativ einfachen Strukturen, die aus Sicht der Datenbankmodellierung und der algorithmischen Verarbeitung deutliche Vorteile bieten. Einen weiteren Vorteil bringt die einfache Struktur der Simplexe mit sich. Da sie in jeder Dimension aus der jeweils einfachsten Geometrie aufgebaut sind, und zusätzlich wiederum aus Simplexen der niedrigeren Dimension, lassen sich auf simplizialen Komplexen effiziente geometrische Algorithmen realisieren.

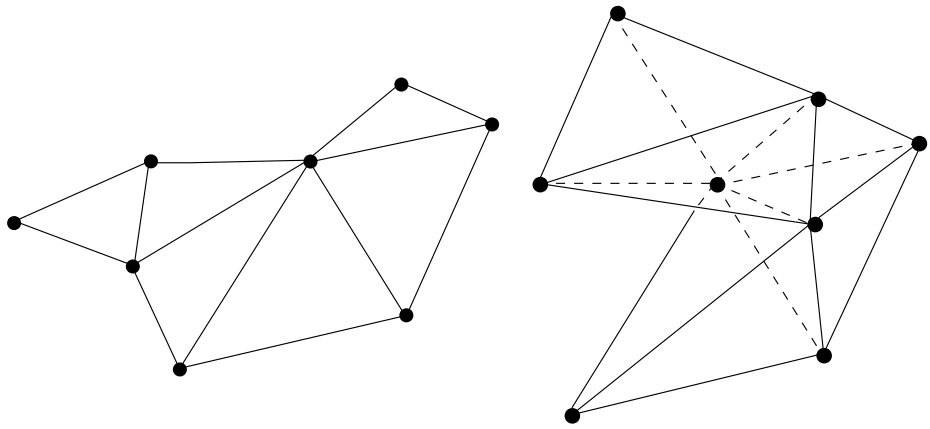

Abbildung 2.8: Beispiele für simpliziale d-Komplexe: simplizialer 2-Komplex (links) und simplizialer 3-Komplex (rechts)

2.1.2 Verhalten von 3D Datenbanksystemen

Es wird ein kurzer Überblick über Zugriffsmethoden für räumliche Datenbanksysteme gegeben. Anschließend werden weitere Techniken vorgestellt, die bei der Verarbeitung geometrischer Daten in Algorithmen angewandt werden. Einige dieser Techniken, die bei geometrischen Daten auftreten, kommen auch außerhalb von Datenbanksystemen in anderen geometrischen Anwendungen zum Einsatz und sind somit nicht rein auf den Datenbankbereich beschränkt.

2.1.2.1 Zugriffsmethoden (Indexe)

Das Gebiet räumlicher Zugriffsmethoden fand, durch steigende Relevanz räumlicher Informationen, in den letzten Jahrzehnten große Beachtung. Es existiert eine Vielzahl von Vorschlägen für Indexstrukturen für 0-dimensionale (Punkte) und höherdimensionale Geodaten. Einen Überblick über die Vielzahl von Vorschlägen geben beispielsweise die Übersichtsveröffentlichungen [93, 11, 38, 4].

Die in Standard-Datenbanksystemen zum Einsatz kommenden Zugriffsstrukturen sind zur Indexierung von eindimensionalen Daten geeignet und basieren auf der Ordnung der Daten anhand dieser Dimension, um eine Unterteilung des Zugriffsweges zu ermöglichen. In räumlichen Datenbanksystemen müssen Datenbankobjekte anhand ihrer Geometrie indexiert werden können. Aufgrund der mehrdimensionalen Struktur dieser geometrischen Daten sind die bisherigen Standard-Zugriffsstrukturen nicht geeignet. Räumliche Indexe sollten im Wesentlichen die folgenden Anfragetypen unterstützen:

- **Raum-Prädikat Anfrage:** Dieser Anfragetyp findet Objekte, die zu einem gege-

benen Raum-Objekt ein bestimmtes räumliches Prädikat erfüllen. Prädikate sind hierbei z.B. *intersects* oder *contains*.

- **Nachbarschaftsanfrage:** Dieser Anfragetyp findet Objekte, die zum Suchobjekt in einer gegebenen Nachbarschaftsbeziehung stehen.
- **Spatial Join:** Dieser Anfragetyp findet Objekt-Paare zweier verschiedener Objektmengen basierend auf einem räumlichen Prädikat (z.B. intersects).

Räumliche Daten in den Geowissenschaften und anderen Anwendungsgebieten besitzen meist keine einheitliche Geometrie und die Berechnung räumlicher Prädikate zwischen einzelnen Datenbankobjekten anhand ihrer exakten Geometrie ist sehr teuer. Räumliche Indexierungen approximieren die geometrischen Objekte daher durch einfachere Geometrieobjekte, wie beispielsweise dem achsenparallelen umschließenden Rechteck des Datenbankobjektes im 2D oder dem achsenparallelen umschließenden Quader im 3D. Diese einfacheren Geometrieobjekte sind dann im räumlichen Index enthalten und verweisen auf ihr zugehöriges Datenbankobjekt. Die Ausnutzung solcher Approximationsgeometrien in Indexen bzw. deren Wirkungsweise in den Stufen *Zelladressierung*, *grober Filter* und *feiner Filter* ist als das „3-Stufen-Modell“ [90] bekannt. Diese Approximation kann prinzipiell mehrstufig sein, so daß Datenbankobjekte in jeder Stufe eine bessere Approximation ihrer realen Geometrie bekommen. Ein Aufbau einer Indexstruktur mit mehreren Approximationen resultiert dann in einer Mehrfach-Prozessierungsstrategie [27] für räumliche Anfragen:

- **Zelladressierung:** Die Index-Struktur wird benutzt, um anhand der Approximationsgeometrie eine Menge an Ergebniskandidaten zu erhalten
- **Grober Filter:** Sind weitere Approximationsgeometrien vorhanden, können diese genutzt werden, um falsche Treffer der Vorgängerapproximation aus den Ergebniskandidaten herauszufiltern. Dieser Schritt wiederholt sich solange, bis keine weiteren Approximationsgeometrien mehr vorhanden sind.
- **Feiner Filter:** Die Ergebniskandidaten werden mit ihrer wirklichen Geometrie getestet, um die endgültige Ergebnismenge zu erhalten.

Grundsätzlich können Indexstrukturen unterteilt werden in *Punkt-Zugriffsstrukturen* zur Organisation mehrdimensionaler Punktobjekte (Point Access Methods - PAM) und in Zugriffsstrukturen für Objekte mit einer räumlichen Ausdehnung, den sogenannten *räumlichen Zugriffsstrukturen* (Spatial Access Methods - SAM).

Eine Taxonomie räumlicher Zugriffsstrukturen kann in die folgenden Gruppen erfolgen [11]:

- Transformationsansatz
  Der Transformationsansatz kann in zwei Kategorien – Transformation in einen multidimensionalen oder in einen eindimensionalen Attributraum – unterteilt werden. Bei der Transformation in einen multidimensionalen Attributraum (*Parameter space indexing*) werden Objekte mit n Vertices in einem k-dimensionalen Raum zu Punkten in einem nk-dimensionalen Raum überführt. Diese durch die Transformation entstandenen Punktobjekte können dann in existierenden Punkt-Indexen (z.B. Gridfile [89]) verwaltet werden. Die zweite Kategorie (*Mapping to single attribute space*) partitioniert den Datenraum in orthogonale Zellen der gleichen Größe, die nach verschiedenen Methoden nummeriert werden. Ein räumliches Objekt wird dann repräsentiert durch eine Menge von Nummern der Zellen in denen es enthalten ist. Hierüber kann dann die Verwaltung der Objekte in Standard-Indexstrukturen (z.B. B-Baum) erreicht werden. Beispiele für Nummerierungsmethoden (Space Filling Methods) sind die Z-Ordnung [97] und die Hilbert-Ordnung [55].

- Ansatz der nicht-überlappenden Raumpartitionierung
  Bei diesem Ansatz erfolgt eine Zerlegung des Raumes ohne Überlappungen der Partitionen (disjunkte Unterräume) und Verwaltung dieser Zerlegung in hierarchischen Baumstrukturen. Die einzelnen Unterräume enthalten Verweise auf die in ihnen enthaltenen indexierten Objekte. Bei diesem Ansatz muß weiterhin zwischen den Kategorien *Objektduplikation* und *Objektzerlegung* unterschieden werden. Bei der Objektduplikation wird ein Datenobjekt in allen Unterräumen indexiert, mit denen es sich schneidet (Beispiel Octree). Im Gegensatz dazu wird bei der Objektzerlegung das ursprüngliche Datenobjekt in mehrere disjunkte kleinere Objekte aufgeteilt, so daß jedes Subobjekt in einem Unterraum komplett enthalten ist (Beispiel R^+-Baum [117]).

- Ansatz der überlappenden Raumpartitionierung
  In diesem Ansatz erfolgt ebenfalls eine Zerlegung des Raumes, wobei eine Überlappung der Unterräume erlaubt ist. Hierdurch können räumliche Objekte immer komplett in einem Unterraum enthalten sein. Zu diesem Ansatz zählen z.B. der R-Baum [48] und der R*-Baum [8].

Im folgenden werden die beiden Indexstrukturen ausführlich beschrieben, die in der prototypischen Implementierung der vorliegenden Arbeit verwendet werden.

R-Bäume R-Bäume sind eine der wichtigsten räumlichen Zugriffstechniken. Sie sind höhenbalancierte Mehrwegebäume, die den Datenraum rekursiv in Unterräume aufteilen. Der hierarchische Aufbau eines R-Baumes ist vergleichbar mit dem eines B-Baumes. Sie können daher auch als mehrdimensionale Verallgemeinerung von B-Bäumen aufgefasst werden.

Die Knoten im R-Baum enthalten im Unterschied zum B-Baum Informationen über die räumliche Ausdehnung einer zugehörigen Menge von Objekten. Somit enthält jeder Knoten eine, die räumliche Ausdehnung beschreibende, Minimum Bounding Box (minimale, umfassende rechteckige Geometrie - Rechteck im 2D, Quader im 3D), die alle Minimum Bounding Boxen ihrer Nachfolgeknoten beinhaltet. Die Knoten stellen eine Untermenge der zu organisierenden Menge räumlicher Objekte dar. In den Blattknoten sind dann die Verweise auf die Datenobjekte enthalten. Zum Aufbau und der Pflege der Baumstruktur sind Lösch-, Einfüge- und Reorganisationsoperationen notwendig. Die Suche im R-Baum erfolgt ausgehend von der Wurzel durch Vergleich der Knoteneinträge mit den (n-dimensionalen) Suchintervallen.

Ein Beispiel für den Aufbau eines R-Baumes zeigt Abbildung 2.9, zur vereinfachten Darstellung an einem zweidimensionalen Beispiel.

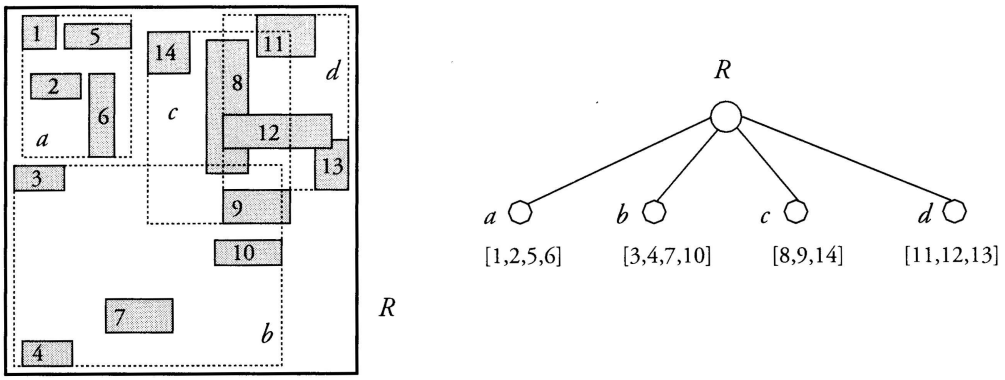

Abbildung 2.9: Aufbau eines R-Baumes im 2D (Quelle: [104])

Da die Suchraumaufteilungen nicht disjunkt sein müssen, d.h. Überschneidungen der Regionen erlaubt sind, ist unter Umständen eine Suche in mehreren Teilbäumen erforderlich. Dies ist auch im Allgemeinen der Fall, weshalb sich Varianten des R-Baumes diesem Sachverhalt von verschiedenen Seiten zu nähern versuchen.

Die zwei wesentlichen zur Verbesserung der Struktur beitragenden Konstruktionsvorschriften sind zum einen, die Überlappungen auszuschließen und zum anderen, die gruppierenden Regionen möglichst kompakt zu halten, also eine weitestgehend geringe Raumüberdeckung zu erreichen. Da sich beide Kriterien gegenseitig ausschließen, ist ihre gleichzeitige Erfüllung unmöglich. Eine Verbesserung stellt der R*-Baum dar. Die Vorschriften zur Konstruktion des R-Baumes sind um Kriterien für das Einfügen neuer Elemente und Splitten von Knoten erweitert. Neue Elemente werden in diejenigen Knoten eingefügt, in denen sie einen minimalen Zuwachs der Raumüberdeckung zur Folge haben. In der Regel kann festgestellt werden, dass der R*-Baum eine Verbesserung des R-Baumes darstellt, allerdings auf Kosten teurerer Einfügeoperationen. Eine andere Variante des R-Baumes ist der R^+-

Baum. Bei seiner Konstruktion sind Überlappungen der Suchräume unzulässig. Objekte können daher in mehr als einem Blattknoten gespeichert werden, wenn sie in Folge der geforderten Disjunktheit geteilt wurden. Das unter Umständen jedoch notwendige rekursive Splitten von den Blattknoten bis zur Wurzel hat eine größere Verbreitung des R^+-Baumes in raumbezogenen Anwendungen verhindert.

Octree Octrees oder Oktalbäume sind die dreidimensionale Variante der Quadtrees [36]. Octrees stellen prinzipiell eine Punkt-Indexstruktur zur Indexierung von Punktobjekten im dreidimensionalen Raum dar, können jedoch auch zur Indexierung räumlich ausgedehnter Objekte verwendet werden. Octrees unterteilen den Datenraum rekursiv in acht reguläre Partitionen (Oktanten) und erzeugen so eine hierarchische Struktur. Die Unterteilung kann bei einer maximalen Anzahl von Teilungen oder bei einer Minimalgröße der Oktanten beendet werden. Abbildung 2.10 zeigt den Aufbau und die Struktur eines Octrees.

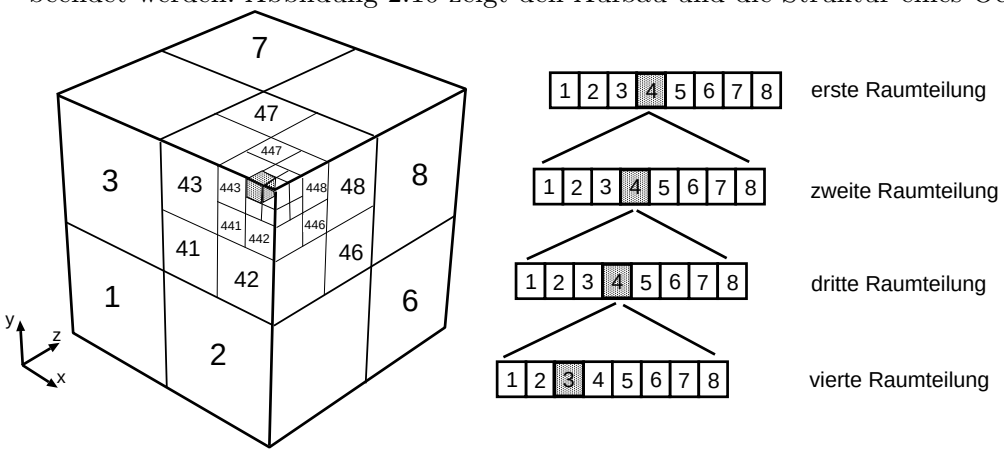

Abbildung 2.10: Octree

Eine Indexierung räumlich ausgedehnter Objekte erfolgt entweder durch eine Objektzerlegung oder durch eine Objektduplikation. Ein Vorteil des Octree gegenüber beispielsweise den R-Baum Varianten ist seine einfache Struktur, die einen sehr effizienten Aufbau ermöglicht. Octrees werden daher oft bei der Ad-hoc Indexierung benutzt, bei der Indexe erst zur Laufzeit aufgebaut werden, wenn die Algorithmen dies erfordern.

2.1.2.2 Geometrische Algorithmen

Grundsätzlich dienen Indexe zur Anfragebeschleunigung in Datenbanksystemen. Hierfür werden die vorgestellten Indexstrukturen mit Filter/Refine-Techniken kombiniert, um die Menge der zu selektierenden Objekte im Datenbanksystem bei einer Anfrage zu reduzieren. Neben der reinen Anfrage an eine Menge von Objekten bieten räumliche Indexstrukturen auch Möglichkeiten zur Unterstützung räumlicher Join-Operationen zwischen zwei

Mengen von Objekten anhand eines räumlichen Prädikats. Bei beiden Nutzungsarten der Indexstruktur wird der Index immer für Gesamtobjekte des Datenbanksystems benutzt. Die deutlich erhöhte Komplexität von Datenbankobjekten in der dritten Dimension macht darüberhinaus auch die Nutzung von internen Indexen zur Strukturierung der Teilobjekte eines Datenbankobjektes interessant.

Interne Indexe Datenbankobjekte im geowissenschaftlichen Bereich können, beispielsweise bei Verwendung des Modells der simplizialen Komplexe, aus vielen Tausend Teilobjekten oder mehr bestehen. Indexe können hier helfen die Menge der benötigten Teilobjekte in von der Datenbank bereitgestellten Operationen deutlich zu reduzieren.

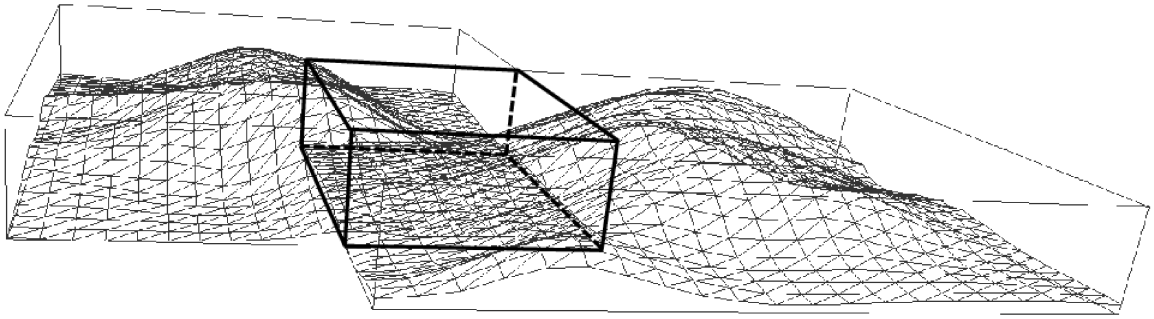

Abbildung 2.11: Schnittüberlappungsbereich zweier großer Flächenobjekte im 3D-Raum (Datengrundlage: Testdaten aus [85])

In Abbildung 2.11 sind zwei große Flächenobjekte mit ihren umschließenden Quadern dargestellt. Sollen beide Objekte auf Schnitt getestet werden, kann die Anzahl der aus dem Sekundärspeicher zu holenden Teilobjekte durch interne Indexe drastisch reduziert werden. Es werden zur Beantwortung des Tests nur diejenigen Teilobjekte beider Datenbankobjekte benötigt, die sich im Schnittbereich der beiden minimal umschließenden Quader befinden. Den Vorteilen interner Indexe bei Operationen auf komplexen Datenbankobjekten stehen allerdings auch Nachteile gegenüber. Zum einen wird ein – je nach Indexstruktur – deutlich höherer Speicherplatzbedarf benötigt. Zum anderen führt die Existenz interner Indexe auch zu erhöhten Verwaltungskosten der Datenbankobjekte, da Datenbankoperationen wie Einfügen bzw. Löschen eines Teilobjektes jeweils eine Indexverwaltungsoperation auslösen. Neben der Möglichkeit einer dauerhaften Indexstruktur besteht hier die Möglichkeit beispielsweise effiziente Ad-hoc Indexstrukturen (z.B. Octree) nur für die Dauer einer oder mehrerer Operationen aufzubauen und zu nutzen.

Robuste und fehlerfreie geometrische Operationen Ein weiteres Problem ist die Bereitstellung von robusten und fehlerfreien geometrischen Operationen in geometrischen

Anwendungen und Datenbanksystemen. Hoffmann geht in seinem Buch *Geometric and Solid Modeling* [56] ausführlich auf die Probleme geometrischer Operationen ein. Die Problematik tritt durch die Repräsentation von Fließkommazahlen (kontinuierliches Zahlensystem) in Computersystemen (binäres endliches Zahlensystem) auf und führt zu einer begrenzten Genauigkeit je nach benutzten Datentyp. Bei einer festgelegten Genauigkeit kommt weiterhin zum Tragen, daß je größer die Anzahl der Ganzzahlstellen einer Zahl ist, desto geringer ist die Anzahl der genau repräsentierbaren Nachkommastellen. Dieses Problem tritt beispielsweise auf, wenn keine lokalen Koordinatensysteme, sondern Referenzsysteme wie Gauß-Krüger oder UTM mit einer großen Anzahl fixer Ganzzahlen verwendet werden.

Güting und Schneider [51, 52] schlagen zur Lösung des Problems eine Struktur vor, die unterhalb der im System definierten Datentypen angesiedelt ist. Diese *geometric realms* genannte Struktur ist dabei als ein planarer Graph (im 2D) über eine Gitterstruktur endlicher Auflösung definiert. Knoten von Geoobjekten dürfen dann nur auf den durch diese Gitterstruktur definierten Schnittpunkten liegen. Ergebnisse von geometrischen Operationen werden immer auf den nächstliegenden Gitterschnittpunkt bezogen und falls nötig abgeändert. Die Auflösung dieses Gitternetzes bestimmt damit die erreichbare Genauigkeit der Geoobjekte.

Eine weitere einfache Methode, um einem Großteil der Genauigkeitsprobleme bei der Verwendung von Fließkommazahlen zu begegnen, ist die Verwendung einer Konstante ϵ (Epsilon) bei allen geometrischen Vergleichsoperationen. Diese Konstante definiert die maximale tolerierbare Abweichung, bei der zwei Koordinaten noch als gleich angesehen werden. Sämtliche geometrischen Vergleiche bzw. Tests auf geometrische Gleichheit innerhalb von Operationen finden dann innerhalb dieses Intervalls statt. Zwei Punkte sind demnach als geometrisch identisch anzusehen, wenn ihre Koordinaten innerhalb von ϵ gleich sind. Die Verwendung dieser Methode löst zwar die Genauigkeitsprobleme bei Verwendung von Fließkommazahlen, führt jedoch nicht automatisch zu robusten geometrischen Algorithmen. Für Beispiele und Ausführungen siehe [56].

2.1.3 Interoperabilität

Bei der Interoperabilität in Computersystemen sind zwei Bereiche zu unterscheiden. Zum einen die *Software-Interoperabilität*, die zu einer Möglichkeit des Zusammenspieles verschiedener Softwaresysteme führen soll. Zum anderen die *Daten-Interoperabilität*, die den Austausch von Daten zwischen Softwaresystemen und Programmen problemlos ermöglichen soll. Beim Austausch von Daten ist weiterhin zwischen der Syntax und der Semantik von Daten zu unterscheiden. Während derzeit existierende und aufzubauende Standardformate zur Datenübertragung in unterschiedlichen Bereichen das Problem einer einheitlichen Syntax von Daten angehen, ist bei der sematischen Interoperabilität zwischen Daten noch

keine Lösung in Sicht. Aktueller Gegenstand der Forschung ist hier die Nutzung von Ontologien [47, 122]. Der Schwerpunkt wird im folgenden auf die Daten-Interoperabilität, speziell unter dem Gesichtspunkt der dritten Dimension, gelegt.

Zum Erreichen von Software-Interoperabilität über Programmiersprachen- und Betriebssystem-Ebenen hinweg existieren schon seit längerem als Middleware konzipierte Software-Architekturen. Eine der früheren Middleware-Architekturen ist die von der Object Management Group (OMG) entwickelte Common Request Broker Architecture (CORBA), eine Programmiersprachen- und betriebssystemunabhängige Softwarearchitektur zur verteilten Objekt-Kommunikation [91]. Eine ähnliche Softwarearchitektur zur entfernten Objekt-Kommunikation, die im Gegensatz zu CORBA größtenteils auf die Programmiersprache Java beschränkt ist, stellt die Remote Method Invocation (RMI) dar. Der Vorteil von RMI liegt vor allem in der deutlich geringeren Komplexität, da hier keine Programmiersprachen unabhängige Zwischendefinition der Objektschnittstellen nötig ist. In den letzten Jahren werden die Lösungen mehr und mehr unter dem Aspekt des Internets und damit verbundener Sicherheitsproblematiken gesehen. Softwarelösungen basieren heute oftmals auf dem Basisprotokoll des Internets, dem HTTP-Protokoll. Vorteil dieses Protokolles ist die standardmäßige Berücksichtigung bei Firewall-Lösungen in Unternehmen. Als derzeitige Lösungen sind hier beispielsweise das Simple Object Access Protokoll (SOAP) [134] im Umfeld der Web-Services zu nennen. Eine den Web-Services ähnliche Art der Dienst-Bereitstellung wird im Geo-Bereich beispielsweise vom Open Geospatial Consortium[1] (OGC) mit den OGC-Services – z.B. Web Map Service, Web Feature Service, Coordinate Transformation Service – verfolgt. Allen aktuellen Lösungen gemein ist die Nutzung von XML [135] für den Datenaustausch und die Spezifikation der Schnittstellen.

Bei der Daten-Interoperabilität existieren auch im 3D mehrere Organisationen, die eine Standardisierung von Datenaustauschformaten, vornehmlich im jeweiligen speziellen Anwendungsbereich, vorantreiben.

Im Bereich des CAD ist hier vor allem die *Industrieallianz für Interoperabilität*[2] (Industrial Alliance for Interoperability - IAI) zu nennen. Diese entwickelt seit 1997 die Industry Foundation Classes (IFC) als Standard für den Datenaustausch zwischen allen Programmen im CAD-Bereich. Mit der aktuellen Version 2.2 [63] finden sich mittlerweile die Definitionen von ca. 600 Klassen zum Datenaustauch im Standard. Diese Klassen dienen nicht nur dem Austausch der Geometrie, sondern auch zum Austausch von anwendungspezifischen Sachdaten aus Bereichen wie Architektur, Haustechnik und Facilities Management. Der Standard ist mittlerweile auch als XML-Schemata ifcXML [63] umgesetzt. Für die Darstellung von Körpern ist beispielsweise die Basisklasse IfcSolidModel definiert. Zur Definition von Körpern in verschiedenen Repräsentationsmodellen sind davon Subklassen abgeleitet. So existiert z.B. mit IfcCsgSolid eine Klasse zur Repräsentation von Körpern

[1] URL: www.opengeospatial.org/
[2] URL: www.iai-international.org

in der CSG-Repräsentation bzw. mit IfcSweptAreaSolid eine Klasse für Körper in der Sweep-Repräsentation (vgl. Kapitel 2.1.1).

Das *Petrotechnical Open Standards Consortium*[3] (POSC) ist ein Zusammenschluß von Firmen und Institutionen der Öl- und Gas-Versorgung und entwickelt und pflegt seit 1990 Standards zum Austausch von Daten in diesem Anwendungsbereich. Die aktuellen Standards sind alle XML-basiert. Beispiele sind hier die WellLogML (Well Log Markup Language) zum Austausch von Well Log Datenbeständen oder PEF XML eine XML-Kodierung des Posc Epicentric Formats zum Austausch von Epicentric Daten. Ein Standard zur Interoperabilität von Anwendung im Bezug auf Objekte des Untergrundes (Subsurface Interpretation Business Objects) existiert seit langer Zeit als Entwurf, wird jedoch offenbar nicht weiterentwickelt. Demgegenüber arbeitet das POSC aktiv in dem Open Geospatial Consortium mit und ist beispielsweise an der Entstehung von GML 3 beteiligt, das eine Erweiterung von GML auf die dritte Dimension darstellt.

Im Bereich der 2D-GIS haben sich in den letzten Jahren eine Reihe von Standards – getragen durch das Open Geospatial Consortium – etabliert und werden mittlerweile von vielen Herstellern unterstützt. Hier kann im Bereich der Daten-Interoperabilität die Geographic Markup Language (GML), im Bereich der Datenübertragung die Spezifikationen für den Web Map Service (WMS) und Web Feature Service (WFS) und im Bereich der Standardisierung von Anfragen die Spezifikationen der Simple Features for SQL (SFS) und die Filter Encoding Language genannt werden. Für 2D-GIS sind seit längerem Geodatenbanksysteme verfügbar, die einen einheitlichen Zugriff über die Simple Features for SQL ermöglichen und die Geodaten im GML 2 Format ausliefern können. Mit der Version 3 der Geographic Markup Language hat das OGC seine XML-Datenbeschreibungssprache, neben vielen Neuerungen wie beispielsweise der Einführung von 2D Features mit Topologie, auch auf die dritte Dimension erweitert. Diese Erweiterung führte ebenfalls zu einem Abgleich mit den Standards der ISO Serie 19100 des ISO Technical Committee 211 (ISO/TC 211 2000), zu welchen GML 3 nun konform ist. Während mit dieser Erweiterung viele Anwendungen (z.B. 3D-Stadtmodelle) bereits unterstützt werden können und auch bereits mit dem Web Terrain Service (WTS) entsprechende Anwendungsspezifikationen existieren, ist GML 3 noch ein gutes Stück von einer generellen Verwendbarkeit in den verschiedensten Gebieten geowissenschaftlicher Anwendungen entfernt.

Für den Bereich der Geowissenschaften fehlen beispielsweise die Definitionen von Simplexen und Simplizial-Komplexen im Standard. In der derzeitigen Version sind Körper durch die Flächen ihrer Begrenzungshülle definiert und bieten damit keine Möglichkeit der Darstellung über Zellzerlegung an. Für den Bereich der Geowissenschaften befindet sich mit der eXploration and Mining Markup Language (XMML) [32] eine Erweiterung von GML 3 hin zu den Anforderungen geowissenschaftlicher Anwendungen in der Entwicklung. XMML

[3] URL: www.posc.org

bietet weitere Schemata beispielsweise zur Modellierung von Körpern durch Zellzerlegung (Tetraeder-Komplexe) oder für spezifische Objekte wie Bohrungen an.

2.1.4 Relevanz für die Arbeit

Die Entscheidung für ein geometrisches Modell im 3D ist abhängig vom Anwendungsgebiet. Da die vorliegende Arbeit im Bereich geowissenschaftlicher Anwendungen entsteht, wird für das Datenbanksystem ein Modell benötigt, welches vor allem naturgeformte Objekte darstellen kann. Neben der Darstellbarkeit sind für die Implementierung in einem Datenbanksystem mit den zugehörigen Operationen auch eine einfache Struktur und einfache Algorithmen von Vorteil. Das in Kapitel 4 vorzustellende Server-Datenbanksystem DB3D basiert auf dem Modell der simplizialen Komplexe und implementiert dieses für die Dimensionen 0 bis 3 im 3D-Raum.

Das Modell der simplizialen Komplexe bietet eine sehr einfache Struktur der Zellzerlegung in Zellen des gleichen Typs. Damit kann eine einfache konstante interne Nachbarschaftstopologie zwischen den Zellen aufgebaut werden, die dann in geometrischen Algorithmen verwendet werden kann. Der Aufbau von Datenbankobjekten aus einer großen Anzahl von Zellen führt zu der Möglichkeit einer internen räumlichen Indexierung der Datenbankobjekte. Vorteil ist hierbei vor allem die Selektion von Zellen, ohne das gesamte Objekt zu durchsuchen. Ein interner räumlicher Index kann in vielen Algorithmen im Zusammenspiel mit der internen Topologie eine deutliche Beschleunigung der Verarbeitung bewirken.

Die Problematik der Darstellung von Fließkommazahlen in Computersystemen beeinflußt eine Vielzahl von Algorithmen in einem Datenbanksystem. Der Aufbau einer internen Topologie für ein neues Datenbankobjekt basiert beispielsweise zu einem Großteil auf Vergleichsoperationen zwischen Punktkoordinaten. In DB3D werden daher jegliche geometrische Vergleichsoperationen über eine definierte Toleranz (Epsilon-Konstante) der Datenbankobjekte geprüft.

Der Bereich der Interoperabilität hat große Relevanz für die vorliegende Arbeit, da nicht eine vollständig uniforme IT-Landschaft zwischen den Server-Datenbanksystemen und den mobilen Datenbanken bzw. den jeweiligen Geräten mit ihren Betriebssystemen garantiert werden kann. In Rahmen dieser Arbeit kann jedoch nicht jeder Aspekt der Interoperabilität berücksichtigt werden. Um die Plattformunabhängigkeit aus Sicht der Betriebssysteme zu erreichen, erfolgen die Implementierungen innerhalb der vorliegenden Arbeit in der Programmiersprache Java™. Eine vollständige Interoperabilität zwischen beliebigen Programmiersprachen und Plattformen wird im Rahmen der Arbeit nicht angestrebt. Aus Sicht der Daten-Interoperabilität werden Daten in einem definierten XML-Format übertragen. Die bereits aufgezeigten Probleme im Bereich der Standardisierung von 3D-Datenformaten führt jedoch dazu, daß bisher noch ein eigenes XML-Austauschformat, optimiert für simpliziale Komplexe, verwendet wird.

2.2 Mobile Datenbanken

Wie in der Einleitung in Kapitel 1.2 bereits aufgeführt, führt die Einführung von mobilen Geräten in die vorhandenen Informationsstrukturen zu neuen Herausforderungen und Möglichkeiten. Zu den dort aufgeführten Herausforderungen kommen im Kontext mobiler Datenbanken weitere hinzu. Für einen ausführlichen Überblick der Herausforderungen und offener Probleme mobiler Datenbanken sei auf [6, 33, 110], und für eine tiefergehende Beschäftigung mit mobilen Datenbanksystemen auf [83, 61] verwiesen.

Im folgenden werden die für die Arbeit relevanten Bereiche der Architektur mobiler Datenbanken und der Modelle zur Transaktionsunterstützung näher ausgeführt. Einführend wird auf zu unterscheidende Anwendungsklassen in mobilen Datenbanken und die möglichen Verbindungszustände zwischen mobilen und stationären Datenbanken eingegangen.

2.2.1 Anwendungsklassen

In [43] werden zwei Klassen von Anwendungen in mobilen Datenbanken unterschieden. Beiden Anwendungsklassen ist gemein, daß die Bearbeitung von Daten auf der mobilen Datenbank im unverbundenen Zustand erfolgt.

Die erste Anwendungsklasse ist die der *nicht-kooperativen Anwendungen*. Allen nicht-kooperativen Anwendungen gemein ist eine größtmögliche Trennung der Aufgabengebiete der einzelnen mobilen Nutzer und damit auch der zugrundeliegenden Datenbasis. Als Beispiel wird in [43] beispielsweise die Arbeit von Außendienstmitarbeitern aufgeführt, deren Aufgabengebiet meist organisatorisch abgegrenzt ist. Diese Abgrenzung schlägt sich auch in der Datenbasis nieder, so daß bei dieser Klasse von Anwendungen mit relativ wenigen Konflikten, also konkurrierende Änderungen am Datenbestand, gerechnet werden kann.

Die *kooperativen Anwendungen* bilden demgegenüber die zweite Anwendungsklasse. In kooperativen Anwendungen greifen verschiedene Nutzer auf einen gemeinsamen Datenbestand zu, womit in dieser Anwendungsklasse mit relativ vielen konkurrierenden Änderungen am Datenbestand gerechnet werden muß. Als Beispiel für kooperative Anwendungen können zum Beispiel Entwurfsanwendungen im Maschinenbau oder ähnlicher Design-Disziplinen aufgeführt werden. Ebenfalls sind die Anwendungen in den Geowissenschaften vornehmlich der kooperativen Anwendungsklasse zuzuordnen. Ein wichtiges Beispiel einer kooperativen Anwendung in den Geowissenschaften ist die gemeinsame Arbeit an einem Untergrundmodell in der Geologie (vgl. Kapitel 1.3.3).

2.2.2 Verbindungszustände und Trennungsmodi

In der Einleitung (Kapitel 1.2.2) wurde bereits auf die Besonderheit der Verbindungen zu Netzwerken bei mobilen Datenbanken hingewiesen. Dabei wurden die drei Verbindungs-

zustände fest verbunden (*connected*), schwach verbunden (*weak connected*) und unverbunden (*disconnected*) unterschieden.

Für den Zustand des unverbundenen Arbeitens können weiterhin mehrere verfeinerte Trennungsmodi unterschieden werden. In [57] beschreiben Holliday et al. diese verfeinerten Trennungsmodi innerhalb eines verteilten mobilen Datenbanksystems. Die Trennungsmodi unterscheiden sich in den verschiedenen Stufen der weiteren Bearbeitungsmöglichkeit der Daten in der getrennten mobilen Datenbank. Sie unterscheiden zwischen den Modi *basic sign-off*, *check-out*, *relaxed check-out* und *optimistic check-out* mit jeweils weiteren Unterteilungen.

Im *basic sign-off* Modus erhält die getrennte mobile Datenbank nur Lesezugriff, während die restlichen Datenbanken ohne Einschränkungen weiterarbeiten dürfen. Im weitergehenden *check-out* Modus wird eine Untermenge des Datenbestandes für Änderungsoperationen auf der mobilen Datenbank ausgewählt. Die drei Unterteilungen *checkout with partition*, *check-out with mobile read* und *check-out with system read* geben die Lese-/Schreibrechte für die mobile und die restlichen Datenbanken im System in dem Unterteilungsmodi wieder. Der *relaxed check-out* Modus ist eine Kombination aus *check-out with mobile read* und *check-out with system read*. Der Modus *optimistic check-out* setzt gegenüber allen bisherigen Modi keinerlei Restriktionen in den Lese-/Schreibrechten. Sowohl für die getrennt arbeitende mobile Datenbank als auch für die verbleibenden Datenbanken im System existieren auf allen Daten Lese- und Schreibrechte. Hier wird das Auftreten von Konflikten bei Updates auf den ausgecheckten Datenbeständen akzeptiert und bei erneuter Verbindung mit dem System aufzulösen versucht.

2.2.3 Architekturen mobiler Datenbanksysteme

Für mobile Informationssysteme existieren verschiedene Architekturen (siehe [99]), ausgehend von der klassischen Client-Server-, über Client-Agent-Server- und Client-Interceptor-Server-Architekturen bis hin zu Peer-To-Peer Modellen, sowie Architekturen, die auf mobilen Agenten basieren. Nicht alle Architekturen sind auch übertragbar auf die mobile Datenverwaltung, in denen Nutzer nicht nur Daten auf mobilen Geräten anfragen, sondern auch bearbeiten wollen. Im folgenden werden Architekturen für die Datenverwaltung mit mobilen Datenbanken ausgeführt.

Verteilte Datenbanksysteme Mobile Datenbanksysteme können als Erweiterung verteilter Datenbanksysteme (siehe [137]) betrachtet werden [33, 60]. Hierbei sind die mobilen Datenbanken als mobile Knoten, neben den stationären Knoten, in das verteilte Datenbanksystem eingebracht.

Im unverbundenen Zustand arbeitet dabei die mobile Datenbank auf replizierten Daten zur Beantwortung lokaler Anfragen. Sie stellt selbst keine Daten als Server im verteilten

System zur Verfügung. Für verteilte Datenbanksysteme existieren mehrere Replikationsverfahren. Die Möglichkeit des Einsatzes dieser Verfahren in einem verteilten Datenbanksystem mit mobilen Knoten wird in [60, 59] behandelt.

Client/Server-Datenbanksysteme In mobilen Datenbanksystemen existiert nicht das klassische Client/Server-Konzept. In klassischen Client/Server-Datenbanksystemen werden Anfragen der Clients alleinig vom Server bearbeitet und ausgeführt oder beispielsweise Änderungstransaktionen nur vom Server abgeschlossen bzw. abgebrochen. Grundlegend in dieser Architektur ist eine häufige Verbindung zwischen Clients und Servern.

Bei mobilen Datenbanksystemen existiert, erweitert zur klassischen Architektur, auch auf den mobilen Geräten eine (vollwertige) Datenbank. Im Gegensatz zu den verteilten Datenbanksystemen übernimmt die mobile Datenbank hier aber nicht die Rolle eines Knotens in einem verteilten Datenbankverbund, sondern ist nur zur lokalen Verwaltung replizierter Datenbestände und zur lokalen Bearbeitung dieser Daten während der unverbundenen Phase zuständig. Somit ist die lokale Datenbank nicht als autonom, wie in einem verteilten Datenbanksystem, anzusehen. Änderungen an den lokal replizierten Daten werden bei einer erneuten Aufnahme einer Verbindung zum Server in den zentralen Datenbestand überführt.

Ein *erweitertes Client/Server-Modell* wird in [44] beschrieben. Das Modell besteht aus einer *konsolidierten* Datenbank, einer Anzahl von Replikationsservern und zwei Arten von Clients, fest verbundenen und mobilen. Kern des Modells ist die konsolidierte Datenbank, in der alle Änderungen von Daten eingebracht werden müssen. Der Datenaustausch zwischen den fest verbundenen und mobilen Clients erfolgt ausschließlich über diese Datenbank, während der Austausch zwischen mobilen Clients in bestimmten Fällen auch über einen gemeinsamen Replikationsserver erfolgen kann. Die mobilen Clients können unabhängig voneinander und von der konsolidierten Datenbank auf den replizierten lokalen Daten arbeiten. Dabei erfolgte Änderungsoperationen müssen später in die konsolidierte Datenbank eingebracht werden. Zur weiteren Entlastung der konsolidierten Datenbank erfolgt der Zugriff von mobilen Clients auf die konsolidierte Datenbank nur über dezidierte Replikationsserver, die jeweils einen Teil der konsolidierten Datenbank repliziert vorhalten und zum Teil auch die Konfliktbehebung und Reintegration übernehmen können. Zur weiteren Last- und Aufgabenverteilung bei der Replikation und Reintegration wird eine baumförmige Hierarchie von Replikationsservern vorgeschlagen.

Peer-To-Peer Architekturen Neben den eher *traditionellen* Architekturen werden in den letzten Jahren vermehrt Peer-To-Peer (P2P) Architekturen für ihre Verwendung in verteilten Datenmanagementsystemen untersucht.

In [113] wird eine Definition von Peer-To-Peer gegeben und die Unterschiede zwischen den

verschiedenen darunter verstandenen Architekturen (Pure-P2P, Hybrid-P2P) und zur klassischen Client/Server-Architektur abgegrenzt. Ein zentraler Punkt ist hierbei, daß Peers entgegen dem Client/Server-Konzept sowohl angebotene Services im P2P-Netzwerk konsumieren (Client-Rolle), als auch selbst Services anbieten (Server-Rolle). P2P-Architekturen bieten Möglichkeiten, die Einschränkungen bisheriger *traditioneller* verteilter Datenbanksysteme, wie statische Netzwerktopologie und hoher Administrationsaufwand in den zentralen Instanzen, zu lösen.

In [46, 10] werden Überblicke über Chancen und Probleme, die sich durch P2P-Architekturen für das Datenmanagement ergeben bzw. ergeben könnten, aufgeführt. Eine Übersicht über aktuelle Forschungen und Projekte in diesem Gebiet geben [109] im Anhang an die Beschreibung ihres XML P2P Datenbanksystems. Dieses System setzt voraus, daß alle Daten als XML-Dokumente mit assoziiertem XML-Schema im P2P-Netzwerk existieren. Demgegenüber geht das in [10] vorgeschlagene und formalisierte Local Relational Model (LRM) davon aus, daß alle Daten im P2P Netzwerk in lokalen (relationalen) Datenbanken vorgehalten werden.

2.2.4 Modelle zur Transaktionsunterstützung in mobilen Umgebungen

Bei der Transaktionsunterstützung in mobilen Umgebungen muß zwischen der Unterstützung für Transaktionen im drahtlos, schwach verbundenem (*weak connected*) und im unverbundenen (*disconnected*) Zustand unterschieden werden.

Die Unterstützung kann in die folgenden Bereiche unterteilt werden:

- Unterstützung für einen transparenten Übergang von laufenden Transaktionen zwischen Funkzellen in drahtlosen Netzwerken.
- Wiederaufnahme bzw. Rücksetzung von Transaktionen bei beabsichtigtem und unbeabsichtigtem Verbindungsabbruch.
- Unterstützung für das Arbeiten im unverbundenen Zustand.
- Unterstützung für langlaufende Transaktionen (größer einer Stunde).

Es existieren mittlerweile eine große Anzahl von Transaktionsmodellen für mobile Datenbanksysteme. Keines der Modelle deckt alle oben aufgeführten Bereiche vollständig ab. Einige Modelle bieten Unterstützung für das Arbeiten im schwach verbundenen Zustand, andere für das Arbeiten im unverbundenen Zustand. Für eine detaillierte Übersicht der letztjährigen Arbeiten auf diesem Gebiet sei auf [128, 118, 77] verwiesen.

2.2.4.1 Transaktionsmodelle

In [128] geben Türker und Zini einen ausführlichen Überblick über den derzeitigen Stand der Transaktionsunterstützung. Dabei gehen sie auch auf die derzeitig existierende Lücke zwischen akademischen Modellen und den verwendeten Modellen in kommerziellen mobilen Datenbanken ein. Diese Trennung soll auch in der folgenden Kurzübersicht über bestehende Modelle beibehalten werden.

Akademische Transaktionsmodelle Unter den akademischen Transaktionsmodellen finden sich sowohl Modelle zur Unterstützung der Mobilität von Transaktionen (Zellübergänge im Funknetz), sowie zur Unterstützung für unverbundenes Arbeiten. Eine Auflistung der Modelle und ihrer Unterstützung für Mobilität und Verbindungslosigkeit zeigt Tabelle 2.1. Für weitergehende Literatur zu den einzelnen Modellen sei auf [128] verwiesen.

	Unterstützung für	
Transaktionsmodell	Mobilität	Verbindungslosigkeit
Reporting/Co	x	-
Isolation-Only	-	x
MDSTPM	-	x
Weak/Strict	-	x
Two-Tier Replication	-	x
Kangaroo	x	-
Pro-Motion	(x)	x
Toggle	x	(x)
Moflex	x	-

Tabelle 2.1: Akademische Transaktionsmodelle (nach [128])

Die meisten Modelle beschränken sich auf die Unterstützung eines Falles. Lediglich die Modelle *Pro-Motion* und *Toggle* bieten Unterstützung für beide Fälle an.

Bei *Pro-Motion* liegt hierbei jedoch der Schwerpunkt auf der Unterstützung von Verbindungslosigkeit. Mobilität der Transaktion soll jedoch durch Erweiterungen auf der Ebene des im Pro-Motion Modell existierenden Mobilitätsmanagers erreicht werden können.

Bei *Toggle* liegt die Unterstützung im Bereich der Mobilität. Die Bearbeitung von Daten während einer Verbindungslosigkeit wird hier nicht unterstützt. Vielmehr werden Verbindungsabbrüche als ein weiterer Transaktionszustand eingeführt. Im Fall eines vorauszusehenden, bzw. herbeigeführten Verbindungsabbruches wird die Transaktion in den Status *disconnected* überführt. Bei späterer Wiederverbindung ist es damit erlaubt die Transaktion weiterzuführen. Bei Verbindungsverlust durch Fehler tritt die Transaktion in den

Zustand *suspended*. Jedoch werden Transaktionen in diesem Zustand nicht abgebrochen, solange sie nicht die Ausführung anderer Transaktionen behindern.

Kommerzielle Lösungen Bei den derzeitig erhältlichen mobilen Datenbanken (siehe Kapitel 2.4) existiert derzeit keine Unterstützung für die Mobilität während laufender Transaktionen. Alle Modelle bieten Unterstützung zur Bearbeitung von Daten im unverbundenen Zustand über Datenreplikationsverfahren. Hierbei werden Transaktionen im unverbundenen Zustand auf den lokalen replizierten Daten vollzogen. Beim Wiederverbinden mit dem Netzwerk erfolgt eine Synchronisation mit dem Server-Datenbanksystem, um einen eindeutigen, konsistenten Datenbankzustand zwischen der mobilen Datenbank und dem Server-Datenbanksystem herzustellen.

Die Synchronisation zwischen mobiler und stationärer Datenbank kann dabei nach zwei prinzipiellen Verfahren erfolgen. Bei dem *Session-based* Synchronisationsverfahren erfolgt die Synchronisation durch eine direkte Verbindung zwischen mobiler und stationärer Datenbank. Nur wenn die Synchronisation komplett ist, kann die mobile Datenbank wieder die Verbindung unterbrechen. Nach Verbindungsunterbrechung sind beide Datenbanken wieder in einem konsistenten Zustand. Das *Message-based* Synchronisationsverfahren hingegen basiert auf einem Messaging-System, welches Informationen zum Abgleich überträgt. Die Synchronisation erfolgt hier während die mobile und stationäre Datenbank unverbunden sind.

Eine vertiefte Beschreibung der in mobilen Datenbanken zu findenden Replikations- und Synchronisationsverfahren findet sich in [128, 82, 83, 61].

2.2.4.2 Replikationsverfahren

Der generelle Ablauf in Replikationsverfahren kann aus der Sicht des Nutzers in die folgenden drei Abschnitte unterteilt werden: Zuerst erfolgt die Übertragung der zu bearbeitenden Daten im verbundenen Modus auf die mobile Datenbank (Replikation). Idealerweise erfolgt dieser Abschnitt nicht über drahtlose Verbindungen, sondern über stabile Netzverbindungen. Im anschließenden Schritt findet die lokale Bearbeitung der Daten auf dem mobilen Gerät im unverbundenen Zustand durch lokale Transaktionen auf der mobilen Datenbank statt. Nach Abschluß der Bearbeitung erfolgt im abschließenden Schritt die Rückübertragung der vollzogenen Änderungen in den Gesamtdatenbestand (Reintegration) mit eventueller Konfliktauflösung.

Bei der Replikation sind zwei Möglichkeiten zu unterscheiden. Zum ersten kann die Replikation als Übertragung einer Teilmenge der Datenbasis des Server-DBMS auf die mobile Datenbank durch Aufbau einer neuen Datenbank verstanden werden. Andererseits kann, bei bereits bestehender Datenbank auf dem mobilen Gerät, die Replikation auch

die Propagierung von Änderungen des Server-DBMS bedeuten. Aus Sicht des Nutzers kann weiterhin zwischen einer *server-orientierten* und *client-orientierten* Replikation unterschieden werden. Bei der server-orientierten Replikation werden nur bestimmte auf dem Server vordefinierte Dateneinheiten zur Replikation angeboten. In der client-orientierten Replikation geht dagegen die Auswahl der zu replizierenden Daten vom Nutzer der mobilen Datenbank aus. Vorteile der server-orientierten Replikation liegen in der Möglichkeit der Definition disjunkter Dateneinheiten, auf denen bei späterer Reintegration wenig Konflikte zu erwarten sind. Beispiel hierfür ist die Definition von Dateneinheiten für die bereits angesprochenen nicht-kooperativen Anwendungen. Die client-orientierte Replikation bietet dagegen einen flexibleren Ansatz, bei dem auch im voraus nicht benötigte Dateneinheiten bei Bedarf repliziert werden können. In derzeitigen kommerziellen Lösungen für mobile Datenbanken findet sich der server-orientierte Replikationsansatz. Ein Ansatz für client-orientierte Replikation beschreibt z.B. Gollmick in [45].

Aus der Bearbeitungssicht des Nutzers kann weiterhin eine *read-only* und *read-write* Replikation unterschieden werden. Die read-only Replikation erlaubt nur Lesetransaktionen, wodurch der Reintegrationsschritt von Änderungen nur einseitig für die Änderungen im Server-Datenbanksystem erfolgen muß. Bei der read-write Replikation dagegen können beim Reintegrationsschritt mit dem Server-Datenbanksystem Konflikte entstehen, die aufgelöst werden müssen.

Die Reintegration von Änderungen ist Teil der Synchronisation zwischen mobiler Datenbank und Server-Datenbanksystem. Hierbei erfolgt neben der Einbringung der lokalen Änderungen auch die Propagierung von Updates des Server-Datenbanksystems. Für die Reintegration können zwei Ansätze (Wiedereinbringungsstrategien) unterschieden werden. Bei der *datenorientierten* Wiedereinbringung werden nur die Änderungen an den Datenelementen betrachtet. Basierend auf dem Wert eines Datenelements nach der letzten Synchronisation und dem derzeitigen Wert, sowie dem aktuellen Wert des Servers, wird die Konflikterkennung und -auflösung durchgeführt. Die *transaktionsorientierte* Wiedereinbringung baut auf der Speicherung aller im unverbundenen Zustand durchgeführten Transaktionsoperationen auf der mobilen Datenbank auf. Die Einbringung erfolgt hier durch eine Ausführung der aufgezeichneten Transaktionsoperationen auf dem Server-Datenbanksystem. Treten bei einer Transaktion Konflikte auf, muß diese Transaktion rückgängig gemacht werden. Alle abhängigen Transaktionen, d.h. Transaktionen die auf Ergebnissen dieser Transaktion aufbauen, dürfen dann ebenfalls nicht eingebracht werden.

2.2.5 Relevanz für die Arbeit

Geowissenschaftliche Anwendungen können im Allgemeinen der kooperativen Anwendungsklasse zugeordnet werden. Die Modellierung von 3D-Modellen erfolgt in gemeinsamer Projektarbeit, bei der mehrere Personen auf ein gemeinsames 3D-Modell Zugriff haben. Ei-

ne Trennung in räumliche Arbeitsbereiche oder sonstige Abgrenzungen der Bearbeitung (nicht-kooperative Anwendungsklasse) ist in bestimmten Fällen möglich, jedoch nicht als der Normalfall anzusehen. Daher wird innerhalb der vorliegenden Arbeit der Verbindungszustand bzw. Trennungsmodus *optimistic-checkout* untersucht. Hierbei soll die Möglichkeit des Lese- und Schreibzugriffes gegeben werden.

Aus Sicht der Architektur ist vor allem die Extended Client-Server Architektur von Bedeutung. Eine Architektur wie bei verteilten Datenbanksystemen, bei der eine mobile Datenbank (bei Verbundenheit) auch Daten bereitstellen kann, wird wegen der Verbindungsproblematiken und der Kurzlebigkeit der Verbindungen nicht in Betracht gezogen. Peer-To-Peer Architekturen ohne zentrale Server-Datenbanksystem-Instanzen zur Koordination scheiden bei dem derzeitigen Status mobiler Geräte und Verbindungsbandbreiten aufgrund der in geowissenschaftlichen 3D Anwendungen sehr großen Datenmengen aus. Es kann bei allen Anwendungsbereichen in den Geowissenschaften eher davon ausgegangen werden, daß die Datenbestände auf festen Rechnerknoten gehalten werden. Eine Erweiterung hin zu einem hybriden Client-Server-Peer Ansatz, in dem mobile Geräte nicht nur als Clients zu einem Server-Datenbanksystem auftreten, sondern auch im Feld ein Ad-hoc Netzwerk zwischen den mobilen Datenbanken bilden können, ist als zukunftsweisend einzustufen. Hiermit könnten Kooperationen zwischen den Clients im Feld ermöglicht werden. Dies könnte ohne eine zentrale Vermittlungsinstanz über Netzwerke für den Nahbereich (z.B. W-LAN, Bluetooth) realisiert werden. Ebenfalls möglich wird dadurch die Benutzung von freien Resourcenkapazitäten auf evtl. stärkeren Client-Geräten im Peer-Netzwerk. Hier sind allerdings noch viele Fragen aus dem Bereich der P2P-Datenbanksysteme zu erforschen. Obwohl diese Ansätze als überaus interessant anzusehen sind, liegen sie jedoch außerhalb der Verwirklichbarkeit innerhalb dieser Arbeit.

Bei den Modellen zur Transaktionsunterstützung ist zwischen Unterstützung für Transaktionsübergabe bei Zellenwechsel und der Unterstützung für unverbundenes Arbeiten zu unterscheiden. Der Aspekt der Transaktionsübergabe wird in der vorliegenden Arbeit nicht weiter betrachtet. Es wird davon ausgegangen, daß die mobilen Geräte bei einer Netzverbindung an einem Standort verbleiben und nicht wie bspw. bei Autoinformationssystemen schnelle Bewegungen mit wahrscheinlichen Zellenwechseln stattfinden. Für die Unterstützung unverbundenen Arbeitens wird bereits in mobilen Datenbanken für Standard-Anwendungen das Replikationsverfahren eingesetzt. Für die in der vorliegenden Arbeit behandelten geowissenschaftlichen Anwendungen erscheint eine Kombination aus diesem Verfahren und dem Einsatz von Versionsmodellen – die im folgenden beschrieben werden – am besten geeignet. Das Replikationsverfahren ersetzt bei Wiedereinbringung die Datenbankobjekte des Server-Datenbanksystems mit den geänderten Objekten der mobilen Datenbank, solange keine Konflikte auftreten. Da wir in geowissenschaftlichen Anwendungen eher von einem hohen Konfliktpotential ausgehen, erscheint ein kombinierter Ansatz mit Versionsmodellen, mit Bildung einer neuen Version bei der Wiedereinbringung,

vielversprechender. Eine evtl. Konfliktauflösung zwischen konkurrierenden Versionen erfolgt hier zu einem späteren Zeitpunkt durch die Nutzer.

2.3 Versionsmodelle

Geowissenschaftliche Modelle, z.B. in der Geologie, bestehen in der Regel nicht aus einer sehr hohen Anzahl von einfachen Objekten, sondern aus einer begrenzten Zahl vieler hochkomplexer Objekte. Da diese Objekte nicht durch eine Observierung der Strukturen der Erdoberfläche wie bei 2D Daten entstehen (vgl. Kapitel 1.3), sind sie einem fortlaufenden iterativen Modellierungsprozess unterworfen. Diese häufige Bearbeitung von Objekten über einen langen Zeitraum erhöht das Konfliktpotential beim reinen Einsatz von Replikationsverfahren in geowissenschaftlichen, mobilen Datenbanken.

Anderere Bereiche, in denen ebenfalls eine fortlaufende Modellierung bzw. Änderung von Objekten unterstützt werden muß, sind z.B. Computer-Aided-Design (CAD), Software-Engineering und temporale/historische Datenbanken. Vornehmlich aus den Bereichen des CADs und des Software-Engineerings stammt das Konzept der Versionsmodellierung zur Unterstützung der fortlaufenden Modellierung von Objekten. Innerhalb dieser Bereiche dient das Versionskonzept zur Unterstützung des Designprozesses, indem komplexe Objekte aus verschiedenen Versionen einfacherer Objekte zusammengesetzt und somit unterschiedliche Konfigurationen getestet werden können. Neben der Design-Unterstützung können Versionen von Objekten jedoch auch zur Unterstützung anderer Problemfelder herangezogen werden.

Im folgenden wird zuerst auf die Versionsmodelle aus dem CAD-Bereich eingegangen, die den Grundstock der Forschung in dieser Richtung legten. Anhand dieser Versionsmodelle werden wichtige Begriffe zum Verständnis der vorliegenden Arbeit vorgestellt und erläutert. Anschließend wird auf die Unterstützung von Versionsmodellen in räumlichen Datenbanksystemen eingegangen.

2.3.1 Übersicht

Oft ist es nützlich in einer Datenbank Informationen über verschiedene Aspekte eines zu modellierenden Objektes zu speichern. Diese Informationen können als Versionen des Objektes gespeichert werden. Es können dabei zwei Versionskategorien unterschieden werden: Versionen auf der *System-Ebene* und der *Benutzer-Ebene*.

Die Versionen auf der *System-Ebene* werden alleinig durch das Datenbanksystem intern erzeugt und genutzt. Diese Versionen werden beispielsweise zur Verbesserung der Synchronisationsmechanismen in Transaktionen (Multiversion Concurrency Control) [132] oder der

Redundanz in verteilten Datenbanksystemen [137] benutzt. Versionen auf der *Benutzer-Ebene* hingegen werden von den Anwendungen für spezifische Nutzungen erzeugt und verwendet. Beispiele dieser Nutzung von Versionen finden sich in den Anwendungsbereichen des CAD/CAM (Computer-Aided-Design/Manufacturing), des VLSI-Designs (Very Large Scale Integration), sowie der Geographischen Informationssysteme und temporaler bzw. historischer Datenbanken.

Die Ursprünge der Versionsmodelle liegen im CAD/CAM-Bereich. Die Vorschläge zu Versionsmodellen aus diesem Bereich reichen bis in die frühen 80er Jahre zurück [31, 66, 65, 73, 105]. Mit dem Aufkommen objekt-orientierter Datenbanksysteme in den 90er Jahren beschäftigten sich viele Veröffentlichungen mit Versionsmodellen in objekt-orientierten Datenmodellen bzw. deren Integration in objekt-orientierte Datenbanksysteme [67, 30, 3, 116, 39]. Ein weiterer Forschungsaspekt in Versionsmodellen stellen die temporalen bzw. historischen Datenbanken dar [12, 13]. An aktuellen Arbeiten auf diesem Gebiet findet sich die Dissertation von Schönhoff 2002 [115], die sich mit Versionsmanagement in Föderierten Datenbanksystemen beschäftigt.

In [65] stellt Katz eine ausführliche Beschreibung seines über die Jahre entwickelten Versionsmodells vor. Neben der Beschreibung untersucht er in dieser Arbeit alle bis 1990 gemachten Vorschläge zu Versionsmodellen und stellt viele Gemeinsamkeiten der Modelle in ihren generellen Konzepten fest. Unterschiede liegen dabei meist in der verschieden angewendeten Terminologie der Begriffe in den Modellen. Die folgenden Ausführungen zum Verständnis von Versionsmodellen basieren teilweise auf den Darstellungen dieser Veröffentlichung.

Im Bereich der Entwurfs-/Modellierungsanwendungen erfolgt nach der Erzeugung eines initialen Entwurf-Objektes ein kontinuierlicher Prozeß der Verbesserung (Revision) des Objektes, den man als Evolution des Entwurf-Objektes umschreiben kann. Im Bereich der geologischen Modellierung bedeutet dies die Verbesserung des Modells durch Hinzuziehung neuerer Erkenntnisse in der Interpretation des Modells oder neuer Meßdaten. Hierbei soll die Veränderung dieses Modells nicht durch eine einfache vollständige Ersetzung des Modells erfolgen, sondern die Schritte der Veränderungen sollen erhalten bleiben und zu einem späteren Zeitpunkt nachvollzogen oder auch wieder zurückgenommen werden können. Zur Unterstützung dieser verschiedenen Stadien der Objektentwicklung werden *Versionen* dieser Objekte gebildet.

2.3.2 Begriffe und Definitionen

In Versionsmodellen wird zwischen dem generischen Teil eines versionierbaren Objektes, dem *DesignObjekt* und seinen *Versionen* unterschieden. Das DesignObjekt stellt hierbei die semantische Bedeutung des versionierbaren Objektes dar und kann als die Vereinigung

all seiner Versionen angesehen werden. Versionen stellen eine bestimmte Ausprägung eines konkreten DesignObjektes zu einem bestimmten Zeitpunkt der Erzeugung bzw. der Modellierung dar.

Versionen und Versionshistorie Versionen können damit Nachfolger existierender Versionen sein, bzw. als Nachfolger für weitere Versionen des DesignObjektes dienen. Lediglich die erste Version eines DesignObjektes besitzt keinen direkten Vorgänger. Es sind prinzipiell folgende Beziehungen zwischen Versionen möglich:

- **initial:** Erste (initiale) Version in der Versionshistorie.
- **revision:** Version stellt einen weiterentwickelten Zustand der Vorgängerversion dar.
- **alternative:** Version stellt einen alternativen Zustand der Vorgängerversion dar. Beide Versionen entwickeln sich ab diesem Zeitpunkt unabhängig voneinander.
- **merge:** Version stellt eine Vereinigung ihrer beiden Vorgängerversionen aus zwei unterschiedlichen Revisionslinien dar.

Die gesamte Menge von Versionen eines DesignObjektes mit ihren Beziehungen untereinander wird *Versionshistorie* genannt. In Abbildung 2.12 ist ein Beispiel für eine vollständige Versionshistorie mit allen existierenden Beziehungen zwischen Versionen dargestellt.

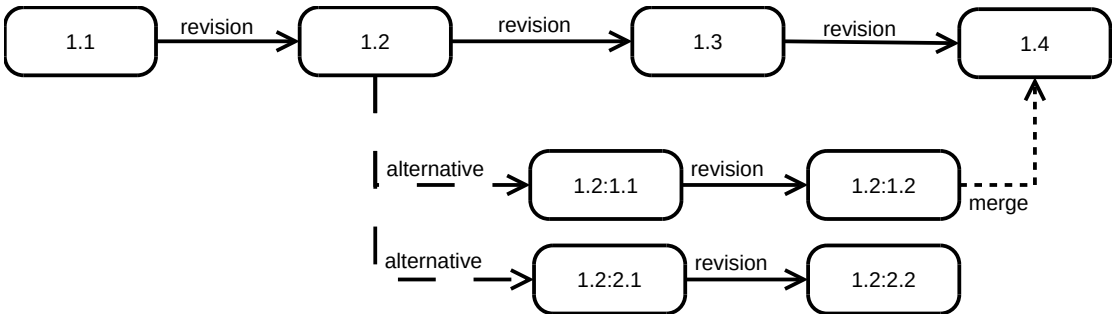

Abbildung 2.12: Beispiel für Versionshistorie

Diese vollständige Versionshistorie entspricht einem Directed Acyclic Graph (DAG). Neben dieser vollständigen Versionshistorie existieren Sonderfälle. Wird in der Versionshistorie beispielsweise nur die Revisions-Beziehung zugelassen, so kann die Versionshistorie als eine lineare Liste der Versionen dargestellt werden.

Workspaces Zusammengehörige DesignObjekte (z.B. alle DesignObjekte eines Untergrundmodelles) werden in Repositories, meist als *Workspaces* bezeichnet, mit ihren Versionen verwaltet. Workspaces sind meist hierarchisch organisiert und Versionen können

zwischen Workspaces durch bestimmte Operationen (CheckIn/CheckOut) migriert werden. Workspaces dienen ebenfalls der Zugriffsregelung auf die enthaltenen Versionen. Je nach Versionsmodell existieren beispielsweise Archiv-, Gruppen- und Private-Workspaces.

Einfache und komplexe DesignObjekte DesignObjekte können in *einfache* und *komplexe* DesignObjekte unterteilt werden. Ein DesignObjekt wird als einfaches DesignObjekt bezeichnet, wenn es nur aus nicht versionierbaren Attributen aufgebaut ist. Komplexe DesignObjekte dagegen stellen eine Aggregation über mehrere DesignObjekte dar, die damit zu Teil-DesignObjekten des komplexen DesignObjektes werden.

In Abbildung 2.13 ist beispielhaft ein komplexes DesignObjekt, aufgebaut aus drei einfachen DesignObjekten (A, B und C), dargestellt.

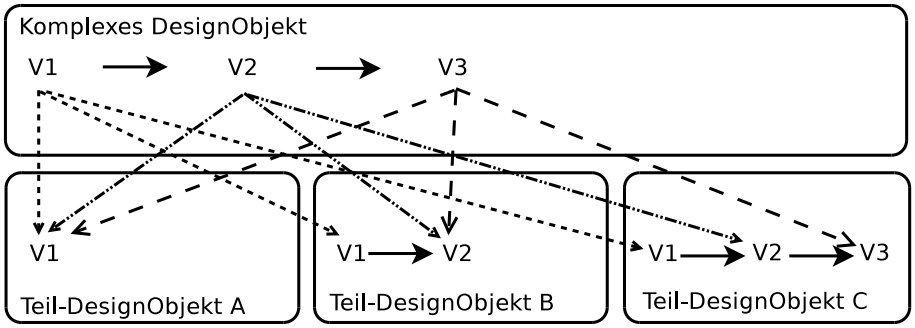

Abbildung 2.13: Beispiel für ein komplexes DesignObjekt

In der Abbildung wird deutlich, daß jedes komplexe DesignObjekt grundsätzlich verschiedene Möglichkeiten der Zusammensetzung durch die einzelnen Versionen der Teil-DesignObjekte besitzt. So können sich Versionen des komplexen DesignObjektes beispielsweise aus den Versionen V1 der einzelnen Teil-DesignObjekte zusammensetzen (Version V1 des komplexen DesignObjektes in der Abbildung), aber auch z.B. aus der Version V1 des Teil-DesignObjektes A sowie den Versionen V2 der Teil-DesignObjekte B und C (vgl. Version V2).

Konfiguration Diese Zusammensetzung von Versionen komplexer DesignObjekte aus bestimmten Versionen seiner Teil-DesignObjekte wird *Konfiguration* genannt. Aufgrund der vielen Möglichkeiten komplexe DesignObjekte zu konfigurieren, sind viele Probleme mit der Konfiguration verbunden. Eine ausführliche Diskussion der Konfigurationsproblematik findet sich in [72]. Die Konfiguration kann also definiert werden als die Bindung zwischen einer Version eines komplexen DesignObjektes und den jeweiligen spezifischen Versionen seiner Teil-DesignObjekte. Es kann dabei zwischen *impliziter* und *expliziter* Konfiguration unterschieden werden. Die implizite Konfiguration wird in verschiedenen Versionsmodellen auch unter dem Begriff *Change Propagation*, z.B. in [65], beschrieben.

Bei der *impliziten* Konfiguration erfolgt bei jeder Erzeugung einer neuen Version eines Teil-DesignObjektes implizit die Erzeugung einer neuen Version des komplexen DesignObjektes, um die neue Version des Teil-DesignObjektes zu integrieren. Erfolgt beispielsweise, ausgehend von Abbildung 2.13, die Bildung einer neuen Version V4 im Teil-DesignObjekt C, so wird automatisch eine neue Version V4 im komplexen DesignObjekt erzeugt. Diese neue Version V4 muß dann auf die bisherigen Versionen der Teil-DesignObjekte A und B sowie auf die neue Version V4 des Teil-DesignObjektes C verweisen.

Bei der *expliziten* Konfiguration erfolgt die Zuordnung der Versionen der Teil-DesignObjekte zu einer Version des komplexen DesignObjektes unabhängig von der Versionserzeugung. Ziel der expliziten Konfiguration nach [72] ist es, eine konsistente Sicht auf eine komplexe Objektstruktur zu realisieren, deren (Teil-)Objekte versioniert sind – und zwar unabhängig von deren Versionserzeugung. Damit überläßt die explizite Konfiguration den Vorgang der Konfiguration einer Version des komplexen DesignObjektes dem Nutzer und ermöglicht somit die Weiterentwicklung von Versionen der Teil-DesignObjekte unabhängig vom komplexen DesignObjekt. Erst zum Konfigurationszeitpunkt muß eine explizite Festlegung der Versionen der Teil-DesignObjekte zu den Versionen des komplexen DesignObjektes erfolgen. Explizite Konfigurationen können auch zur Zusammenfassung von Versionen nicht zusammenhängender DesignObjekte verwendet werden (übergreifende Konfiguration).

2.3.3 Unterstützung in räumlichen Datenbanksystemen

Im Bereich der Geographischen Informationssysteme und der räumlichen Datenbanksysteme können Versionsmodelle neben der Unterstützung von Entwurfs-/Modellierungsprozessen auch zur Lösung des Problems langer Transaktionen eingesetzt werden. Während jedoch im Design-Bereich die Verwendung von Versionsmanagement bis in die frühen 80er Jahre zurückreicht, kam die Verwendung von Versionsmanagement für räumliche Datenbanksysteme und GIS vorwiegend erst Anfang der 90er Jahre auf [34, 86, 88, 80, 87, 29, 81, 79]. Arbeiten der letzten Jahre zur Nutzung von Versionen in GIS sind beispielsweise [98, 7]. Eine Übersicht über die Unterstützung von Versionsmanagement in Datenbanksystemen ist im Kapitel 2.4 zu finden. Allerdings werden im Datenbankbereich oft sehr unterschiedliche Begriffe für Versionsmanagement benutzt. So wird im GIS Smallworld bspw. von Alternativenmanagement [34] und im Datenbanksystem Oracle von Workspace-Management [96] gesprochen.

2.4 Verwandte Forschungsgebiete und -arbeiten

Die vorliegende Arbeit ist in der Schnittmenge mehrerer Forschungsgebiete anzusiedeln. Sie versucht, unter dem Aspekt geowissenschaftlicher Anwendungen, die Ergebnisse und

Erfahrungen mehrerer Arbeitsgebiete zu nutzen, um eine Problemlösung für die mobile Datenverwaltung drei dimensionaler geowissenschaftlicher Daten zu erreichen.

Hierbei sind vor allem die Arbeitsgebiete Datenbanksysteme, räumliche Datenbanksysteme, mobile Datenbanken und Versionsmodelle relevant. Die Arbeitsgebiete müssen dabei jeweils unter dem Aspekt der dritten Raumdimension gesehen werden. Im Bereich von kooperativen Anwendungen kann auch der Forschungsbereich Computer Supported Cooperative Work (CSCW) [114] genannt werden. Dieser Bereich beschäftigt sich zwar primär nicht mit räumlichen Daten, hat jedoch die Zusammenarbeit von mehreren Personen an einem Objekt zum zentralen Gegenstand.

3D Datenbanksysteme Während räumliche 2D Datenbanksysteme mittlerweile von allen wichtigen Datenbank Herstellern als Erweiterungen angeboten werden, existieren im 3D-Umfeld bisher nur Forschungsprototypen aus dem universitären Bereich.

Mit *GeoStore* und *GeoToolKit* [16, 20, 5] wurden am Institut für Informatik III der Universität Bonn in Zusammenarbeit mit dem Geologischen Institut Prototypen eines objekt-orientierten 3D/4D Geodatenbanksystems entwickelt. GeoStore ist dabei ein eng an den Anwendungsbereich der Geologie angelehntes 3D-Geoinformationssystem zur Verwaltung von räumlichen und damit verbundenen thematischen Informationen geologischer 3D-Modelle des Untergrundes. Die Speicherung der Daten erfolgt dabei durch Erweiterung des objekt-orientierten Datenbanksystems ObjectStore© mit speziellen räumlichen 3D Geometrieobjekten. Aus der zunächst speziell auf die Modellierung geologischer Objekte abgestimmten Entwicklung ging in der Folge das GeoToolKit hervor. Während GeoStore speziell auf die Geologie zugeschnittene 3D-Geoobjekte bereitstellte, abstrahierte man bei GeoToolKit die nötige Funktionalität für ein 3D-Datenbanksystem in abstrakte Geometrie- und Verwaltungsklassen und stellte somit eine anwendungsunabhängige Funktionalität bereit. Damit ist das GeoToolKit als ein GIS-Baukasten für zukünftige Geoinformationssysteme als Geodatenbankkern zu verstehen, auf dem verschiedene, unterschiedlich spezialisierte 3D Geodatenbanksysteme aufgesetzt werden können. GeoToolKit ist als eine C++ Klassenbibliothek realisiert. Das zugrundeliegende Datenmodell basiert auf dem Modell Simplizialer Komplexe und stellt damit erweiterbare Klassen für 0D bis 3D Geometrieobjekte im 3D bereit. Eine Erweiterung um zeitabhängige Geometrieobjekte wird in [121] und [5] beschrieben. Weiterhin existieren Klassen zur logischen Verwaltung und zur Indexierung der Daten. Die Persistenz der Klassenbibliothek wird auf Basis der objekt-orientierten Datenbanksoftware ObjectStore© erreicht.

An der Universität Münster wurde Mitte der 90er Jahre das räumliche Datenbanksystem *GOODAC* entwickelt. GOODAC stellt die prototypische Implementierung des *Object-Oriented Geographic Data Model* (OOGDM) [130] auf Basis der objekt-orientierten Datenbanksoftware ObjectStore dar. GOODAC bietet über die Implementierung des Datenmodells OOGDM die Unterstützung von 2D und 3D-Objekten sowohl im Vektor- als auch

im Rastermodell an. Der Bereich der 3D Unterstützung basiert dabei auf einem Vektor-Randmodell zur Modellierung von Polyedern durch eine Menge von 3D Polygonen, die die Begrenzungsflächen (faces) der Körper darstellen. In [131] wird eine zeitliche Erweiterung des Datenmodells (T/OOGDM) beschrieben. Die Erweiterung führt die gültige Zeit (valid time) für Datenbankobjekte ein. Damit ist es möglich, Datenbankobjekten bzw. den einzelnen Attributen (sowohl Geometrie- als auch Sachdatenattributen) Zeitstempel oder Zeitintervalle ihrer Gültigkeit zuzuordnen. Die Anfragesprache wurde erweitert, um Eingrenzungen in der Gültigkeit der Zeit zuzulassen.

Ebenfalls Mitte bis Ende der 90er Jahre wurde an der TU Clausthal das *Baugeologisch-Geotechnische Informationssystem BAGIS* [69] entwickelt. BAGIS stellt ein Anwendungssystem dar, das sich aus einem Datenbanksystem zur Verwaltung der Datenbasis und einer externen Anwendung zur Modellierung und Visualisierung auf Basis des 3D CAD-Systems Microstation zusammensetzt. Grundlage der Datenverwaltung ist ein, ebenfalls auf Basis von ObjectStore entwickeltes, 3D Datenbanksystem. Das Datenmodell teilt sich dabei in zwei Datenmodelle auf. Das grundlegende Datenmodell *Clausthal Object-Relational Database Modell* (CORE) dient der Beschreibung von thematischen Daten. Darauf aufbauend entstand das Geodatenmodell GeoCORE [68] zur Modellierung von Geoobjekten. Das Datenmodell ist über eine Datendefinitionssprache DDL festgelegt, welche unter Verwendung fester Abbildungsvorschriften über einen Präcompiler auf das zugrundeliegende Datenbanksystem ObjectStore abgebildet wird. GeoCORE ist in drei Ebenen unterteilt. Die Thematische Ebene basiert auf CORE, während eine topologische und metrische Ebene von GeoCORE hinzugefügt wird. Die topologische Ebene ermöglicht die explizite Speicherung topologischer Beziehungen, wie z.B. Nachbarschaft oder Umschließung, zwischen Objekten. Die metrische Ebene basiert auf dem Vektor-Randmodell zur Darstellung von GeoObjekten im 3D. Im Gegensatz zu GOODAC werden 3D-Körper bzw. ihre Begrenzungsflächen jedoch nicht über Polygone definiert, sondern basieren auf Flächenrepräsentation in B-Spline Techniken (parametrisierte kubische Bögen und bikubische Flächen).

Als letztes Datenbanksystem sei hier noch kurz das in Kapitel 4 zu beschreibende Datenbanksystem *DB3D* angesprochen. DB3D basiert wie GeoToolKit auf dem Datenmodell der Simplizialen Komplexe. Es baut auf den Erfahrungen mit GeoToolKit auf, ist jedoch von Anfang an auf einen Internet-basierten Zugriff ausgelegt. Die gesamte Interaktion mit dem Datenbanksystem wird in DB3D durch eine Menge von Diensten ermöglicht, die ebenfalls von mobilen Systemen verwendet werden können.

Mobile Datenbanken In [82] und [83] vergleichen Mutschler und Specht derzeitige kommerzielle, mobile Datenbanksysteme. Mobile Datenbanksysteme existieren für XML-Datenbanksysteme beispielsweise mit Tamino Mobile und für relationale Datenbanksysteme – Vertreter sind hier z.B. DB2 Everyplace, Oracle 9i Lite und Sybase UltraLite. Wie bereits erwähnt bieten alle Systeme Unterstützung zum unverbundenen Arbeiten auf

replizierten Daten. Bei den mobilen, relationalen Datenbanksystemen wird nicht der gleiche Funktionsumfang wie bei den zugehörigen Server-Datenbanksystemen unterstützt. Die Möglichkeit der Definition eigener Datentypen und damit verbundener Operationen fehlt größtenteils vollständig. Allgemein ist festzustellen, daß die derzeitigen mobilen Datenbanken nur für die sogenannten Standard-Anwendungen geeignet sind. Es wird weder eine Erweiterungsmöglichkeit hinsichtlich der Datentypen, Indexe und Operationen geboten, noch eine fertige Unterstützung bzw. Erweiterungen für räumliche Daten.

Die Kommunikation bei der Synchronisation erfolgt bei diesen Produkten meist über die *Synchronization Markup Language* SyncML [95]. SyncML ist eine XML-Spezifikation zur herstellerunabhängigen Synchronisation von Daten in einem mobilen Client/Server-Szenario. Ziel der Synchronisation ist dabei die Herstellung eines konsistenten Zustandes zwischen mobiler DB und Server-DBMS. Die Synchronisation läuft dabei meist in zwei Phasen ab. Eine Reintegrationsphase überträgt Datenänderungen von der mobilen DB an das Server-DBMS und anschließend erfolgt eine Rückübertragungsphase, bei der Änderungen auf dem Server-DBMS an die mobile DB übertragen werden. Die Konfliktbehandlung erfolgt nach relativ einfachen Strategien. Beispiele hierfür sind die *Kapitulation*, bei der bei Konflikten die Synchronisation abgebrochen wird oder die *Gewinner*-Strategie, bei der von Anfang an festgelegt ist, welcher Synchronisationspartner bei Konflikten gewinnt.

Mit Produkten wie Versant FastObjects[4] oder PSEPro für Java[5] existieren vollwertige auf mobilen Geräten lauffähige objekt-orientierte Datenbanken. Alle diese Produkte sind jedoch als eigenständige Datenbanksysteme anzusehen und bieten keine Funktionalität für den Einsatz als mobile Datenbank im Zusammenspiel mit ihren jeweiligen Pendants an Server-Datenbanksystemen. Die jeweilige Middleware zum Bereitstellen der Replikations- bzw. Synchronisationsfunktionalität muß hier also selbst entwickelt werden. Vorteil objektorientierter Datenbanksysteme ist die Unterstützung von beliebig komplexen Datenmodellen auch auf mobilen Geräten.

Datenbanken mit Versionsunterstützung Mittlerweile sind verschiedene Datenbanksysteme mit Versionsunterstützung auf dem Markt. In vielen 2D Geodatenbanksystemen wird Versionsunterstützung für lange Transaktionen bei Updates eingesetzt. Weitere Anwendung ist die Unterstützung der Entstehungshistorie von Datenbeständen, sowie die Unterstützung stufenweiser Modellierung von Daten. Bei den räumlichen Datenbanksystemen ist hier vor allem das dem GIS Smallworld zugrundeliegende Datenbanksystem [86] zu nennen. Es kann als Pionier der Versionsunterstützung im GIS-Bereich angesehen werden. Weiterhin bieten die Datenbank-Middleware ArcSDE [124] und auch Oracle mit seinem Produkt Workspace-Manager [2] Versionsunterstützung an. Neben diesen objekt-relationalen Datenbanksystemen unterstützen bzw. unterstützten auch mehrere objekt-

[4] URL: www.versant.com/en_US/products/fastobjects
[5] URL: www.progress.com/realtime/products/

orientierte Datenbanksysteme Versionsmodelle. Bis zur Version 4 bspw. beinhaltete das OODBMS ObjectStore ein Versionsmodell. Weiterhin sind viele Modelle zur Versionsunterstützung in objekt-orientierten Forschungsdatenbanken beschrieben [3]. Microsoft bietet mit dem Microsoft Repository [9] einen objekt-orientierten Aufsatz mit Versionsmanagement für ihr Produkt SQL-Server an.

Kapitel 3

Geodatenverwaltung in mobilen Datenbanken

Das vorangegangene Grundlagenkapitel hat den derzeitigen Stand im Bereich der räumlichen Datenbanksysteme und mobilen Datenbanken, sowie die Anforderungen von Seiten geowissenschaftlicher Anwendungsfelder beschrieben. Dieses Kapitel geht nun auf die daraus resultierende Anforderung an die Verwaltung von Geodaten mit mobilen Datenbanken sowie auf existierende Problemfelder ein und zeigt einen Lösungsansatz auf.

3.1 Anforderungen

Wie in der Einleitung in Kapitel 1 beschrieben, bringen die großen Fortschritte im Bereich mobiler Geräte und drahtloser Kommunikation viele neue Möglichkeiten für die Geowissenschaften mit sich. Auf Energieverbrauch optimierte Prozessoren, integrierte drahtlose Netzwerkadapter und eine radikal verringerte Größe und Gewicht heutiger mobiler Geräte machen die Vision vom Arbeiten des Geowissenschaftlers im Gelände, unabhängig von weiteren Energiequellen, fast erreichbar.

Auf der Seite der Software werden die Spezifikation von Geo-Diensten für den mobilen Datenzugriff immer ausgereifter und sind weitreichend etabliert. Die Standards des Open Geospatial Consortium (OGC) ermöglichen den Zugriff auf Geodaten in einer interoperablen Art und Weise – auch von mobilen Geräten aus. Prinzipiell scheint es, daß die Infrastruktur vorbereitet ist, um mobile Applikationen zu entwickeln, die vollständig in die heutigen Geoinformationsstrukturen integriert sind. Neben dem reinen Zugriff auf Geodaten benötigen mobile Applikationen jedoch auch Verarbeitungs- und Updatemöglichkeiten. Hier werfen vor allem die Updatemöglichkeiten von Geodaten mehrere Problemfelder auf. Bei der Betrachtung mobiler Applikationen müssen wir unterscheiden zwischen online und

offline Updates auf den Geodaten. Der erste Fall des Online-Updates ist heutzutage - eine gute Netzverbindung vorausgesetzt - bereits mit den OGC Standards möglich, wobei diese größtenteils noch auf den zweidimensionalen Bereich beschränkt sind. Der zweite Bereich der offline Updates im unverbundenen Zustand beinhaltet jedoch immer noch Probleme. In der vorliegenden Arbeit wird dieser Bereich der offline Updates vertiefend behandelt und ein Lösungsansatz vorgestellt.

Eine mobile Applikation wird in der vorliegenden Arbeit als vollständig funktionsfähig betrachtet, wenn sie in beiden Modi – offline und online – arbeiten kann. In einem *online Modus*, in dem Geodaten durch Dienste der Geoinformationsstrukturen zugänglich gemacht werden und in einem *offline Modus*, der die Möglichkeit bietet im unverbundenen Zustand geänderte Geodaten wieder in die Datenbestände der Geoinformationsstrukturen zu integrieren.

Hierfür müssen mobile Applikationen eine effiziente und konsistente Möglichkeit der Speicherung von Geodaten auf mobilen Geräten anbieten. Mobile Datenbanken erscheinen für diese Aufgabe am besten geeignet, da hiermit alle Vorteile von Datenbanksystemen auch auf den mobilen Geräten zur Verfügung gestellt werden. Jedoch löst die reine Benutzung von mobilen Datenbanken auf mobilen Geräten nicht das Problem der offline Updatefähigkeit von Geodaten. Hierfür muß die mobile Datenbank mit den Server-Geodatenbanksystemen der Geoinformationsstrukturen integriert sein. Wichtige Funktionalitäten sind hier beispielsweise eine Synchronisationsmöglichkeit mit den aktuellsten Datenbeständen der Server-Geodatenbanksysteme. Weiterhin müssen die geänderten Geodaten der mobilen Datenbank rückintegrierbar in die Server-Geodatenbanksysteme sein. Dies bedeutet, daß Integrationsfähigkeiten in Verbindung mit Konfliktverarbeitung existieren müssen, da in der Zeit der offline Bearbeitung Daten auch von anderen Nutzern bereits geändert worden sein könnten.

3.2 Probleme

Das in der vorliegenden Arbeit betrachtete Problem mobiler Anwendungsszenarien ist die Unterstützung des offline oder unverbundenen Modus für die Bearbeitung von Geodaten auf mobilen Geräten. Hintergrund der Notwendigkeit dieser Unterstützung liegen im technischen und ökonomischen Bereich der drahtlosen Netzverbindungen, die jedoch nicht spezifisch für das Anwendungsfeld mobiler Datenbanken sind. Neben den technischen Problemen der Qualität der Verbindung bezogen auf Stabilität, Bandbreite und der ungenügenden Netzabdeckung außerhalb städtischer Gebiete müssen jedoch weitere Problemfelder betrachtet werden. Derzeitige drahtlose Übertragungstechnologien haben noch einen nicht zu unterschätzenden Energieverbrauch, so daß die Nutzer nicht ganztägig im online Modus arbeiten können. Weiterhin zwingen die hohen Kosten, beispielsweise für

UMTS-Verbindungen (ökonomische Komponente), ebenfalls dem Nutzer einen offline Modus für die Bearbeitung von Geodaten auf mobilen Geräten bereitzustellen. In nächster Zeit sind keine grundlegenden Veränderungen bei diesen Problemen zu erwarten und damit wird die Benutzung des offline Modus der Normalzustand für mobile Applikationen sein.

Unverbundenes Arbeiten Die Unterstützung des offline Modus bedingt – wie bereits ausgeführt – die Bereitstellung von Update-Möglichkeiten auf lokalen Geodaten in mobilen Anwendungen und deren Rückintegrierbarkeit in Server-Geodatenbanksysteme. Wir gehen im folgenden von der Annahme aus, daß auf dem mobilen Gerät eine mobile Datenbank existiert und ebenfalls ein Server-Datenbanksystem zur Verfügung steht. Davon ausgehend stellt sich die Frage, wie eine Reintegration geänderter Geodaten verwirklicht werden kann.

Bei ganztägigen Arbeiten können Bearbeitungsphasen über mehrere Stunden andauern. Gehen wir von einem Ansatz mit Transaktionen (pessimistischer Ansatz) aus, würde die Bearbeitung lokaler Geodaten in einer mobilen Datenbank einer langandauernden Datenbanktransaktion auf dem Server-DBS entsprechen. In einer solchen Transaktion sind die replizierten Datenbestände auf dem Server-Datenbanksystem für andere Transaktionen gesperrt. Dieses Problem langer Transaktionen ist allgemein bekannt und im GIS-Umfeld zum Beispiel von Newell & Batty [87] ausführlich beschrieben. Wird der unverbundene Bearbeitungszustand also als eine normale Transaktion angesehen, müssen alle replizierten Daten auf dem Server-DBS gesperrt werden. Dies führt jedoch im allgemeinen zu einer Degradierung des Server-Datenbanksystems hin zu einem Ein-Benutzer System. Um dies zu vermeiden verfolgen existierende mobile Datenbanken (vgl. Kapitel 2.4) den Ansatz sogenannter optimistischer Verfahren. Hierbei stellt der gesamte Ablauf von Replikation bis zu einer späteren Reintegration/Synchronisation keine Transaktion dar. In der Reintegrationsphase wird versucht, die geänderten Daten entweder Datensatz- oder transaktionsorientiert in das Datenbanksystem auf dem Server zu integrieren. Erfolgte aufgrund der fehlenden Sperre durch das optimistische Verfahren bereits eine Änderung der jeweiligen Daten durch weitere Benutzer, so muß eine Konfliktauflösung stattfinden. Das Problem der derzeitigen mobilen Datenbanken liegt in der Art, wie sie die Konfliktbehandlung durchführen. Im allgemeinen wird davon ausgegangen, daß keine Konflikte entstehen und entstandene Konflikte werden durch relativ einfache Mechanismen aufgelöst (siehe z.B. [44, 45, 61]). Beispiel hierfür ist die einfache Strategie *Server-Updates win*, wodurch grundsätzlich Änderungen von mobilen Applikationen im Konfliktfall nicht durchgeführt werden. Unter bestimmten Vorraussetzungen ebenfalls möglich sind sogenannte konfliktvermeidende Verfahren. Diese Verfahren verhindern jedoch nur sehr spezifische Probleme (z.B. Schlüssel-Konflikte durch Zuweisung von Key-Pools, siehe [61]) oder sind nur auf sehr spezielle Datenarten (z.B. Escrow-Verfahren [92]) anwendbar. Es kann festgestellt werden, daß die zur Verfügung stehenden Konfliktauflösungsstrategien bzw.

-vermeidungsstrategien nicht generell für räumliche Daten, bei denen ein hohes Konfliktpotential der Normalfall ist, nutzbar sind.

Versionsmanagement ist in räumlichen Datenbanksystemen weitgehend als Lösung zum Problem langer Transaktionen akzeptiert [7]. Die meisten Anbieter von Geodatenbanksystemen bieten Lösungen auf dieser Basis an (siehe Kapitel 2.4). Jedoch existieren keine mobilen, räumlichen Datenbanken, die über Versionsmanagement mit einem Server-Datenbanksystem integriert sind. Die Integration von mobilen Datenbanken als Clients zu Server-Datenbanksystemen über Versionsmanagement erscheint ein vielversprechender Ansatz. Vorteil dieser Vorgehensweise ist, daß die Konfliktauflösung nicht durch automatisierte Verfahren auf dem Server-Datenbanksystem stattfindet, sondern durch die Bearbeiter während einer Zusammenführung von verschiedenen Versionen eines Datenbankobjektes. Das Konzept zur Unterstützung von Versionsmanagement in geowissenschaftlichen Datenbanksystemen wird im weiteren Verlauf der Arbeit in Kapitel 5 beschrieben.

Mobile Datenbanken Die Realisierung mobiler Datenbanken muß aufgrund der eingeschränkten Leistungsfähigkeiten mobiler Geräte (vgl. Kapitel 1.2) in verschiedenen Bereichen andere Wege als bei den Server-Datenbanksystemen gehen. Zentrale Bereiche sind hier Fragen der Arbeitsverteilung zwischen Server-Datenbanksystem und mobilen Datenbanken, sowie Fragen der Optimierung von Algorithmen und Datenstrukturen in mobilen Datenbanken. Hierbei lassen sich die Bereiche Optimierung und Arbeitsverteilung nicht voneinander getrennt betrachten, was im folgenden an einem Beispiel ausgeführt wird.

Als Beispiel dient die zeitliche Optimierung des initialen Objektaufbaus für ein komplexes Geoobjekt in der mobilen Datenbank, wenn ein Objekt repliziert wird. Hintergrund des Zusammenwirkens von Optimierung und Arbeitsverteilung ist, daß die Operationen im Server-Datenbanksystem in ihren Algorithmen auf dem internen räumlichen Index oder der aufgebauten internen Nachbarschaftstopologie aufbauen. Die Effizienz des Objektaufbaus hängt zum größtenteil vom Aufbau des internen Indexes und der Topologie ab. Um eine zeitliche Optimierung des Objektaufbaues zu erreichen kann entweder der interne Index weggelassen werden oder optimiert werden. Eine Entfernung des internen Index aus der Objektstruktur in der mobilen Datenbank hätte zur Folge, daß viele Operationen eine deutlich schlechtere Laufzeit haben bzw. gar nicht mehr angewendet werden können. Indexe, die effizienter im Aufbau sind (z.B. Octree), besitzen durch weniger Optimierungen bzw. Re-Strukturierungen des Indexes beim Aufbau im Allgemeinen ein schlechteres Zeitverhalten beim Zugriff. Bei Verwendung eines solchen Indexes kann eine mobile Datenbank alle Operationen benutzen, jedoch nicht mit dem gleichen Laufzeitverhalten wie im Server-DBS. Hier muß eine Abwägung stattfinden, ob spezifische Operationen, die auf bestimmte interne Objektstrukturen angewiesen sind, in der mobilen Datenbank benötigt werden, oder ob solche Operationen nur auf dem Server-Datenbanksystem stattfinden sollen.

3.3 Geowissenschaftliche Beispielanwendungen

Die folgenden geowissenschaftlichen Anwendungen dienen als Beispiel für die Evaluation der entworfenen und prototypisch implementierten Konzepte für die 3D Geodatenverwaltung mit mobilen Datenbanksystemen in Kapitel 6.

3.3.1 Geologische Modellierung der Schaumburg-Lippeschen Kreidemulde

Die Geologische Modellierung und Kartierung der Schaumburg-Lippeschen Kreidemulde erfolgte innerhalb des DFG-Antragbündels „Interoperable Geowissenschaftliche Informationssysteme - IOGIS“ durch das Geologische Institut der Universität Bonn [22]. Die Schaumburg-Lippesche Kreidemulde liegt im südlichen Norddeutschen Becken am Nordrand des Wesergebirges. Die Eignung dieses Gebietes als Untersuchungsgebiet für geologische Modellierung des Untergrundes ist auf die Existenz einer hinreichend große Datendichte aufgrund der durchgeführten Explorationen auf Kohlenwasserstoffe zurückzuführen. Eine Beschreibung des Projektes und der durchgeführten Arbeiten findet sich in [23, 21].

Innerhalb des DFG-Projektes erfolgte die Konstruktion von 3D-Strukturmodellen des Untergrundes anhand der vorhandenen Daten. Hierbei wurden zwei Modellierungsverfahren kombiniert. Es erfolgte eine geometrisch-geologische Modellierung mit dem Modellierungsprogramm GOCAD [78]. Das resultierende 3D-Modell besteht aus den geologischen Grenzflächen (Schichtgrenzen und Störungsflächen) und einem daraus abgeleiteten 3D-Volumenmodell. Weiterhin wurde eine geophysikalische 3D-Modellierung mit IGMAS (Interactive Gravity and Magnetic Application System) [49, 71] auf der Basis gravimetrischer Daten durchgeführt. Die Konstruktion und Validierung des 3D-Modells erfolgte durch iterativen Austausch der Ergebnisse der jeweiligen Modellierung.

Von besonderen Interesse für die vorliegende Arbeit ist die geometrisch-geologische Konstruktion des 3D-Modells. Dieses Modell besteht aus mehreren (mehr oder weniger) horizontal verlaufenden Schichtflächen, die von verschiedenen Störungs- und Verwerfungsflächen durchdrungen werden. Bei der Konstruktion der Modelle muß, vereinfacht dargestellt, jede Schichtfläche mit jeder Störungsfläche verschnitten und aneinander angepaßt werden. Abbildung 3.1 zeigt ein fertiges Untergrundmodell der Schaumburg-Lippeschen Kreidemulde.

Jeder dieser Schnitt- und Anpassungsvorgänge kann in einzelne Schritte zerlegt werden, die jeweils ein Zwischenergebnis darstellen. In dem beschriebenen Projekt erfolgte entweder keine Speicherung der vielen Zwischenergebnisse bzw. nur eine Ablage der groben Schritte im Dateisystem. Hiermit ergeben sich Probleme, wenn einzelne Schritte später für Änderungen nicht mehr zur Verfügung stehen bzw. der Ablauf nicht mehr nachvollzogen werden kann.

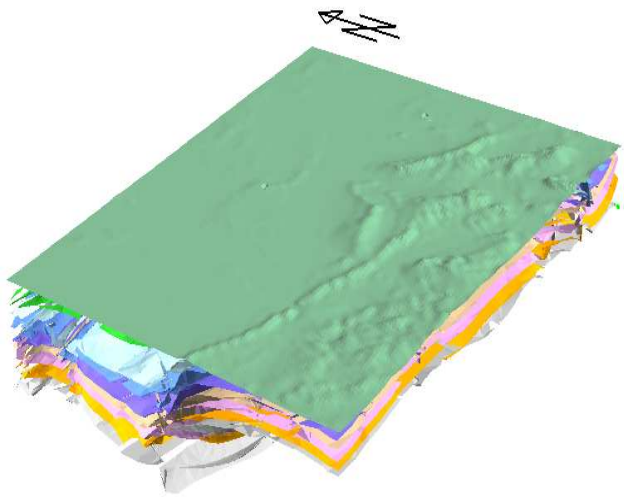

Abbildung 3.1: Untergrundmodell der Schaumburg-Lippeschen Kreidemulde (Visualisierung mit GOCAD basierend auf Daten des DFG-Forschungsprojektes IOGIS [22])

Versionsmanagement in geowissenschaftlichen Datenbanksystemen kann hier eine Hilfestellung bei der Modellentwicklung geben. Die einzelnen Schritte im Verlauf der Modellierung können innerhalb der Versionsverwaltung als Versionen erhalten werden. Somit sind jederzeit die einzelnen Schritte nachvollziehbar und können für Änderungen weiterhin benutzt werden. Ebenfalls kann die Entstehungsgeschichte der Modelle aus den Informationen der Versionshistorie rekonstruiert und nachvollzogen werden. Das Untergrundmodell wird in Kapitel 6 zur Evaluation des Laufzeitverhaltens des Datenbanksystems auf komplexen 3D-Modellen herangezogen.

3.3.2 Hangrutschungen am Albtrauf in Balingen

Ein weiterer geowissenschaftlicher Anwendungsbereich, der von mobiler Geodatenverwaltung unterstützt werden kann, sind Hangrutschungen. Innerhalb des Projektes „Weiterentwicklung von Geodiensten“ (siehe Kapitel 1.1) erfolgte die Entwicklung von online nutzbaren datenbankbasierten Geodiensten. Das Anwendungsszenario des Projektes sind die Hangrutschungen am Albtrauf in Balingen. Innerhalb der vorliegenden Arbeit wird das Konzept der mobilen Geodatenbank anhand dieses Anwendungsszenarios evaluiert.

Die Hangrutschungen finden am Winkelgrat, einem markanten Vorsprung des Albtraufs südwestlich von Laufen im Zollernalbkreis (Baden-Württemberg), statt. Hier werden seit vielen Jahren tiefreichende Felsbewegungen beobachtet (siehe Abbildung 3.2), welche eine potentielle Gefährdung einer Kreisstraße darstellen [106].

Seit einigen Jahren ist hier ein automatisches Überwachungssystem im Einsatz. Mit fünf

Abbildung 3.2: Felsspalte am Winkelgrat in Balingen

hochempfindlichen Extensometern (Abbildung 3.3) werden die Bewegungen der Teilschollen kontinuierlich aufgezeichnet und an einen Feld-Rechner gemeldet. Dieser löst bei Überschreitung gewisser Grenzwerte einen Alarm aus, die Kreisstraße wird mittels zweier Ampeln gesperrt und der zuständige Geologe benachrichtigt. Nach Besichtigung vor Ort kann entweder die Sperrung aufgehoben werden, oder es wird entschieden, welche weiteren Maßnahmen zur Sicherung der Straße notwendig sind.

Abbildung 3.3: In einer Felsspalte installiertes Extensometer

Für die vorliegende Arbeit werden die Spaltenveränderungen durch die Bewegungen der Teilschollen in der mobilen Datenbank verwaltet. Auf diese Weise können Änderungen in den Spaltengeometrien im Gelände erfasst und als neue Version in das Versionsmanagement des Datenbanksystems überführt werden. Die erfassten Änderungen der Spaltengeometrien können anschließend, bei signifikanten Änderungsschritten, als Aktualisierung in die Hauptdatenbank überführt werden. Die verschiedenen Versionen können weiterhin zur automatischen Konstruktion des 4D-Modelles der Spaltenbewegungen im Datenbanksystem genutzt werden.

3.4 Konzept

Dieser Abschnitt gibt einen Überblick über das in dieser Arbeit verfolgte Konzept zur Verwaltung geowissenschaftlicher 3D Daten in mobilen Datenbanksystemen. Die Verwaltung der geowissenschaftlichen 3D Daten soll dabei im offline Modus durch die Nutzung von mobilen 3D Datenbanken integriert mit den Server-Datenbanksystemen möglich sein. Die Konzepte werden als Prototypen basierend auf dem geowissenschaftlichen 3D-Datenbanksystem DB3D [17, 18] vorgestellt und diskutiert.

Die grundlegende Idee ist die Integration von mobilen Datenbanken über ein Versionsmanagement im Server-Datenbanksystem. In diesem Konzept erfolgt keine automatische Konfliktauflösung bei der Reintegration von geänderten Geodaten, sondern die Änderungen werden als neue Version des ursprünglichen Datenbankobjektes im Datenbanksystem gespeichert. Etwaige Konflikte zwischen Versionen werden vom Nutzer bei der Propagierung durch eine Hierarchie von Ebenen des Versionsmanagements in Kooperation aufgelöst.

Wie bereits erwähnt, basieren geowissenschaftliche Anwendungen im allgemeinen auf einem kooperativen Arbeitsansatz. Vornehmlich im Anwendungsgebiet der Geologie – z.B. bei der Modellierung von 3D-Modellen – erfolgt eine sehr starke iterative Verbesserung der Modelle. Hier stellt sich die Frage, ob bei Änderungen im Modell nur eine Konfliktauflösung bei der Reintegration gewünscht ist oder ob es sinnvoll ist, auch bestimmte Modellzustände beizubehalten, um nicht die bisherigen Arbeiten zu *überschreiben*. In der Arbeit wird davon ausgegangen, daß es für die Unterstützung der 3D-Modellierung nötig ist, die Speicherung von Modellzuständen anzubieten. Soll ein alter Modellzustand nicht beibehalten werden, so liegt die Aufgabe der Konfliktauflösung – falls in der Zwischenzeit auch andere Änderungen gemacht wurden – bei den Anwendern. Die Fachexperten entscheiden damit in Kooperation, welcher Modellzustand im Konfliktfall Vorrang hat bzw. wie ein gemeinsamer Modellzustand durch eine Vereinigung der Modellzustände erreicht werden kann. Diese Entscheidungen können nicht automatisch durch das Datenbanksystem erfolgen. Das Datenbanksystem kann hier Hilfestellung durch Bereitstellung von Verfahren zum Erkennen von Konflikten und zum Berechnen von Unterschieden zwischen Modell-

zuständen anbieten. Hierdurch wird es den Anwendern ermöglicht zu entscheiden, welcher Modellzustand bestehen bleiben soll, bzw. wie die Modellzustände vereinigt werden können.

Die prototypische Realisierung basiert vollständig auf objekt-orientierter Datenbanktechnologie. Hierdurch ist es möglich, die Programmcode-Basis des Datenbanksystems DB3D für die Entwicklung der mobilen Datenbank wiederzuverwenden. Durch die Verwendung von objekt-orientierten Datenbanken ist weiterhin die Entwicklung komplexer Datenmodelle auf mobilen Geräten möglich. Somit kann beispielsweise das Datenbankobjektmodell von dem Server-Datenbanksystem größtenteils auch für die mobile Datenbank übernommen werden. Innerhalb der wiederverwendeten Codebasis können dann Optimierungen für mobile Datenbanken vorgenommen werden.

Im anschließenden Kapitel wird das Datenbanksystem für geowissenschaftliche 3D Anwendungen DB3D ausführlich behandelt. Hier wird der Aufbau des Datenbankkerns mit dem Datenbankobjektmodell auf Basis Simplizialer Komplexe beschrieben und auf die Dienst-Architektur für den Zugriff eingegangen. Darauf aufbauend erläutert Kapitel 5 die Erweiterung dieses Datenbanksystems um ein Versionsmanagement für 3D-Geoobjekte zur Nutzung mit mobilen Datenbanken. Hier werden zum einen die Möglichkeiten der Einbindung von Versionsmanagement und die Repräsentation von Versionen für die Datenbankobjekte diskutiert und beschrieben. Zum anderen wird der Aufbau einer mobilen Datenbank auf Basis des Server-Datenbanksystems und deren Integration mit dem Versionsmanagement vorgestellt. Der weitere Verlauf der Arbeit 6 beschäftigt sich mit der Evaluation der prototypischen Implementierung anhand der geowissenschaftlichen Beispielanwendungen.

Kapitel 4

Datenbanksystem für geowissenschaftliche 3D Anwendungen

DB3D dient in der vorliegenden Arbeit als Grundlage zum Aufbau eines Prototypen für ein Server-Datenbanksystem mit Versionsmanagement, auf das die mobilen Datenbanken zugreifen können.

4.1 Ausgangspunkt und Architektur

In einer verteilten, mobilen Umgebung muß ein geowissenschaftliches 3D Datenbanksystem dem Bearbeiter, sowohl im Büro als auch im Gelände, Zugriff und Operationen auf komplette 3D-Modelle bereitstellen. Als ein zentraler Punkt wird hierbei die Nutzbarkeit des Datenbanksystems mit mobilen Geräten gesehen. Dies wird bei DB3D durch die Bereitstellung des Zugriffs über Dienste erreicht. Diese Dienste bilden die vollständige Funktionalität des Datenbanksystems ab und stellen dem Nutzer diese, ohne höherwertige Funktionalität von dem mobilen Endgerät zu erwarten, zur Verfügung. Um einen offenen Austausch der Daten innerhalb des Datenbanksystems zu gewährleisten erfolgt der gesamte Datenaustausch in einem XML-Format. Dienste zum Transformieren dieses XML-Formates bieten damit die Möglichkeit eine Vielzahl von im Detail variierenden Datenformaten zu unterstützen.

Der objekt-orientierte Ansatz hat sich in den letzten Jahren auch beim geometrischen Modellieren sehr gut etabliert. Auf dem Gebiet der Datenbanksysteme haben es objekt-orientierte Datenbanksysteme jedoch nicht erreicht, den relationalen bzw. objekt-relationalen Ansatz abzulösen. Wie die Evaluation sowohl objekt-relationaler als auch objekt-

orientierter Datenbanksysteme auf ihre Eignung für die prototypische Entwicklung eines 3D Datenbanksystems gezeigt hat [53, 18], besitzt der objekt-orientierte Ansatz jedoch deutliche Vorteile gegenüber objekt-relationalen Systemen. Die Übertragung komplexer objekt-orientierter 3D-Datenmodelle auf das objekt-relationale Datenmodell verursacht, trotz gewisser objekt-orientierter Unterstützung, immernoch den bekannten *impedance-mismatch*. Der größte Vorteil liegt in der deutlich reduzierten Entwicklungszeit mit objekt-orientierten Datenbanksystemen, da hier die Implementierung des nötigen *Mapping-Codes* zwischen den Datenstrukturen entfällt. Dieser Faktor ist gerade in zeitlich begrenzten Forschungsprojekten von entscheidender Bedeutung.

DB3D baut auf einem kommerziellen objekt-orientierten Datenbanksystem mit Page-Server Architektur auf (ObjectStore [74]). Die Softwarearchitektur ist in mehrere horizontale Schichten unterteilt. In Abbildung 4.1 sind die einzelnen Schichten schematisch dargestellt.

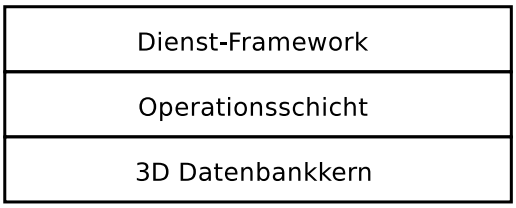

Abbildung 4.1: Schichten der Softwarearchitektur von DB3D

Basierend auf dem objekt-orientierten Datenbanksystem wurde ein Datenbankkern zur Verwaltung geometrisch/thematischer 3D-Objekte konzipiert. Der Datenbankkern beinhaltet nur elementare 3D-Geometrieobjekte, das 3D-Objektmodell sowie Zugriffsstrukturen. In der Operationsschicht existieren Operationen basierend auf dem 3D-Objektmodell, wie z.B. Input/Output- oder geometrische Schnittoperationen zwischen 3D-Objekten. Diese verwenden zur Realisierung ihrer komplexen Funktionalität die Datenstrukturen, die vom 3D Datenbankkern zur Verfügung gestellt werden. Die oberste Schicht in der Softwarearchitektur stellen die Dienste dar. Diese Schicht realisiert einen Framework, der es ermöglicht, Datenbankfunktionalität basierend auf den implementierten Operationen als Dienste zur Verwendung über das Internet zur Verfügung zu stellen.

Skalierung des Datenbanksystems In objekt-orientierten Page-Server Architekturen liegt die Hauptlast des Systems bei der Applikation bzw. dem Applikationsserver. Der Page-Server ist alleinig für die Speicherung und Auslieferung von Seiten-Einheiten (Pages) zuständig und kann dadurch eine Vielzahl von Anwendungen als Clients bedienen. Abbildung 4.2 zeigt die Verteilung des Datenbanksystems auf mehrere Rechnerknoten im Netzwerk, wodurch eine Skalierung zur Lastverteilung erreicht werden kann.

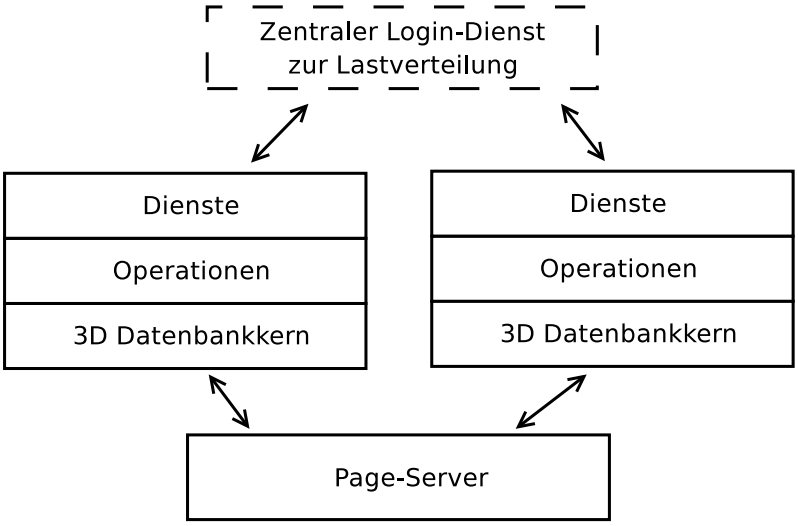

Abbildung 4.2: Verteilung der Softwarearchitektur auf Rechnerknoten

Der zentrale Login-Dienst ist für die transparente Zuordnung der Clients zu den einzelnen Instanzen der Datenbanksystem-Applikationen auf verschiedenen Rechnerknoten im Netzwerk zuständig. Innerhalb dieser Zuordnung der Clients erfolgt die Lastverteilung (Loadbalancing) zur gleichmäßigen Auslastung des Systems.

4.2 3D-Datenbankkern

Der Datenbankkern implementiert die Grundfunktionalität von DB3D. Hierfür stellt er eine Schnittstelle zur Verbindungsaufnahme, Organisations- und Zugriffsstrukturen sowie das geometrisch-thematische Datenmodell bereit.

4.2.1 Abstraktion

Um eine Abhängigkeit von einem konkreten objekt-orientierten Datenbanksystem zu vermeiden wird eine Abstraktionsschicht verwendet. Hierdurch ist eine relativ leichte Portierung des Systems auf weitere OODBMS möglich. Die Abstraktionsschicht stellt abstrahierte Schnittstellen zur Verbindungsaufnahme und zu Datenstrukturen zur Verwaltung von Objektmengen bereit.

Die Schnittstelle zur Verbindungsaufnahme definiert die Erzeugung von Verbindungen, sowie die benötigten Schnittstellen zur Interaktion mit dem Datenbanksystem (Connection, Session und Transaktionsmanagement). Die Schnittstellen sind als *Interfaces* definiert und werden über eine *Factory* zur Erzeugung von Verbindungen bezogen. Konkrete OODBMS

werden dann durch die Bereitstellung einer Implementierung der Factory und der Interfaces unterstützt.
Die benötigten Datenstrukturen zur Verwaltung von Mengen von 3D-Geoobjekten sind ebenfalls in einer Factory zusammengefaßt. Innerhalb des Datenbankkerns werden die Datenstrukturen nur über diese Factory eingebunden. Für jedes konkrete OODBMS erfolgt wiederum die Implementierung dieser Factory. Werden bestimmte Datenstrukturen nicht durch das konkrete OODBMS bereitgestellt, können diese als Applikationsobjekte des Datenbanksystems implementiert werden.

Bei der beschriebenen Vorgehensweise existiert spezifischer Datenbanksystem-Code nur noch an den beschriebenen Schnittstellen. Softwarekomponenten, die auf dem Datenbankkern aufsetzen sind damit unabhängig von einem konkreten OODBMS eines Herstellers.

4.2.2 Organisation

Zur organisatorischen Unterteilung der Datenbestände beinhaltet eine Datenbank eine Menge von Projekten. Ein Projekt hat einen eindeutigen Namen und ID innerhalb der Datenbank. Projekte können eine beliebige Anzahl von *Space*s (Räumen), in denen die logisch zusammengehörigen Datenbankobjekte verwaltet werden, enthalten. Abbildung 4.3 stellt die Organisationsstruktur einer Datenbank in UML-Notation dar.

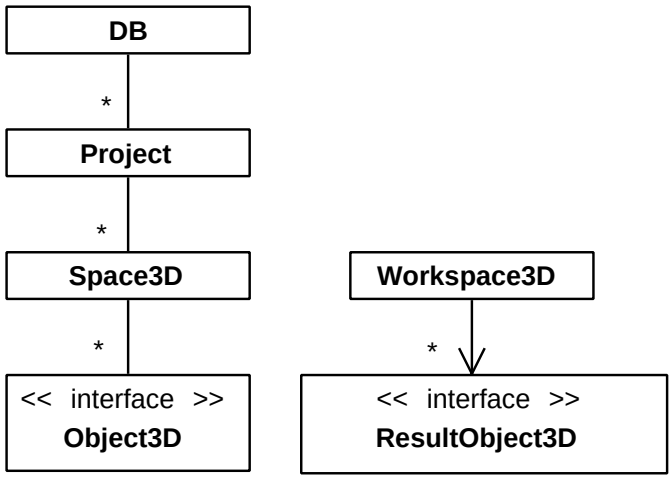

Abbildung 4.3: Organisationsstruktur einer Datenbank im 3D-Datenbankkern

Space: Ein Space stellt ein Container-Objekt zur Verwaltung einer Menge von Datenbankobjekten gleicher Dimension dar. Bei der Erzeugung eines Space werden bestimmte Definitionen festgelegt, die für alle Objekte in diesem Space gelten. Für einen Space3D – einem Space für 3D-Geoobjekte – sind folgende Definitionen vorhanden:

- Informationen: Besitzer, Erstellungsdatum, Namen, Beschreibung etc.
- Integritätsbedingungen des Raumes: Geometrische und thematische Integritätsbedingungen, die durch das Einfügen eines Objektes nicht verletzt werden dürfen.
- Koordinatensystem: Koordinatensystem im Raum - lokales oder Referenzsystem.
- Genauigkeitskonstante: Kleinster von Null verschiedener Skalar für geometrische Vergleichsoperationen.

Ein Space ist für die konsistente und effiziente Speicherung der enthaltenen Datenbankobjekte verantwortlich. Hierfür existieren Strukturen zur Indexierung der Objekte nach thematischen und geometrischen Eigenschaften, um eine effiziente Selektion von Objekten aus einem Space zu ermöglichen. Grundsätzlich können beliebige Objekte in einem Space verwaltet werden. Dies beinhaltet Objekte gleicher oder geringerer Dimension als die Dimension des Space. Die Einschränkung enthaltener Objekte in einem Space erfolgt durch dessen Integritätsbedingungen. Als Integritätsbedingungen können zum einen Bedingungen definiert werden, die sich nur auf das einzufügende Objekt beziehen. Beispielsweise Integritätsbedingungen, die nur die Einfügung von Objekten der Dimension X oder Thematik Y bzw. einer Kombination zulassen. Zum anderen können Integritätsbedingungen in Bezug auf die bereits im Space enthaltenen Objekte definiert und damit z.B. überschneidungsfreie Räume garantiert werden.

Workspace: Neben dem Space existiert zusätzlich das Workspace-Konzept. Ein Workspace stellt den temporären Arbeitsraum einer Datenbank-Verbindung (Session) dar. Innerhalb des Workspaces können Zwischenergebnisse beispielsweise von Konstruktionsoperationen gespeichert werden. Die Workspaces können transient (speicherbasiert) oder persistent (datenbankbasiert) aufgebaut werden. Hiermit besteht die Möglichkeit, die Ergebnisobjekte eines Dienstes konsistent zwischenzuspeichern und einem weiteren nachfolgenden Dienst zur weiteren Bearbeitung zur Verfügung zu stellen. Erfolgt der Aufbau persistent, ergibt sich dadurch eine transaktionsorientierte Abarbeitung von *Dienst-Ketten* (vgl. Kapitel 4.4.3), bei dem nach jedem Dienst ein konsistenter Datenbankzustand erreicht wird. Abbildung 4.3 zeigt einen Workspace3D zur Verwaltung einer Menge an 3D-Ergebnisobjekten.

4.2.3 Datenmodell

Das geometrische Datenmodell basiert auf dem Modell simplizialer Komplexe (siehe Kapitel 2.1.1). Es werden Datenbank-Objekte für 0D, 1D, 2D und 3D-Objekte sowie eine Hüllenrepräsentation auf Basis geschlossener Dreiecksnetze (simpliziale 2-Komplexe) im

3D Raum unterstützt. Im folgenden wird das Datenmodell ausgehend von der zugrundeliegenden Geometriebibliothek beschrieben.

4.2.3.1 Geometriebibliothek

Die Geometriebibliothek stellt die 3D-Geometrieobjekte und analytischen Geometrieobjekte bereit. Abbildung 4.4 zeigt ausgewählte Klassen dieser Bibliothek. Die Klassen *Vector3D*, *Line3D* und *Plane3D* modellieren Vektoren, Geraden und Ebenen im 3D. Der *ScalarOperator* besitzt Methoden zum Vergleich von Skalaren unter einer gegebenen Genauigkeitskonstante *Epsilon*. Alle geometrischen Vergleiche in den Methoden der Klassen der Geometriebibliothek basieren auf diesem Prinzip, um Fehler durch ungenaue Repräsentation von Fließkommazahlen abzufangen.

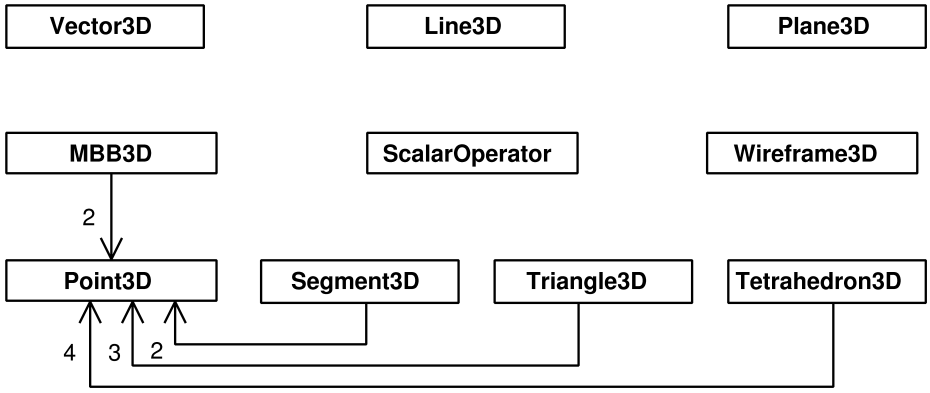

Abbildung 4.4: Ausgewählte Klassen der Geometriebibliothek (vereinfacht)

Mit *Point3D*, *Segment3D*, *Triangle3D* und *Tetrahedron3D* werden die Simplexe bereitgestellt. Aus Effizienzgründen sind die Implementierungen der Simplexe nicht über ihre d-1 dimensionalen Seitensimplexe definiert, sondern nur über ihre Knoten. Die Erzeugung der Seitensimplexe erfolgt zur Laufzeit, wenn Algorithmen diese benötigen. Mit *MBB3D* steht eine Klasse für minimale umschließende Quader (Minimum Bounding Box) in der dritten Dimension zur Verfügung. Die Klasse *Wireframe3D* modelliert ein Drahtgitter-Objekt mit einzelnen Knoten sowie Verbindungskanten. Diese Klasse dient als Ergebnisobjekt für eine Vielzahl von Operationen zwischen den Klassen der Geometriebibliothek. So kann beispielsweise der Schnitt zwischen zwei Tetraedern als Ergebnis entweder einen Point3D, Segment3D, Triangle3D oder Tetrahedron3D als definierten Simplex liefern, jedoch auch ein beliebiges Schnittergebnis, das dann als Wireframe3D repräsentiert wird. Abbildung 4.5 zeigt ein Beispiel für ein Schnittergebnis, das nur als Wireframe repräsentiert werden kann.

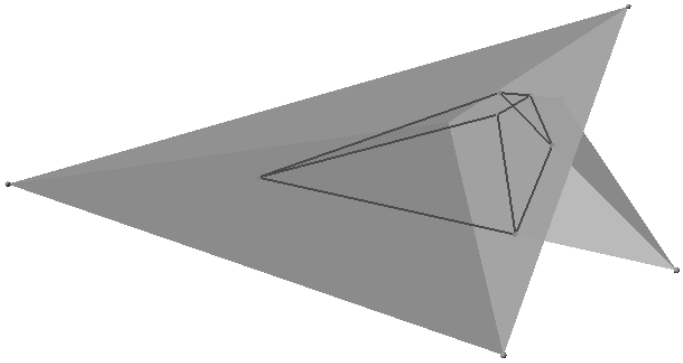

Abbildung 4.5: Schnitt zweier Tetraeder: Ergebnis als Wireframe dargestellt

4.2.3.2 3D-Datenbankobjekte

Abbildung 4.6 zeigt die beiden 3D-Datenbankobjekte *Object3D* und *ResultObject3D*. Object3D dient als die Schnittstelle für alle definierten Datenbankobjekte im 3D, die in der Datenbank verwaltet werden können, und ResultObject3D als Schnittstelle für Ergebnisobjekte von Operationen.

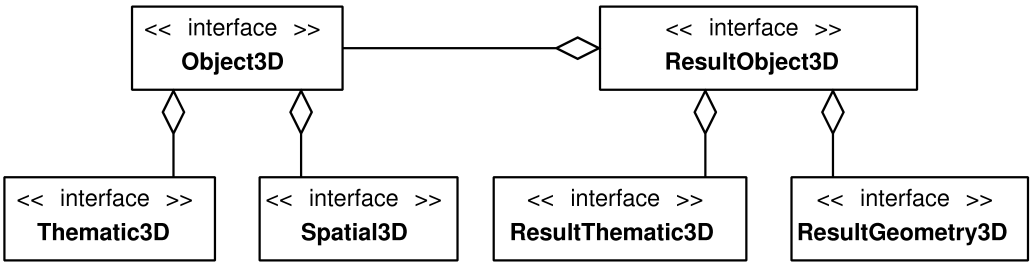

Abbildung 4.6: Schnittstellen der Datenbankobjekte

Das Interface Object3D beinhaltet als Aggregation die Interfaces *Thematic3D* und *Spatial3D*, die die thematischen und geometrischen Eigenschaften repräsentieren. Ein ResultObject3D-Objekt kann ein Object3D-Objekt als Ergebnis einer Operation beinhalten. Hiermit werden Operationen ermöglicht, die ein Ergebnis nach dem simplizialen Komplex-Modell liefern. Die Mehrzahl der Operationen im 3D liefert jedoch keine Ergebnisse nach diesem Modell, sondern eine Menge an Geometrieobjekten (Simplexe, Wireframes) unterschiedlicher Dimension mit ihrer Thematik. Für diese Ergebnisse stellt ResultObject3D die Interfaces *ResultThematic3D* und *ResultGeometry3D* bereit. ResultGeometry3D-Objekte sind somit als Ergebnisobjekte mit unbekannter Dimension und Inhalt zu verstehen. Me-

thoden zur Bestimmung des Inhaltes und zur Filterung nach Typ bzw. Dimension der Geometrieobjekte ermöglichen in einem weiteren Schritt, nur die gewünschten Geometrieobjekte als Ergebnis zu erhalten.

Als Beispiel für die Notwendigkeit solcher Ergebnisobjekte für Operationen im 3D soll hier der Schnitt eines Dreiecksnetzes (simplizialer 2-Komplex) mit einer Ebene ausgeführt werden: Als *normales* Ergebnis dieser Operation wird ein Segmentnetz (simplizialer 1-Komplex) erwartet. Je nach Lage der einzelnen Dreiecke zur Schnittebene können jedoch auch Punkte (Dreieck berührt Ebene mit einer Ecke) bzw. Dreiecke (Dreieck liegt auf der Ebene) im Ergebnis der Operation auftauchen. Mit Hilfe der Ergebnisobjekte wird es dem fachkundigen Anwender ermöglicht das jeweilige gewünschte Ergebnis zu bekommen. Wird nur das Segmentnetz benötigt, so kann dies aus dem ResultGeometry3D-Objekt gefiltert werden. Ist der Anwender jedoch an z.B. den auf der Schnittebene liegenden Dreiecken interessiert, so kann er auch diese über die Filtermethoden erhalten.

Geometriemodell Im folgenden wird das implementierte Geometriemodell für 3D-Geoobjekte beschrieben, soweit es für die weitere Arbeit zum Verständnis relevant ist. Das implementierte Modell der simplizialen Komplexe baut auf der Geometriebibliothek auf. Die Simplex-Klassen *PointElt3D*, *SegmentElt3D*, *TriangleElt3D* und *TetrahedronElt3D* sind von den Geometrieobjekten Point3D, Segment3D, Triangle3D und Tetrahedron3D abgeleitet und erweitern diese um Identifikation im simplizial Komplex und Nachbarschaftstopologie. Ein vereinfachtes Klassendiagramm des Geometriemodells der Datenbankobjekte zeigt Abbildung 4.7.

Das Interface Spatial3D definiert die gemeinsamen Methoden aller Geoobjekte im 3D. Für jeden simplizialen k-Komplex existiert weiterhin ein spezialisiertes Interface (z.B. *Surface3D* für simpliziale 2-Komplexe) mit speziellen Methoden. Die oberste Implementierungsklasse stellt die abstrakte Klasse *SpatialObject3D* dar, die das Interface Spatial3D implementiert. Von ihr abgeleitet sind die jeweiligen konkreten Klassen der einzelnen Dimensionen, die wiederum die Interface-Spezialisierungen von Spatial3D implementieren.

Die Struktur der Implementierung ist symmetrisch über die Dimensionen. Sei für ≪Simplex≫ je nach Dimension Point3D, Segment3D, Triangle3D und Tetrahedron3D eingesetzt, so kann die Implementierung für alle Dimensionen wie folgt beschrieben werden:
Die Klassen ≪*Simplex*≫*Net3D* stellen die Modellierung eines Netz-Objektes bestehend aus einer beliebigen Anzahl untereinander nicht topologisch zusammenhängender Netzkomponenten – ≪*Simplex*≫*Net3DComp* – dar. Die Klassen ≪Simplex≫Net3DComp modellieren dann den einzelnen simplizialen k-Komplex durch eine Menge von adjazenten, disjunkten ≪*Simplex*≫*Elt3D*-Objekten der jeweiligen Dimension k. Innerhalb einer solchen Netzkomponente sind die Simplexe topologisch zusammenhängend, d.h. es existieren keine isolierten Simplexe.

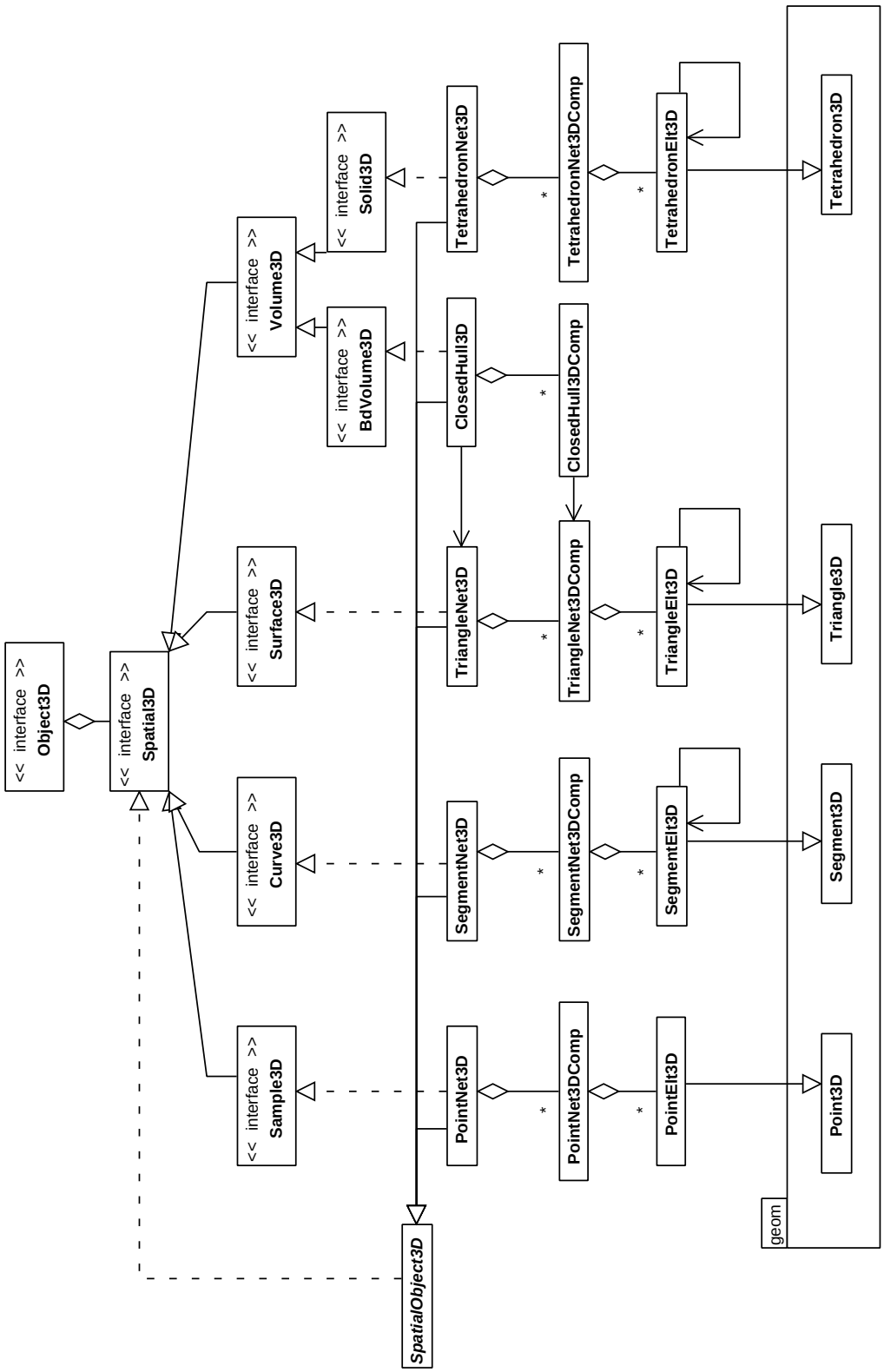

Abbildung 4.7: Geometriemodell der Datenbankobjekte in UML-Notation

Einen Sonderfall in der Implementierung nimmt die Unterstützung von Körpern durch eine geschlossene Hüllenrepräsentation auf Basis von simplizialen 2-Komplexen ein, die mit den Klassen *ClosedHull3D* und *ClosedHull3DComp* realisiert sind. Diese Klassen werden als Wrapper um die Klassen *TriangleNet3D* und *TriangleNet3DComp* verwendet. Flächenspezifische Methodenaufrufe werden, nach Prüfung auf geschlossenes Netz, an diese Klassen weitergeleitet. Volumenspezifische Methoden (z.B. inside-Test) sind in der Wrapper-Klasse implementiert.

Die ≪Simplex≫Elt3D-Objekte sind in ≪Simplex≫Net3DComp räumlich indexiert. Hierdurch kann bei einer Vielzahl von räumlichen Operationen ein selektiver räumlicher Zugriff auf die einzelnen Simplexe bzw. die räumliche Simplex-Menge erreicht werden. Beim Aufbau eines ≪Simplex≫Net3DComp-Objektes werden alle Simplexe im Index registriert und anschließend unter Ausnutzung der aufgebauten Indexstruktur die Nachbarschaftstopologie zwischen den Simplexen aufgebaut. Die räumliche Indexstruktur wird abstrahiert über das Interface *SAM* (Spatial Access Method), und über die bereits angesprochene Factory für Datenstrukturen eingebunden. Der derzeitige Prototyp bietet zwei Implementierungen an. Mit der Klasse *RStar* existiert eine Implementierung auf R*-Baum Basis, die eine gleichbleibende Anfrageeffizienz bei mit zunehmender Größe steigenden Einfügezeiten bietet. Weiterhin wird mit der Klasse *Octree* ein Ansatz der nichtüberlappenden Raumpartitionierung mit Objektduplikation angeboten. Vorteil des Octrees ist der deutlich beschleunigte Einfügevorgang, da hier die Baumreorganisationsoperationen nicht stattfinden (vgl. Kapitel 2.1.2.1).

Thematisches Modell Abbildung 4.8 zeigt das thematische Modell mit den wichtigsten Klassen als UML-Diagramm. Die Thematik (*Thematic3D*) eines 3D-Geoobjektes (*Object3D*) wird implementiert durch die Klasse *ThematicObject3D*. Über diese Klasse kann ein Geoobjekt zu einer thematischen Gruppe (*ThematicGroup*) im Space gehören und für die in der jeweiligen Gruppe definierten Attribute Sachdaten enthalten. ThematicObject3D beinhaltet dabei die Thematik des Geoobjektes (Assoziation *Object3DThematic*) und der einzelnen Simplexe im Geoobjekt (Assoziation *SimplexThematic*).

Damit erlaubt das Datenmodell, neben einer sich auf das gesamte 3D-Geoobjekt beziehenden Thematik, auch die Bereitstellung einer Thematik für die Simplexe. Es können eine Reihe von Simplex-Attributen definiert werden und jedes Simplex kann für diese Attribute Werte enthalten. Auf diese Weise ist es möglich Zustände wie z.B. die physikalische Dichte in Volumenkörpern, modelliert als simpliziale 3-Komplexe, zu unterstützen. Als Datentypen für Attribute in den Objekt- und Simplex-Thematiken werden die Datentypen Fließkomma- und Ganzzahlen, boolsche Werte, Text und Vektoren unterstützt. Die Unterstützung von Vektoren als Datentyp ermöglicht es auch in *statischen* 3D-Modellen Bewegungsrichtungen und -größen durch thematische Angaben zu modellieren.

Um beliebige Sachdaten in der Thematik zu ermöglichen, wird eine zur Laufzeit definierba-

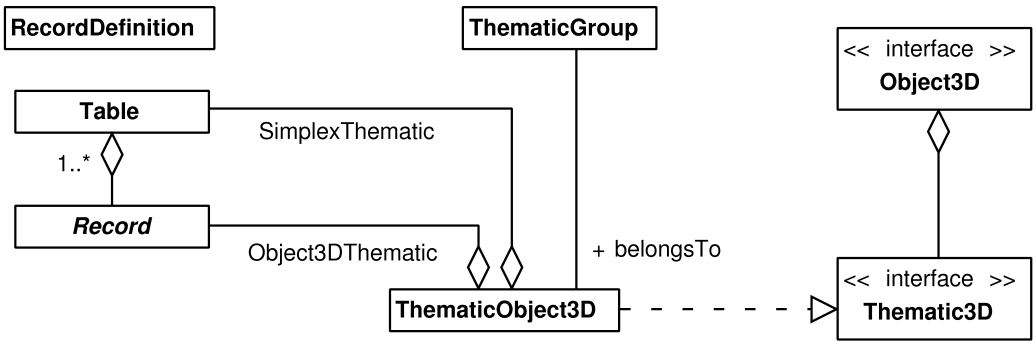

Abbildung 4.8: Thematisches Modell der Datenbankobjekte in UML-Notation

re Tabellenstruktur benutzt. Die Struktur besteht aus einer Tabelle (Klasse *Table*), die eine indexierbare Sammlung von Datensätzen (Klasse *Record*) beinhaltet, die einer bestimmten Typdefinition (Klasse *RecordDefinition*) entsprechen. Diese Typdefinition enthält die Spaltenanzahl, den Spaltennamen und die Spaltendatentypen.

4.2.4 Elementare Operationen

Im Datenbankkern sind nur elementare Operationen enthalten. Weitergehende Operationen wie Konstruktion von Geoobjekten, Ein-/Ausgabe-Operationen oder geometrische Operationen zwischen Geoobjekten basieren auf dem Datenmodell und sind außerhalb des Datenbankkerns verwirklicht (vgl. Kapitel 4.3).

Im Datenbankkern werden die folgenden Operationen auf Simplexen bzw. zwischen Simplexen zur Verfügung gestellt.

- Tests topologischer Beziehungen anhand der Bounding Box.
- Tests topologischer Beziehungen mit exakter Geometrie.
- Schnittberechnungen.
- Projektionsberechnungen auf Ebenen.
- Tests für strikte Gleichheit und geometrische Äquivalenz.
- Tests auf Validität, Regularität und Wohlgeformtheit.
- Berechnung der Seiten-Simplexe.
- Berechnung spezifischer Kenngrößen (Fläche, Volumen, Zentrum, Winkel, etc.).

Weiterhin enthält der Datenbankkern folgende elementare Operationen auf 3D-Geoobjekten.

- Tests topologischer Beziehungen zwischen 3D-Geoobjekten und Simplexen.
- Berechnung der Randelemente jeder Dimension.
- Berechnung spezifischer Kenngrößen (Gesamtfläche/-volumen, Anzahl der Randelemente, Anzahl der Simplexe/Seiten-Simplexe pro Dimension).

4.2.5 XML-Modell

Im Kapitel 2.1.3 Interoperabilität wurde bereits auf das Problem der Standardisierung von Datenformaten im 3D eingegangen. Da derzeit kein standardisiertes Datenformat für die Repräsentation von Simplizialen Komplexen existiert, wurde für das Datenbanksystem DB3D ein eigenes XML-Format (DB3D-XML) entworfen.

Das XML-Format ermöglicht die Repräsentation von Metadaten der Datenbank und die Repräsentation der unterstützten Objekttypen. Es stellt das Standardaustauschformat des Datenbanksystems dar. Ausgehend von diesem Format kann der Import und Export weiterer Ascii-Formate über XSL-Transformation – auch durch den Nutzer – erweitert werden. Derzeitig unterstützt der Prototyp über XSL-Transformation das Format VRML beim Export sowie X3D beim Import und Export. Das XML Schema für das DB3D XML-Format ist im Anhang A zu finden.

4.3 Externe Operationen

Die externen Operationen können in die Kategorien Ein-/Ausgabeoperationen und Geometrieoperationen unterteilt werden. Diese Operationen setzen auf dem Datenmodell des Datenbankkerns auf und bieten eine höherwertige Funktionalität für die von der Datenbank bereitgestellten 3D-Geoobjekte an. Alle externen Operationen erben von der abstrakten Klasse *AbstractOperation*, durch die jeder Operation ein *OperationContext* Objekt zur Verfügung gestellt wird. Der OperationContext bietet den Operationen die Möglichkeit zum Zugriff auf das Datenbanksystem, den aktuellen Workspace sowie verschiedene Methoden zum Erzeugen von persistenten Ergebnisobjekten in der Datenbank. Abbildung 4.9 stellt einen Auszug der Klassen des Operationenmodells dar. In der Abbildung sind verschiedene Beispieloperationen zu sehen, auf die im folgenden näher eingegangen wird.

4.3.1 Ein-/Ausgabeoperationen

Grundsätzlich können Datenformate über eine Implementierung der Konvertierung der Datenbankobjekte in ein Datenformat unterstützt werden. Diese Möglichkeit hat jedoch

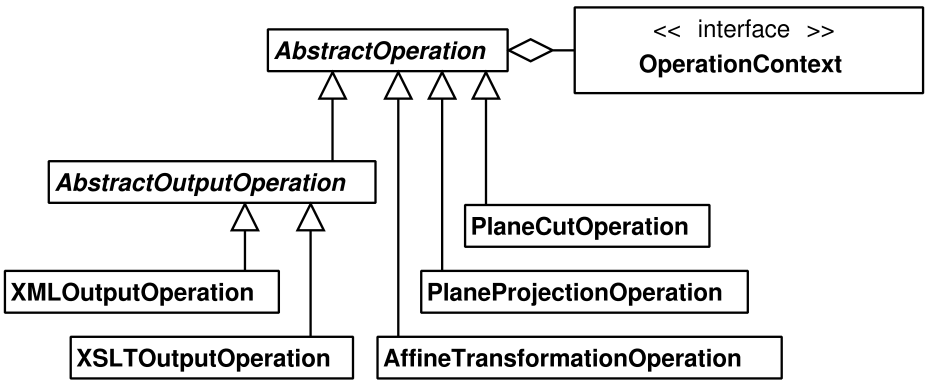

Abbildung 4.9: Operationenmodell mit Beispiel-Operationen in UML-Notation

entscheidende Nachteile. Zum einen ist eine Implementierung eng mit dem Datenbanksystem verbunden. Erfolgt beispielsweise eine Änderung in der Implementierung des Datenbankmodells, müssen alle Datenformatimplementierungen angepasst werden. Zum anderen hat der Nutzer eines solchen Systems keinerlei Einflußnahme auf unterstützte Datenformate. DB3D beinhaltet deshalb zusätzlich die Möglichkeit Formate durch XSL-Transformation zu unterstützen.

Implementierung Die Implementierung von Formaten beschränkt sich auf essentielle Datenformate bzw. auf durch XSL-Transformation nicht realisierbare Datenformate. Derzeitige Implementierungen existieren für den Austausch mit dem 3D-Modellierungsprogramm GOCAD [78] (GOCAD Ascii) und für das eigene XML-Format, das die Basis für die Unterstützung von XSL-Transformationen darstellt. Die Implementierungsmethode wird immer für die Unterstützung binärer Datenformate benötigt. Weiterhin können z.B. unstrukturierte Ascii-Formate nicht über XSLT verarbeitet werden und müssen somit über eine Implementierung zumindest als Eingabeformat Unterstützung finden.

XSL-Transformation Die Methode der XSL-Transformation ermöglicht die offene Unterstützung für beliebige XML/Ascii-Formate bei der Ausgabe und für XML-Formate bei der Eingabe. Benutzer haben damit die Möglichkeit, eigene Datenformate durch selbst erstellte XSLT-Skripte zu unterstützen und als Eingabe-/Ausgabeformat zu verwenden. Hierzu werden spezielle XSLT-Dienste (z.B. *XSLTOutputOperation*) bereitgestellt, die mit einem vom Nutzer zur Verfügung gestellten XSLT-Skript arbeiten. Derzeit erfolgt die Unterstützung von VRML 2.0 (Ausgabe) und X3D (Ein-/Ausgabe) über XSL-Transformation.

4.3.2 Geometrieoperationen

Jede Geometrieoperation stellt genau eine definierte Datenbankoperation bereit. Mit Hilfe der Geometrieoperationen können z.B. topologische Beziehungen zwischen 3D-Objekten oder das Schnittergebnis zwischen zwei Volumen-Objekten berechnet werden. Jede Geometrieoperation wird als einzelner Dienst über den in Kapitel 4.4 beschriebenen Dienst-Framework Nutzern zur Verfügung gestellt. Die im folgenden beschriebenen Operationen sind Bestandteil des im späteren Verlauf vorgestellten zusammengesetzten Anwendungsdienstes 3Dto2D-Dienst.

PlaneCutOperation Die Operation *PlaneCutOperation* berechnet das Schnittresultat einer gegebenen Anzahl von 3D-Geoobjekten mit einer Ebene. Die Eingabe besteht aus einem oder mehreren Objekten und der beliebig im Raum liegenden Ebene, das Resultat sind Ergebnisobjekte vom Typ ResultObject3D. Für jedes Eingabeobjekt wird, wenn es sich mit der Ebene schneidet, ein Ergebnisobjekt erzeugt. Dieses Ergebnisobjekt beinhaltet eine Anzahl von elementaren Geometrieobjekten. Beim Schnitt eines simplizialen 2-Komplexes mit einer Ebene kann dieses Ergebnis aus Point3D, Segment3D und Triangle3D Objekten bestehen. Ergebniselemente vom Typ Point3D und Triangle3D entstehen, wenn Simplexe des ursprünglichen Komplexes an einem Knoten die Ebene schneiden bzw. ein Simplex auf der Ebene liegt. Werden im weiteren Verlauf nur *reguläre* Ergebniselemente (Segment3D-Objekte) benötigt, so kann das Ergebnisobjekt z.B. durch eine *SimplexFilterOperation* gefiltert werden.

PlaneProjectionOperation Die Operation *PlaneProjectionOperation* projiziert alle Simplexe eines 3D-Geoobjektes auf die gegebene Ebene. Die Eingaben zur Operation und die Ausgaben sind die gleichen wie bei der PlaneCutOperation. Diese Operation projiziert rein geometrisch und nicht in einem kartographischen Sinne, so daß keine Verdeckungsberechnungen erfolgen.

AffineTransformationOperation Die *AffineTransformationOperation* stellt die Operation einer beliebigen affinen Transformation zur Verfügung. Eingabe ist hier eine beliebige Anzahl von 3D-Geoobjekten und eine Transformationsmatrix, die Ausgabe sind die Ergebnisobjekte.

4.4 Dienst-Framework

Der Dienst-Framework stellt den Zugriff auf das Datenbanksystem und dessen Funktionalität über eine Objekt zu Objekt Kommunikation bereit. Der Framework ist auch von

mobilen Geräten mit geringen Ressourcen nutzbar. Um eine Suche nach verfügbaren Datenbanken und Datenbankdiensten zu ermöglichen, wird auf die vorhandene Funktionalität des Jini Lookup-Services [123] zurückgegriffen. Dieser Registrierungsdienst ermöglicht die Suche nach publizierten Datenbanken, ihren Diensten und deren Bezug.

4.4.1 Architektur

Neben einem *ConnectionService* für den Aufbau und den Abbruch einer Verbindungssession mit dem Datenbanksystem stehen auf dem Jini Lookup-Service die verschiedenen Dienste für die Funktionalitäten des Datenbanksystems bereit. Diese Dienste können nach erfolgreichem Einloggen bezogen werden. Dabei wird dem Nutzer ein Proxy für den jeweiligen Dienst ausgegeben, der vor der erstmaligen Benutzung aktiviert werden muß. Abbildung 4.10 zeigt in einem Sequenzdiagramm die Abfolge der Benutzung des Dienst-Frameworks vom Zeitpunkt des Einloggens über den Bezug und die Aktivierung eines Dienstes bis zum Ausloggen aus dem Datenbanksystem.

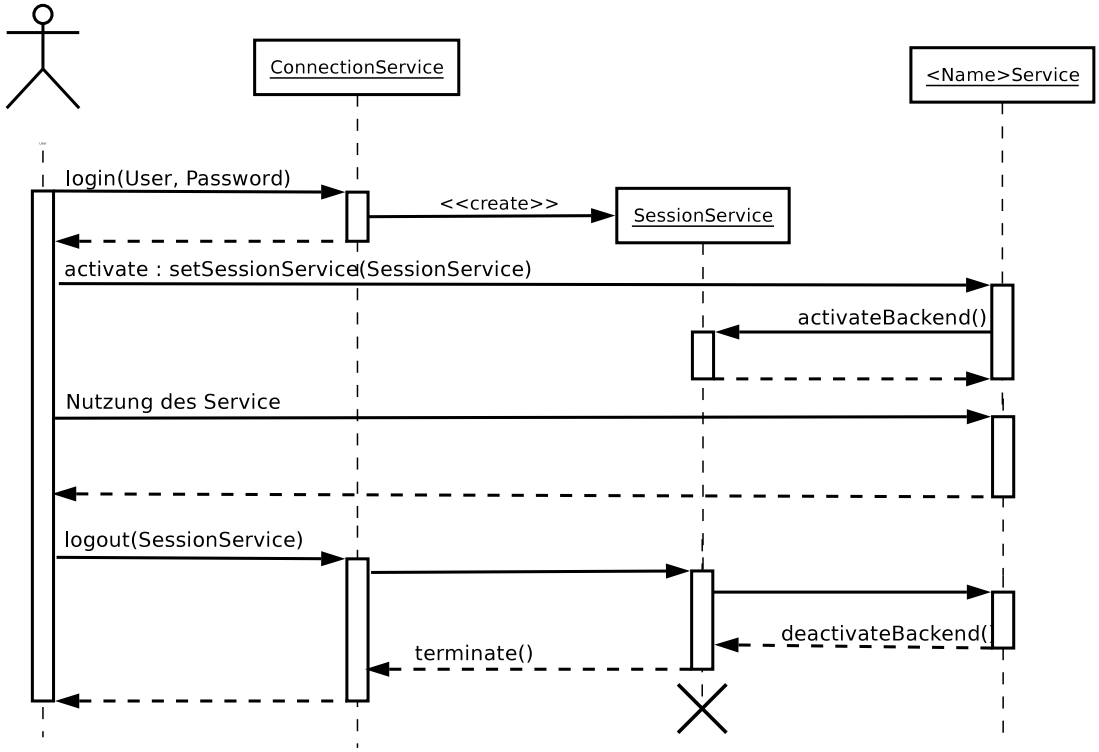

Abbildung 4.10: Sequenzdiagramm der Benutzung des Dienst-Frameworks

Grundlage der Benutzung ist eine etablierte Verbindung zum Jini Lookup-Service. Dieser Dienst stellt für die verschiedenen Datenbanken des Datenbanksystems je einen Connec-

tionService zur Verfügung. Der Nutzer holt sich den gewünschten ConnectionService und loggt sich mit den Zugangsdaten ein. Bei korrekten Zugangsdaten wird ein *SessionService* für die dadurch aufgebaute Datenbanksession des Benutzers erzeugt und als Rückgabe der Login-Prozedur an den Nutzer zurückgegeben. Dieser SessionService übernimmt während der gesamten Laufzeit der Datenbanksession die Aktivierung der vom Benutzer über den Jini Lookup-Service bezogenen Datenbankdienste. Diese als Dienst-Proxy bezogenen Datenbankdienste werden durch Übergabe des SessionService aktiviert. Bei dieser Aktivierung werden vom SessionService die Zugriffsrechte des Nutzers auf diesen Dienst überprüft und das zugehörige Dienst-Backend auf dem Server zum Zugriff freigeschaltet. Nach dieser Aktivierung kann der Dienst benutzt werden (siehe Abbildung 4.10: Nutzung des Dienst-Frameworks). Diese Prozedur muß für jeden Dienst innerhalb einer Datenbanksession einmalig erfolgen. Die Beendigung der Datenbanksession erfolgt durch Ausloggen beim ConnectionService. Dieser sendet eine Terminierungsnachricht an den SessionService, welcher wiederum alle aktivierten Dienst-Backends auf dem Server terminiert und die gesamten Ressourcen freigibt. Durch die notwendige einmalige Aktivierung jedes Dienstes werden keine unnötigen Ressourcen des Servers für nicht benutzte Dienste belegt. Weiterhin ist auf diesem Wege eine Rechtevergabe für die Dienste möglich, wodurch für bestimmte Benutzer einzelne Dienste (z.B. Administrationsdienste) gesperrt werden können.

Alle während einer Datenbanksession aktivierten Dienste greifen auf die gleichen Ressourcen des Datenbanksystems zu. Um dies zu ermöglichen, muß die Entwicklung von Diensten nach einer vorgegebenen Struktur erfolgen. Abbildung 4.11 zeigt ein Klassendiagramm der allgemeinen Struktur der Dienste mit den benötigten Schnittstellen und abstrakten Implementierungsklassen.

Alle Dienste bestehen aus einem Client- und Server-Teil. Die Grundfunktionalität für Sessionmanagement und Aktivierung von Diensten wird durch die Interfaces und abstrakten Klassen *CommonService*, *CommonServiceProxy*, *CommonServiceBackend* und *CommonServiceBackendImpl* bereitgestellt. Jeder Dienst muß diese abstrakten Klassen erweitern, um die benötigte Grundfunktionalität zu erhalten.

Der Client-Teil wird dabei von dem Interface CommonService bereitgestellt. Dieses Interface definiert die gemeinsame Methode zum Aktivieren eines Dienstes und CommonServiceProxy stellt hierfür die notwendige Implementierung bereit. Auf der Serverseite stellt das Interface CommonServiceBackend gemeinsame Methoden zur Verfügung. Die Implementierung durch CommonServiceBackendImpl ermöglicht den Zugriff für alle Dienste auf einen gemeinsamen *SessionContext*, der den Zugriff auf das Datenbanksystem erlaubt.

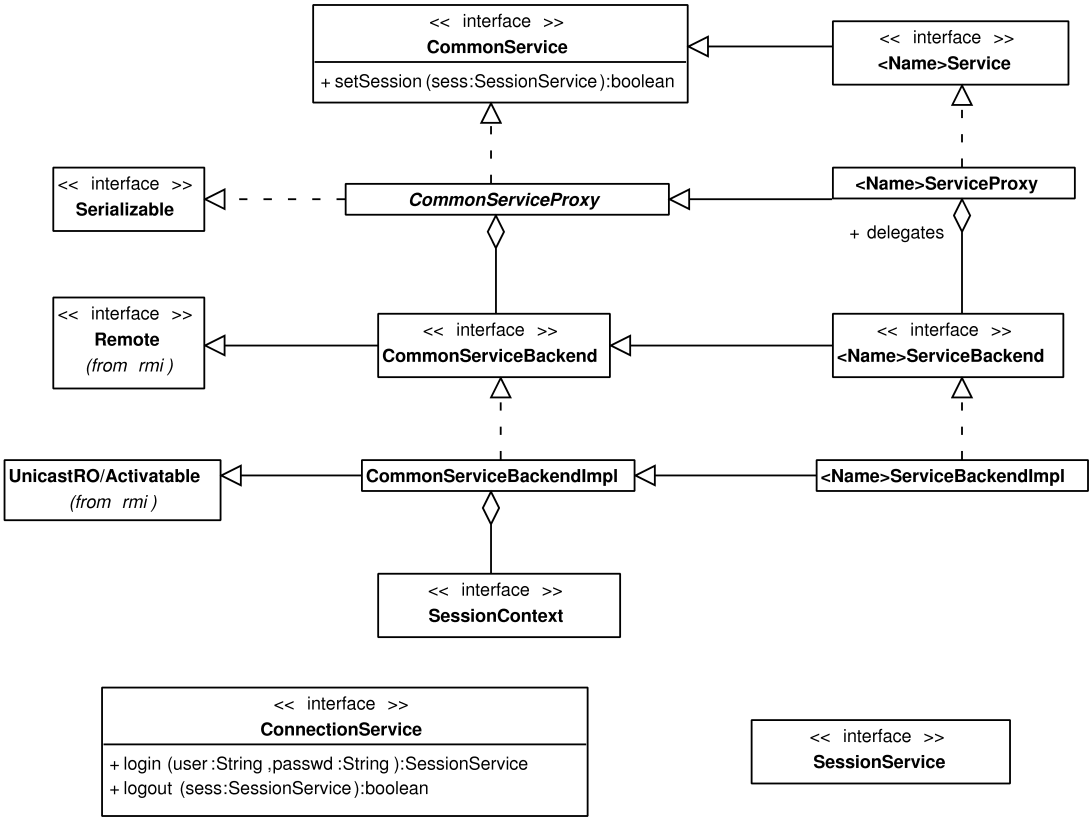

Abbildung 4.11: Aufbau eines Dienstes nach der Dienststruktur

4.4.2 Atomare Dienste

Die einzelnen Dienste des Datenbanksystems sind als atomare Dienste anzusehen, die jeweils nur eine eindeutige Operation des Datenbanksystems bereitstellen. Erst die Ausführung einer Abfolge einzelner Dienste führt zu einer komplexen Funktionalität in der Datenbank.

Die atomaren Dienste laufen als abgeschlossene Transaktionen im Datenbanksystem ab. Dienste, die neue Datenbankobjekte bzw. Zwischenergebnisse erzeugen (z.B. bei Schnittoperationen), speichern diese Ergebnisse in dem der Session zugeordneten Workspace. Die Übergabe von Daten zwischen den atomaren Diensten erfolgt durch die Bereitstellung von direkten Zugriffsadressen. Jeder atomare Dienst erwartet als Eingabeparameter für zu verarbeitenden Daten die jeweiligen Zugriffsadressen und gibt als Ergebnis die Zugriffsadressen der Ergebnisse zurück. Auf diese Weise erfolgt bei dem sequentiellen Aufruf mehrerer Dienste keine unnötige Datenübertragung zwischen Client und Server. Es werden nur die jeweiligen Zugriffsadressen zwischen Client und Server ausgetauscht und erst das Endergebnis einer Menge von Dienstaufrufen vom Server zum Client übertragen. Die

transaktionelle Verarbeitung der atomaren Dienste ermöglicht es, daß einzelne Dienstabbrüche in einer Abfolge von Diensten nicht die komplette Neuausführung der Dienstkette erforderlich macht. Es reicht, wenn die Dienstkette ab dem abgebrochenen Dienst wieder ausgeführt wird.

Im folgenden werden einige wichtige Dienste vorgestellt, bevor im Anschluß auf einen Anwendungsdienst eingegangen wird.

Retrieve-Dienst: Der Retrieve-Dienst selektiert Datenbankobjekte über ihre ID oder räumliche Anfragen in der Datenbank. Er liefert als Ergebnis die direkten Zugriffsadressen der gefundenen Datenbankobjekte. Diese können dann als Eingabe für weitere Dienste zur Prozessierung oder Ausgabe benutzt werden.

Operationen-Dienste: Dienste für Datenbankoperationen nehmen als Eingabe eine Menge von Zugriffsadressen von Datenbankobjekten entgegen, die beispielsweise durch den Retrieve-Dienst bestimmt werden. Die Dienste führen für diese Menge von Objekten die jeweilige Operation aus und legen die Ergebnisobjekte in dem der Datenbanksession zugeordneten Workspace ab. Rückgabe der Operationen-Dienste sind die Zugriffsadressen der Ergebnisobjekte. Diese dienen als Eingabe für weitere Operationen-Dienste oder Ausgabe-Dienste.

Ausgabe-Dienste: Die Ausgabe-Dienste geben die über Zugriffsadressen aus vorangegangenen Diensten spezifizierten Datenbank- oder Ergebnisobjekte in ihrem Datenformat aus. Zur Zeit existieren Ausgabe-Dienste für die XML-, VRML-, X3D- und GOCAD/Ascii-Ausgabe.

4.4.3 Verkettung von Diensten

Die atomaren Dienste – bis auf die Ausgabe-Dienste – legen ihre Ergebnisobjekte im Workspace ab und geben nur die Zugriffsadressen der Objekte weiter. Dies ermöglicht eine effiziente Verkettung von Diensten, ohne daß ein Datenaustausch zwischen Client und Server stattfinden muß. Als Beispiel für eine Verkettung atomarer Dienste zu einem komplexen Anwendungsdienst wird hier der 3Dto2D-Dienst vorgestellt.

Anwendungsdienst 3Dto2D-Dienst: Innerhalb des Datenbanksystems werden ausschließlich 3D-Objekte bzw. 3D-Modelle verwaltet. In geologischen Anwendungen werden jedoch häufig beliebige 2D-Schnittdarstellungen durch ein 3D-Modell zur Interpretation benötigt. Ebenfalls sind viele mobile Endgeräte derzeit nicht in der Lage 3D-Objekte zu visualisieren. Ein Dienst zur Berechnung beliebiger 2D-Schnittdarstellungen eröffnet

auch auf diesen eingeschränkten Geräten die Möglichkeit der Nutzung von 3D-Daten im Gelände. Der 3Dto2D-Dienst stellt die notwendige Funktionalität zur Berechnung von Schnittdarstellung mit zusätzlicher Projektion benachbarter Geoobjekte auf das resultierende 2D-Profil zur Verfügung. Der Dienst setzt sich aus den folgenden atomaren Diensten zusammen:

- Retrieve-Dienst - Auswahl der Objekte des 3D-Modells.
- PlaneCut-Dienst - Operation für Ebenenschnitt durch Objekte.
- BufferCut-Dienst - Filterungsoperation für Objekte nach Distanzkriterium.
- PlaneProjection-Dienst - Projektionsoperation für Objekte.
- AffineTransformation-Dienst - Affine Transformationsoperation für Objekte.
- GMLOutput-Dienst - Ausgabeoperation für GML Format.

Das berechnete Ergebnis ist eine Karte in der xy-Ebene. Für die Weiterverarbeitung der 2D-Daten existiert ein Ausgabe-Dienst, der das resultierende Ergebnis in das GML Format überführt. Abbildung 4.12 zeigt den schematischen Ablauf des Dienstes. Im linken Bildteil ist ein 3D-Modell mit zwei Bohrprofilen b1 & b2, sowie der Lage des Ebenenschnittes zwischen den Punkten A & B zu sehen. In der Mitte ist das 3D-Modell mit Sicht auf das gewünschte Profil dargestellt. Der rechte Bildteil stellt das erwartete Ergebnis des Dienstes dar – ein Profilschnitt in der xy-Ebene mit den beiden projizierten Bohrprofilen als zusätzliche Informationen.

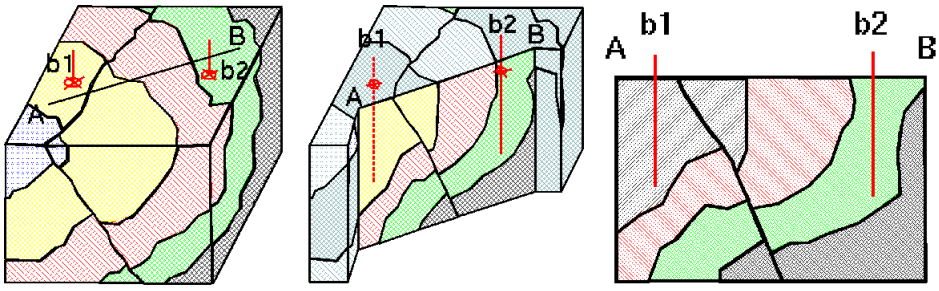

Abbildung 4.12: Prinzipieller Ablauf des 3Dto2D-Dienstes

Abbildung 4.13 zeigt einen Ausschnitt des 3D-Modells der Schaumburg-Lippeschen Kreidemulde in Niedersachsen, Datensatz Porta[1] (siehe auch Projektbeschreibung in Kapitel

[1]erstellt von Seidemann/Kroll unter Anleitung von A. Siehl, Institut für Geologie, Universität Bonn

3.3). Das 3D-Untergrundmodell besteht aus mehreren Schichtflächen und einigen Störungsflächen. Die Darstellung ist 3-fach überhöht und enthält die Lagen von aufeinanderfolgenden versetzten Ausführungen des 3Dto2D-Dienstes. Das Ergebnis eines der berechneten Profilschnitte auf diesem Datensatz zeigt Abbildung 4.14. Auf der Abbildung sind der Schnitt durch das Oberflächengeländemodell, sowie die Schichtflächen und einige der Störungsgrenzen zu sehen.

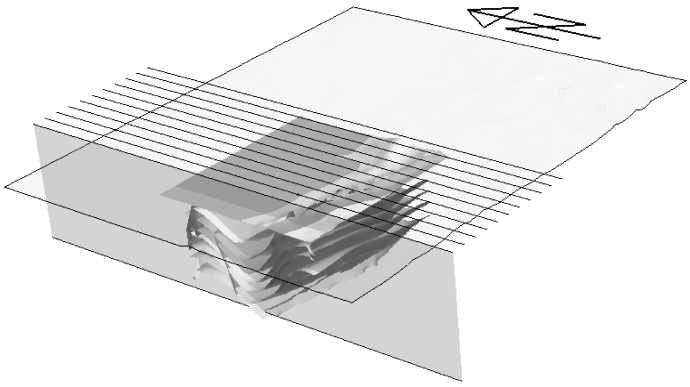

Abbildung 4.13: Porta Datensatz mit Lageangabe der berechneten Profilschnitte

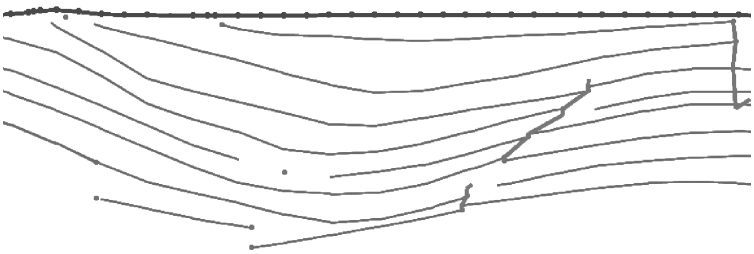

Abbildung 4.14: Ausschnitt eines berechneten Profilschnittes durch den Porta Datensatz

Kapitel 5

Erweiterung um ein Versionsmanagement zur Nutzung mit mobilen Datenbanken

Im folgenden wird ein Versionsmanagement für geowissenschaftliche 3D-Datenbanksysteme vorgestellt. Durch die Verwendung von Versionsmanagement wird eine *Reintegration* der auf dem mobilen Gerät in der Datenbank geänderten Geodaten als neue Versionen in das Server-Datenbanksystem ermöglicht.

Neben der Definition grundlegender Begriffe wird auf die Integrationsmöglichkeiten von Versionsmanagement in Datenbanksystemen eingegangen. Der Schwerpunkt liegt in der Beschreibung des konzipierten Versionsmodells, der umgesetzten Repräsentation der geowissenschaftlichen Objekte sowie der notwendigen Operationen und Algorithmen für das Versionsmanagement.

5.1 Definitionen

Die hier vorgestellten Definitionen für das Versionsmanagement basieren auf den bereits vorgestellten Konzepten von DesignObjekt, Version und Workspace in Kapitel 2.3. In Analogie zu dem bereits existierenden Workspace-Konzept im Datenbanksystem DB3D wird im weiteren Verlauf der Begriff *VersionWorkspace* für die Workspaces des Versionsmanagements verwendet.

DesignObjekt (DO) Das DesignObjekt entspricht dem semantischen Teil eines versionierbaren Objektes. Es enthält die von der Versionsbildung unabhängigen Bestandteile

(versionsunabhängige Attribute) des Objektes. Ist das DesignObjekt ein komplexes DesignObjekt, d.h. aus mehreren weiteren DesignObjekten aufgebaut, so enthält es den Aufbau aus diesen DesignObjekten als Beziehungen.

Komplexes DesignObjekt Ein DesignObjekt, das aus mehreren DesignObjekten aufgebaut ist. Hier sind komplexe, flache und komplexe, verschachtelte DesignObjekte zu unterscheiden. Unter komplexen, flachen DesignObjekten werden DesignObjekte verstanden, die selbst aus nicht komplexen DesignObjekten aufgebaut sind. Demgegenüber sind komplexe, verschachtelte DesignObjekte aus ebenfalls komplexen DesignObjekten aufgebaut.

Version (V) Die Versionen stellen den veränderlichen Teil eines versionierbaren Objektes dar. Sie stellen dabei eine Ausprägung des DesignObjektes zu einem bestimmten Zeitpunkt der Erzeugung dar. Versionen können damit Nachfolger existierender Versionen sein bzw. als Nachfolger für weitere Versionen dienen. Lediglich die erste Version (initiale Version) eines DesignObjektes besitzt keinen direkten Vorgänger. Es sind prinzipiell die Beziehungen *revision*, *alternative* und *merge* zwischen Versionen möglich (siehe Kapitel 2.3).

VersionWorkspace (VWS) Ein VersionWorkspace stellt den Verarbeitungsraum von DesignObjekten und ihren Versionen dar. VersionWorkspaces sind für die Gruppierung von Arbeiten unter verschiedenen Aspekten zuständig. So können VersionWorkspaces beispielsweise unterteilt werden nach dem Bearbeitungsstand (z.B. Initial, In Work, Release), bzw. zur Einteilung von Arbeitsgruppen oder zur Zugriffsregelung benutzt werden. VersionWorkspaces sind hierarchisch organisiert mit einem Root-VersionWorkspace. Jeder VersionWorkspace besitzt dabei genau einen Vorgänger-VersionWorkspace und kann mehrere Nachfolger-VersionWorkspaces beinhalten.

Versionen können zwischen VersionWorkspaces migriert werden. DesignObjekte sind in einem VersionWorkspace sichtbar, sobald eine ihrer Versionen im VersionWorkspace enthalten ist. Abbildung 5.1 zeigt die prinzipiellen Beziehungen zwischen DesignObjekt, Version und VersionWorkspace.

5.2 Integration des Versionsmanagements

Das Versionsmanagement kann sowohl direkt in das Datenmodell eines Datenbanksystems integriert oder basierend auf einem Datenmodell aufgesetzt verwirklicht werden. Hierbei bieten beide Integrationsarten einige Vorteile und Nachteile in Bezug auf die Flexibilität, die Performanz und die Anwendbarkeit für bestehende Systeme.

Direkte Integration: Bei der direkten Integration wird das Versionsmanagement im zugrundeliegenden Datenmodell des Datenbanksystems verwirklicht. Datenbankobjekte

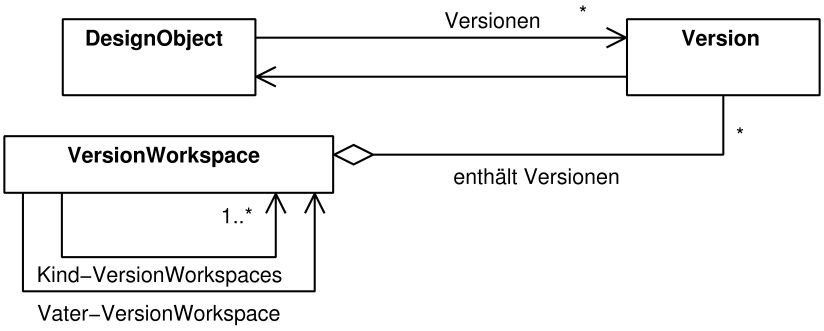

Abbildung 5.1: Beziehungen zwischen DesignObjekt, Version und VersionWorkspace

sind damit grundsätzlich versionierbar und es stehen Konstrukte zur Versionserzeugung, -verwaltung und -löschung zur Verfügung.

Vorteile der direkten Integration liegen vor allem in der Verwendung von Versionen als elementare Datenbankobjekte in sämtlichen Ebenen des Datenbanksystems. Operationen und Anfragen im Datenbanksystem sind damit auf Versionen der Datenbankobjekte möglich. Aus der Sicht bestehender Systeme sind für eine solche Unterstützung jedoch grundlegende Änderungen auf allen Datenbankebenen nötig. Eine direkte Integration ist somit nicht möglich, wenn das bestehende Datenmodell unverändert bleiben soll bzw. muss. Ein weiterer Nachteil ist der vorhandene Overhead für die Versionierung auch bei Datenbankobjekten, für die keine Versionen benötigen werden.

Aufsetzende Integration: Bei der aufsetzenden Integration basiert das Versionsmanagement auf dem bestehenden Datenmodell. Die Versionsverwaltung erfolgt als eigene Ebene auf (*on-top*) dem Datenbanksystem. Es existieren zusätzliche Dienste für die Verwaltung und Anfrage von Versionen von Datenbankobjekten. Eine Konvertierungsmöglichkeit zwischen Versionsmodell und existierendem Datenmodell ermöglicht die Verwendung der vorhandenen Datenbankfunktionalität auch für die Versionen von Datenbankobjekten.

Der entscheidende Vorteil der aufsetzenden Integration liegt darin, daß das bestehende Datenbanksystem nicht verändert werden muß. Damit ist sowohl der weitere Betrieb des Datenbanksystems im *normalen* Modus als auch als Datenbanksystem mit Versionsmanagement möglich. Existierende Anwendungen, die auf dem Datenbanksystem aufbauen, bleiben weiterhin funktionsfähig. Anwendungen, die das Versionsmanagement nutzen wollen, müssen diese Neuerungen implementieren bzw. mit den neu zur Verfügung gestellten Diensten für Versionsmanagement zusammenarbeiten können.

Die vorliegende Arbeit verfolgt den Ansatz der *aufsetzenden Integration* im Versionsmodell und der prototypischen Implementierung. Entscheidend hierfür ist, daß keine Änderungen

im Datenbanksystem auf der internen Ebene erfolgen müssen. Hierdurch wird eine getrennte Weiterentwicklung und Wartbarkeit des Datenbanksystems und der Erweiterung um ein Versionsmanagement ermöglicht. Die Unterstützung von Konvertierungen zwischen Versionsmodell und Datenmodell des Datenbanksystems ermöglicht eine vollständige Verarbeitung von Versionen von Datenbankobjekten in den implementierten Operationen des Datenbanksystems.

5.3 Komplexe versus einfache DesignObjekte

Wie in Kapitel 4.2.3 beschrieben, wird im Datenbanksystem DB3D als Datenmodell für 3D-Objekte das Simpliziale d-Komplex Modell verwendet. Ein 3D-Geoobjekt setzt sich damit aus einfacheren 3D-Geometrien zusammen, die miteinander in Nachbarschaftsbeziehungen stehen. Wie im Kapitel 2.3 zu den Versionsmodellen bereits ausgeführt wurde, bedingt die Verwaltung von Versionen komplexer DesignObjekte das Problem der Konfiguration von komplexen DesignObjekten. Im folgenden wird diskutiert, ob für den komplexen Aufbau der Datenbankobjekte komplexe DesignObjekte benötigt werden, oder ob auch ein einfaches DesignObjekt-Modell verwendet werden kann.

5.3.1 Repräsentation als komplexes DesignObjekt

Im folgenden wird die Repräsentation von Datenbankobjekten als komplexe DesignObjekte beschrieben und diskutiert. Als Grundlage wird in Abbildung 5.2 der Aufbau der Klasse Object3D für einen simplizialen 2-Komplex vereinfacht dargestellt.

Die Klassen, deren Operationen zu Änderungen am Objekt und damit zu einer neuen Version führen, stellen die für das Versionsmanagement entscheidenden Klassen dar. Jede dieser Klassen wird im Versionsmanagement durch eine DesignObject-Subklasse und eine Version-Subklasse repräsentiert. In Abbildung 5.2 können mehrere solcher Klassen identifiziert werden. Im thematischen Teil stellt die Klasse *ThematicObject3D* Operationen zum Hinzufügen, Ändern und Löschen von Attributen für das gesamte Objekt und die einzelnen Simplexe bereit. Im geometrischen Objektteil sind zwei Klassen relevant. Die Klasse *TriangleNet3D* stellt Operationen zum Hinzufügen, Löschen und Aufteilen ihrer Netzkomponenten zur Verfügung, während die Klasse *TriangleNet3DComp* die Operationen zum Ändern der einzelnen Netzkomponenten bereitstellt. Die anderen Klassen stellen die Einheiten der Änderungen dar. So ist die Klasse *TriangleElt3D* beispielsweise die Einheit der Änderung bei Netzkomponenten. Instanzen dieser Klasse können zu Netzkomponenten hinzugefügt oder entfernt werden. Abbildung 5.3 zeigt eine Übersicht dieser Klassen und des entsprechenden komplexen DesignObjektes mit den Versionsklassen. Zur Vereinfachung werden die Aggregationsklassen und die implementierten Interfaces weggelassen.

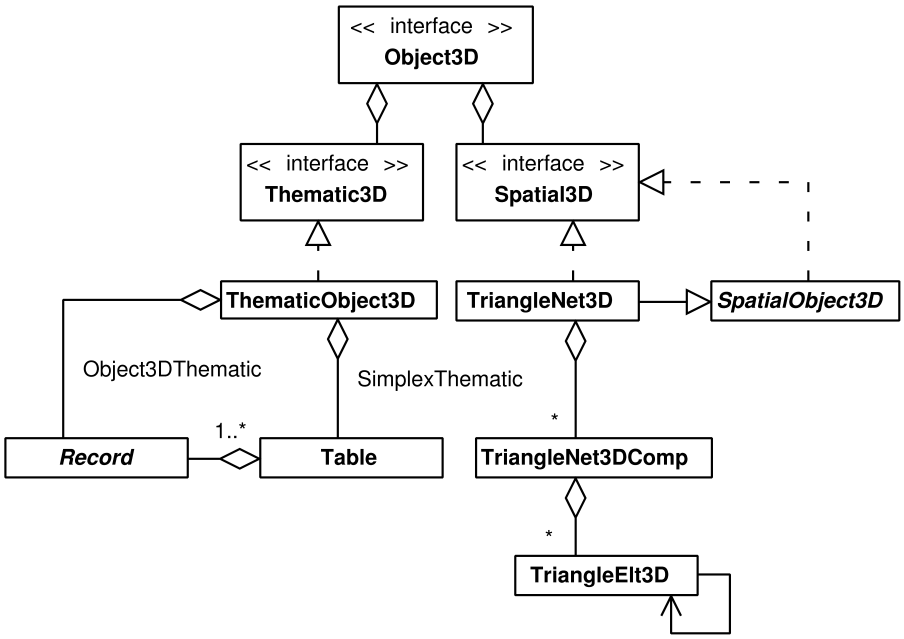

Abbildung 5.2: Aufbau von Object3D für simplizialen 2-Komplex (vereinfacht)

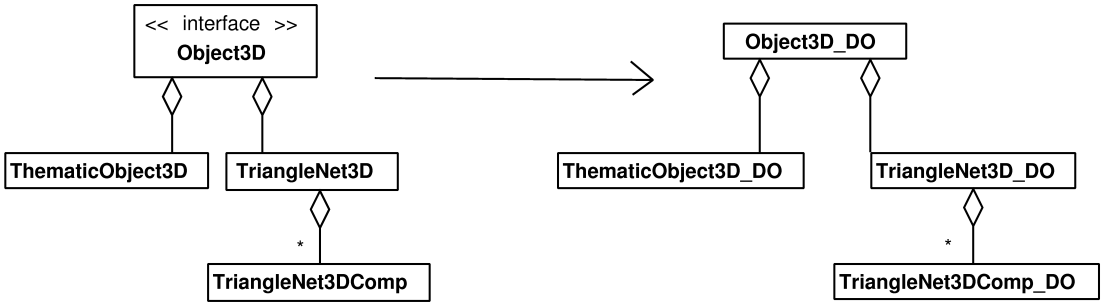

Abbildung 5.3: Bildung eines komplexen DesignObjektes am Beispiel Object3D

Zusätzlich zu der Zusammensetzung von Object3D aus mehreren Klassen (Aggregation) sind auch die Assoziationsbeziehungen zwischen den Teilobjekten zu berücksichtigen. Beispielsweise enthält ThematicObject3D neben den thematischen Daten für das gesamte Objekt auch die Sachdaten für alle einzelnen Simplexe. Die Verbindung ist dabei über die Simplex-ID realisiert, die eindeutig im gesamten Netz ist. Innerhalb eines komplexen DesignObjektes müssen auch diese Beziehungen berücksichtigt werden. Im vorliegenden Beispiel bedingt dies eine Propagierung von Simplex-Änderungen zum ThematicObject3D.

Durch die Zusammensetzung eines komplexen DesignObjektes aus mehreren weiteren DesignObjekten müssen bei der Bildung einer neuen Version in diesen DesignObjekten auch

neue Versionen der übergeordneten DesignObjekte gebildet werden, die diese neue Version einbeziehen (Konfiguration).

Das Problem komplexer DesignObjekte wird anhand der Abbildung 5.4 beispielhaft verdeutlicht. Die Abbildung stellt ein Object3D mit zwei Netzkomponenten dar. Anhand einer einzelnen Änderungsoperation in dem DesignObjekt TriangleNet3DComp_DO-2 werden die Auswirkungen und die damit verbundenen Versionsneubildungen gezeigt. Im Beispiel wird das DesignObjekt TriangleNet3D_DO weggelassen und angenommen, Object3D_DO beinhalte direkt die Netzkomponenten. Hierdurch ergibt sich eine einfaches komplexes, flaches DesignObjekt für die Diskussion und eine damit verbundene übersichtliche Darstellung. Die Realität für Object3D würde jedoch ein komplexes DesignObjekt bedeuten, das selbst aus komplexen DesignObjekten (TriangleNet3D_DO) besteht und somit noch deutlich komplexer ist.

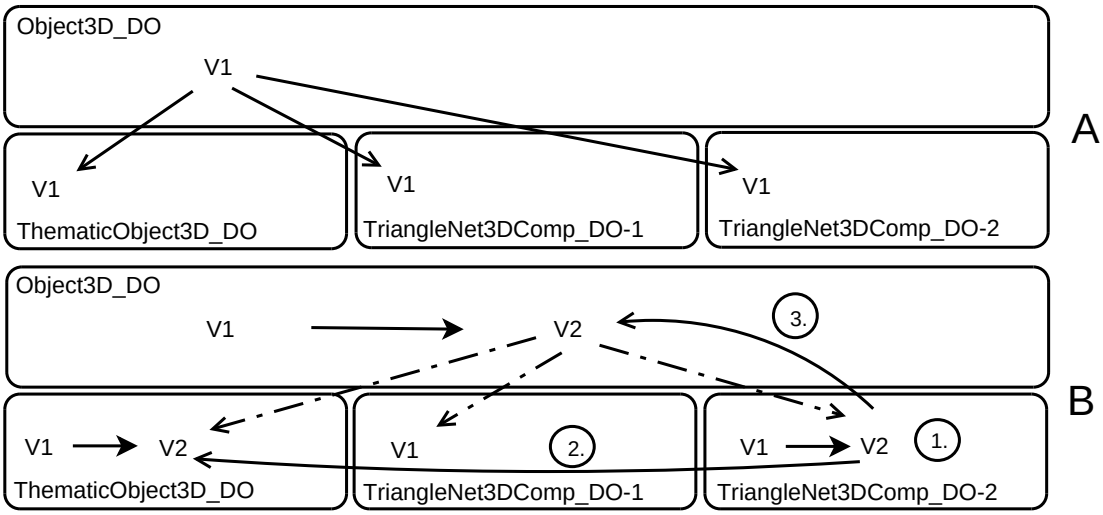

Abbildung 5.4: Beispiel für die Auswirkungen einer Änderungsoperation auf die Versionierung bei einem komplexen DesignObjekt am Beispiel vom (vereinfachten) komplexen Object3D_DO.

In Abbildung 5.4 stellt V1 die Version vor und V2 die Version nach der Änderungsoperation dar. Die Zusammensetzung der Object3D_DO Versionen aus den Versionen seiner DesignObjekte ist durch Pfeilsymbole gekennzeichnet.

A: Anfangsversion V1 mit den zugehörigen Versionen V1 seiner DesignObjekte.

B: Löschung eines Simplexes in der Netzkomponente TriangleNet3DComp_DO-2:

1. Löschung Simplex → Bildung Version V2 mit gelöschtem Simplex.

2. Propagierung der Löschung zu ThematicObject3D_DO → Bildung der Version V2 mit gelöschter Thematik des Simplex.
3. Propagierung der Löschung zu Object3D_DO → Bildung der neuen Version V2 mit Verweisen auf die jeweiligen Versionen von ThematicObject3D_DO und TriangleNet3DComp_DO-2.

Dieses Beispiel verdeutlicht, wie einfache Änderungen an den untergeordneten DesignObjekten, die dort neue Versionen erzeugen, in den übergeordneten DesignObjekten ebenfalls neue Versionen erzwingen. Diese neuen Versionen beinhalten oft keine eigentlichen Änderungen, sondern nur die Aktualisierung der Verweise auf die Versionen der untergeordneten DesignObjekte. Weiterhin erzwingt die Beziehung zwischen den Simplexen und ihrer Thematik eine Versionsbildung in dem DesignObjekt ThematicObject3D_DO, wenn Simplexe gelöscht oder hinzugefügt werden. Wird nicht von der Annahme eines vereinfachten komplexen, flachen DesignObjekts ausgegangen, so ergeben sich noch weitere Versionsbildungen. Bei jeder Änderung in einer Netzkomponente erfolgen neben der Bildung einer neuen Version auch Versionsbildungen des übergeordneten Netz-DesignObjekts und des diesem übergeordneten Object3D-DesignObjekts.

5.3.2 Repräsentation als einfaches DesignObjekt

Die geschilderten Probleme bei der Verwendung komplexer DesignObjekte zur Repräsentation der Datenbankobjekte machen es naheliegend keine komplexen DesignObjekte zu verwenden. Bei der Repräsentation als einfache DesignObjekte kann nur ein einziges DesignObjekt Object3D_DO, ohne untergeordnete DesignObjekte, existieren (siehe Abbildung 5.5).

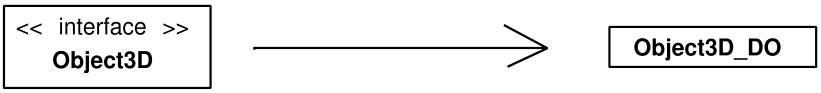

Abbildung 5.5: Bildung eines einfachen DesignObjektes am Beispiel Object3D

Hierfür muß das DesignObjekt Object3D_DO alle im normalen Datenmodell vorhandenen Operationen zum Ändern von Object3D-Objekten und damit zur Erzeugung neuer Versionen bereitstellen. Diese Operationen müssen die jeweiligen Beziehungen, die im vorangegangenen Abschnitt vorgestellt wurden, in ihrer Umsetzung berücksichtigen.

Ein DesignObjekt Object3D_DO muß Operationen für folgende Änderungen bereitstellen:

- Hinzufügen/Löschen einer Netzkomponente.
- Hinzufügen/Löschen eines Simplex zu/von einer Netzkomponente.

- Hinzufügen/Löschen/Ändern der Thematik des Objektes.
- Hinzufügen/Löschen/Ändern der Thematik eines Simplexes.

Das Hinzufügen eines Simplexes zu einer Netzkomponente kann dabei kombiniert mit dem Hinzufügen der Thematik eines Simplexes erfolgen. Das Löschen einer gesamten Netzkomponente bzw. eines Simplexes aus einer Netzkomponente muß das entsprechende Löschen der Simplexthematik(en) auslösen. Die Operationen zum Ändern der Thematik des Objektes bzw. der Simplexe kann prinzipiell in eine Kombination von Löschen und Hinzufügen überführt werden, so daß hier keine extra Implementierung nötig ist.

5.4 Versionsmodell

Das im folgenden ausgeführte Versionsmodell zur Verwaltung von DesignObjekten und ihren Versionen wurde ausgehend von dem durch Schönhoff in [115] beschriebenen Versionsmodell entworfen. Im Gegensatz zu dem Modell von Schönhoff, das auf den Bereich des CAD/CAM ausgelegt ist, bezieht sich das folgende Versionsmodell auf den geowissenschaftlichen Anwendungsbereich. Hauptanforderung hier ist eine Unterstützung von Modellierungsschritten sowie konkurrierende Entwicklungslinien/Updates durch ein Versionsmanagement. Anforderungen aus dem CAD/CAM-Bereich, wie zum Beispiel die Unterstützung von komplexen DesignObjekten, werden im geowissenschaftlichen Modell nicht dringlich benötigt (vgl. Diskussion im Kapitel 7).

Die Beschreibung des Versionsmodells geht zuerst auf die interne Verwaltung der Historie von DesignObjekten ein. Diese Modellierung erfolgt durch das History-Modell und wird vom Versionsmanagement zur Verwaltung der Beziehungen zwischen den Versionen eines DesignObjektes benutzt. Aufbauend darauf erfolgt die Vorstellung der Konzepte für die Modellierung von DesignObjekten, Konfigurationen und der Verwaltung dieser in Version-Workspaces.

5.4.1 History

Zu jedem Datenbankobjekt, das dem Versionsmanagement im Datenbanksystem unterliegt, ist ein DesignObjekt zugeordnet. Ein DesignObjekt besteht, neben den versionsunabhängigen Informationen, aus einer Anzahl von Versionen. Diese Versionen bilden die Entwicklungsgeschichte des DesignObjektes; ausgehend von einer initialen Version bis zur (zeitlich gesehen) letzten, aktuellen Version.

Die Aufgabe des History-Modells ist die Verwaltung aller Versionen eines DesignObjektes anhand deren Entwicklungsgeschichte sowie die Bereitstellung von Navigations- und Ab-

fragemöglichkeiten. Abbildung 5.6 stellt die wichtigsten Elemente im History-Modell in UML-Notation dar.

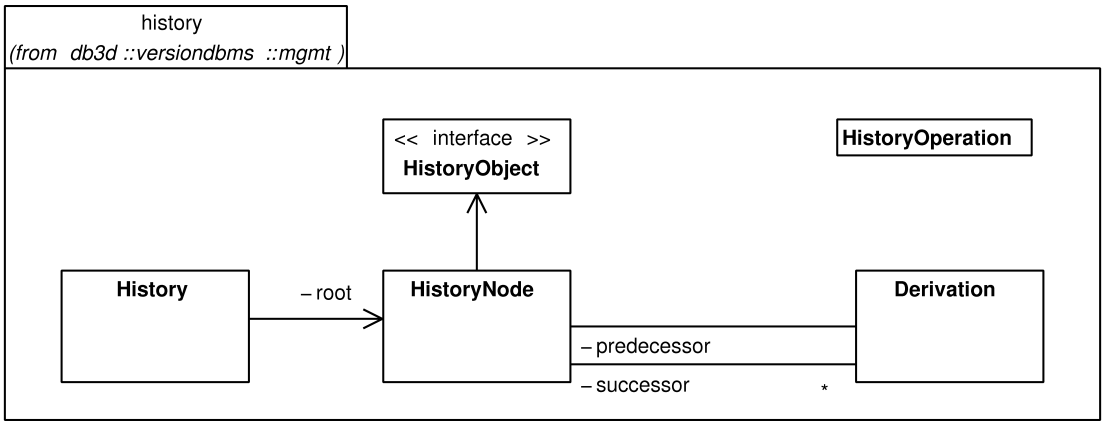

Abbildung 5.6: History-Modell zur Verwaltung der Versionen von DesignObjekten

Im History-Modell wird die Versionshistorie eines DesignObjektes von der Klasse *History* repräsentiert. Im Versionsmanagement enthält jedes DesignObjekt ein History-Objekt. Ein History-Objekt beinhaltet das initiale *HistoryNode*-Objekt, das die initiale Version repräsentiert. Alle weiteren Versionen in der History werden über Beziehungen zwischen HistoryNode-Elementen modelliert. Die Beziehungen sind typisiert – Beziehungen: Revision, Alternative und Merge – und durch die Klasse *Derivation* bereitgestellt. Jedes HistoryNode-Objekt kann mehrere Vorgänger- und Nachfolger-Beziehungen enthalten.

Die Verbindung zwischen einem HistoryNode-Objekt und einer bestimmten Version im Versionsmanagement erfolgt über das Interface *HistoryObject*. Dieses Interface wird im Versionsmanagement von den Versionsklassen implementiert. Der Aufbau einer Historie für ein DesignObjekt erfolgt durch die Methoden der Klasse History. Methoden zur Navigation und Abfrage der Historie werden durch die Klasse *HistoryOperation* bereitgestellt. Beide Klassen stellen die einzigen Einstiegspunkte in das History-Modell dar und werden ausschließlich im Versionsmanagement intern genutzt und eingebunden.

5.4.2 Versionsmanagement

Die Verwaltung der Versionen von DesignObjekten im Versionsmanagement baut auf dem History-Modell auf. Das Versionsmanagement-Modell definiert die DesignObjekte, Versionen und Konfigurationen zur Modellierung von Versionen von Datenbankobjekten. Eine vereinfachte Darstellung des Modells stellt Abbildung 5.7 in UML-Notation dar.

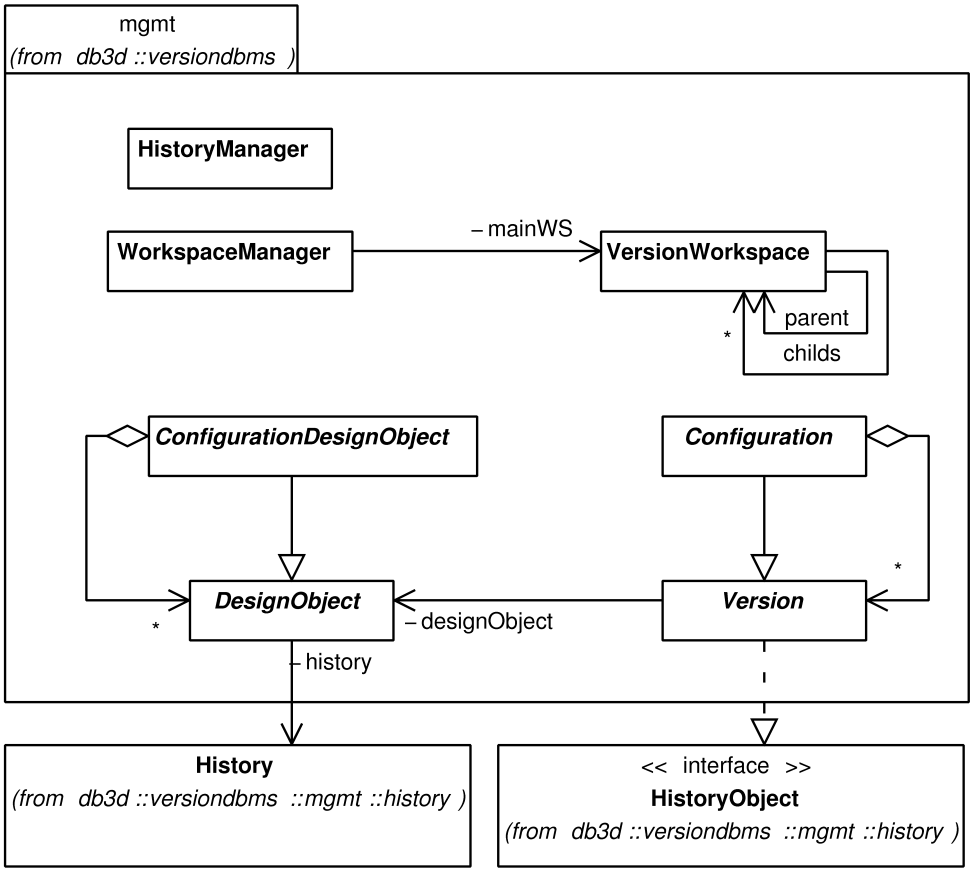

Abbildung 5.7: Versionsmanagement-Modell mit dem Bezug zum History-Modell

DesignObjekte und Versionen: Das Versionsmodell unterstützt nur einfache DesignObjekte. Ein DesignObjekt wird durch eine Instanz der Klasse *DesignObject* repräsentiert. Die Versionen eines DesignObjektes sind durch die Klasse *Version* modelliert und werden über ein *History*-Objekt aus dem History-Modell verwaltet. Hierfür implementiert die Klasse Version das *HistoryObject*-Interface. Die Klassen DesignObject und Version stellen abstrakte Klassen dar. Für jeden Datenbankobjekt-Typ existiert eine Implementierung. Die abstrakten Klassen stellen die Infrastruktur zur Versionsverwaltung bereit. Die konkreten Implementierungen legen die Art fest, wie Versionen in der Datenbank repräsentiert und gespeichert werden sollen bzw. können. In Kapitel 5.5 wird ein mögliches Modell für die Umsetzung für 3D-Geoobjekte vorgestellt, das im Prototyp implementiert ist.

Verwendung von Konfigurationen: Das Konzept der Konfiguration kann in zwei Ausprägungen verwendet werden (siehe auch [115, 133]). Zum einen als ein Konzept die

derzeitig zusammengehörigen Versionen eines komplexen DesignObjektes zu identifizieren. Zum anderen auch aus der Sichtweise des Anwenders um Sammlungen von DesignObjekten, die aus Anwendungssicht zusammengehören, und deren Versionen zu unterstützten.

Durch die Einschränkung im Versionsmodell auf einfache DesignObjekte wird kein Konzept zur Konfiguration im Sinne komplexer DesignObjekte benötigt. Das Versionsmodell verwendet jedoch das Konzept der Konfiguration im zweiten Sinne, um eine Gruppierung spezifischer Versionen einer Anzahl von DesignObjekten zu ermöglichen. Ein wichtiges Konzept der Konfigurationen im Modell ist die Möglichkeit die in einer Konfiguration enthaltenen Versionen von DesignObjekten durch die Definition von Constraints einzuschränken. Es können somit beispielsweise räumliche Einschränkungen – z.B. das keine Objektüberlappung existiert – auf den enthaltenen Versionen der DesignObjekte in einer Konfiguration definiert werden.

Als Anwendungsbeispiel für den geowissenschaftlichen Bereich sind die Repräsentation von 3D-Modellen im Versionsmanagement oder die Übertragung des Space-Konzeptes des Datenbanksystems in das Versionsmanagement zu nennen. Im weiteren Verlauf der Arbeit wird auf diese Verwendung des Konfigurationskonzeptes zur Modellierung von Spaces eingegangen. Im Sinne von 3D-Modellen ermöglichen Konfigurationen ebenfalls, eine explizite Ausprägung des 3D-Modells (als Version einer Konfiguration), für Operationen im Versionsmanagement – Export, Propagierung, Anwendung von Datenbankoperationen – zu verwenden.

Die Konfigurationsunterstützung wird durch eine Erweiterung von *DesignObject* und *Version* erreicht. Die abstrakten Klassen *ConfigurationDesignObject* und *Configuration* implementieren die notwendige Infrastruktur. In ConfigurationDesignObject sind die in der Konfiguration enthaltenen DesignObjekte festgelegt. Die Configuration-Objekte enthalten, analog zu Versionen, jeweils die spezifische Version dieser DesignObjekte.

VersionWorkspaces: Die Verwaltung der DesignObjekte erfolgt in sogenannten VersionWorkspaces zusammen mit ihren Versionen. Im Versionsmanagement ist genau ein Haupt-VersionWorkspace vorhanden. In diesem Haupt-VersionWorkspace erfolgt die initiale Erzeugung von DesignObjekten mit ihren initialen Versionen bei Überführung von Datenbankobjekten in den Versionsraum. Für die Weiterentwicklung der Versionen von DesignObjekten durch Revisionen und Alternativen können beliebig hierarchisch verbundene VersionWorkspaces erzeugt werden. Jeder VersionWorkspace (siehe Klasse *VersionWorkspace* in Abbildung 5.7) besitzt genau einen Vorgänger- und einen oder mehrere Nachfolger-VersionWorkspaces. DesignObjekte sind in einem VersionWorkspace sichtbar sobald eine ihrer Versionen enthalten ist.

Im Gegensatz zu anderen Versionsmodellen, bei denen die Historie eines DesignObjektes vollständig unabhängig von den VersionWorkspaces ist, wird im vorliegenden Ver-

sionsmodell eine Beziehung zwischen VersionWorkspaces und der Historie forciert. Das Versionsmodell erlaubt nur eine lineare Weiterentwicklung von Versionen innerhalb eines VWS, innerhalb eines VersionWorkspaces existieren somit nur Revisionen. Eine Entwicklung von alternativen Versionen, die deutliche Änderungen zu der bisherigen Entwicklung eines DesignObjektes bedeuten, müssen als alternativer Entwicklungsstrang in einem Nachfolger-VWS erfolgen. Die Beziehung zwischen der Vorgängerversion und der neuen Version im Nachfolger-VersionWorkspace ist dabei die Alternativ-Beziehung. Innerhalb des Nachfolger-VWS kann diese alternative Version durch beliebige Revisionen weiterentwickelt werden. Um eine Alternativ-Version oder eine ihrer Revisionen zu einem späteren Entwicklungszeitpunkt wieder zum Vorgänger-VWS zurück zu migrieren, muß ein Merge der beiden aktuellsten Revisionen in beiden VWS erfolgen. Hierbei entsteht im Vorgänger-VWS eine Merge-Version, die zwei Vorgängerversionen besitzt. Eine Überführung einer Version in einen Nachfolger-VWS kann auch für ein konfliktfreies Arbeiten an DesignObjekten verwendet werden. Den Zusammenhang zwischen der Historie eines DesignObjektes und der VersionWorkspace-Hierarchie stellt Abbildung 5.8 schematisch dar.

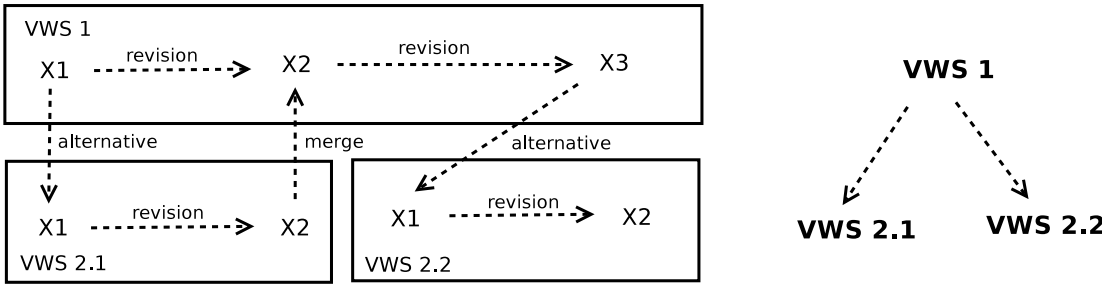

Abbildung 5.8: Beziehung zwischen Historie von DesignObjekt und VersionWorkspace

Ein ähnliches Modell, wie die hier beschriebene Modellierung der Alternative/Merge-Beziehung in der VWS-Hierarchie, ist im Datenbanksystem des GIS Smallworld zu finden [86]. Dort werden die VersionWorkspaces des vorliegenden Modells Alternativen genannt, da sie alternative Entwicklungsstränge repräsentieren.

Im Versionsmanagement dienen die VersionWorkspaces neben der Verwaltung von DesignObjekten und Versionen auch der Zugriffsreglementierung auf Versionen. So können in den meisten Versionsmodellen *private*, *geteilte* und auch *release* Workspaces gefunden werden. Im vorliegenden Modell ist derzeit nur ein sehr einfaches Zugriffsmodell, das zwischen privaten und öffentlichen/geteilten VersionWorkspaces unterscheidet, integriert. In einem privaten VWS sind nur Änderungen durch die erzeugende bzw. besitzende Person möglich, während in öffentlichen/geteilten VWS Änderungen durch beliebige Anwender erlaubt sind.

5.4.3 Operationen

Alle Operationen im Versionsmanagement erfolgen über zwei Einstiegsklassen in das Versionsmodell – die Manager für VersionWorkspaces und Histories.

Jegliche Administrationsoperationen bezogen auf VersionWorkspaces und deren Hierarchie erfolgen durch den *WorkspaceManager*. Dieser stellt Operationen zum Erzeugen und Löschen von VersionWorkspaces bereit. Zugriffe auf VersionWorkspaces erfolgen ebenfalls über den Manager, wodurch eine einfache Umsetzung von Zugriffsreglementierungen auf der Ebene von VersionWorkspaces erreicht werden kann. Der *HistoryManager* stellt die Operationen zur Verwaltung der Historie von DesignObjekten bereit. Operationen sind hier die Einfügung von Revisions-, Alternativ- und Mergeversionen in die Historie eines DesignObjektes, sowie Operationen zum Löschen einzelner Versionen aus der Versionshistorie bzw. ganzer DesignObjekte mit ihren Versionen. Durch die zentrale Bereitstellung der Operationen auf den Historien ist der Manager verantwortlich für die Konsistenz der Historien im Versionsmanagement. Als grundlegende Konsistenzbedingungen können hier genannt werden, daß eine Version nur mit einer Revisions- jedoch mit mehreren Alternativ-Versionen verbunden sein darf. Bei der Löschung einzelner Versionen muß weiterhin die Navigationsfähigkeit in der Historie erhalten werden und die korrekte Speicherung der Versionen (siehe Kapitel 5.5) garantiert werden.

5.4.3.1 Workspace-Operationen

Erzeugung eines VersionWorkspace: Ein VersionWorkspace wird im Versionsmanagement immer bezogen auf einen Vorgänger-VWS erzeugt. Die Operation zur Erzeugung des VersionWorkspace erwartet als Parameter den Vorgänger-Workspace, die gewünschte Zugriffsberechtigung, einen Namen und eine weitere Beschreibung. Der VersionWorkspace wird erzeugt und beim Vorgänger-VWS als Nachfolger-VWS registriert und in das Verzeichnis der VersionWorkspaces im Versionsmanagement aufgenommen.

Löschung eines VersionWorkspace: Die Löschung eines VersionWorkspaces bedingt gleichzeitig die Löschung aller im VWS enthaltenen Versionen. Ein VersionWorkspace kann nur gelöscht werden, wenn kein Nachfolger-VWS existiert bzw. alle existierenden Nachfolger-VWS gelöscht werden. Bei der Löschung eines VersionWorkspaces werden alle Versionen der sichtbaren DesignObjekte aus der Versionshistorie gelöscht und alle vom Vorgänger-VWS ausgehenden Alternativ- und in den Vorgänger-VWS zurückgehenden Merge-Beziehungen der Versionshistorie gelöscht. Anschließend wird der Workspace im Vorgänger-VWS deregistriert.

Zugriff auf VersionWorkspaces: Jeglicher Zugriff auf VersionWorkspaces im Versionsmanagement erfolgt über die WorkspaceManager-Schnittstelle. Diese stellt die Möglichkeit bereit über die VersionWorkspaces ausgehend vom Haupt-VersionWorkspace zu iterieren oder Zugriff auf einen VersionWorkspace über dessen Namen zu erhalten. In den bereitgestellten Möglichkeiten erfolgt die Überprüfung auf eine ausreichende Zugriffsberechtigung des Aufrufers.

5.4.3.2 History-Operationen

Erzeugung einer initialen Version: Bei der Erzeugung einer initialen Version muß zwischen der Operation zur Erzeugung des DesignObjektes, der Version und deren Registrierung als initiale Version eines DesignObjektes im Versionsmanagement unterschieden werden. Die Operation zur Erzeugung eines DesignObjektes und Version wird im Abschnitt 5.5.5 behandelt.

Die Registrierung initialer Versionen überprüft für das übergebene DesignObjekt, ob dieses bereits im Haupt-VersionWorkspace sichtbar und damit registriert ist. In diesem Fall wird abgebrochen. Ansonsten wird die übergebene Version als initiale Version im DesignObjekt sowie das DesignObjekt im Haupt-VersionWorkspace registriert.

Bildung einer Revisionsversion: Die Bildung einer Revisionsversion stellt den Normalfall bei der Entwicklung von DesignObjekten dar. Revisionsversionen werden immer bezogen auf die letzte Version eines DesignObjektes im VersionWorkspace erstellt. Eingabe ist der VersionWorkspace, die zu registrierende Revisionsversion und die aktuelle Version im VWS. Zuerst erfolgt die Überprüfung der übergebenen Parameter auf Korrektheit. Es wird geprüft, ob die aktuelle Version die letzte im VWS ist bzw. existiert und beide Versionen sich auf das gleiche DesignObjekt beziehen. Bei positiver Überprüfung erfolgt die Registrierung der Version als Revisionsversion in der Versionshistorie und wird mit dem Beziehungstyp *revision* mit der Vorgängerversion verbunden.

Die Abfolge der Registrierung ist im Versionsmanagement für die verschiedenen Versionsbeziehungen sehr ähnlich. Abbildung 5.9 stellt den detaillierten Ablauf der Registrierung einer Revisionsversion über die beteiligten Versionskonzepte als Sequenzdiagramm dar.

Bildung einer Alternativ-Version: Die Bildung von Alternativ-Versionen erfolgt beim Aufbau eines neuen VersionWorkspaces zur Modellierung alternativer Entwicklungen von DesignObjekten. Für alle Versionen, die in diesen neuen VersionWorkspace überführt werden sollen, müssen neue Alternativ-Versionen erstellt werden. Es können beliebig viele Alternativ-Versionen von einer Version ausgehen, solange diese in unterschiedlichen Nachfolger-VersionWorkspaces gebildet werden. Der Ablauf der Registrierung als Alternativ-Version folgt dem gleichen Schema wie die Registrierung einer Revisionsversion.

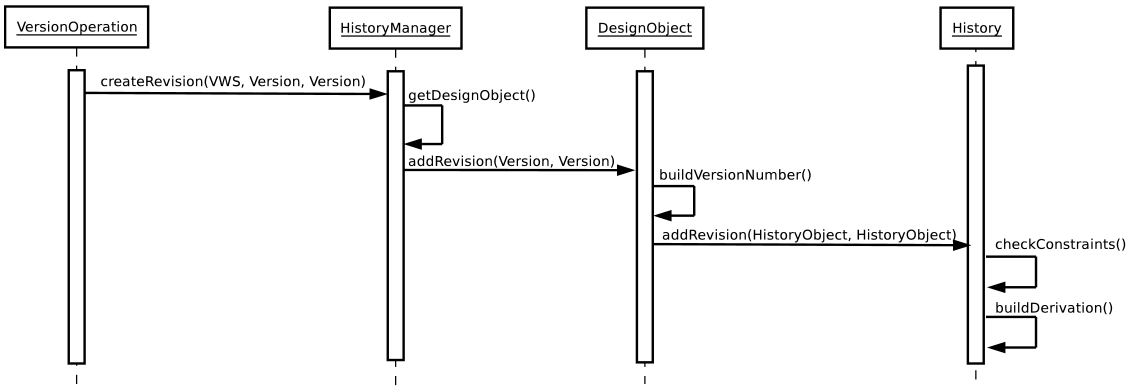

Abbildung 5.9: Sequenzdiagramm des Ablaufs der Registrierung einer Revisionsversion

Bildung einer Merge-Version: Die Bildung einer Merge-Version stellt die Propagierung einer Version eines DesignObjektes zurück in den Vorgänger-VWS, also *nach oben* in der VersionWorkspace-Hierarchie, dar. Die Propagierung entspricht der Bildung einer Merge-Version zwischen der letzten Revisionsversion im Vorgänger-VersionWorkspace und der zu propagierenden Version im aktuellen VersionWorkspace. Die Bildung von Merge-Versionen kann solange erfolgen, bis der Haupt-VersionWorkspace erreicht ist.

Zwischen der aktuellen Revisionsversion im Vorgänger-VWS und der zu propagierenden Version existieren, aufgrund des alternativen Charakters einer Entwicklung in einem Nachfolger-VWS, meistens Konflikte. Diese Konflikte müssen entweder aufgelöst werden oder die zu propagierende Version die Änderungen im Vorgänger-VWS überschreiben. Diese Entscheidungen sind durch Kooperation der Nutzer und der VersionWorkspace-Besitzer zu treffen. Ist die Propagierung einer Version in den Vorgänger-VWS, mit oder ohne Konflikte, entschieden erfolgt die Bildung dieser Version und deren Registrierung nach dem gleichen Schema wie bei einer Revisionsversion nur mit dem Beziehungstyp *merge*.

Löschung von Versionen: Die Löschung einer Version erfolgt durch die Löschung der gespeicherten Versionsdaten selbst und der Deregistrierung in der Versionshistorie des DesignObjektes. Die Löschung einer Version ohne Auswirkungen auf die Versionshistorie kann nur erfolgen wenn keine Nachfolgerversion existiert, es sich also um die aktuelle Alternative-, Merge- oder Revisionsversion handelt. Existieren Nachfolger bedingt die Löschung einer Version die Veränderung der Versionshistorie bzw. eine Veränderung der Beziehungen der bisherigen Vorgänger und Nachfolger in der Historie. Somit sind bei der Betrachtung der Versionslöschung die in Abbildung 5.10 aufgeführten Konstellationen zwischen Versionen zu unterscheiden.

Von den 10 möglichen Konstellationen erlauben nur die Konstellationen 1, 2 und 3 eine

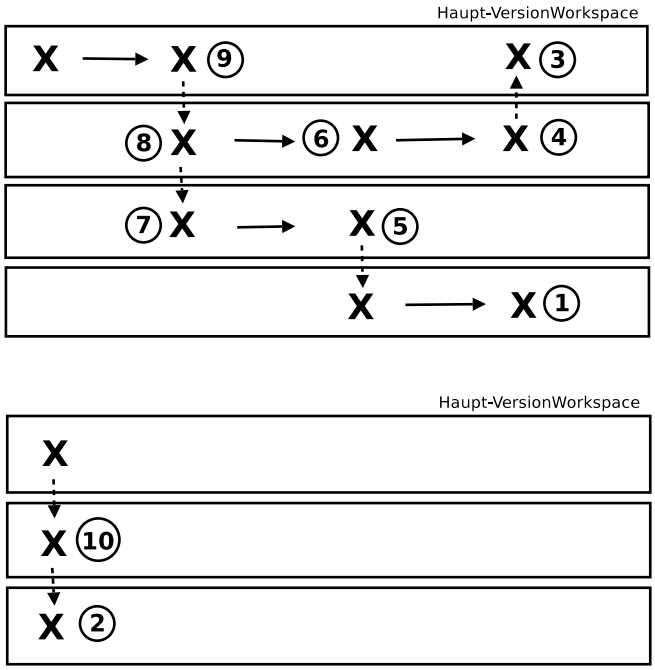

Abbildung 5.10: Existierende Versionsbeziehungen für die Versionslöschung

Versionslöschung ohne Veränderung der Versionshistorie, da bei diesen keine Nachfolgerversionen existieren. Ein weiterer Spezialfall bildet die 6. Konstellation, bei der die Version nur mit Revisionen in die Versionshistorie eingebunden ist. Die Löschung könnte hier durch die Übertragung der Änderungsoperationen auf die nächste Revisionsversion erfolgen, ohne die Semantik der Versionshistorie zu verändern. So besitzt die Vorgänger-Version auch nach der Löschung eine Revisionsbeziehung zu ihrem Nachfolger und ebenso die Nachfolger-Version eine Revisionsbeziehung zu ihrem Vorgänger. Alle anderen Konstellationen bedingen immer eine Veränderung in der Semantik der Versionshistorie. So bedingt beispielsweise die Löschung einer Version in der 4. Konstellation das sich die Versionsbeziehung der Vorgänger-Version ändert. Vor der Löschung besitzt die Vorgänger-Versionen eine Revisionsbeziehung zum Nachfolger, nach der Löschung eine Mergebeziehung. Ähnliches gilt für die jeweiligen anderen Konstellationen.

Aus Sicht der technischen Umsetzung kann im Prinzip eine Löschung in jeder Konstellation stattfinden und die jeweiligen Änderungsoperationen der zu löschenden Version an die jeweilige von ihr abstammende Nachfolger-Version übertragen werden. Die Entscheidung, ob eine Veränderung der Entstehungshistorie von den Versionen eines DesignObjektes erlaubt werden soll, ist jedoch von der jeweiligen Art der Anwendung abhängig. Soll das Versionsmanagement verwendet werden um die schrittweise Entwicklung von Objekten abzubilden, bedeutet jegliche Löschung einer Version mit Nachfolger einen nicht akzepta-

blen Informationsverlust in der Historie. Demgegenüber stehen andere Anwendungstypen, die beispielsweise nicht mehr benötigte feingliedrige Zwischenschritte in Revisionen (4. Konstellation) ohne Informationsverlust löschen können. Die Entscheidung, welche Konstellationen löschbar sein sollen ist somit abhängig von der jeweiligen Anwendung. Die Löschung in jeglichen Konstellationen sollte damit im Modell prinzipiell vorhanden sein und für jede Anwendung konfigurierbar sein.

Löschung eines DesignObjektes: Die Löschung eines DesignObjektes hängt wiederum davon ab, ob es in einem Nachfolger-VWS sichtbar ist oder nicht. Existiert das DesignObjekt nicht in einem Nachfolger-VWS erfolgt die Löschung durch die Löschung aller seiner Versionen im VWS (siehe vorangegangenen Abschnitt). Das DesignObjekt wird mit der Löschung seiner letzten Version im VersionWorkspace automatisch deregistriert und ist nicht mehr sichtbar. Existiert das DesignObjekt auch in einem Nachfolger-VWS muß es dort zuerst gelöscht werden. Die vollständige Löschung eines DesignObjektes mit allen Versionen aus dem Versionsmanagement erfolgt durch Löschung des DesignObjektes aus allen VersionWorkspaces, ausgehend vom letzten VersionWorkspace hin zum Haupt-VWS.

5.5 Umsetzung für 3D-Geoobjekte

Im folgenden wird die Umsetzung des Versionsmodelles für 3D-Geoobjekte beschrieben. Ausgehend von der Speicherung von Versionen erfolgt die Beschreibung der Repräsentation von Object3D als einfaches DesignObjekt im Versionsmodell. Die Nutzung des Konzepts zur Konfiguration wird am Beispiel der Repräsentation von Space3D Objekten erläutert.

5.5.1 Speicherung der Versionen

Die Grundlage vieler Operationen im Versionsmanagement – wie die Erzeugung einer Revisions- oder Mergeversion sowie die Konflikterkennung – ist die Funktionalität Änderungen zwischen zwei gegebenen Versionen festzustellen.

Für diese Funktionalität hat die Repräsentation von Versionen im Versionsmodell und ihre Art der Speicherung im Datenbanksystem eine entscheidende Bedeutung. Hierbei existieren zwei prinzipielle Möglichkeiten. Einerseits die Speicherung jeder Version als vollständiges Datenbankobjekt und andererseits lediglich die Änderungen zwischen zwei aufeinanderfolgenden Versionen zu speichern.

Speicherung als Datenbankobjekt: Die Speicherung als Datenbankobjekt (Vollspeicherung) hat den Vorteil, daß keine neue Objektrepräsentation für Versionen entwickelt

werden muß. Versionen sind vollwertige Datenbankobjekte und damit im Datenbanksystem in den Operationen (z.B. Import/Export) bereits unterstützt.

Diesem Vorteil stehen jedoch mehrere Nachteile gegenüber. Zum einen ist der hohe Speicherplatzverbrauch, vor allem bei *geringfügigen* Änderungen zwischen Versionen, zu erwähnen. Weiterhin bedingt die Komplexität der 3D-Objekte eine relativ lange Aufbauzeit (durch interne Indexierung, Topologieaufbau – vgl. Kapitel 6). Aus Effizienzgründen ist jedoch die fehlende Speicherung von Änderungen zwischen Versionen als größter Nachteil anzusehen. Operationen, die Änderungen zwischen Versionen benötigen, müssen diese erst aus den Datenbankobjekten durch geometrische Algorithmen berechnen. Finden Operationen zwischen zwei in der Historie nicht aufeinanderfolgenden Versionen statt, müssen die Änderungen für jedes Versionspaar zwischen den beiden Versionen berechnet und zusammengeführt werden. Weiterhin möglich ist eine Speicherung des Datenbankobjektes zusätzlich mit den erfolgten Änderungsoperationen zur Vorgängerversion. Dies führt jedoch zu einem noch höheren Speicherplatzverbrauch als bei der Vollspeicherung.

Änderungsspeicherung: Bei der Änderungsspeicherung (Delta-Storage) werden jeweils nur die Änderungen, die eine Version in Bezug zu ihrer Vorgängerversion ausmachen, gespeichert. Geht man davon aus, daß die meisten Versionsbildungen nicht eine komplette Änderung des Objektes beinhalten, ist der Speicherplatzbedarf bei der Änderungspeicherung als deutlich geringer als bei der Vollspeicherung anzusehen.

Da in der Änderungspeicherung keine Datenbankobjekte als Versionen gespeichert werden, müssen diese, um in Operationen des Datenbanksystems verwendet werden zu können, aus den Versionen erzeugt werden können. Ein Algorithmus hierfür ist der Aufbau des Datenbankobjektes, vom initialen Datenbankobjekt ausgehend, durch das Einpflegen der gesamten Änderungen in der Versionshistorie bis zur gewünschten Version (vgl. Pseudocode 5.1). Um relativ konstante Laufzeiten auch bei langen Versionshistorien zu garantieren, können beispielsweise in definierten Abständen vollständige Datenbankobjekte zusätzlich in den Versionen gespeichert werden. Grundlegender Vorteil der Änderungsspeicherung ist die direkte Verwendbarkeit der Versionsrepräsentation bei Operationen im Versionsmanagement. Jegliche Operationen, die auf den Unterschieden und Änderungen zwischen Versionen basieren, wie z.B. Konflikterkennung oder Änderungsanalyse, können damit ohne aufwendige Berechnungen beantwortet werden.

Umsetzung im Versionsmodell: Änderungen zwischen Versionen werden im Versionsmodell als eine Menge von Änderungsoperationen für den jeweiligen DesignObjekt-Typ modelliert. Unter einer Änderungsoperation ist ein Tupel aus Art der Änderungsoperation (z.B. Löschen) und dem veränderten Datum (z.B. ID des zu löschenden Simplex) zu verstehen.

Abbildung 5.11 zeigt die Umsetzung von Änderungsspeicherung im Versionsmodell. Es wird eine generische Unterstützung für die Änderungsspeicherung modelliert, die für jeden Datenbankobjekt-Typ (DBTyp – siehe Abbildung 5.11) bezogen auf seine Änderungsoperationen erweitert werden muß. Die Konkretisierung der Klasse DesignObject beinhaltet die Referenz auf das initiale Datenbankobjekt, aus dem das DesignObjekt hervorgeht. Jede Version enthält dann ein Objekt der Klasse *ChangeSet*, das die Menge der Änderungsoperationen zur vorangegangenen Version vom Typ *ChangeOperation* verwaltet. Implementierungen für die einzelnen Datenbankobjekt-Typen können von ChangeSet ableiten, um kontrollieren zu können, welche Änderungsoperationen hinzugefügt werden können.

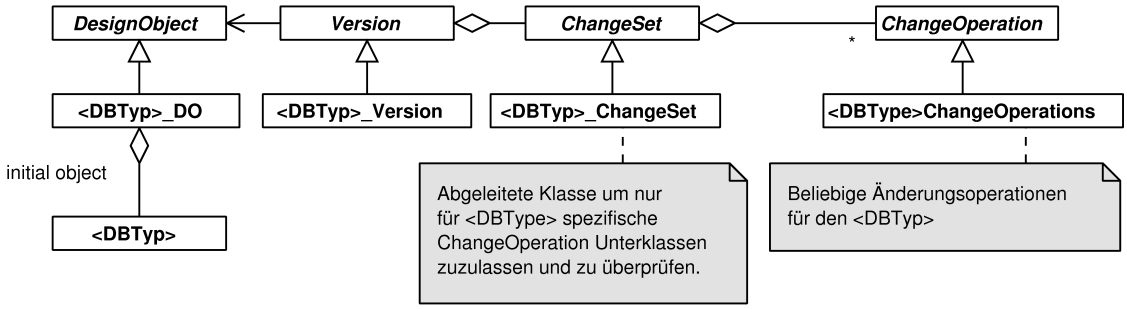

Abbildung 5.11: Änderungsspeicherung für Versionen im Versionsmodell

Die Komplexität von Änderungsspeicherung ist abhängig von dem Datenbankobjekt-Typ, der unterstützt werden soll. In Abhängigkeit vom zugrundeliegenden Datenmodell kann die Konzeption von Änderungsoperationen von einfach bis sehr komplex reichen. Das in der vorliegenden Arbeit zugrundeliegende Datenmodell der simplizialen d-Komplexe ermöglicht eine einfache Umsetzung der Änderungsoperationen.

5.5.2 Repräsentation von Object3D

Die Modellierung des Datenbankobjekt-Typs *Object3D* erfolgt durch die beiden Klassen *Object3D_DO* als Konkretisierung von DesignObject und *Object3D_Version* als Ableitung von Version. Die Klasse ChangeSet zur Speicherung der Menge von Änderungsoperationen wird mit der Klasse *Object3D_ChangeSet* erweitert, um aufzunehmende Änderungsoperationen auf die korrekte Zugehörigkeit überprüfen zu können. Abbildung 5.12 zeigt den Aufbau für den Datenbankobjekt-Typ Object3D in UML-Notation.

Änderungsoperationen sind als Unterklassen von *ChangeOperation* modelliert. Ausgehend von den Feststellungen in Kapitel 5.3.2 werden zum einen Operationen für die Behandlung von Netzkomponenten inklusive thematischer Sachdaten benötigt. Diese Operationen beziehen sich auf Änderungen, die eine komplette Löschung oder Hinzufügung einer Netzkomponente beinhalten. Wichtig ist hier die Integration der Thematik für die Simplexe

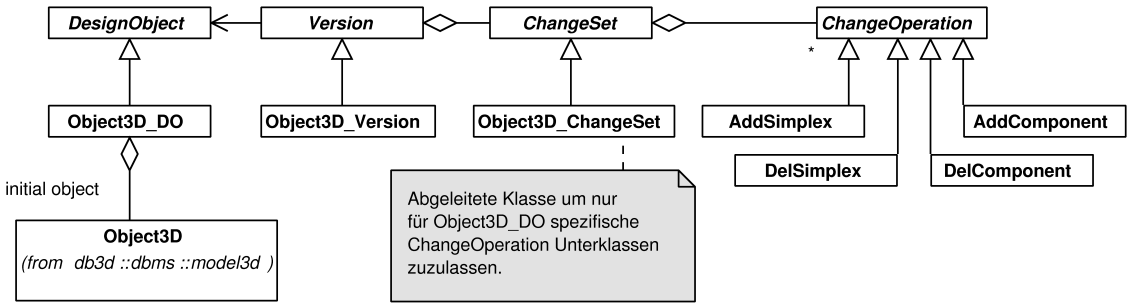

Abbildung 5.12: Modell der Änderungsspeicherung für Object3D-Datenbankobjekte

der Netzkomponente in die Änderungsoperation. Die Anwendung der Änderungsoperation muß nicht nur die Netzkomponente zum Object3D hinzufügen bzw. löschen, sondern auch die zugehörigen thematischen Daten im ThematicObject3D-Objekt von Object3D entsprechend ändern.

Zum anderen werden Operationen für die Modifizierung von Netzkomponenten benötigt. Die Grundoperationen sind das Hinzufügen bzw. Löschen eines Simplexes der Netzkomponenten mit den zugehörigen thematischen Daten. Neben der Art der Operation (Hinzufügen/Löschen) und dem jeweiligen Simplex muß auch noch die Netzkomponente, auf die diese Operation angewendet werden soll, angegeben werden. Auch hier muß die jeweilige Änderungsoperation die Zusammenhänge zwischen Geometrie und Thematik im Datenbank-Objektmodell berücksichtigen.

Um ebenfalls eine Änderung der thematischen Daten ohne gleichzeitige Geometrieänderung zu ermöglichen, müssen Operationen zum Hinzufügen, Löschen und Modifizieren von thematischen Daten für Simplexe anhand ihrer ID bereitgestellt werden. Da die Simplex ID eindeutig im gesamten Netz ist, wird keine weitere Angabe zur Netzkomponente benötigt, in der der Simplex enthalten ist. Zur Änderung der Objektthematik reicht die Angabe der Änderung aus, da jedes Objekt nur ein zugeordnetes Sachdatum besitzt.

5.5.3 Repräsentation von Space3D

Die Unterstützung des Space-Konzeptes im Datenbanksystem wird auf das Versionsmanagement mittels Konfigurationen übertragen. Exemplarisch wird im folgenden die Umsetzung des Space3D-Konzeptes, im Datenbanksystem zur Verwaltung von 3D-Modellen verwendet, auf das Konfigurationskonzept beschrieben. Space3D-Konfigurationen können genutzt werden um definierte, konsistente Sammlungen von Versionen von DesignObjekten zusammenhängend in den Operationen des Versionsmanagements zu verarbeiten.

Konfigurationen werden im Versionsmodell durch die von DesignObject und Version abgeleitenden Klassen *ConfigurationDesignObject* und *Configuration* repräsentiert. Confi-

gurationDesignObject referenziert die Menge an enthaltenen DesignObjekten und Configuration die spezifische Version jedes enthaltenen DesignObjektes. Die Konfigurationsklassen stellen allgemeine Konsistenzprüfungen bereit. Diese beinhalten beispielsweise die Prüfung, daß ein DesignObjekt nur einmal in einer Konfiguration enthalten sein darf und Versionen nur in einer Configuration referenziert werden dürfen in der auch ihr zugehöriges DesignObjekt enthalten ist. Parallel zur Repräsentation von DesignObjekten müssen auch für Konfigurationen spezifische Ableitungen, jeweils pro Konfigurationstyp der im Versionsmanagement unterstützt werden soll, erfolgen. Diese Erweiterungsklassen können die allgemeinen Konsistenzprüfungen um spezifische Einschränkungen erweitern. Abbildung 5.13 stellt den Ausschnitt des Versionsmodells für Konfigurationen mit den Konfigurationstyp bezogenen Ableitungsklassen dar.

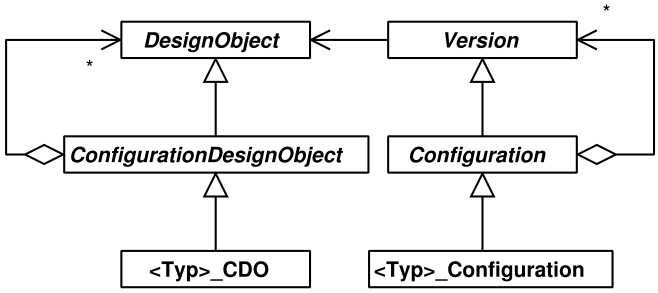

Abbildung 5.13: Konfigurationen im Versionsmodell

Die Repräsentation eines Space3D erfolgt durch spezifische Klassen für diesen Konfigurationstyp – *Space3D_CDO* und *Space3D_Configuration*. Space3D_CDO erweitert die allgemeinen Konsistenzprüfungen um erlaubte Dimensionen für DesignObjekte (0D, 1D, 2D und 3D). Eine Erweiterung der Konsistenzprüfungen von Space3D_Configuration ist grundsätzlich ebenfalls möglich. Es muß jedoch beachtet werden, daß zur Überprüfung der Konsistenz nur das Modell der Änderungsspeicherung zur Verfügung steht. Zum Beispiel können, um eine Konsistenzbedingung *Schnittfreier Raum* in einer Space3D-Konfiguration zu garantieren, nicht die vorhandenen Operationen für Test auf Schnitt des Datenbanksystems angewendet werden. Um diese Datenbankoperationen anzuwenden muß zuerst eine Konvertierung der enthaltenen und der zu überprüfenden Version in das Datenbank-Objektmodell erfolgen. Es muß beachtet werden, daß eine Konvertierung in das Datenbank-Objektmodell hohe Laufzeiten bedingt.

Hier ist festzustellen, daß unabhängig von den Vorteilen der Änderungsspeicherung für die Verwaltung von Versionen und bei Versionsoperationen im Versionsmanagement, die Änderungsspeicherung Probleme für die Anwendbarkeit normaler geometrischer Algorithmen verursacht. Als weiterer Ansatz kommt neben der bereits angesprochenen Konvertierung der enthaltenen Versionen in das Datenbank-Objektmodell eine eingeschränkte Vollspei-

cherung der Versionen in der letzten Konfigurationsrevision in Betracht. Werden neue Revisionen einer Konfiguration gebildet, wird das Datenbankobjekt der Vollspeicherung durch die Änderungen in den enthaltenen Versionen aktualisiert und auf die neue Revision der Konfiguration übertragen.

Eine weitere Möglichkeit stellt die Entwicklung von Algorithmen für die Überprüfung der jeweiligen Konsistenzbedingung auf dem Modell der Änderungsspeicherung dar. Problematisch hierbei ist die Möglichkeit von Volumenobjekten in Hüllenrepräsentation. Überprüfungen, ob Änderungen innerhalb oder außerhalb des Volumenobjektes liegen, sind in diesem Modell nicht durchführbar. Kann jedoch die Annahme getroffen werden, daß keine Volumenobjekte in Hüllenrepräsentation erlaubt sind (in vielen geologischen Anwendungen wird die Volumenrepräsentation, nicht die Hüllenrepräsentation benötigt), können auch auf dem Modell der Änderungsspeicherung effiziente Algorithmen entwickelt werden.

5.5.4 Konvertierung zwischen Datenmodellen

Es existieren zwei Konvertierungsoperationen pro Datenbankobjekt-Typ: Die Konvertierung vom Versionsmodell zum Datenbank-Objektmodell und die Konvertierung in die entgegengesetzte Richtung. Bei der Konvertierung vom Versionsmodell zum Datenbank-Objektmodell wird aus einer Version eines DesignObjektes ein Datenbankobjekt erzeugt. Dieses kann anschließend mit den Operationen im Datenbanksystem verarbeitet oder in die unterstützten Datenformate exportiert werden. Die Konvertierung vom Datenbank-Objektmodell in das Versionsmodell hat zwischen zwei Fällen zu unterscheiden: Existiert zum gegebenen Datenbankobjekt bereits ein DesignObjekt, so erfolgt die Bildung einer neuen Revisions-, Alternativ- oder Merge-Version, bezogen auf eine bereits existierende Version dieses DesignObjektes. Ist das gegebene Datenbankobjekt noch nicht unter Versionskontrolle, so muß dafür ein neues DesignObjekt mit einer initialen Version erzeugt werden.

Versionsmodell zu Datenbank-Objektmodell: Die Konvertierung von dem Versionsmodell zum Datenbank-Objektmodell basiert auf dem Aufbau eines DB-Objektes aus einem initialen DB-Objekt und der sequenziellen Ausführung aller Änderungsoperationen der Versionshistorie bis zur gewünschten zu konvertierenden Version. Das initiale DB-Objekt wird im Versionsmodell vom DesignObjekt der Version referenziert. Ergebnis der Konvertierung ist ein DB-Objekt äquivalent zur gegebenen Version. Für den Datenbankobjekt-Typ Object3D erfolgt der Aufbau aus dem initialen Object3D-Objekt und den in Versionen gespeicherten typspezifischen Object3D-Änderungsoperationen.

Ein allgemeiner Algorithmus (siehe Pseudocode 5.1) kann folgendermaßen beschrieben werden: Eingabe ist die zu konvertierende Version v, als Ergebnis wird ein Objekt o

vom entsprechendend Datenbankobjekt-Typ zurückgegeben. Für den Datenbankobjekt-Typ Object3D wird ein Object3D-Objekt zurückgegeben.

```
Input: Version v
Output: DB-Object o

begin
    DesignObject do = v.designObject;
    DB-Object initial = do.initialObject;
    DB-Object o ← copy of initial;
    List versions = RetrieveVersionChainInitialToGiven(v);
    foreach Version ver ∈ versions do
        ChangeSet cs ← ver.changeSet;
        foreach ChangeOperation op ∈ cs do
            apply op to o;
    return o;
end
```

Pseudocode 5.1: Konvertierung: Versionsmodell → Datenbank-Objektmodell

Über das DesignObject der Version v wird das initiale DB-Objekt erhalten. Über eine Operation im Versionsmanagement (siehe Pseudocode 5.2 und Erklärung im Text) wird für die gegebene Version eine Liste aller Versionen von der initialen Version des Haupt-VersionWorkspace bis zur übergebenen Version erzeugt. Jede dieser Versionen enthält ein ChangeSet-Objekt mit den Änderungsoperationen zur vorangegangenen Version. Für jede Version in der Reihenfolge der Liste werden die Änderungsoperation im ChangeSet-Objekt auf dem DB-Objekt ausgeführt. Nach der Anwendung aller Änderungsoperationen wird das DB-Objekt als Ergebnis zurückgegeben.

Die Funktion *RetrieveVersionChainInitialToGiven* ist in der Klasse *HistoryOperation* des History-Modells implementiert. Die Funktion erstellt eine Liste der Versionen in der Versionshistorie eines DesignObjektes, von der initialen Version bis zur übergebenen Version. Die Funktion verwendet eine weitere Funktion *RetrieveLinearRevision*, um der Liste alle Versionen innerhalb der linearen Historie eines VersionWorkspace hinzuzufügen (alle Revisionen bis zur ersten Version im VWS).

Der Algorithmus der Funktion *RetrieveVersionChainInitialToGiven* funktioniert wie folgt: Eingabe ist die Version v, für die eine Versionsliste erstellt werden soll. Rückgabe der Funktion ist diese Liste mit den Versionen, geordnet von der initialen zur übergebenen Version.

Über die Schnittstelle zur History (*HistoryObject*) wird das HistoryNode-Objekt der Version erhalten und an die Funktion *RetrieveLinearRevision* übergeben. Diese Funktion fügt alle Versionen der linearen Versionshistorie im aktuellen VersionWorkspace in die Liste

```
RetrieveVersionChainInitialToGiven(Version v) : List
begin
    List list = ∅;
    HistoryNode actual = null;
    // go back first in the linear revision line
    actual = RetrieveLinearRevision (version.historyNode, list);
    // as long as there is a variant predecessor - move along
    while actual has a variant predecessor do
        actual = RetrieveLinearRevision (actual.variantPredecessor, list);
    return list
end

RetrieveLinearRevision(HistoryNode node, List list) : HistoryNode
begin
    list.add(0, node.historyObject);
    HistoryNode actual = node;
    while actual has more revision predecessors do
        actual = actual.revisionPredecessor;
        list.add(0, actual.historyObject);
    return actual
end
```

Pseudocode 5.2: Funktion: Erzeugung der Versionslinie aus einer Version v

ein. Nach Ablauf erfolgt eine Prüfung, ob die erste Version des VersionWorkspace eine Alternativ-Version als Vorgänger besitzt. Solange eine Alternativ-Version existiert, werden die Versionen der Revisionslinie in dem VersionWorkspace der Alternativ-Version über die Funktion *RetrieveLinearRevision* der Liste hinzugefügt. Dies wird wiederholt, bis keine Alternativ-Version als Vorgänger mehr existiert und somit die initiale Version im Haupt-VersionWorkspace erreicht ist. Die Rückgabe ist die geordnete Liste mit allen Versionen von der initialen Version bis zur übergebenen Version.

Wie im Abschnitt 5.5.1 Änderungsspeicherung bereits ausgeführt wurde, ist die Laufzeit des Konvertierungsalgorithmus abhängig von der Länge der Versionshistorie bzw. der Anzahl an Änderungsoperation bis zur angefragten Version. Diese Abhängigkeit kann durch zwei unabhängig voneinander anwendbaren Optimierungen im Algorithmus entgegnet werden.

Ein Ansatz ist eine zusätzliche Speicherung von Versionen im Datenbank-Objektmodell in definierten Abständen der Versionshistorie. Der Konvertierungsalgorithmus muß dadurch nicht mit dem initialen DB-Objekt des DesignObjektes beginnen, sondern kann auf dem jeweils letzten gespeicherten DB-Objekt in der Versionshistorie aufbauen. Für die Auswahl,

in welchen Versionen der Versionshistorie zusätzlich eine Versionsvollspeicherung erfolgen soll, kommen verschiedene Schemata in Betracht. Zum einen sind definierte Abstände bzw. Abstände anhand der Anzahl von Änderungsoperationen denkbar. Weiterhin kann eine Speicherung im Datenbank-Objektmodell der jeweils ersten Version eines DesignObjektes pro VersionWorkspace festgelegt werden. Das Schema einer Versionsvollspeicherung nach einer festgelegten Anzahl von Änderungsoperationen ermöglicht die Bestimmung eines worst-case Laufzeitverhaltens des Algorithmus.

Eine weitere Optimierung kann durch die Minimierung der Anzahl an Änderungsoperationen, die auf dem DB-Objekt angewendet werden, erreicht werden. Die Ausführung von Änderungsoperationen auf dem DB-Objekt kann laufzeitintensiv werden, wenn dadurch Aktualisierungen des internen Objektzustandes (z.B. Aktualisierung des internen Indexes) ausgelöst werden. Um möglichst wenig Änderungsoperationen anzuwenden, kann auf der geordneten Liste der Änderungsoperationen im Algorithmus eine Überprüfung auf Aufhebung von Operationen durch Operationen späterer Versionen erfolgen. Zum Beispiel werden alle Änderungen an einer Netzkomponente aufgehoben durch eine spätere Änderungsoperation, die diese Netzkomponente löscht. In diesem Fall können alle Änderungsoperationen, die sich auf diese Netzkomponente beziehen, im Algorithmus übergangen werden. Es muß jedoch aufgepasst werden, daß eine Überprüfung auf Aufhebung von Operationen nicht eine höhere Laufzeit als die Anwendung der Änderungsoperationen selbst bedingt. Die Überprüfung anhand einfacher Heuristiken wie die Löschung kompletter Netzkomponenten in späteren Versionen kann jedoch sehr effizient erfolgen.

Datenbank-Objektmodell zu Versionsmodell: Die Konvertierung vom Datenbank-Objektmodell in das Versionsmodell wird für die Unterstützung externer auf das Versionsmanagement aufsetzender Applikationen benötigt. Der Austausch zwischen Applikationen und dem Versionsmanagement kann auf zwei Ebenen erfolgen: Applikationen können zum einen im Datenbank-Objektmodell mit dem Versionsmanagement Daten austauschen. In diesem Fall werden benötigte Versionen in DB-Objekte konvertiert und an die Applikation übergeben bzw. DB-Objekte von der Applikation empfangen. Zum anderen können Applikationen mit dem Versionsmanagement direkt auf Basis des Versionsmodells Daten austauschen.

Im ersten Fall wird für die Erzeugung einer Revisions-, Alternativ- oder Merge-Version aus dem DB-Objekt der externen Applikation, eine Konvertierung vom Datenbank-Objektmodell in das Versionsmodell benötigt. Im Kapitel 6.2 wird eine Anwendung des Versionsmanagements mit dem Modellierungssystem GOCAD, deren Datenaustausch auf Basis von DB-Objekten basiert, vorgestellt. Die im anschließenden Kapitel 5.6 beschriebene mobile Datenbank beinhaltet dagegen auch einen auf dem Versionsmodell basierenden Datenaustausch. Die mobile Datenbank speichert lokale Änderungen als Änderungsoperationen und überträgt, beim Erzeugen einer neuen Version im Versionsmanagement, nur die Menge an

Änderungen zur vorangegangenen Version.

Die Operationen zur Bildung der Versionsbeziehungen in der Versionshistorie wird im folgenden Abschnitt behandelt. Im folgenden wird der Algorithmus zur Berechnung der Menge an Änderungsoperationen zwischen einem DB-Objekt und einer gegebenen Version ausgeführt. Eingabe in den Algorithmus (siehe Pseudocode 5.3) ist das DB-Object o und die Version $prev$, zu der die Änderungsoperationen berechnet werden sollen.

Der Algorithmus basiert auf dem Vergleich der Simplexe zwischen dem DB-Objekt und der Version. Der Ablauf ist unterteilt in zwei Abschnitte: Im Teil A wird für jede Netzkomponente die Menge ihrer Simplexe aufgebaut. Dies erfolgt für die Netzkomponenten des DB-Objektes und der Version. Diese Simplex-Mengen stellen das gemeinsame Datenmodell zwischen dem DB-Objekt und der Version dar. Im Teil B werden in der ausgelagerten Funktion *ComputeChangeSet* (siehe Pseudocode 5.4) zwischen den Simplex-Mengen des DB-Objektes und der Version die Menge an Änderungsoperationen berechnet und als ChangeSet-Objekt zurückgegeben. Als Rückgabe des Algorithmus wird anschließend eine Version mit dem ChangeSet-Objekt und der übergebenen Version $prev$ als Vorgänger erzeugt. Der beschriebene Algorithmus bezieht sich dabei nur auf geometrische Unterschiede, Unterschiede in den thematischen Sachdaten der Objekte bzw. der Simplexe werden im beschriebenen Algorithmus nicht berücksichtigt.

Teil A: In einem ersten Schritt werden die Simplex-Mengen der Version aufgebaut. Dies erfolgt durch den Aufbau einer Simplex-Menge pro Netzkomponente des initialen Datenbankobjektes, das von dem DesignObjekt referenziert wird. Auf diesen Simplex-Mengen werden anschließend die Änderungsoperationen der Versionslinie von der initialen Version bis zur gegebenen Version eingebaut. Im zweiten Schritt werden die Simplex-Mengen für das DB-Objekt aufgebaut. Die in diesen Schritten erzeugten Simplex-Mengen werden in zwei *Map*-Datenstrukturen für das DB-Objekt und die Version, mit der Netzkomponenten-ID als Schlüssel und der Simplex-Menge der Netzkomponente als Wert, vorgehalten.

Teil B: Die Berechnung der Änderungsoperationen erfolgt auf den übergebenen Map-Datenstrukturen des DB-Objektes und der Version. Als Ergebnis wird das aus den Änderungsoperationen erzeugte ChangeSet-Objekt zurückgegeben. Im Algorithmus werden die beiden Map-Strukturen für jede Netzkomponenten-ID durchlaufen. In jedem Durchlauf werden die Simplex-Mengen für die aktuelle Netzkomponenten-ID wie folgt verarbeitet: Existiert für eine Netzkomponenten-ID nur eine Simplex-Menge für das DB-Objekt, so ist diese Komponente komplett neu hinzugekommen. Es wird eine *AddComponent*-Änderungsoperation erzeugt. Eine *DelComponent*-Änderungsoperation wird dagegen erzeugt, wenn nur die Simplex-Menge für die Version existiert, also die Netzkomponente im DB-Objekt gelöscht wurde. Existieren die Simplex-Mengen einer Netzkomponenten-ID sowohl für das DB-Objekt als auch die Version, dann werden die Paare gleicher Simplexe in den beiden Mengen berechnet und aus den Simplex-Mengen entfernt. Für die verbliebenen Simplexe in den Simplex-Mengen gilt folgendes: Simplexe, die noch in der Simplex-Menge

```
Input: DB-Object o, Version prev
Output: Version v

// Teil A: Aufbau der Simplex Mengen von Version und DB-Objekt

// Maps with key=Komponent-ID/value=Simplex-Set der Komponente
Map simplicesVersion = new Map();
Map simplicesObject = new Map();

// Aufbau Simplex-Sets in Map simplicesVersion für Version prev

Set components1 = v.designObject.initialObject.components;
List versions = RetrieveVersionChainInitialToGiven(prev);

foreach Component comp ∈ components1 do
    build set from simplexes of comp;
    put set into simplicesVersion for its component id;

// Änderungsoperationen der Versionen einbauen
foreach Version ver ∈ versions do
    ChangeSet cs = ver.changeSet;
    foreach ChangeOperation op ∈ cs do
        if op instanceof AddSimplex then
            add Simplex to simplices set for given component id;

        else if op instanceof DelSimplex then
            remove Simplex from simplices set for given component id;
        else if op instanceof AddComponent then
            put set of simplices into simplicesVersion for given component id;
        else
            remove set of simplices from simplicesVersion for given component id;

// Aufbau Simplex-Sets in Map simplicesObject für DB-Object o

Set components2 = o.components;
foreach Component comp ∈ components2 do
    build set from simplexes of comp;
    put set into simplicesObject for its component id;

// Teil B: Berechnung ChangeSet zwischen den Simplex-Sets

ChangeSet cs = ComputeChangeSet (simplicesVersion, simplicesObject);

Build Version v for DesignObject from Version prev with ChangeSet cs;
return v;
```

Pseudocode 5.3: Konvertierung: Datenbank-Objektmodell → Versionsmodell

```
Input: Map simplicesVersion, Map simplicesObject
Output: ChangeSet cs

cs = new ChangeSet();

foreach component id existent as key in either simplicesVersion or
simplicesObject do
    Set versionSimplices = simplicesVersion for component id;
    Set objectSimplices = simplicesObject for component id;

    if versionSimplices = ∅ and objectSimplices ≠ ∅ then
        // Diese Komponente ist neu hinzugekommen
        add AddComponent-ChangeOperation for objectSimplices to ChangeSet cs;
        continue;
    if objectSimplices ≠ ∅ and objectSimplices = ∅ then
        // Diese Komponente wurde gelöscht
        add DelComponent-ChangeOperation for component id to ChangeSet cs;
        continue;

    // Berechne für gleiche Simplex-Paare wenn beide Sets ≠ ∅
    Set equalPairs = ComputeEqualPairs(versionSimplices, objectSimplices);

    // Entferne alle gleichen Paare von den Sets
    foreach Pair(a,b) ∈ equalPairs do
        remove a from versionSimplices;
        remove b from objectSimplices;

    // versionSimplices enthält nur noch Simplices die gelöscht werden müssen
    // objectSimplices enthält nur noch Simplices die hinzugefügt werden müssen
    foreach element ∈ versionSimplices do
        add DelSimplex-ChangeOperation for element to ChangeSet cs;
    foreach element ∈ objectSimplices do
        add AddSimplex-ChangeOperation for element to ChangeSet cs;

return cs;
```

Pseudocode 5.4: Berechnung des ChangeSet zwischen Simplex-Mengen

der Version sind, wurden im übergebenen DB-Objekt gelöscht. Für diese Simplexe wird eine *DelSimplex*-Änderungsoperation erzeugt. Analog dazu wird für die Simplexe aus der DB-Objekt Simplex-Menge eine *AddSimplex*-Änderungsoperation erzeugt. Aus den berechneten Änderungsoperationen wird das ChangeSet erzeugt und zurückgegeben.

Die Berechnung der geometrisch äquivalenten Paare von Simplexen für die Netzkomponenten erfolgt durch die Funktion *ComputeEqualPairs*. Der Algorithmus dieser Funktion ist entscheidend für das Laufzeitverhalten des Konvertierungsalgorithmus. In der prototy-

Input: Set *versionSimplices*, Set *objectSimplices*
Output: Set *equalPairs*

// Aufbau von zwei R-Bäumen mit den Simplexen
RTree *vTree* = new RTree(*versionSimplices*);
RTree *objTree* = new RTree(*objectSimplices*);

// Spatial-Join zur Bestimmung der geometrisch äquivalenter Paare
Set *equalPairs* = *vTree*.equalsJoin(*objTree*);

return *equalPairs*;

Pseudocode 5.5: Berechnung geometrisch äquivalenter Simplexe

pischen Implementierung basiert der Algorithmus auf einem ad hoc aufgebauten R-Baum zum Abgleich geometrisch äquivalenter Simplexe. Der Pseudocode 5.5 zeigt den zugehörigen Algorithmus. Eingabe in den Algorithmus sind die Simplex-Mengen jeweils für die Version *versionSimplices* und das DB-Objekt *objectSimplices*. Als Ergebnis wird die Menge der geometrisch äquivalenten Simplexpaare *equalPairs* zurückgegeben.

5.5.5 Operationen im Versionsmanagement

Um innerhalb des Versionsmanagements arbeiten zu können, müssen entweder existierende Objekte des Datenbanksystems in das Versionsmanagement für die weitere Modellierung überführt oder Objekte aus externen Applikationen in das Versionsmanagement eingebracht werden können. Der vorliegende Abschnitt beschäftigt sich mit der Erstellung der DesignObjekte und Versionen für die Umsetzung des Versionsmodells mit Änderungsspeicherung.

Überführung DB-Objekt in das Versionsmanagement: Ein existierendes Objekt der Datenbank muß zuerst in das Versionsmanagement überführt und registriert werden, bevor eine Bearbeitung mittels Versionen erfolgen kann. Die Überführung eines Datenbankobjektes erfolgt durch die Bildung eines DesignObjektes und der Erstellung der initialen Version im Haupt-VersionWorkspace.

Das gebildete DesignObjekt referenziert das DB-Objekt als das initiale Ausgangsobjekt. Die Registrierung im Versionsmanagement erfolgt über die Bildung der initialen Version im Haupt-VersionWorkspace. Diese initiale Version enthält keine Änderungsoperationen und ist somit identisch mit dem DB-Objekt. Die Bildung dieser initialen Version führt zur Sichtbarkeit des DesignObjektes im Haupt-VersionWorkspace. Ausgehend von der initialen Version kann die Modellierung des Datenbankobjektes im Versionsmanagement durch Revisionsversionen oder durch eine Alternativ-Version in einem Nachfolger-VWS erfolgen.

Überführung neuer Objekte in das Versionsmanagement: Die Arbeit an 3D-Modellen im Versionsmanagement erfolgt nicht ausschließlich durch die Weiterentwicklung der bestehenden DesignObjekte des Versionsmanagements. Vielmehr werden Änderungen und Ergänzungen am 3D-Modell zu neuen Objekten im Modell und zur Entfernung von existierenden Objekten im Modell führen. Hierfür muß die Möglichkeit bereitgestellt werden, neue Objekte von externen Applikationen, z.B. dem Modellierungssystem GOCAD, als neue DesignObjekte in das Versionsmanagement zu überführen. Für diese neuen DesignObjekte existieren keine vorhandenen DB-Objekte im Datenbanksystem.

Um ein neues Objekt von externen Applikationen in das Versionsmanagement als neues DesignObjekt zu überführen, muß für dieses externe Objekt ein DesignObjekt mit initialer Version gebildet werden. Im Versionsmodell benötigen DesignObjekte eine Referenz auf ein initiales DB-Objekt, von dem ausgehend die Änderungen in den Versionen als Änderungsoperationen verwaltet werden. Hierfür muß das externe Objekt im Versionsmanagement als initiales DB-Objekt im Datenbank-Objektmodell gespeichert werden. Die Erzeugung des DesignObjektes und der initialen Version findet im Haupt-VersionWorkspace statt und die initiale Version wird zum eigentlichen Arbeits-VersionWorkspace, in den das externe Objekt importiert werden soll, propagiert.

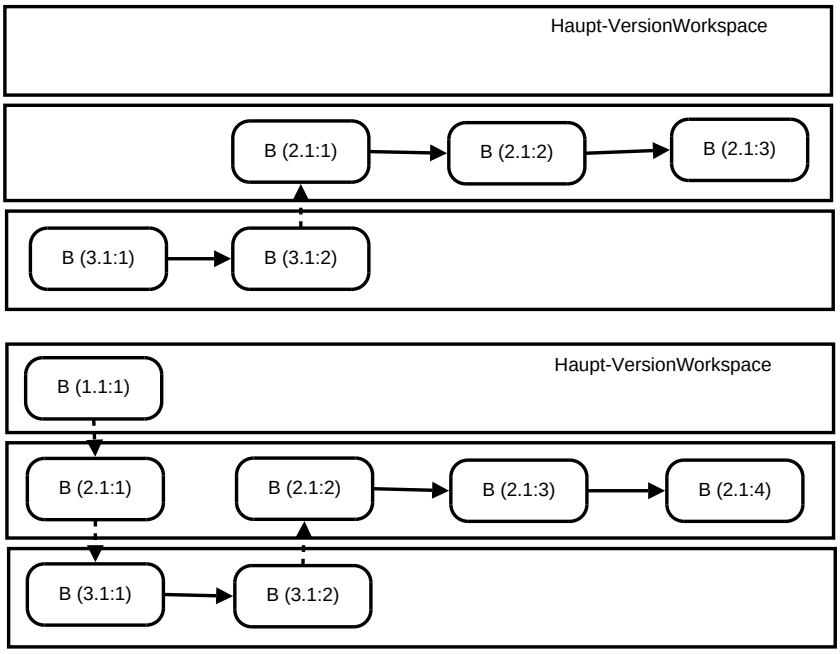

Abbildung 5.14: Integration neuer DesignObjekte in das Versionsmanagement (oben: Integration in den Arbeits-VWS; unten: Integration in den Haupt-VersionWorkspace)

Eine Erzeugung des DesignObjektes und der initialen Version im Haupt-VersionsWorkspace mit anschließender Propagierung in den aktuellen VersionWorkspace ist nötig, um die Ver-

sionshistorie für die Algorithmen konsistent zu halten. Abbildung 5.14 stellt die Integration externer Objekte in den Arbeits-VWS (obere Graphik) und den Haupt-VersionWorkspace (untere Graphik) dar. In der oberen Darstellung wird das externe Objekt direkt in den Arbeits-VWS mit dem erzeugten DesignObjekt und der initialen Version integriert (initiale Version B 3.1:1). Nach einer ersten Bearbeitung (Revisionsversion 3.1:2) erfolgt eine Migration in den übergeordneten VersionWorkspace. Diese migrierte Version stellt die initiale Version in diesem VersionWorkspace in der linearen Versionshistorie dar. Ausgehend von dieser Version können weitere Revisionen im VersionWorkspace erfolgen.

Die Inkonsistenz in der Versionshistorie tritt auf, wenn beispielsweise die Version B 2.1:3 in das Objektmodell konvertiert werden muß: Die Beschaffung der Versionslinie des DesignObjektes B endet nach dem beschriebenen Algorithmus (siehe Pseudocode 5.2) bei Version B 2.1:1, die die erste Version in diesem VersionWorkspace darstellt. Diese Version ist jedoch nicht die initiale Version des DesignObjektes und enthält somit keine Referenz auf das initiale DB-Objekt für den Konvertierungsalgorithmus.

Erfolgt die Integration externer Objekte in das Versionsmanagement ausgehend vom Haupt-VersionWorkspace, wie in der unteren Darstellung der Abbildung 5.14 dargestellt, existiert immer eine Versionslinie hin zu der initialen Version des DesignObjektes im Haupt-VersionWorkspace.

5.5.6 Konfliktverarbeitung

Konflikte treten im Versionsmanagement nur bei der Propagierung von Versionen eines DesignObjektes in den Vorgänger-VersionWorkspace auf. Ziel der Propagierung ist die Bildung einer Merge-Version zwischen der aktuellen Revisionsversion im Arbeits-VersionWorkspace und der letzten Revisionsversion im Vorgänger-VWS. Die Auflösung der auftretenden Konflikte zwischen diesen beiden Versionen erfolgt durch die Nutzer des Versionsmanagements. Das Versionsmanagement stellt dabei Funktionalität zur Verfügung um Konflikte zu erkennen und kann zur Unterstützung der Anwender diese auch graphisch aufbereitet darstellen. Die Anwender müssen etwaige Konflikte zuerst auflösen, bevor eine Merge-Version erstellt werden kann.

Eine Auflösung von Konflikten kann auf zwei Arten erfolgen: Zum einen kann eine neue Revisionsversion im Arbeits-VWS aufgebaut werden, in der die in Konflikt stehenden Änderungen zur Version im Vorgänger-VWS zurückgenommen bzw. abgeändert werden. Zum anderen existiert die Möglichkeit explizit die aktuelle Version in den Vorgänger-VWS, ohne eine Konfliktauflösung, zu propagieren. Diese Möglichkeit ist notwendig, wenn beispielsweise die alternative Entwicklungslinie für die weitere gemeinsame Modellierung im Vorgänger-VersionWorkspace benutzt werden soll. In diesem Fall werden Konflikte ignoriert und eine Merge-Version auf Basis der aktuellen Version im Arbeits-VWS gebildet. Welche Benutzer das Recht haben, Versionen im Vorgänger-VWS auch im Konfliktfall

zu überschreiben, muß über die Schreibrecht-Vergabe auf dem Vorgänger-VWS geregelt werden. Im allgemeinen hat nur ein Benutzer uneingeschränktes Schreibrecht auf einem VersionWorkspace. Dieser Benutzer muß mit den anderen Anwendern, die alternative Modellierungsrichtungen in den Nachfolger-VWS bearbeiten, Propagierungen in Kooperation durchführen.

Damit das Versionsmanagement Konflikte zwischen Versionen erkennen und Hilfe bei der Darstellung von Konflikten geben kann, müssen Regeln zur Konflikterkennung definiert werden. Bei Standard-Anwendungen von mobilen Datenbanken in relationalen Datenbanksystemen werden die Datenobjekte (Tupel) als eine Einheit der Änderung in der Konflikterkennung verwendet. Allgemeine Konfliktmuster sind hierbei beispielsweise die Löschung eines Tupels in der mobilen Datenbank und dessen Modifikation im Server-Datenbanksystem. Konflikte treten somit bei beidseitigem Zugriff auf ein gemeinsames Tupel im Datenraum auf. Bei der Verwaltung von geowissenschaftliche Objekten im vorliegenden Datenbanksystem DB3D und dem Versionsmanagement stellen die 3D-Objekte *Object3D* die Datenobjekte im Datenbanksystem dar. Prinzipiell können damit Konflikte wie in den Standard-Anwendungen als beidseitiger Zugriff auf ein gemeinsames 3D-Objekt definiert werden. Im Gegensatz zu den Standard-Anwendungen mit einfachen, kleinen Datenobjekten, verwenden geowissenschaftliche Anwendungen meist eine geringere Anzahl komplexer Datenobjekte (siehe auch Kapitel 1.3). Die Größe dieser Datenobjekte bedingt oftmals eine *konkurrierende* Bearbeitung an z.B. geometrisch disjunkten Bereichen. Eine solche Bearbeitung durch zwei unterschiedliche Bearbeiter in alternativen Modellierungslinien kann dabei – je nach Anwendung und Aufgabe – als Konflikt oder kein Konflikt angesehen werden. Im allgemeinen kann dieser Tatsache durch die Definition von anwendungsspezifischen Konfliktfällen Rechnung getragen werden. Hierfür kann auf den Aufbau der komplexen 3D-Objekte aus grundlegenderen Elementen zurückgegriffen werden. Aus der Sicht der Konflikterkennung können hierbei zum einen die Unterteilung in mehrere, topologisch zusammenhängende Komponenten und die Unterteilung dieser Komponenten in einzelne Simplexe (allgemein Basisobjekte) unterschieden werden.

Diese Unterteilung der komplexen 3D-Objekte kann zur Bewertung konkurrierender Bearbeitung und der Definition von Konfliktfällen verwendet werden. Auf Grundlage der Unterteilung der 3D-Objekte – in Komponenten und Simplexe von Komponenten – sind drei Arten der konkurrierenden Bearbeitung unterscheidbar:

- Konkurrierende Bearbeitung unterschiedlicher Komponenten eines Objektes
- Konkurrierende Bearbeitung einer Komponente ohne Geometrieüberschneidungen
- Konkurrierende Bearbeitung einer Komponente mit Geometrieüberschneidungen

In Abhängigkeit der spezifischen Anwendung kann einmal die konkurrierende Bearbeitung eines Objektes an unterschiedlichen Komponenten einen Konflikt darstellen und ein ande-

res Mal eine gewünschte, abgestimmte Möglichkeit der gemeinsamen Bearbeitung. Ebenso kann z.B. die konkurrierende Bearbeitung einer großen Schichtgrenzfläche an abgesprochenen, räumlich disjunkten Bereichen der gleichen Komponente keinen Konflikt darstellen. Einzig die konkurrierende Bearbeitung einer Komponente mit räumlichen Überschneidungen stellt grundsätzlich einen Konflikt zwischen Versionen dar.

Um eine flexible Verwendung des Versionsmanagements im geowissenschaftlichen Bereich zu ermöglichen kann eine definierbare Konfliktregelung verwendet werden. Die Regeln, was in einer bestimmten Anwendung einen Konflikt darstellt, werden dabei vom Administrator des Versionsmanagements für die jeweilige Anwendung festgelegt. Prinzipiell ist auch eine Festlegung auf Basis der DesignObjekte denkbar. Im folgenden wird eine möglichst einfache Definition von erlaubten konkurrierenden Bearbeitungen gegeben:

1. Konkurrierende Bearbeitung unterschiedlicher Komponenten
2. Räumlich disjunkte konkurrierende Bearbeitung einer Komponente

Definiert der Administrator für eine Anwendung die erste Konfliktregel, so werden konkurrierende Bearbeitungen an unterschiedlichen Komponenten eines Objektes nicht als Konflikt erkannt bzw. gewertet. Die Definition von Konfliktregeln kann dabei auch weiter unterschieden werden. Anstatt einen Konflikt auf Basis der allgemeinen Menge an Änderungen zu definieren, kann dies auch einzeln für die existierenden Änderungsoperationen wie z.B. *AddComponent* oder *DelComponent* erfolgen. Damit können beispielsweise die Konfliktregeln erweitert werden, so daß das Einfügen neuer Komponenten keinen Konflikt, das Löschen einer Komponente jedoch einen Konflikt darstellt. Analog kann eine feingliedrige Regelung für die Änderungsoperationen der Simplexe erstellt werden.

5.6 Mobile 3D Datenbank

5.6.1 Konzeption einer mobilen 3D Datenbank

Wie in Kapitel 2.4 bereits ausgeführt, existieren derzeit keine kommerziellen mobilen Datenbanken für die Verwaltung räumlicher Daten – auch nicht für zwei dimensionale Geodaten. Die existierenden relationalen mobilen Datenbanken, nutzbar für Standardanwendungen, sind derzeit nicht um neue Datentypen oder Indexstrukturen erweiterbar. Demgegenüber ermöglichen viele objekt-orientierte Datenbanken auch auf stark eingeschränkten mobilen Geräten eine Unterstützung beliebig komplexer Datenmodelle. Die Integration zwischen mobiler Datenbank und dem 3D-Datenbanksystem auf dem Server (Replikation, Synchronisation) muß hier durch den Datenbankentwickler bzw. Anwendungsentwickler erfolgen.

Die Konzeption der mobilen Datenbank auf objekt-orientierter Datenbanktechnologie hat daher, wie auch in der vorliegenden prototypischen Implementierung deutlich wird, wesentliche Vorteile. Unter anderem besteht die Möglichkeit einen Großteil des Datenmodells des Server-Datenbanksystems wiederzuverwenden. Im folgenden wird zuerst kurz auf die Unterschiede einer mobilen Datenbank gegenüber einem Server-Datenbanksystem eingegangen und anschließend zum Aufbau der mobilen Datenbank übergegangen.

5.6.1.1 Unterschiede zum Server-Datenbanksystem

Bei den Unterschieden zwischen mobiler Datenbank und dem Server-Datenbanksystem ist zwischen der Sicht des Nutzers und des Anwendungsentwicklers zu unterscheiden. Aus Sicht des Nutzers existieren Unterschiede nur in der Funktionalität, die von der mobilen Datenbank bereitgestellt wird. Für den Anwendungsentwickler dagegen existieren Unterschiede im Zugriff auf die Datenbank und in der Verwendung der mobilen Datenbank.

Aus *Nutzersicht* ist die Funktionalität zur Verwaltung von 3D-Geoobjekten dieselbe wie im Server-Datenbanksystem. Unterschiede existieren darin, daß die mobile Datenbank ein Ein-Benutzersystem darstellt und direkt mit dem Server-DBS gekoppelt ist. Die Benutzung als isoliertes Datenbanksystem auf einem mobilen Endgerät kann als Erweiterung erreicht werden. Die Koppelung mit dem Server-DBS hat zur Folge, daß die Nutzung der mobilen Datenbank immer als eine Abfolge der Interaktion mit dem Versionsmanagement zu sehen ist. Der Nutzer repliziert Datenbestände aus dem Versionsmanagement auf die lokale mobile Datenbank bzw. synchronisiert sie mit den letzten Änderungen. Danach erfolgt eine Bearbeitungsphase im *verbindungslosen* Zustand. Nach dieser Bearbeitungsphase schließt die Re-Integrationsphase an, in der die geänderten lokalen Datenbankobjekte als neue Revisionen bzw. neue lokale Datenbankobjekte als initiale Versionen in das Versionsmanagement überführt werden.

Aus der *Entwicklersicht* stellt die mobile Datenbank als eine sogenannte *embedded database* eine Schnittstelle (API - Application Programmer Interface) für den Anwendungsentwickler zur Interaktion bereit. Im Gegensatz zum Server-Datenbanksystem, bei dem jeglicher Zugriff auf die Datenbankfunktionalität über die Datenbankdienste erfolgt, wird die mobile Datenbank über diese API in die zu entwickelnden Anwendungen fest eingebunden. Durch die Einbettung der Datenbank erfolgt die Verwaltung der mobilen Datenbank durch die jeweilige Anwendung. Über die bereitgestellte Datenbank-API kann die Anwendung mobile Datenbanken öffnen und schließen und erhält Zugriff auf die enthaltenen Datenbankobjekte sowie die Transaktionssteuerung. Spezielle Funktionen für die Interaktion der mobilen Datenbank mit dem Server-Datenbanksystem ermöglichen die Kommunikation mit den Versionsdiensten des Server-Datenbanksystems. Hiermit können die Replikations-, Synchronisations- und Reintegrationsphasen mit dem Versionsmanagement kontrolliert werden.

5.6.1.2 3D-Funktionalität und Versionenverwaltung

Die mobile Datenbank beinhaltet das Geometriemodell (0D, 1D, 2D, 3D Simplexe) und das 3D-Objektmodell auf Basis der Simplizialen-*d*-Komplexe aus dem Datenbanksystem DB3D (vgl. Kapitel 4). Die Unterschiede zum Server-Datenbanksystem liegen in zwei Bereichen. Zum einen muß die mobile Datenbank in der Lage sein, die zusätzlichen Versionsinformationen für seine Datenbankobjekte zu verwalten. Jedes Datenbankobjekt in der mobilen Datenbank muß mit seinem DesignObjekt und seiner Ursprungsversion im Versionsmanagement des Serverdatenbanksystems assoziiert sein. Zum anderen ist die mobile Datenbank in Teilbereichen für mobile Geräte optimiert.

API-Erweiterungen: Die drei grundsätzlich notwendigen Erweiterungen der Schnittstellen zeigt Abbildung 5.15. Die mobile Datenbank enthält gegenüber dem Server-Datenbanksystem nur ein Projekt und einen Space (DefaultProject/DefaultSpace). Es erfolgt somit eine direkte Zuordnung der Datenbank zu einem versionierten Projekt bzw. Space des Server-Datenbanksystems. Hierfür wird das Interface *DBMS* erweitert, um den direkten Zugriff auf das Standardprojekt und den Standardspace, sowie die assoziierten Versionsmanagementattribute zu ermöglichen. Weiterhin muß in der Zugriffs-API die *Session*-Schnittstelle erweitert werden, um den direkten Zugriff auf die *MobileDBMS*-Schnittstelle bereitzustellen. Das *Object3D* Interface wird erweitert, um die zugehörigen Versionsattribute wie DesignObjekt-ID, Versionsnummer der übertragenen Version sowie Beschreibung und Zeitstempel zu speichern.

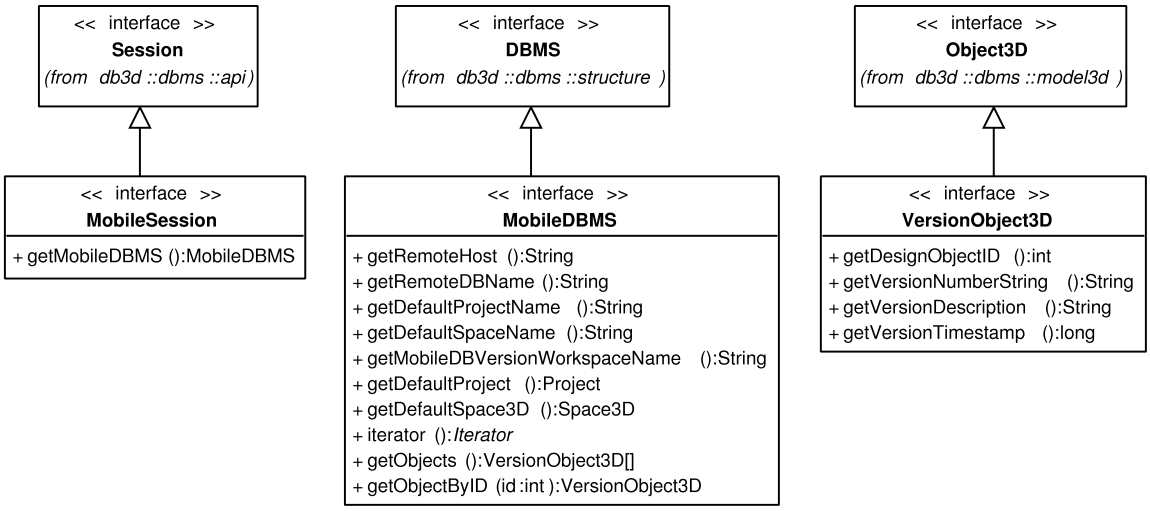

Abbildung 5.15: API-Erweiterungen der mobilen Datenbank

Für den Import/Export von Versionen in der mobilen Datenbank sowie der optimier-

ten Kommunikation zwischen mobiler Datenbank und Versionsmanagement werden zwei Erweiterungen vorgenommen. Zum einen werden die XML Import/Export Routinen erweitert, um Versionen verarbeiten zu können. Zum anderen wird das XML-Format um eine Beschreibung von Versionen anhand ihrer Änderungsoperationen erweitert. Hiermit kann eine effiziente Übertragung zwischen den Versionen der mobilen Datenbank und dem Versionsmanagement während der Synchronisations- und Reintegrationsphasen erfolgen. Für die initiale Replikationsphase, bei der die Versionen in die (leere) mobile Datenbank überführt werden, muß jedoch weiterhin die Übertragung auf den vollständigen Versionen basieren.

Optimierungen Die Verwaltung der Objekte in der mobilen Datenbank muß gegenüber der Implementierung des Server-Datenbanksystems optimiert für die eingeschränkten Ressourcen der mobile Geräte erfolgen. Dies beinhaltet die Optimierung auf bestimmte Benutzungsmuster (z.B. größtenteils keine Nutzung von Operationen auf Geometrieobjekten), die in der mobilen Datenbank einen anderen Schwerpunkt haben. Die Optimierungen müssen dabei in der internen Schicht der Datenbank erfolgen und bringen somit keine Änderung der Schnittstellen des Objektmodells mit sich. Ein Beispiel ist die Optimierung des internen räumlichen Indexes der Datenbankobjekte, der die einzelnen Simplexe indexiert. Hierauf wird im Rahmen der Evaluation (siehe Kapitel 6.1 Evaluation Objektaufbau) des Prototypen näher eingegangen.

Ein weiterer kritischer Punkt beim Objektaufbau stellt der Aufbau der internen Topologie der Simplizialen Komplexe dar. Eine Möglichkeit diesen Engpaß im Objektaufbau zu vermeiden bzw. zu verlagern ist die Verschiebung des Topologieaufbaus auf einen späteren Zeitpunkt nach dem Import des neuen 3D-Geoobjektes in die mobile Datenbank. Der Aufbau der internen Topologie wird damit aus der Import-Transaktion herausgenommen und läuft in einer internen Verwaltungstransaktion im Hintergrund ab. Die Verschiebung des Topologieaufbaus reduziert die Aufbauzeit der Datenbankobjekte wesentlich (siehe Tabelle 6.3 Kapitel 6.1). Entscheidender Nachteil dieser Methode ist jedoch, daß Operationen, die auf die interne Topologie in ihren Algorithmen angewiesen sind, nicht sofort nach dem Objekt-Import ausgeführt werden können.

5.6.2 Integration mit dem Server-Datenbanksystem

Die Kommunikation zwischen mobilen Datenbanken und dem Versionsmanagement des Server-DBS erfolgt über Dienste für die Versionsverwaltung. Diese Dienste ermöglichen den Zugriff auf die Metadaten des Versionsmanagements, das Holen und Senden von Versionen und die Migration von Versionen innerhalb der VersionWorkspace-Hierarchie.

Bei der Kommunikation zwischen mobiler Datenbank und Versionsmanagement kann zwischen erstmaliger *Replikation*, *Reintegration* und *Synchronisation* bestehender mobiler Da-

tenbanken unterschieden werden.

Bei der erstmaligen **Replikation** wird eine neue mobile Datenbank auf dem mobilen Gerät angelegt. Die für die Replikation spezifizierten Versionen werden zur mobilen Datenbank übertragen und dort als normale Datenbankobjekte mit Versionsinformationen importiert. Hierfür müssen aus den Versionen normale Datenbankobjekt-Repräsentationen mit zusätzlicher Versionsinformation für die Übertragung erzeugt werden (siehe Pseudocode 5.1).

Bei der **Reintegration** werden die Änderungen an den Datenbankobjekten der mobilen Datenbank als neue Versionen in das Server-Datenbanksystem überführt. Hierzu existieren zwei verschiedene Möglichkeiten. Zum einen kann das mobile Datenbanksystem das komplette Datenbankobjekt an das Versionsmanagement übertragen. Im Versionsmanagement auf dem Server erfolgt dann die Berechnung der erfolgten Änderungsoperationen im Bezug auf die letzte Version (siehe Pseudocode 5.4) und eine neue Version erzeugt. Eine Reduzierung der zu übertragenen Daten, sowie die Einsparung der Berechnung der Änderungsoperationen kann in diesem Schritt durch die alleinige Übertragung der erfolgten Änderungsoperationen in der mobilen Datenbank erreicht werden. Um diesen Ansatz nutzen zu können, müssen die in der mobilen Datenbank vollzogenen Änderungen an den Datenbankobjekten zusätzlich auch als die korrespondierenden Änderungsoperationen gespeichert werden. Dies kann durch eine interne Erweiterung des Datenmodells der mobilen Datenbank implementiert werden. Die im Datenmodell existierenden Update-Operationen werden in ihrer Implementierung dahingehend erweitert, daß nicht nur die Datenbankobjekte modifiziert, sondern gleichzeitig die jeweilige Änderungsoperation(en) zusätzlich gespeichert werden.

Die **Synchronisation** einer bestehenden mobilen Datenbank mit den eventuell erfolgten Änderungen im Versionsmanagement bringt die mobile Datenbank und das Versionsmanagement auf den gleichen Versionenstand. Für die Synchronisation sind wiederum beide Übertragungsmöglichkeiten denkbar. Im Gegensatz zur erstmaligen Replikation können bei der Synchronisation erfolgte Weiterentwicklungen an den DesignObjekten im Server-Datenbanksystem auch als Änderungsoperationen übertragen werden.

Grundsätzlich sollte das Versionsmanagements des Server-Datenbanksystems immer beide Ansätze zur Übertragung implementieren um auch externen Applikationen die Verwendung des Versionsmanagements zu ermöglichen. Diese sind, da sie keine Kenntnisse über das verwendete Änderungsmodell besitzen, auf den Austausch im Standard-Objektmodell (DB3D-XML Format) des Datenbanksystems angewiesen.

Mobile Datenbank als Client des Versionsmanagements: Wird die mobile Datenbank als spezieller Client gegenüber dem Versionsmanagement des Server-Datenbanksystems betrachtet, so ergibt sich eine natürliche Integration der mobilen Datenbank mit

dem Server-Datenbanksystem. Für jede mobile Datenbank existiert ein privater der mobilen Datenbank zugeordneter VersionWorkspace im Versionsmanagement. Dieser VersionWorkspace kann nur durch die mobile Datenbank benutzt werden, wodurch während der Replikations-, Synchronisations- und Reintegrationsphasen keine Konfliktsituationen auftreten können. Abbildung 5.16 zeigt die Struktur der Integration externer Clients und mobiler Datenbanken in das Versionsmanagement.

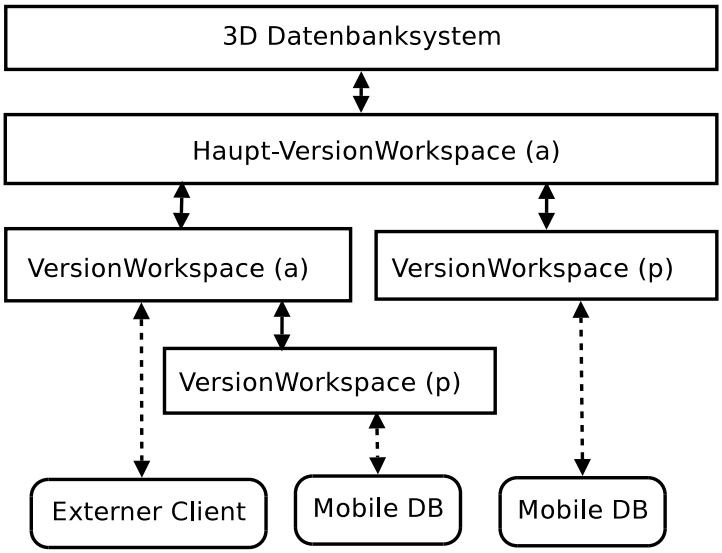

Abbildung 5.16: Struktur der Integration von Clients in das Versionsmanagement
Zugriffsrechte: (a) allgemeiner und (p) privater Zugriff

Die *normale* Datenbank beinhaltet in ihren Projekten und Spaces *fertige* Datenbankobjekte. Sie wird daher im folgenden als Releasedatenbank bezeichnet. Ausgehend von der Releasedatenbank können Projekte und ihre Spaces unter Versionskontrolle gestellt werden. Die Verwaltung der DesignObjekte und ihrer Versionen erfolgt dann in einer Hierarchie von VersionWorkspace-Repositories mit einem dem versionierten Space zugeordneten Haupt-VersionWorkspace. Soll ein Datenbankobjekt aus der Releasedatenbank zur weiteren Bearbeitung in den Versionsraum überführt werden, so wird für dieses Objekt ein DesignObjekt erzeugt und eine initiale Version im Haupt-VersionWorkspace abgelegt. Ausgehend von dieser initialen Version können nun neue Versionen als Revisionen oder als Alternativen in weiteren Nachfolger-VersionWorkspaces erzeugt und bearbeitet werden.

Ein normaler Arbeitszyklus mit einer mobilen Datenbank kann wie folgt beschrieben werden: In einer *Replikationsphase* werden ausgewählte Versionen aus dem der mobilen Datenbank zugeordneten VersionWorkspace in die mobile Datenbank übertragen bzw. mit dem aktuellen Stand synchronisiert. Die anschließende *Bearbeitungsphase* auf dem mobilen Gerät findet ohne eine Netzverbindung im offline Modus als lokale Transaktionen auf der

mobilen Datenbank statt. Ist die Bearbeitungsphase zu Ende, werden in der *Reintegrationsphase* die Änderungen der mobilen Datenbank zum Versionsmanagement übertragen und dort Revisionen der zuvor replizierten Versionen erzeugt.

Die Propagierung/Migrierung von Versionen aus dem der mobilen Datenbank zugeordneten VersionWorkspace mit eventuellen Konfliktauflösungen durch die VersionWorkspace-Hierarchie erfolgt durch die Migrationsdienste im online Modus. Der Schritt der Propagierung der Versionen im VersionWorkspace der mobilen Datenbank kann bei stabiler Netzwerkverbindung erfolgen. Diese kooperative Phase der Propagierung von Versionen in der VWS-Hierarchie unterscheidet sich dabei nicht zwischen mobilen Datenbanken und externen Applikationen.

Kapitel 6

Evaluation

Das folgende Kapitel evaluiert das Konzept und den Prototypen einer über Versionsmanagement eingebundenen, mobilen Datenbank zur Verwaltung geowissenschafticher dreidimensionaler Daten. Es wird zuerst auf Laufzeitaspekte des Server-Datenbanksystems und der mobilen Datenbank mit 3D-Testdaten eingegangen und anschließend die Einbindung der mobilen Datenbank über das Versionsmanagement anhand einer geowissenschaftlichen Beispielanwendung diskutiert.

6.1 Laufzeitverhalten

Laufzeitverhalten für komplexe 3D-Operationen: Die höhere Komplexität von 3D-Objekten ergibt erwartungsgemäß deutlich höhere Laufzeiten bei der Ausführung von geometrischen Operationen als bei vergleichbaren Operationen auf 2D-Objekten. Um ein Gefühl für die Laufzeiten von Operationen mit realen geowissenschaftlichen 3D-Modellen im 3D-Datenbanksystem zu vermitteln, wird im folgenden das Laufzeitverhaltens des komplexen 3Dto2D-Dienstes (vgl. Abschnitt 4.4.3) evaluiert und bewertet.

Das verwendete 3D-Modell (siehe Kapitel 3.3.1) *GESAMT* enthält 10 Schichtgrenzflächen und ein digitales Geländemodell als oberste Deckschicht und besteht aus 129.806 Simplexen. Die Ergebnisse stellen Durchschnittswerte mehrerer Berechnungsabfolgen verschiedener Profilschnitte (Profilschnittsequenz 1 und 2) dar. Die Abfolgen der Profilschnitte wurden durch wiederholte Ausführung des 3Dto2D-Dienstes mit einer gleich orientierten Ebene in einem Abstand von 100 Metern auf dem 3D-Modell berechnet.

In Tabelle 6.1 sind die Laufzeiten einzelner Basis-Dienste des 3Dto2D-Dienstes aufgeführt. Die benötigte Zeit für den Basis-Dienst *PlaneCutOperation* beträgt durchschnittlich um die 15 Sekunden pro Profilschnitt. Die anschließende Ausgabe des Profilschnittes als XML

und dessen Konvertierung über XSLT zu GML hängt von der Anzahl der in der Schnittmenge vorhandenen Elemente (die die Ebene schneiden) ab. Die Basis-Dienste für die Projektion (*PlaneProjectionOperation*) der weiteren Information auf die Profilebene (wie z.B. Bohrprofile) und für die affine Transformation des Ergebnisses in die XY-Ebene (*AffineTransformationOperation*) sind hier nicht aufgeführt, da die Ausführungszeiten dieser Basis-Dienste vernachlässigbar sind gegenüber der benötigten Zeiten für den Ebenenschnitt und die XSL-Transformation (vgl. Tabelle 6.2).

Basis-Dienst Zeiteinheit	PlaneCut Ø in ms	Simplexe Ø Anzahl	XML Ausgabe Ø in ms	XSLT Ø in ms
Profilschnittsequenz 1	15.565	1.441	186	3.812
Profilschnittsequenz 2	15.363	1.180	149	2.085

Tabelle 6.1: Durchschnittliche Laufzeiten verschiedener Basis-Dienste im 3Dto2D-Dienst

Tabelle 6.2 zeigt dagegen die Laufzeiten der Basis-Dienste *PlaneCutOperation*, *PlaneProjectionOperation* und *AffineTransformationOperation* für die beiden Profilsequenzen bezogen auf ein einzelnes Simplex. In den Evaluationsergebnissen ist hier deutlich die Auswirkung des zugrundeliegenden objekt-orientierten Datenbanksystems mit client-seitiger Cache-Architektur erkennbar. Der erste Durchlauf jeder Abfolge von Profilschnitten benötigt deutlich mehr Zeit als die nachfolgenden Profilschnittberechnungen, da Teile der internen Indexstrukturen und der simplizialen Komplexe bereits im Client-Cache vorhanden sind. Die Auswirkungen des Caches lassen die Zeiten für den ersten Profilschnitt und die folgenden Profilschnitte ungefähr um den Faktor 2 differieren.

	Basis-Dienst	Erstes Profil t/Simplex in ms	Folgende Profile t/Simplex in ms
Profilschnittsequenz 1	PlaneCut	17,63	8,45
	PlaneProjection	0,41	0,32
	AffineTransformation	0,04	0,03
Profilschnittsequenz 2	PlaneCut	20,78	10,08

Tabelle 6.2: Laufzeiten pro Simplex der Ergebnismenge für verschiedene Basis-Dienste

Bei der Profilschnittsequenz 2 sind hier nur noch die Messungen für den Basis-Dienst *PlaneCutOperation* aufgeführt, da die Ausführungszeiten der anderen Basis-Dienste des 3Dto2D-Dienste gegenüber der Gesamtlaufzeit vernachlässigbar sind. Weitere Tests mit dem 3D-Modell GESAMT und anderen Modellen zeigten, daß die Zeit pro mit der Ebene schneidenden Simplex für den PlaneCutOperation Basis-Dienst immer zwischen 6 bis 14 ms liegt. Die Spanne läßt sich hierbei mit der variierenden Komplexität der Schnittbe-

rechnung aufgrund unterschiedlicher Lagen der Simplexe zu der Ebene im 3D-Raum und der hohen Anzahl von Spezialfällen in der Berechnung erklären.

Die Evaluation zeigt, daß neben der Schnittoperation auch die XSL-Transformation der Ergebnisse in das Endformat den Hauptteil der Gesamtdauer des Dienstes ausmacht. Der XSLT-Ansatz erlaubt jedoch eine große Flexibilität und die Unterstützung verschiedener XML und GML verwandter Graphikformate (GML, SVG, Herstellerspezifische XML-Formate) als Ausgabeformat. Weiterhin ist anzumerken, daß die in der Evaluierung verwendete Implementierung der Basis-Dienste des 3Dto2D-Dienstes beliebig im Raum orientierte 3D-Ebenen prozessieren können. Da die Implementierung nicht für den in der Evaluation verwendeten achsenparallelen Spezialfall optimiert wurde, sind die erhaltenen Ergebnisse auch für beliebig im Raum orientierte Ebenen gültig.

Evaluation des Objektaufbaus in der mobilen Datenbank: Wie im Abschnitt 5.6 ausgeführt, bestimmt der initiale Objektaufbau zu großen Teilen das Laufzeitverhalten der mobilen Datenbank. Dies trifft vor allem auf die erstmalige Replikation von 3D-Objekten zur mobilen Datenbank zu. Spätere Synchronisationen mit dem Versionsmanagement des Server-Datenbanksystems können effizient auf Basis der Änderungsoperationen unterstützt werden.

Im Prototypen der mobilen Datenbank wurde eine Optimierung des internen räumlichen Indexes der Datenbankobjekte vorgenommen. Die Optimierung erfolgte unter der Zielsetzung einer verbesserten Aufbauzeit für die Datenbankobjekte. Die Verbesserung der Aufbauzeit des internen räumlichen Indexes wird durch die Ersetzung des im Server-Datenbanksystem verwendeten R*-Baum Indexes durch eine Octree-Implementierung erreicht. Nachteil dieser Ersetzung sind höhere Zugriffszeiten auf die einzelnen Elemente der Simplizialen Komplexe gegenüber dem Zugriff über einen R*-Baum Index. Dieser Nachteil wird vor allem in komplexen Operationen zwischen Datenbankobjekten wie z.B. Schnittberechnungen sichtbar, die auf eine effiziente räumliche Vorselektion bzw. räumlichen Verbund angewiesen sind. Für eine mobile Datenbank kann diese Laufzeitverschiebung jedoch in Kauf genommen werden, da im normalen Betrieb keine komplexen Berechnungen zwischen Datenbankobjekten durchgeführt werden. Komplexere geometrische Operationen werden im Allgemeinen auf dem Server-Datenbanksystem durchgeführt. Die Tests, unter Berücksichtigung verschiedener Knotenladungen beim R*-Baum bzw. maximaler Füllung einer Zelle beim Octree, haben eine Verbesserung der Laufzeit des Indexaufbaus bei Nutzung des Octrees um den Faktor 5 ergeben.

Tabelle 6.3 zeigt die benötigten Zeiten (in Sekunden) für den Aufbau künstlich generierter 3D-Objekte in der mobilen Datenbank. Die benötigte Zeit ist aufgeschlüsselt nach dem Aufbau des Objekt-Indexes, dem Aufbau der Topologie und der benötigten Commit-Zeit des zugrundeliegenden objekt-orientierten Datenbanksystems. Die Testplatform ist

ein Sharp Zaurus® PDA mit einer XScale 255 CPU und 64 MB RAM. Die angegebenen Zeiten stellen das Mittel über fünf Testläufe dar.

# Dreiecke im Objekt	Aufbau Octree in Sekunden	Aufbau Topologie in Sekunden	Commit in Sekunden	Gesamt in Sekunden
2.500	25	86	19	130
5.000	57	180	42	279
7.500	83	268	60	411
10.000	108	374	76	558
20.000	240	742	166	1.178

Tabelle 6.3: Aufbauzeiten für 3D-Objekte in der mobilen Datenbank auf einem PDA

Die Optimierung der mobilen Datenbank durch die Nutzung des Octree-basierten Objekt-Indexes reicht nicht aus, um auf einem PDA eine – aus Sicht des Laufzeitverhaltens – akzeptable mobile Datenbank bereitzustellen. Selbst die Umsetzung weiterer Optimierungen, wie beispielsweise die in Kapitel 5.6.1.2 für die interne Topologie beschrieben, lassen kein wirklich akzeptables Laufzeitverhalten erwarten. Die Ergebnisse der Evaluation lassen lediglich den Schluß zu, daß es möglich ist ein mobile Datenbank für komplexe 3D-Objekte auf einem PDA zu betreiben, jedoch mit den derzeitig verfügbaren Geräten nur mit sehr kleinen 3D-Objekten bzw. inakzeptablen Zeitverzögerungen. Die derzeitig verfügbaren Geräte besitzen hier noch deutliche Limitierungen in der CPU-Leistungsstärke und vor allem langsamen Haupt- und Sekundärspeicher (Speicherkarten). Es kann jedoch davon ausgegangen werden, das zukünftige Entwicklungen in der Hardware eine mobile 3D-Datenbank schon bald ermöglichen werden (vgl. Tabelle 6.4).

# Dreiecke im Objekt	Aufbau Octree in Sekunden	Aufbau Topologie in Sekunden	Commit in Sekunden	Gesamt in Sekunden
10.000	2,2	4,6	1,6	8,4
20.000	4,8	9,2	2,4	16,4
50.000	11,4	23,0	5,2	39,6
100.000	23,1	45,9	11,6	80,6

Tabelle 6.4: Aufbauzeiten für 3D-Objekt in der mobilen Datenbank auf Subnotebook

Tabelle 6.4 zeigt den gleichen Objektaufbau auf einem Subnotebook (IBM X40, 512 MB RAM). Wie zu sehen ist erfolgt der Objektaufbau eines 3D-Objektes mit 20.000 Simplexen in 16,4 Sekunden, gegenüber 1.178 Sekunden auf dem PDA. Neben dem deutlich schnelleren Objektaufbau ist mit einem Subnotebook als mobilen Endgerät auch die Verwaltung von deutlich komplexeren 3D-Objekten mit 100.000 und mehr Simplexen möglich.

6.2 Ergebnisse mit einer realen Anwendung

Für die Evaluation der prototypischen mobilen Datenbank in Verbindung mit dem Versionsmanagement des Server-Datenbanksystems wurde eine graphische Anwendung zur Verwendung und Steuerung der mobilen Datenbank und der Synchronisation mit dem Server-Datenbanksystem implementiert.

6.2.1 Testapplikation

Eine graphische Nutzerschnittstelle bietet die Möglichkeit zur Interaktion mit den Versionsdiensten des Server-Datenbanksystems und integriert den Prototyp einer mobilen Datenbank (vgl. Kapitel 5.6) als *embedded database*. Der Zugriff auf die mobile Datenbank erfolgt über die beschriebene Programmierschnittstelle (API).

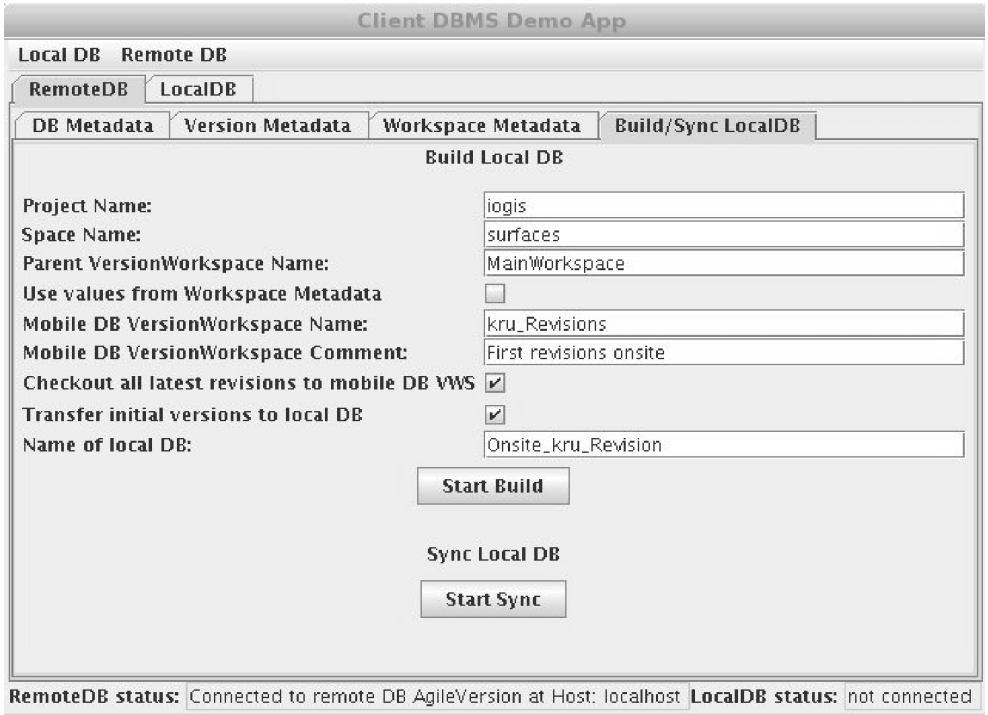

Abbildung 6.1: Aufbau bzw. Synchronisation der mobilen Datenbank

Die Applikation beschränkt sich auf die Steuerung der mobilen Datenbank und die Interaktion mit dem Versionsmanagement, sowie die vergleichende Visualisierung von lokalen Objekten mit Versionen des Versionsmanagements. Die Modellierung von Updates der 3D-Geoobjekte wird von externen Modellierungsprogrammen (z.B. GOCAD) erledigt. Anschließend werden die geänderten 3D-Geoobjekte in der mobilen Datenbank aktualisiert. Abbildung 6.1 verdeutlicht die Oberfläche zum Aufbau einer mobilen Datenbank

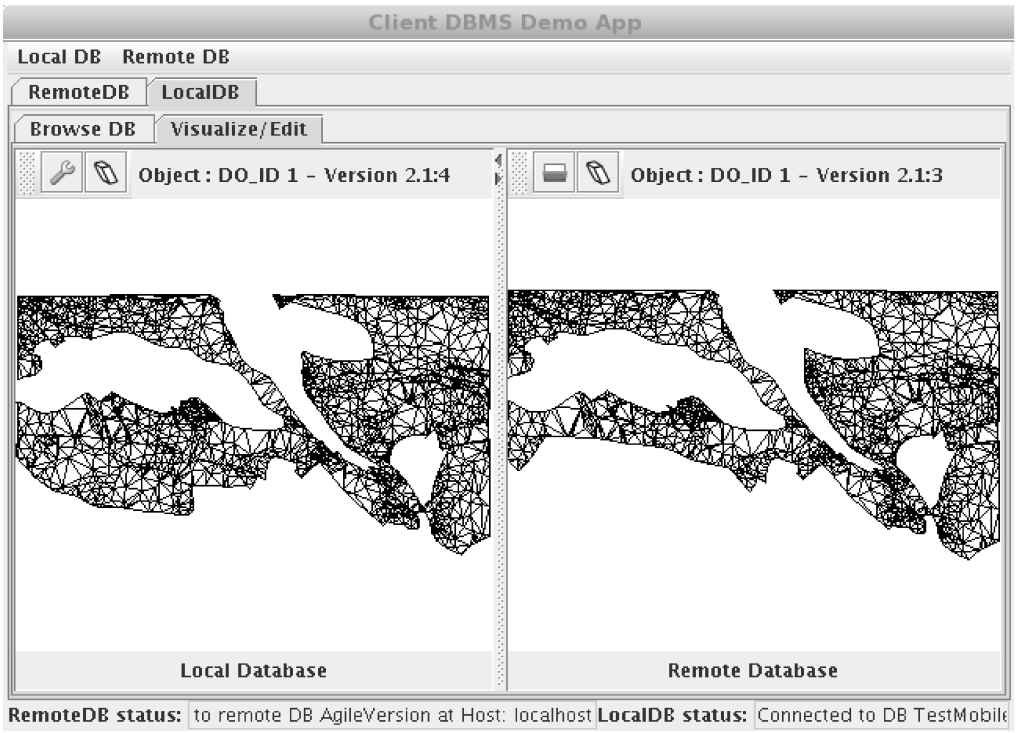

Abbildung 6.2: Vergleichende Visualisierung zwischen lokalem Objekt und Version

bzw. zur Synchronisation mit einem VersionWorkspace des Versionsmanagements. Abbildung 6.2 verdeutlicht die vergleichende Visualisierung zwischen einem lokal geänderten 3D-Geoobjekt und einer Vorgängerversionen im Versionsmanagement.

6.2.2 Aufnahme von Spaltenbewegungen

Zur Evaluation eines realen Arbeitsablaufes mit der mobilen Datenbank im Freiland wurde die Aufnahme von Bewegungen der Spalten am Albtrauf (siehe Anwendungsbeschreibung Kapitel 3.3.2) anhand einer Simulation nachvollzogen.

Die im Server-Datenbanksystem vorhandenen Modelldaten des Albtraufes (Abbildung 6.3) wurden zur weiteren Bearbeitung mit dem mobilen Datenbanksystem im Feld in das Versionsmanagement überführt. Der Aufbau der mobilen Datenbank mit den im Feld benötigten Spalten erfolgt über die graphische Applikation der mobilen Datenbank (vgl. vorherige Abbildung 6.1). Die Applikation erstellt den benötigten VersionWorkspace für die mobile Datenbank im Versionsmanagement auf dem Server-DBS und repliziert die initialen Versionen aus diesem VersionWorkspace zur mobilen Datenbank.

In der aufgebauten mobilen Datenbank sind anschließend die replizierten 3D-Objekte enthalten und können visualisiert (Abbildung 6.4) bzw. mit externen 3D-Modellierungs-

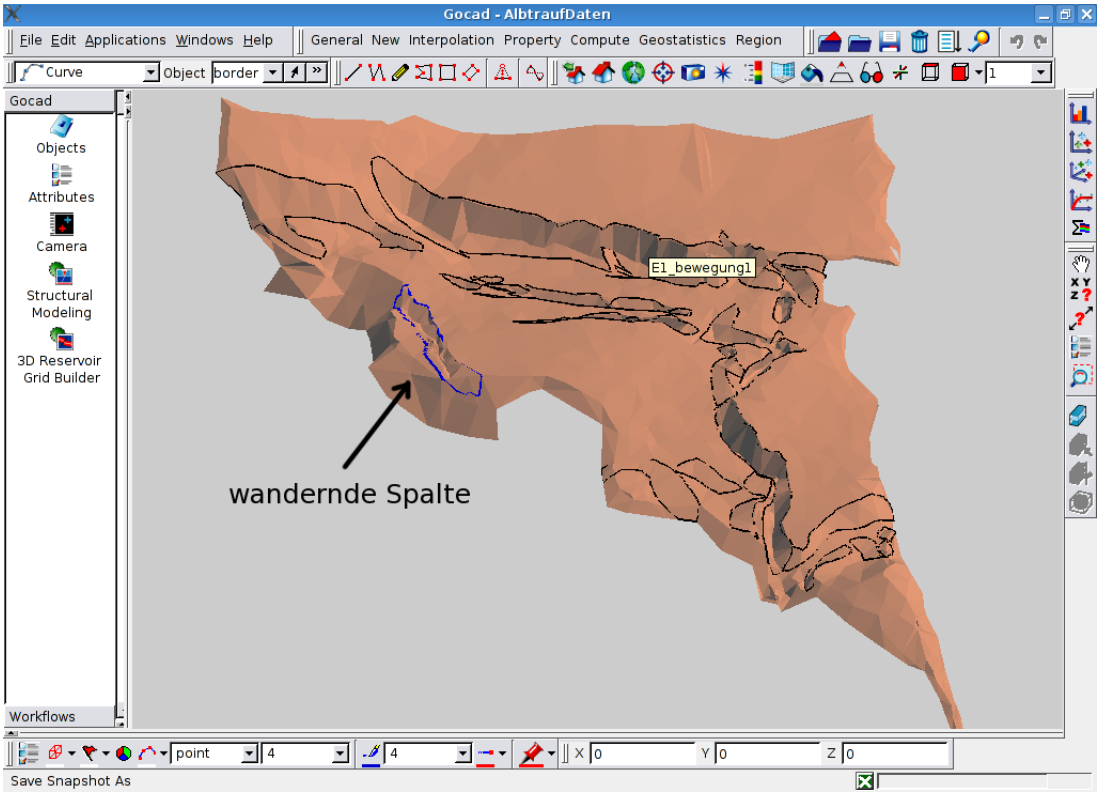

Abbildung 6.3: Albtrauf mit Spalten und wandernder Spalte in GOCAD

applikationen (z.B. GOCAD) entsprechend den realen Spaltenbewegungen abgeändert und in der mobilen Datenbank upgedated werden.

Für die Evaluation der Kopplung der mobilen Datenbank mit dem Versionsmanagement auf dem Server-DBS wurden 4 Spaltenbewegungen mit dem Modellierungssystem GOCAD simuliert und jede Spaltenaufnahme als Update in die mobile Datenbank übernommen. Anschließend erfolgte eine Reintegration des modifizierten Spaltenobjekts der mobile Datenbank mit dem Versionsmanagement. Nach Abschluß der Reintegration existiert das geänderte Spaltenobjekt der mobilen Datenbank als neue Revisionsversion im Version-Workspace des Versionsmanagements. Die graphische Oberfläche der mobilen Datenbank erlaubt – bei Verbindung zum Server-Datenbanksystem - bereits im Felde eine visuelle Überprüfung der Änderungen in der mobilen Datenbank gegenüber allen Vorgängerversionen des Versionsmanagements. Abbildung 6.5 zeigt den Vergleich zwischen dem lokal geänderten Spaltenobjekt (Version: 2.1:5) mit zwei Vorgängerversionen (2.1:2 in rot und 2.1:4 in blau dargestellt) aus dem Versionsmanagement.

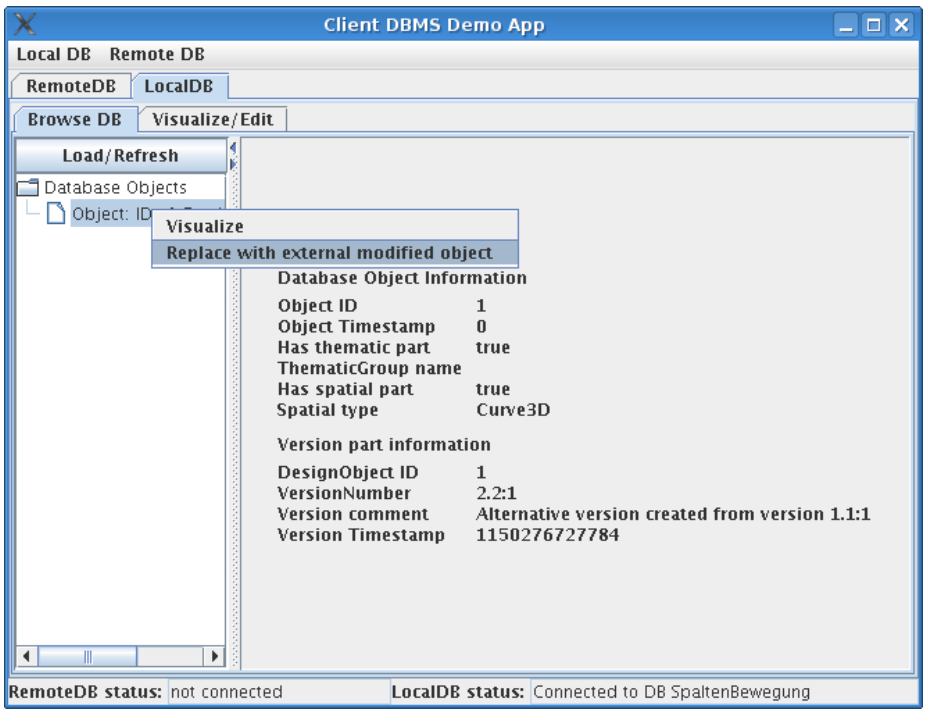

Abbildung 6.4: Visualisierung und Update von Objekten in der mobilen Datenbank

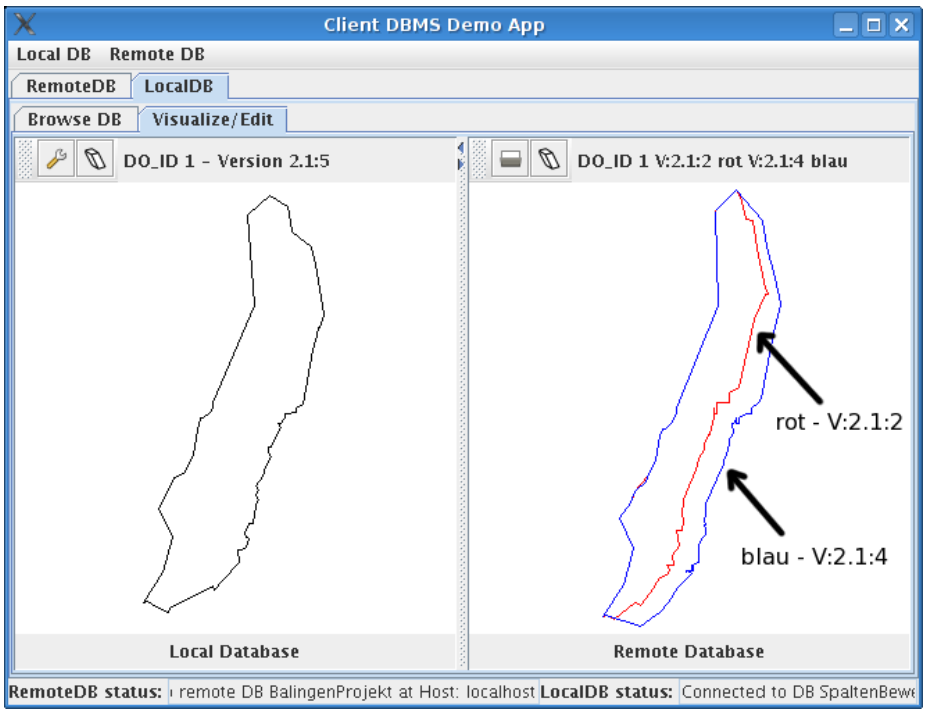

Abbildung 6.5: Vergleich mit Versionen im Versionsmanagement

Die fortwährende Reintegration der Änderungen der lokalen mobilen Datenbank mit dem serverseitigen Versionsmanagement ergibt dort eine lineare Versionshistorie in dem der mobilen Datenbank zugeordneten VersionWorkspace. Aufgrund der linearen Historie ist die Entwicklungsgeschichte der Objekte der mobilen Datenbank für den Anwender einfach nachzuvollziehen und die vergleichende Visualisierung bietet eine einfache Entwicklungskontrolle. Die Modifizierbarkeit der 3D-Objekte in der mobilen Datenbank – basierend auf den unterstützten Import/Export-Formaten – ermöglicht die Kopplung mit leistungsfähigen 3D-Modellierungssystemen für die Editierung und Weiterentwicklung von 3D-Objekten und ganzen Modellen. Die Änderungen der Spaltenobjekte im Evaluationsbeispiel erfolgte beispielsweise mit dem Modellierungssystem GOCAD auf einem Subnotebook.

Kapitel 7

Diskussion

Einschränkungen der Arbeit In der vorliegenden Arbeit erfolgten verschiedene Einschränkungen bei der Umsetzung der vorgestellten Konzepte. So erfolgte die prototypische Implementierung der Änderungsspeicherung, der Algorithmen für die Versionskonvertierung und der Import-/Export-Funktionalität nur für die in der Evaluation verwendeten Objektdimensionen (Segmentnetze und Dreiecksnetze). Ebenfalls sind in der prototypischen Umsetzung des Versionsmanagements keine Operationen zum Löschen von Versionen, DesignObjekten oder VersionWorkspaces enthalten.

Auf Seiten der mobilen Datenbank wurde von den ausgeführten Optimierungen die Ersetzung des internen R*-Baums durch einen Octree umgesetzt. Die Optimierung des Topologieaufbaus hätte den Rahmen dieser Arbeit gesprengt.

Unterschiede zum Versionsmodell von Schönhoff: Das verwendete Versionsmodell enthält grundsätzliche Unterschiede zum Versionsmodell von Schönhoff, das als Grundlage der Versionsmodellierung herangezogen wurde. Diese liegen zum einen in der Strukturierung der Versionshistorie in Verbindung mit der VersionWorkspace-Hierarchie, sowie darin, daß Beziehungen zwischen Versionen verschiedener DesignObjekte nicht unterstützt werden.

Im vorliegenden Versionsmodell ist die Struktur der Versionshistorie eines DesignObjektes an die VersionWorkspace-Hierarchie gebunden. Die Bildung einer neuen alternativen Entwicklungslinie (Revisionslinie) kann nur in einem neuen VersionWorkspace erfolgen. Innerhalb eines VersionWorkspaces existieren nur Revisionen der Versionen eines DesignObjektes. Damit erfolgt durch das VersionWorkspace-Konzept direkt eine logische Modellierungsstruktur für den Anwender. Im Modell von Schönhoff dagegen besitzt jeder Workspace[1] seine eigene Versionshistorie. Ein DesignObjekt enthält also für jeden

[1] Schönhoff verwendet den Begriff Workspace, anstelle von VersionWorkspace in der vorliegenden Arbeit

Workspace eine Versionshistorie, in dem eine Version des DesignObjektes existiert und damit dieses sichtbar macht. Innerhalb eines Workspaces können damit eigene alternative Versionslinien (sog. *Branches*) aufgebaut werden. Obwohl dieser Ansatz auf den ersten Blick flexibler erscheint, ist eine vorgegebene Strukturierung von Entwicklungsrichtungen – wie sie in dem Versionsmodell der vorliegenden Arbeit verfolgt wird – einfacher in der Anwendung durch die Nutzer. Hinzu kommt, daß beim Modell von Schönhoff eine Version in mehreren Workspaces sichtbar und bearbeitbar sein kann. Hierfür wird für jede Version in einem Workspace eine eigene Versionsnummer vergeben, damit Anwender Versionen in verschiedenen Workspaces und die daraus entstandenen Nachfolge-Versionen überhaupt auseinander halten können. Neben dem Problem der Übersichtlichkeit besteht ebenfalls die Gefahr, daß Bearbeiter nicht voneinander wissen, wenn sie in verschiedenen privaten Workspaces an der gleichen Version arbeiten. Auf diese Weise können alternative Entwicklungslinien entstehen, die in der Versionshistorie des DesignObjektes jedoch nicht als alternative Entwicklungsstränge erkennbar sind.

Während Schönhoff in seinem Modell komplexe DesignObjekte basierend auf Beziehungen zwischen Versionen verschiedener DesignObjekte unterstützt, existiert im vorliegenden Versionsmodell nur die Unterstützung von Konfigurationen über eine Menge von DesignObjekten. Diese Art von Konfiguration kann als *Ersatz* für komplexe DesignObjekte verstanden werden. Wie im Kapitel 5.3 Komplexe versus einfache DesignObjekte ausgeführt wird, ist im Bereich geowissenschaftlicher Anwendungen die Unterstützung von komplexen DesignObjekten in der Regel nicht notwendig. Komplexe Objektbeziehungen eines DB-Objekttyps können auch über einfache DesignObjekte und eine Speicherung von Änderungsoperationen modelliert werden. Da Schönhoff in seiner Arbeit nicht auf die Speicherungsweise (Voll-/Deltaspeicherung) von Versionen eingeht, ist anzunehmen, daß Versionen als *normale* Datenbankobjekte gespeichert werden. Die Komplexität in seinem Versionsmodell legt dies auch nahe, da aufgrund der vielfältigen Möglichkeiten von Versionsbeziehungen zwischen den verschiedene Versionshistorien eine Delta-Speicherung über Änderungsoperationen nicht verwendet werden kann. Für die Delta-Speicherung über Änderungsoperationen wird davon ausgegangen, daß eine Vorgänger-/Nachfolger-Beziehung zwischen nur zwei Versionen existiert, was im Modell von Schönhoff nicht garantiert werden kann.

Änderungsspeicherung & Progressive Meshes Geowissenschaftliche Anwendungen basieren oft auf einer relativ geringen Anzahl hoch komplexer Objekte (z.B. Untergrundmodelle großer Kohleflöze). Um eine effiziente und schnelle Übertragung und Visualisierung von solchen detaillierten Modellen zu erreichen, existieren verschiedene Verfahren aus dem Bereich der *Progressive Meshes* [58]. Dieses algorithmische Fachgebiet beschäftigt sich mit der geometrischen Vereinfachung, Kompression und progressiven Übertragung von polygonbasierten Darstellungen von Flächen und Körpern. Auch hier stellt das Datenmodell

der simplizialen Komplexe einen Teilbereich dar. Die Anwendung progressiver Übertragung bei Dreiecksnetzen [120] kann wie folgt beschrieben werden: Ein Dreicksnetz wird durch die Anwendung spezifischer Algorithmen in ein relativ grobmaschiges Dreiecksnetz plus eine Anzahl von Delta-Operationen zerlegt. Die Anwendung der Delta-Operationen in spezifischer Reihenfolge auf dem grobmaschigen Ausgangsnetz ermöglicht die Rekonstruktion des Netzes bis zum vollständigen Detaillierungsgrad. Für die effiziente Visualisierung über Netzwerke wird das grobmaschige Netz übertragen. Anschließend erfolgt die Übertragung der Delta-Operationen, welche auf der Clientseite angewendet eine Verfeinerung des Netzes bewirken. Vorteil der progressiven Übertragung ist, daß der Nutzer sofort eine Visualisierung erhält. Diese wird immer weiter verfeinert und kann dabei auch vom Nutzer unterbrochen werden, wenn beispielsweise der Detailgrad ausreichend ist. Für eine häufige Verwendung ist deshalb die Vorausberechnung der Delta-Operationen und die Speicherung dieser im Versionsmanagement denkbar.

Das vorgestellte Modell zur Repräsentation von 3D-Objekten mittels Änderungsspeicherung unterstützt nur elementare Änderungsoperationen basierend auf dem Datenmodell der simplizialen d-Komplexe. Diese Operationen sind atomar, d.h. sie können innerhalb einer Version (also die Menge an Operationen, die eine Version ausmacht) in beliebiger Reihenfolge auf dem 3D-Objekt angewendet werden, um das neue 3D-Objekt zu generieren. Demgegenüber sind die Delta-Operationen, wie sie im Bereich der progressive meshes verwendet werden, nicht atomar. Diese Operationen sind nur in einer definierten Anwendungsreihenfolge definiert. Weiterhin muß die Anwendung direkt auf dem zu ändernden Objekt erfolgen.

Grundsätzlich können progressive meshes im Versionsmanagement unterstützt werden. Die notwendige Beibehaltung der Reihenfolge in der Anwendung der Delta-Operationen muß jedoch in dem Modell zur Änderungsspeicherung speziell berücksichtigt werden. In der derzeitigen Implementierung erfolgt keine sequentielle Speicherung der Änderungsoperationen. So ist zum Beispiel die Bereitstellung eines speziellen VersionWorkspaces zur Speicherung benötigter vorabberechneter progressive meshes des geowissenschaftlichen Modells denkbar. Auf dem Versionsmanagement aufsetzende Anwendungen können damit Modelle der Release-Datenbank den Client-Anwendungen als progressive meshes den Client-Anwendungen bereitstellen.

Aufgetretene Probleme Die prototypische Implementierung des Versionsmanagements und der mobilen Datenbank zeigte zwei Problembereiche auf. Zum einen stellte sich heraus, daß die verwendeten geologischen Modelle und deren Bearbeitung mit dem Modellierungssystem GOCAD geometrische Ungenauigkeit mit sich brachten. Zum anderen traten durch die Verwendung objekt-orientierter Datenbanktechnologie Problematiken bei den Export-Operationen im Versionsmanagement auf.

Geometrische Ungenauigkeiten Die verwendeten geologischen 3D-Daten enthalten durch

die Georeferenzierung Koordinatenwerte mit mehreren gleichbleibenden Ziffern in den ersten Stellen der Rechts- und Hochwerte. Diese großen Koordinatenzahlen führten in verschiedenen Visualisierungen (GOCAD, X3D/VRML-Browser) zu Problemen, z.B. zu Artefakten in der Visualisierung bzw. nicht funktionierenden Auswahlwerkzeuge (z.B. Picking). Um eine bessere Genauigkeit zu erreichen sollte für die interne Speicherung im Datenbanksystem ein lokales Koordinatensystem angedacht werden. Hierfür müßte eine Transformation bei Import und Export der Daten erfolgen. Vorteile hierbei sind eine höhere Genauigkeit der Nachkommastellen in den Koordinaten durch eine Eliminierung der gleichbleibenden Ziffern in den Hoch- und Rechtswerten.

Degenerierte Simplexe Weitere geometrische Probleme lagen in der Erzeugung von degenerierten Simplexen durch die externen Modellierungsprogramme in deren Algorithmen. Innerhalb des Datenbanksystems erfolgt eine Überprüfung auf Validität und Regularität der Simplexe, um geometrische Fehler in Operationen innerhalb des Datenbanksystems auszuschließen. Die Überprüfung dieser Eigenschaften der Simplexe erfolgt dabei in Abhängigkeit der epsilon-Konstante. Die optimalen Definitionen von Validität (z.B. $length = 4 * epsilon$ bei einem 1-Simplex) wurden in der vorliegenden Arbeit nicht weiter untersucht, sondern vernünftig erscheinende Definitionen verwendet. Problematisch erwiesen sich beispielsweise Neuvermaschungen im Modellierungssystem bei der Erzeugung von Schnittflächen in Untergrundmodellen. Hierbei ergaben sich eine Vielzahl von stark degenerierten Simplexen. Um diese Modelldaten in die Datenbank überführen zu können, mußten zuerst im Modellierungssystem eine Vermaschungsüberprüfung und Korrektur erfolgen.

Objektexport von Versionen Die Referenzierung des initialen Datenbankobjektes im DesignObjekt führt zu einer Koppelung zwischen Datenbanksystem und Versionsmanagement. Die Operationen zur Konvertierung einer Version in das Objektmodell greifen in ihren Algorithmen auf das initiale Datenbankobjekt als Ausgangsobjekt zu, was je nach Transaktionsart zu Konflikten mit dem normalen Ablauf im Datenbanksystem führen kann. Weiterhin darf bei dieser Art der Koppelung keine Löschung dieser Objekte im Datenbanksystem erfolgen. Prinzipiell bestehen zwei Möglichkeiten zur Entkoppelung. Das initiale Datenbankobjekt wird als identische Kopie im Versionsmanagement gespeichert und durch das DesignObjekt referenziert oder die initiale Version enthält das Datenbankobjekt komplett in der Repräsentation durch Änderungsoperationen. Die Entkopplung führt damit zu einer Erhöhung des Speicherplatzverbrauches, wobei dieser als minimal im Bezug auf die Menge von Versionen bezeichnet werden kann.

Optimierungen Die prototypische Implementierung erfolgte ohne Verfolgung optimaler Laufzeiten, sondern eines schnellen Prototyping. Demzufolge ergeben sich noch einige Möglichkeiten Laufzeitverbesserungen sowohl im Datenbanksystem DB3D als auch im Versionsmanagement und der mobilen Datenbank zu erreichen.

So können beispielsweise Optimierungen bei der Konvertierung von Versionen in das Datenbank-Objektmodell aufgrund der Änderungsspeicherung erfolgen. Die Änderungsoperationen des Versionsmodells können auch innerhalb der Versionslinie angewendet werden, um beispielsweise die Menge an Änderungsoperationen zu minimieren bevor sie auf dem initialen Datenbankobjekt ausgeführt werden. Sei beispielsweise angenommen, daß in einer Version verschiedene Löschungen von Simplexen in der Netzkomponente A vollzogen werden und die nächste Version eine Löschung der kompletten Komponente A enthält. Dann kann die Menge der nötigen Änderungsoperationen zwischen den beiden Versionen konsolidiert werden, um nur die minimale Menge an Änderungen zu erhalten. Solche Konsolidierung der Änderungsoperationen zwischen beliebigen Versionen innerhalb der Versionshistorie sind weiterhin für die Änderungserkennung interessant.

Verwendbarkeit des Ansatzes Zum Abschluß der vorliegenden Arbeit stellt sich die Frage der Verwendbarkeit des Ansatzes von Versionsmanagement zur Einbindung von mobilen Datenbanken in Server-Datenbanksysteme. Grundlegend hierbei ist die Art der jeweiligen Anwendung. Im allgemeinen ist eine Verwendung von Versionen dann angebracht, wenn eine kooperative Anwendung betrachtet wird (siehe Kapitel 2.2.1). Es muß dabei sichergestellt werden, daß den Nutzern zumindest bei der Zusammenführung eventuell konkurrierender Versionen eine geeignete Kommunikationsmöglichkeit zur Konfliktauflösung zur Verfügung gestellt wird. Verschiedene unterstützende Tools zur Konfliktauflösung wie Merge-Assistenten für konkurrierende Versionen wurden in der vorliegenden Arbeit nicht betrachtet.

Aus Sicht der Laufzeit ist klar festzustellen, daß eine mobile 3D-Datenbank derzeitig keine aktzeptable Nutzbarkeit auf PDA-Geräten ermöglicht. Jedoch stellt die Datenbank nicht das alleinige Problem dar, da auch 3D-Modellierungssysteme zur Modifikation komplexer 3D-Daten im Gelände auf dem mobilen Endgerät lauffähig sein müssen. Die Zielgruppe mobiler, geländetauglicher Notebooks bietet jedoch sowohl von der Prozessorleistung, Arbeits- und Festplattenspeicherkapazität und den benötigten 3D-Graphikfähigkeiten alles für einen akzeptablen Betrieb Notwendige (vgl. Kapitel 6.1).

Wird die Änderungsspeicherung bei den Versionen verwendet, so kann bei entsprechender Ausnutzung in der Implementierung eine deutliche Reduktion der Synchronisationzeit und benötigten Übertragungsbandbreite zwischen der mobilen Datenbank und dem Versionsmanagement erreicht werden. Daneben kann zur optimalen Ausnutzung auch eine Übertragung basierend auf binär kodiertem XML bzw. binärer Serialisation der Änderungsoperationen untersucht werden. Die Möglichkeit einer optimalen Reduktion der zu übertragenen Datenmenge durch die Verwendung der Änderungsspeicherung ist vor allem vor dem Hintergrund des Einsatzes mobiler Funktechniken mit hohen Kostenfaktoren interessant.

Kapitel 8

Zusammenfassung und Ausblick

Aufgrund der jüngsten technischen Weiterentwicklungen mobiler Endgeräte ist der Einsatz von Datenbanken auf diesen Geräten realistisch geworden. Die vorliegende Arbeit beschäftigt sich mit den sich daraus ergebenden Möglichkeiten für die Realisierung mobiler, räumlicher Datenbanken. Der Schwerpunkt lag dabei auf der Verwaltung geowissenschaftlicher 3D-Daten mit Hilfe mobiler Datenbanken.

8.1 Zusammenfassung

Den Hintergrund für die vorliegende Arbeit lieferte das Projekt „Entwicklung von Komponenten-Software für den Internet-basierten Zugriff auf Geodatenbank-Dienste“ [17, 18]. Ziel des Projektes war es, eine Datenbankunterstützung für raum- und zeitbezogene Objekte (3D/4D) mit Internet-Zugriff über Geodatenbank-Dienste zu realisieren. Die vorliegende Dissertation erweitert die Untersuchungen hinsichtlich der Einbeziehung von Datenbanken auf mobilen Endgeräten. Es wird ein Konzept und dessen prototypische Implementierung für die Verwaltung von 3D-Daten mit mobilen Datenbanken in den Geowissenschaften vorgestellt.

Die Arbeit berücksichtigt dabei die Spezifika der Verwaltung geowissenschaftlicher 3D-Daten und Modelle. Diese sind insbesondere darin begründet, daß sich geowissenschaftliche Anwendungen vornehmlich mit dem nicht für direkte Beobachtungen zugänglichen Untergrund beschäftigen. Es werden meist *echte* 3D-Modelle benötigt, die immer auf Schätzungen geologischer Interpretation und fachlichem Hintergrundwissen basieren. Der Modellierungsprozeß geowissenschaftlicher 3D-Modelle ist dabei ein kontinuierlicher Prozeß der Interpretation der geologischen Meßdaten und deren Integration in das Modell. Eine Besonderheit stellt die gleichzeitige kooperative Arbeit mehrerer Gruppen am gleichen Modell dar – die Bearbeitungszeiten können hierbei sehr lange sein. Aufgrund der kooperativen

Modellentwicklung können Ansätze für eine mehrbenutzer-orientierte Bearbeitung mittels Versionsmanagement verfolgt werden.

Im Grundlagenkapitel wird zuerst auf die derzeitigen Ansätze zur Modellierung geometrischer 3D-Datenmodelle und den Stand der Forschung bei 3D-Datenbanksystemen eingegangen. Ein Schwerpunkt wird auf den Überblick existierender Ansätze und Forschungen auf dem Gebiet mobiler Datenbanken gelegt. Weiterhin werden die möglichen Verbindungszustände und Trennungsmodi zwischen mobiler Datenbank und Server-Datenbanksystem behandelt. Neben einer Übersicht über mögliche Architekturen mobiler Datenbanksysteme, wird auf die derzeitig verfügbaren Modelle zur Transaktionsunterstützung in mobilen Umgebungen eingegangen. Es zeigt sich, daß viele Modelle nur den Übergang zwischen Funkzellen in Mobilfunktechnologien behandeln. Der wichtige unverbundene Zustand bei mobilen Datenbanken wird größtenteils über Replikationsverfahren unterstützt. Abschließend erfolgt eine Darstellung der für die Arbeit wichtigen Versionsmodelle als Grundlage für das verwendete Versionsmanagement.

Die Einbeziehung mobiler Datenbanken als Clients innerhalb eines Datenbanksystems erfolgt ausgehend von der Beschreibung der Konzeption und der Implementierung des Server-Datenbanksystems für 3D/4D-Geodaten. Das Konzept basiert auf der Integration mobiler Datenbanken über ein Versionsmanagement im Server-Datenbanksystem. Vorteil des Versionsansatzes ist, daß keine automatisierte Konfliktauflösung durch das Server-Datenbanksystem bei der Reintegration der geänderten Geodaten erfolgt. Vielmehr werden die Änderungen als neue Versionen der ursprünglichen Datenbankobjekte im Versionsmanagement des Datenbanksystems abgelegt. Auftretende Konflikte zwischen Versionen werden von den Bearbeitern bei der Propagierung durch eine Hierarchie von Ebenen des Versionsmanagements in Kooperation aufgelöst.

Hierfür wird das Server-Datenbanksystem um ein Versionsmanagement erweitert. Das konzipierte Versionsmodell erlaubt eine intuitive Modellierung von Revisionen und alternativen Entwicklungen von 3D-Modellen in den Geowissenschaften. Die Möglichkeit der Repräsentation komplexer 3D-Geoobjekte im Versionsmanagement wird diskutiert und eine Umsetzung für das verwendete Datenmodell der simplizialen Komplexe beschrieben. Für die effiziente Speicherung und Verwaltung von Versionen wird ein Konzept der Versionsspeicherung über Änderungsoperationen aufgestellt. Neben der effizienten Speicherung ermöglicht diese Speicherungsart auch effiziente Algorithmen zur Konflikterkennung und Änderungsanalyse. Im weiteren Verlauf werden anschließend die benötigten Algorithmen und Operationen zur Konvertierung zwischen dem Datenmodell für Versionen und dem für Datenbankobjekte ausführlich beschrieben. Die Datenmodell-Konvertierung ermöglicht die Nutzung aller existierenden Operationen des Datenbanksystems auch mit Versionen der Datenbank-Objekte. Die Behandlung der benötigten Operationen im Versionsmanagement zur Verwaltung der Versionshistorien und die Diskussion der Konfliktverarbeitung schließt die Beschreibung des Versionsmanagements ab. Hierbei werden verschiedene Kon-

fliktarten auf 3D-Geoobjekten beschrieben und die Möglichkeit der Definition von Konfliktregeln je nach Anwendung vorgeschlagen. Weiter wird eine einfache Konfliktregelung für eine konkurrierende, jedoch kooperative Bearbeitung von 3D-Geoobjekten vorgestellt und Möglichkeiten zur Verfeinerung der Regeln aufgezeigt.

Ein weiterer Bereich der Arbeit beschäftigt sich mit der Datenbank auf den mobilen Endgeräten. Die Umsetzung der prototypischen Implementierung basiert dabei in weiten Teilen auf dem Quellcode der objekt-orientierten Server-Datenbank. Unterschiede bestehen in einer Erweiterung zur Verwaltung der Versionsinformationen der lokalen Datenbankobjekte und in den bereitgestellten Schnittstellen zur Nutzung der Dienste des Versionsmanagements auf dem Server-Datenbanksystem. Für die Optimierung auf die eingeschränkte Leistungsfähigkeit der mobilen Endgeräte werden Ansätze diskutiert und vereinzelt im Prototypen implementiert. Die Integration der mobilen Datenbank mit dem Server-Datenbanksystem beschreibt den generellen Ablauf eines Bearbeitungszyklus zwischen der mobilen Datenbank und dem Versionsmanagement.

Das Konzept des Versionsmanagements wird, basierend auf der prototypischen Implementierung der mobilen Datenbank und der Versionserweiterung des Server-Datenbanksystems, evaluiert. Die Evaluierung der Nutzbarkeit des Versionsansatzes erfolgt anhand eines realen Anwendungsszenarios aus den Geowissenschaften. Im Anwendungsgebiet finden Hangrutschungen mit tiefreichenden Felsbewegungen statt, die sich durch Spaltöffnungen an der Oberfläche zeigen. Diese Spalten unterliegen Veränderungen durch die Bewegungen der Teilschollen und werden in der mobilen Datenbank verwaltet. Auf diese Weise können Änderungen in den Spaltgeometrien im Gelände erfasst und als neue Version in das Versionsmanagement des Datenbanksystems überführt werden. Die Evaluation zeigt die prinzipielle Anwendbarkeit des Ansatzes bei der Modellierung und Verwaltung geowissenschaftlicher Prozesse.

Den Abschuß der Arbeit bildet die Diskussion der Konzepte und Ergebnisse der Evaluation. Es werden die Unterschiede des Versionsmodells zum Ausgangsmodell diskutiert und aufgetretene Probleme während der prototypischen Implementierung und der Evaluation angesprochen. Die Diskussion der Anwendbarkeit des Ansatzes zur Integration mobiler Datenbanken mit Server-Datenbanksystemen zeigt, daß diese vor allem für kooperative Anwendungen gegeben ist, wie sie häufig in den Geowissenschaften auftreten. Das Konzept der Versionsspeicherung über Änderungsoperationen eignet sich besonders für die meistens nur geringfügigen Veränderungen zwischen Versionsschritten. Die Umsetzung der Konzepte mit mobilen Datenbanken auf stark eingeschränkten Endgeräten wie PDAs führte jedoch nicht zum erhofften Ergebnis. Hier zeigte sich eine immer noch zu geringe Leistungsfähigkeit zur effizienten Nutzung einer 3D-Geodatenbank. Die Verwendbarkeit des Ansatzes bei Nutzung eines leistungsfähigeren Endgerätes, wie beispielsweise einem Subnotebook, konnte in der Evaluation jedoch bestätigt werden.

8.2 Ausblick

Die Umsetzung der prototypischen Implementierung erfolgte als objekt-orientiertes Datenbanksystem. Der entscheidende Vorteil dieser Implementierung liegt in der deutlich reduzierten Entwicklungszeit zur Umsetzung der Konzepte. Aufgrund der weitreichenden Verbreitung und Verwendung objekt-relationaler Datenbanken erscheint eine Implementierung der Konzepte als objekt-relationales Datenmodell als nützlich. Die entwickelten Konzepte sind unmittelbar auch im objekt-relationalen Modell umsetzbar. Problematisch gestaltet sich hier allerdings die Umsetzung der mobilen Datenbank, da bisher keine erweiterbaren objekt-relationalen mobilen Datenbanken auf dem Markt verfügbar sind. Auch die Umsetzung der Architektur für die Datenbank-Dienste auf eine standardisierte Webservice-Architektur würde eine bessere Integration in die heutigen IT-Strukturen und Unternehmensnetzwerke ermöglichen. Aus wissenschaftlicher Sicht stellt die konkrete Umsetzung des Konzepts der Datenbank-Dienste jedoch nur ein Implementierungsdetail dar und ist damit nicht von vordergründigem Interesse.

Dagegen erscheint die Erweiterung des Datenbanksystems durch die Verwendung generischer topologischer Repräsentationen (z.B. G-Maps) zur Modellierung komplexer 3D-Geometrien interessant. Diese Strukturen zur Beschreibung von Zellkomplexen [28, 76] beruhen auf mathematischen Konzepten der kombinatorischen Topologie und erlauben eine einfache, sehr generelle und weitgehend dimensionsunabhängige Repräsentation. Die Verwendbarkeit von G-Maps als generische topologische Repräsentation in Geodatenbanksystemen wird derzeit im DFG-Forschungsprojekt „Modellierung und Analyse der Topologie in Multiple Representation Databases“ [1] (siehe [126]) untersucht.

[1] Projektseite: www.igf.uos.de/projekte/abstraktion.html

Kapitel 9

Summary

The use of databases on mobile devices becomes realistic these days, due to the recent technical advancements in this area. The thesis concerns with the possibilities for the realization of mobile spatial databases. The emphasis of the thesis is thereby on the management of geoscientific 3D data by mobile databases.

The background of this thesis is provided by the German research project „Development of software components for the internet-based access to geodatabase services“ [17, 18]. The main goal of the project is to provide database support for space and time-related objects (3D/4D) for the internet-based access via geodatabase services. This thesis extends the work of the project by investigations regarding databases on mobile devices. A concept and its prototypical implementation for the management of 3D data with mobile databases in the geosciences is presented.

The work thereby considers the specific requirements of geoscientific 3D data and models. In particular these are reflected in the fact that geoscientific applications primarily deal with underground models, which are not accessible for direct observations. Furthermore, this usually means that *real* 3D models of the underground are needed. These are always based on estimations, geological interpretation and technical background knowledge. Therefore, the modelling process of geoscientific 3D models is best described as a continuous process of interpretation of the geological measuring data and their repeated integration into the model. A further characteristic represents the simultaneous cooperative work of several groups on the same model. Due to the cooperative work model a multi-user oriented development model can be pursued by means of version management.

The section on the research basics first presents the available data models for geometric 3D data and reviews the current state of research within the area of spatial 3D database systems. An emphasis is put on the overview of the available models and ongoing research in the area of mobile databases. Two different application classes of mobile applications are pointed out. Furthermore, the possible connection and disconnection states between

mobile databases and the server database system are discussed. An overview of possible architectures of mobile database systems is given. An important part is covered by the presentation of currently available models for transaction support in mobile environments. Many of these models only support transaction transitions during changing radio cells on movement. The important case of disconnected operation for mobile databases is mostly supported by the means of replication operation. Finally, a presentation of different aspects of version models as a basis for the later defined concept of version management is given.

The inclusion of mobile databases as clients within a database system takes place on the basis of an existing server database system for 3D/4D geodata. The concept is based on the integration of mobile databases by version management through the server database system. An advantage of this conept is that no automated dissolution of conflicts takes place during the reintegration of modified geodata. Rather the changes are stored as new versions of the original database objects in the version management on the server. Arising conflicts between versions are dissolved by the users in cooperation as the propagate versions through a hierarchy of levels in the version management.

The server database system is extended by version management. The conceived version model permits an intuitive modelling of revisions and alternative developments of 3D models in the geosciences. The possibility of the representation of complex 3D geoobjects in the version management is discussed. Furthermore, an implementation concept for the used data model of simplical complexes is given. A concept is defined which allows efficient storage and management of versions by storage of change operations between versions. Apart from efficient storage this concept also enables efficient algorithms of conflict detection and change analysis. In the following the necessary algorithms and operations are described in detail to convert between the data model of versions and of database objects. The possibility to convert between the data models makes the usage of existing operations of the database system also available to versions. The section closes by a discussion of the conflict management. The definition of conflict types on 3D geoobjects is discussed and the possibility of defining conflict rules on an application level is suggested.

A further part of the thesis describes the database for mobile devices. The prototypical implementation is based on the source code of the object-oriented server database system. Differences insist in an extension for the management of the needed version information of the local database objects and in the provided interfaces for the use of the services of the version management on the server. Different possibilities of optimization of the mobile database, to cope with the restrictions of mobile devices, are discussed and partly implemented in the prototype. A presentation of the integration of the mobile database with the server database system describes the general workflow of a session between the mobile database and the version management.

The concept of the version management is evaluated based on the prototype of the mobile database and the version management extension of the server database system. The

usability evaluation of the version concept is done on the basis of an application scenario from the geosciences. The effects of the landslide are visible by several cervices at the surface. These cervices are subject to changes by the movements of the underlying partial rocks. The geometrical changes in the cervices in this area are administered with the mobile database. Changes are captured in the field and stored as new versions in the version management. The evaluation shows the applicability of the principle concept for the modelling and management of data in geoscientific processes.

The thesis closes with the discussion of the given concepts and results of the evaluation. The arisen problems during the prototypical implementation and the evaluation are presented. The usability of the presented concept for the integration of mobile databases with server database systems is discussed. The conclusion is that the concept is particularly applicable to cooperative applications as frequently found in the geosciences. The concept of version storage by the use of change operations shows its suitability in geoscientific models that only change slightly between version steps related to their complete sizes. The usage of mobile databases on rather limited devices – such as PDAs – did not fulfil the expectations. It is shown that the devices are still not powerful enough to run a 3D geodatabase. However, the applicability of the concept could be confirmed by the evaluation using a more powerful mobile subnotebook.

Literaturverzeichnis

[1] A. Chance, D.G. T.: Smallworld GIS: An Object-oriented GIS - Issues and Solutions. In: *Proc. of the EGIS, Amsterdam*, 1990, S. 179–188

[2] Agarwal, S. ; Arun, G. ; Chatterjee, R. ; Speckhard, B. ; Vasudevan, R.: Long transactions in an RDBMS. In: *7th Annual Conference and Exhibition: A GEOSPATIAL ODYSSEY (GITA 2003)* Geospatial Information and Technology Association (GITA), 2003

[3] Agrawal, R. ; Buroff, S.J ; Gehani, N.H. ; Shasha, D.: Object Versioning in Ode. In: *Proceedings of the IEEE 7th Int. Conf. on Data Engineering.* Kobe, Japan, 1991, S. 446–455

[4] Ahn, H.K. ; Mamoulis, N. ; Wong, H.M.: A Survey on Multidimensional Access Methods / Institute of Information and Computing Sciences - University Utrecht. 2001 (UU-CS-2001-14). – Forschungsbericht

[5] Balovnev, O. ; Bode, T. ; Breunig, M. ; Cremers, A.B. ; Müller, W. ; Pogodaev, G. ; Shumilov, S. ; Siebeck, J. ; Siehl, A. ; Thomsen, A: The Story of the GeoToolKit - An Object-Oriented Geodatabase Kernel System. In: *GeoInformatica* 8 (2004), Nr. 1, S. 5–47

[6] Barbara, D.: Mobile Computing and Databases - A Survey. In: *IEEE Transactions on Knowledge and Data Engineering* Bd. 11, 1999, S. 108–117

[7] Batty, P. M.: Version Management Revisited. In: *Proceedings of GITA Annual Conference.* Florida, 2002

[8] Beckmann, N. ; Kriegel, H.-P. ; Schneider, R. ; Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: *Proc. 1990 ACM SIGMOD Intern. Conf. on Management of Data*, 1990, S. 322–331

[9] Bergstaesser, T. ; Bernstein, P.A. ; Pal, S. ; Shutt, D.: Versions and Workspaces in Microsoft Repository. In: *Proceedings of the 1999 ACM SIGMOND Int. Conf. on Management of Data* ACM, 1999, S. 532–533

[10] Bernstein, P.A. ; Giunchiglia, F. ; Kementsietsidis, A. ; Mylopoulos, J. ; Serafini, L. ; Zaihrayeu, I.: Data Management for Peer-to-Peer Computing: A Vision. In: *WebDB Workshop on Databases and the Web 2002*. Madison, USA, June 2002, S. 6

[11] Bertino, E. ; Ooi, B.C. ; Sacks-Davis, R. ; Tan, K.-L. ; Zobel, J. ; Shidlovsky, B. ; Catania, B.: *Indexing techniques for advanced database systems*. Kluwer, 1997 (Kluwer Int. Series on Advances in Database Systems)

[12] Biliris, A.: Database Support for Evolving Design Objects. In: *26th ACM/IEEE Design Automation Conference*, 1989, S. 258–263

[13] Blanken, H.: Implementing version support for complex objects. In: *Data & Knowledge Engineering* (1991), Nr. 6, S. 1–25

[14] Breunig, M.: *Integration räumlicher Informationen für Geo-Informationssysteme*, Mathematisch-Naturwissenschaftliche Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn, Diss., 1995

[15] Breunig, M.: *On the Way to Component-based 3D/4D Geoinformation Systems*. Heidelberg : Springer, 2001 (Lecture Notes in Earth Sciences)

[16] Breunig, M. ; Bode, T. ; Cremers, A.B.: Implementation of elementary geometric database operations for a 3D-GIS. In: *Sixth Internat. Symposium on Spatial Data Handling*. Edinburgh, 1994, S. 604–613

[17] Breunig, M. ; Bär, W. ; Thomsen, A.: Services for Geoscientific Applications based on a 3D-Geodatabase Kernel. In: *Information Systems in Earth Management*. Potsdam, 2004 (Geotechnologien Science Report 4), S. 35–39

[18] Breunig, M. ; Bär, W. ; Thomsen, A.: Usage of Spatial Data Stores for Geo-Services. In: *Proceedings of the 7th AGILE Conference on Geographic Information Science*. Heraklion, Greece, April 29th - May 1th 2004, S. 687–696

[19] Breunig, M. ; Bär, W. ; Thomsen, A. ; Hering-Coelho, A. ; Staub, G. ; Wursthorn, S.: Ein Blick in die Zukunft: Datenbankunterstützung für mobile AR Systeme. In: König-Ries, Klein (Hrsg.): *Mobile Datenbanken: heute, morgen und in 20 Jahren*. BTW 2005 Karlsruhe, 2005 (8. Workshop des GI-Arbeitskreises Mobile Datenbanken und Informationssysteme), S. 85–96

[20] Breunig, M. ; Cremers, A.B. ; Flören, J. ; Müller, W. ; Siebeck, J.: New Methods for Topological Clustering, Spatial Access and Test Set Generation in 3D Databases - Extensions to GeoToolKit. In: *Proceedings of the 9th ACM Intern. Symposium on Advances in Geographic Information Systems*. Atlanta, GA, 2001, S. 10

[21] Breunig, M. ; Cremers, A.B. ; Götze, H.-J. ; Schmidt, S. ; Seidemann, R. ; Shumilov, S. ; Siehl, A.: First Steps Towards an Interoperable GIS - an Example from Southern Lower Saxony. In: *Phys. Chem. Earth* 24 (1999), Nr. 3, S. 179–189

[22] Breunig, M. ; Cremers, A.B. ; Götze, H.-J. ; Schmidt, S. ; Seidemann, R. ; Shumilov, S. ; Siehl, A.: Abschlußbericht: Geologische Kartierung mit GIS auf der Grundlage von 3D-Modellen mit thematischem Bezug zum Bündelantrag Interoperable geowissenschaftliche Informationssysteme. (2002), S. 49. – Abschlußbericht Deutsche Forschungsgemeinschaft DFG Az Si 73/15-2, 1.8.1998-31.12.2001

[23] Breunig, M. ; Cremers, A.B. ; Götze, H.-J. ; Seidemann, R. ; Schmidt, S. ; Shumilov, S. ; Siehl, A.: Geologische Kartierung mit GIS auf Grundlage von 3D-Modellen. In: *Proceedings Umweltinformatik - 13. Int. Symposium Informatik für den Umweltschutz*, 1999, S. 417–428

[24] Breunig, M. ; Malaka, R. ; Reinhardt, W. ; Wiesel, J.: Advancement of Geoservices. In: *Information Systems in Earth Management - Kick-Off-Meeting University of Hannover*, 2003 (Geotechnologien - Science Report 2), S. 37–50

[25] Breunig, M. ; Malaka, R. ; Reinhardt, W. ; Wiesel, J.: Entwicklung mobiler Geodienste. In: L. Bernhard, K. S. (Hrsg.): *Geodaten- und Geodienste-Infrastrukturen - von der Forschung zur praktischen Anwendung, Beiträge zu den Münsteraner GI-Tagen*, 2003 (IfGIprints 18), S. 253–265

[26] Brinkhoff, T.: *Geodatenbanksysteme in Theorie und Praxis.* Wichmann Verlag, 2005

[27] Brinkhoff, T. ; Kriegel, H.-P. ; Schneider, R. ; Seeger, B.: Multi-step processing of spatial joins. In: *Proc. ACM SIGMOD International Conference on Management of Data*, 1994, S. 197–208

[28] Brisson, E.: Representing geometric structures in d dimensions: topology and order. In: *Discrete and Computational Geometry* (1993), Nr. 9, S. 387–426

[29] C.B. Medeiros, F. P.: Databases for GIS. In: *SIGMOD Record* 23 (1994), Nr. 1, S. 107–115

[30] Cellary, W. ; Jomier, G.: Consistency of Versions in ObjectOriented Databases. In: *Proc. of the 16 th VLDB*, 1990, S. 432–441

[31] Chou, Hong-Tai ; Kim, Won: A Unifying Framework for Version Control in a CAD Environment. In: *Proceedings of the 12th Internat. Conf. on Very Large Data Bases.* Kyoto, 1986, S. 336–344

[32] Cox, Simon J.: Geologic Data Transfer Using XML. In: *Proceedings of the International Association of Mathematical Geology - IAMG.* Cacum, September 2001, S. 17

[33] Dunham, M.H. ; Helal, A.: Mobile Computing and Databases: Anything New? In: *ACM SIGMOND Record* 24 (1995), Dec., Nr. 4, S. 5

[34] Easterfield, M. E. ; Newell, R.G. ; Theriault, D.G.: Version Management in GIS - Applications and Techniques. In: *Proc. of the European Conf. on Geographical Information Systems.* Amsterdam, 1990, S. 288–297

[35] Elmasri, R. ; Navathe, S.: *Fundamentals of Database Systems.* 3. Addison Wesley, 2000

[36] Finkel, R.A. ; Bentley, J.L.: Quad trees: A data structure for retrieval on composite keys. In: *Acta Informatica* (1974), Nr. 4, S. 1–9

[37] Forman, G.H. ; Zahorjan, J.: The Challenges of Mobile Computing. In: *IEEE Computer* 27 (1994), Nr. 4, S. 38–47

[38] Gaede, V. ; Günther, O.: Survey on Multidimensional Access Methods / Department of Economics and Business Aministration - Humboldt University Berlin. 1997 (ISS-16). – Forschungsbericht

[39] Gancarski, S. ; Jomier, G.: Managing Entity Versions within their Context: a Formal Approach. In: Karagiannis, D. (Hrsg.): *Proc. DEXA '94.* Athens, September 1994 1994, S. 400–409

[40] GEOTECHNOLOGIEN Science Report 2: Information Systems in Earth Management. Potsdam, 2003. – Forschungsbericht. Kick-Off-Meeting, University of Hannover, 19 February 2003

[41] GEOTECHNOLOGIEN Science Report 4: Information Systems in Earth Management. Potsdam, 2004. – Forschungsbericht. Status Seminar, RWTH Aachen University, 23-24 March 2004

[42] Giguere, E.: Mobile Data Management: Challenges of Wireless and Offline Data Access. In: *17th Int. Conf. on Data Engineering.* Heidelberg, Germany : IEEE Computer Society, 2001, S. 227–228

[43] Gollmick, C.: Anwendungsklassen und Architektur Mobiler Datenbanksysteme. In: Klein, H.-J. (Hrsg.): *12. GI-Workshop Grundlagen von Datenbanken*, Christian-Albrechts-Universität Kiel, Bericht Nr. 2005, 2000, S. 36–40

[44] Gollmick, C.: Unterstützung mobiler Clients durch erweiterte Client/Server-Datenbanksysteme. In: *Tagungsband zum 13. GI-Workshop Grundlagen von Datenbanken*, Fakultät für Informatik, Universität Magdeburg, 2001, S. 43–47

[45] Gollmick, C.: Client-Oriented Replication in Mobile Database Environments / Institut für Informatik. Friedrich-Schiller-Universität Jena, 2003 (Math/Inf/08/03). – Jenaer Schriften zur Mathematik und Informatik

[46] Gribble, S. ; Halevy, A. ; Zachary, I. ; Rodrig, M. ; Suciu, D.: What can Databases Do for Peer-to-Peer? In: *WebDB Workshop on Databases and the Web 2001*, 2001, S. 6

[47] Guarino, N.: Formal Ontology and Information Systems. In: *Proceedings FOIS'98*. Trento, Italy : IOS Press, 1998, S. 3–15

[48] Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: *Proc. 1984 ACM SIGMOD Intern. Conf. on Management of Data*. Boston, MA, 1984, S. 47–57

[49] Götze, H.-J.: Ein numerisches Verfahren zur Berechnung der gravimetrischen Feldgrößen drei-dimensionaler Modellkörper. In: *Arch. Met. Geophy. Biokl.* Ser. A (1978), Nr. 25, S. 195–215

[50] Güting, R.H.: An Introduction to Spatial Database Systems / FernUniversität Hagen. Fachbereich Informatik, 1994 (165). – Forschungsbericht

[51] Güting, R.H. ; Schneider, M.: Realms: A Foundation for Spatial Data Types in Database Systems. In: *Symposium on Large Spatial Databases*, 1993, S. 14–35

[52] Güting, R.H. ; Schneider, M.: Realm-Based Spatial Data Types: The ROSE Algebra. In: *VLDB Journal* 4 (1995), Nr. 2, S. 243–286

[53] Hammerich, D.: *Konzeption und Implementierung der Verwaltung von 3D Geo-Objekten in einem objekt-relationalen Datenbanksystem*, Hochschule Vechta, Diplomarbeit, 2003

[54] Heuer, A. ; Saake, G. ; Sattler, K.-U.: *Datenbanken kompakt.* mitp-Verlag, 2001

[55] Hilbert, D.: Über die stetige Abbildung einer Linie auf ein Flächenstück. In: *Mathematische Annalen* 38 (1891), S. 459–460

[56] Hoffmann, C.M.: *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann, 1989 (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)

[57] Holliday, J. ; Agrawal, D. ; Abbadi, A.: Disconnection Modes for Mobile Databases. In: *Wireless Networks* 8 (2002), Juli, Nr. 4, S. 391–402

[58] Hoppe, H.: Efficient Implementation of Progressive Meshes / Microsoft Research. Redmond, WA 98052, 1998 (MSR-TR-98-02). – Forschungsbericht

[59] Höpfner, Hagen: Grundlagen von Replikationstechniken für mobile Datenbanken. In: Endig, H. (Hrsg.) ; Herstel, T. (Hrsg.): *Tagungsband zum 13. GI-Workshop Grundlagen von Datenbanken.* Fakultät für Informatik, Universität Magdeburg, 2001, S. 68–72

[60] Höpfner, Hagen: *Replikationstechniken für mobile Datenbanken*, Otto-von-Guericke-Universität Magdeburg, Diplomarbeit, 2001

[61] Höpfner, H. and Türker, C. and König-Ries, B.: *Mobile Datenbanken und Informationssysteme - Konzepte und Techniken.* dpunkt.verlag, 2005

[62] Imieliski, T. (Hrsg.) ; Korth, H.F. (Hrsg.): *The Kluwer International Series in Engineering and Computer Science.* Bd. 353: *Mobile Computing.* Boston : Kluwer Academic Publishers, 1996

[63] Industrial Alliance for Interoperability. *IFC 2x Edition 2 Specification.* 2003. Url: http://www.iai-international.org/Model/IFC(ifcXML)Specs.html

[64] Karlsson, J.S. ; Lal, A. ; Leung, C. ; Pham, T.: IBM DB2 Everyplace: A Small Footprint Relational Database System. In: *17th Int. Conf. on Data Engineering.* 17th Int. Conf. on Data Engineering : IEEE Computer Society, 2001, S. 230–232

[65] Katz, Randy H.: Toward a Unified Framework for Version Modeling in Engineering Databases. In: *ACM Computing Surveys* 22 (1990), Nr. 4, S. 375–408

[66] Katz, R.H. ; Chang, E.: Managing Change in a Computer-Aided Design Database. In: *Proceedings of the 13th VLDB Conference.* Brighton, 1987, S. 455–462

[67] Kent, William: An Overview of the Versioning Problem. In: *Proc. SIGMOD.* Portland Oregon, Mai 1989, S. 3

[68] Kesper, B.: Das Geo-Datenmodell GeoCORE. In: *Kurzfassungen - 9. Workshop: Grundlagen von Datenbanken.* Königslutter, 1997. – Technischer Bericht, Universität Dortmund, Nr. 643, S. 51–55

[69] Kesper, B. ; Möller, D.P.F.: BAGIS - Baugeologisch-Geotechnisches Informationssystem. In: [101], S. 57–66

[70] Kleinschmidt, A.: *Simulation of wireless networks: GPRS, IEEE 802.11, UMTS*, University College London, Department of Computer Science, Diplomarbeit, 2001

[71] Klesper, Ch.: IVIS-3D and IGMAS: tools for an interactive visualization of gravity models. In: *Physics and Chemistry of the Earth* 23 (1998), Nr. 3, S. 279–283

[72] Käfer, W. ; Ritter, N. ; Schöning, H.: Konfigurationskonzepte für DB-basierte technische Entwurfsanwendungen. In: *Informatik Forsch. Entw.* (1998), Nr. 13, S. 1–17

[73] Käfer, W. ; Schöning, H.: Mapping a Version Model to a Complex-Object Data Model. In: *Proceedings of the 8th International Conference on Data Engineering*, IEEE Computer Society, 1992, S. 348–357

[74] Lamb, C. ; Landis, G. ; Orenstein, J. ; Weinreb, D.: The Objectstore Database System. In: *Communications of the ACM* 34 (1991), Nr. 10, S. 50–63

[75] Leebmann, J. ; Hering-Coelho, A. ; Bähr, H.-P. ; Staub, G ; Wiesel, J.: Augmented Reality im Katastrophenmanagement. In: Zipf, Coors (Hrsg.): *3D-Geoinformationssysteme - Grundlagen und Anwendungen.* Heidelberg : Wichmann-Verlag, 2005

[76] Lienhardt, P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. In: *Computer Aided Design* 23 (1991), Nr. 1, S. 59–82

[77] Madria, S.K.: Transaction Models for Mobile Computing. In: *Int. Database Engineering and Application Symposium*, 1998, S. 92–102

[78] Mallet, J.-L.: GOCAD - A Computer Aided Design Program for Geological Applications. In: [127], S. 123–142

[79] Medeiro, C.B. ; Bellosta, M.J. ; Jomier, G.: Managing Multiple Representations of Georeferenced Elements. In: *DEXA Workshop*, 1996, S. 364–370

[80] Medeiros, C.B. ; Jomier, G.: Managing Alternatives and Data Evolution in GIS. In: *Proc. ACM/ISCA Workshop on Advances in Geographic Information Systems*, 1993, S. 34–37

[81] Medeiros, C.B. ; Jomier, G.: Using Versions in GIS. In: *Database and Expert Systems Applications (DEXA)*, Springer, 1994, S. 465–474

[82] Mutschler, B. ; Specht, G.: Implementierungskonzepte und Anwendungsentwicklung kommerzieller mobiler Datenbanksysteme - Ein Vergleich. In: *Mobilität und Informationssysteme - Workshop des GI-Arbeitskreises Mobile Datenbanken und Informationssysteme* Institut für Informationssysteme, ETH Zürich, 2003 (Technical Report 422), S. 70–79

[83] Mutschler, B. ; Specht, G.: *Mobile Datenbanksysteme.* 1. Heidelberg : Springer-Verlag, 2004

[84] Mäntylä, Martti: *An Introduction to Solid Modelling.* Computer Science Press, 1988 (Principles of computer sciences)

[85] Müller, W.: *Skalierbare Datenstrukturen für große Mengen räumlicher 3D-Daten*, Universität Bonn, Institut für Informatik III, Diplomarbeit, 2000. – Betreuer: Prof. Dr. Martin Breunig

[86] Newell, R. G. ; Easterfield, M.: Version Management - the problem of the long transaction / Smallworld Systems Ltd. 1990 (4). – Technical Paper

[87] Newell, R.G. ; Batty, P.M.: GIS Databases are different. In: *Proceedings AM/FM International Annual Conf. XVII.* Denver, 1994, S. 279–288

[88] Newell, R.G. ; Theriault, D. ; Easterfield, M.: Temporal GIS - Modeling the evolution of spatial data in time. In: *Computers & Geosciences* 18 (1992), Nr. 4, S. 427–433

[89] Nievergelt, J. ; Hinterberg, H. ; Sevcik, K.C.: The grid file: An adaptable, symmetric multikey file structure. In: *ACM Transactions on Database systems* 9 (1984), Nr. 1, S. 38–71

[90] Nievergelt, J. ; Widmayer, P.: Spatial Data Structures: Concepts and Design Choices. In: van Kreveld; Nievergelt; Roos; Widmayer (Hrsg.): *Algorithmic Foundations of Geographic Information Systems.* Springer, 1997, S. 153–197

[91] Object Management Group. *CORBA/IIOP Specification.* 2004. Url: http://www.omg.org/technology/documents/formal/

[92] O'Neil, P.E.: The escrow transactional method. In: *TODS* 11 (1986), Nr. 4, S. 405–430

[93] Ooi, B.H. ; Sacks-Davis, R. ; Han, J. *Indexing in Spatial Databases.* Url: http://citeseer.nj.nec.com/319577.html

[94] Open Geospatial Consortium. *Simple Features Specification (SFS).* 2001. Url: http://www.opengeospatial.org/

[95] Open Mobile Alliance. *SyncML Specification.* 2005. Url: http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html

[96] Oracle. *Oracle Database 10g Workspace Manager Technical White Paper.* 2005. – Url: www.oracle.com/technology/products/database/workspacemanager

[97] Orenstein, J.A.: Spatial Query Processing in an Object-Oriented Database System. In: *11th ACM International Conference on Management of Data*, 1986, S. 326–336

[98] Peerbocus, A. ; Jomier, G. ; Badard, T.: A Methodology for Updating Geographic Databases using Map Versions. In: C. Armenakis, Y.C. L. (Hrsg.): *Geospatial Theroy, Processing and Applications, ISPRS Commission IV, Symposium 2002*. Ottawa, Canada, July 2002

[99] Pitoura, E. ; Samaras, G. ; Elmagarmid, A.K. (Hrsg.): *Data Management for mobile Computing*. Kluwer Academic Publisher, 1998 (Advances in Database Systems)

[100] Raper, J. (Hrsg.): *Three dimensional applications in Geographical Information Systems*. Taylor & Francis, 1989

[101] Reik, G. & Paehge, W. (Hrsg.): *Raumbezogene Informationssysteme für geologische, bau- und geotechnische Aufgaben*. TU Clausthal, 1999 (FIS-Forum Clausthal - Oktober 1997)

[102] Requicha, A.A.G.: Representations for Rigid Solids: Theory, Methods and Systems. In: *Computing Surveys* 12 (1980), Nr. 4, S. 437–464

[103] Requicha, A.A.G. ; Voelcker, H.B.: Solid Modeling: Current Status and Research Directions. In: *IEEE Comp. Graph. Appl.* Bd. 3, 1983, S. 25–37

[104] Rigaux, P. ; Scholl, M. ; Voisard, A.: *Spatial databases with application to GIS*. San Francisco : Morgan Kaufmann, 2002

[105] Ritter, Norbert: *DB-gestützte Kooperationsdienste für technische Entwurfsanwendungen*. infix, 1997 (DISDBIS 33). – PhD-Thesis

[106] Ruch, C. *Georisiken: Aktive Massenbewegungen am Albtrauf*. LGRB-Nachrichten No.8/2002, Landesamt für Geologie, Rohstoffe und Bergbau (LGRB), Baden-Württemberg. 2002

[107] Samet, H.: *Applications of spatial data structures*. Addison-Wesley, 1990 (Addison-Wesley Reading)

[108] Samet, H.: *The design and analysis of spatial data structures*. Addison-Wesley, 1990 (Addison-Wesley Reading)

[109] Sartiani, C. ; Manghi, P. ; Ghelli, G. ; Conforti, G.: XPeer: A Self-organizing XML P2P Database System. In: *Proceedings of the First Int. Workshop on Peer-to-Peer Computing and Databases (P2P&DB)*. Heraklion - Crete, Greece, 2003, S. 17

[110] Satyanarayanan, M.: Fundamental Challenges of Mobile Computing. In: *ACM Symposium on Principles of Distributed Computing*, 1995, S. 7

[111] Sayda, F. ; Reinhardt, W. ; Wittmann, E.: Positionsbezogene Dienste zur Unterstützung von Bergsteigern und Wanderern. In: Zipf, A. & Strobl, J. (Hrsg.): *Geoinformation mobil.* Heidelberg : Wichmann Verlag, 2002, S. 127–137

[112] Schmidt-Belz, B. ; Zipf, A. ; Poslad, S. ; Laamen, H.: Location-based mobile tourist services - first user experiences. In: *ENTER 2003 - Int. Congress on Tourism and Communications Technologies.* Helsinki, Finnland : Springer Computer Science. Heidelberg, Berlin, 2003

[113] Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: *Proceedings of the First Int. Conf. on Peer-to-Peer Computing P2P 2001*, IEEE, 2002, S. 101–103

[114] Schwabe, G. (Hrsg.) ; Streitz, N. (Hrsg.) ; Unland, R. (Hrsg.): *CSCW-Kompendium: Lehr- und Handbuch zum computerunterstützten kooperativen Arbeiten.* Berlin : Springer Verlag, 2001

[115] Schönhoff, Martin: *Version Management in Federated Database Systems.* infix, 2002 (DISDBIS 81). – PhD-Thesis

[116] Sciore, Edward: Versioning and Configuration Management in an Object-Oriented Data Model. In: *VLDB Journal* (1994), Nr. 3, S. 77–106

[117] Sellis, T. ; Roussopoulos, N. ; Faloutsos, C.: The R+-tree: A dynamic index for multi-dimensional objects. In: *Proc. 13th International Conference on Very Large Data Bases VLDB*, 1987, S. 507–518

[118] Serrano-Alvarado, P. ; Roncancio, C. ; Adiba, M.: Mobile Transaction Supports for DBMS: An Overview / Laboratoire LSR-IMAG, Grenoble. 2001 (RR 1033-1-LSR 16). – Rapport de Recherche

[119] Shekhar, S. ; Chawla, S.: *Spatial Databases: A Tour.* Prentice Hall, 2002

[120] Shumilov, S. ; A., Thomsen ; Cremers, A. B. ; Koos, B.: Management and visualization of large, complex and time-dependent 3D objects in distributed GIS. In: *GIS '02: Proceedings of the 10th ACM international symposium on Advances in geographic information systems.* New York, NY, USA : ACM Press, 2002, S. 113–118

[121] Siebeck, J.: *Concepts for the Representation, Storage, and Retrieval of Spatio-Temporal Objects in 3D/4D Geo-Information-Systems*, Rheinische Friedrich-Wilhelms-Universität Bonn, Diss., 2004

[122] Staab, S. (Hrsg.) ; Studer, R. (Hrsg.): *Handbook on Ontologies.* Heidelberg : Springer Verlag, 2004 (Springer Series: International Handbooks on Information Systems)

[123] Sun Microsystems, Inc. *JINI specifications.* 2001.
Url: http://www.sun.com/software/jini/specs/

[124] T.E. Lonski, R. P.: Transaction Management and Versioning for Enterprise-wide ESRI Implementations. In: *Proceedings of the ESRI Userconference 2002*, 2002

[125] Thomsen, A. ; Breunig, M. ; Bär, W. ; Cremers, A.B. ; Siehl, A.: Datenbankunterstützung für geologische Anwendungen. In: Zipf, Coors (Hrsg.): *3D-Geoinformationssysteme.* Heidelberg : Wichmann-Hüthig Verlag, 2005, S. 354–375

[126] Thomsen, A. ; M., Breunig: Some remarks to topological abstraction in multi representation databases. In: *Proceedings of Internat. Workshop on Information Geographical Information Systems.* St. Petersburg, 2006. – accepted, S. 12

[127] Turner, A.K. (Hrsg.): *Three-Dimensional Modeling with Geoscientific Information Systems - Proceedings of the NATO Advanced Research Workshop, Santa Barbara, CA., December 10-15, 1989.* Kluwer Academic Publishers in cooperation with NATO Scientific Affairs Division, 1989 (Mathematical and Physical Sciences)

[128] Türker, C. ; Zini, G.: A Survey of Academic and Commercial Approaches to Transaction Support in Mobile Computing Environments / ETH Zurich, Department of Computer Science. 2003 (429). – Forschungsbericht

[129] Ullman, Jeffrey D.: *Principles of computer sciences.* Bd. 1: *Principles of database and knowledgebase systems.* Computer Science Press, 1988

[130] Voigtmann, A. ; Becker, L. ; Hinrichs, K.H.: An Object-Oriented Data Model and a Query Language for Geographic Information Systems / Universität Münster, Institut für Informatik. 1995 (15/95-I). – Forschungsbericht

[131] Voigtmann, A. ; Becker, L. ; Hinrichs, K.H.: Temporal extensions for an Object-Oriented Geo-Data-Model / Universität Münster, Institut für Informatik. 1996 (6/96-I). – Forschungsbericht

[132] Weikum, G. ; Vossen, G.: *Transactional Information Systems - Theory, Algorithms, and the Practice of Concurrency Control and Recovery.* Morgan Kaufmann Publishers, 2002

[133] Westfechtel, B. and Conradi, R.: Software Configuration Management and Engineering Data Management: Differences and Similarities. In: *Lecture Notes in Computer Science* 1439 (1998), S. 95+

[134] World Wide Web Consortium W3C. *SOAP Specification*. 2003. Url: http://www.w3.org/TR/soap/

[135] World Wide Web Consortium W3C. *Extensible Markup Language (XML) 1.1 (Second Edition)*. 2006. Url: http://www.w3.org/TR/2006/REC-xml11-20060816/

[136] Zipf, A. (Hrsg.) ; Strobl, J. (Hrsg.): *Geoinformation mobil*. Heidelberg, Germany : Wichmann Verlag, 2002

[137] Özsu, M.T. ; Valduriez, P.: *Principles of Distributed Database Systems*. 2. Prentice Hall, 1999

Anhang A

DB3D-XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema elementFormDefault="qualified"
  targetNamespace="http://www.geoservices.uni-osnabrueck.de/db3d"
  xmlns:db3d="http://www.geoservices.uni-osnabrueck.de/db3d"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:element name="DB3DXML">
  <xsd:complexType>
   <xsd:choice>
    <xsd:element ref="db3d:DBMS"/>
    <xsd:element ref="db3d:Project"/>
    <xsd:element ref="db3d:_Space"/>
    <xsd:element ref="db3d:Object3D"/>
    <xsd:element ref="db3d:ResultObject3D"/>
    <xsd:sequence>
     <xsd:element ref="db3d:Object3DArray" minOccurs="0"/>
     <xsd:element ref="db3d:ResultObject3DArray" minOccurs="0"/>
    </xsd:sequence>
   </xsd:choice>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="DBMS">
  <xsd:annotation>
   <xsd:documentation>
    The DBMS element is the root element in a complete DBMS XML export document.
    There can only be one DBMS element. A DBMS element can contain an arbitary
    number of Projects as Project elements.
   </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
```

```
    <xsd:element ref="db3d:Description" minOccurs="0"/>
    <xsd:element name="Projects">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element maxOccurs="unbounded" minOccurs="0" ref="db3d:Project"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="Project">
  <xsd:annotation>
   <xsd:documentation>
    A Project element contains all project relevant data in the DBMS. A Project has a name and
    an id for identification in the DBMS. It contains a Description elements and an arbitrary
    number of Spaces as abstract _Space elements (Currently only Space3D is existent).
   </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="db3d:Description" minOccurs="0"/>
    <xsd:element name="Spaces">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element maxOccurs="unbounded" minOccurs="0" ref="db3d:_Space"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="projectID" type="xsd:positiveInteger"/>
  </xsd:complexType>
 </xsd:element>

 <xsd:element abstract="true" name="_Space" type="db3d:abstractSpaceType">
  <xsd:annotation>
   <xsd:documentation>
    The abstract _Space element is the head element of the substitution group for SpaceXX
    elements. It is of the abstract type definition db3d:abstractSpaceType which contains the
    common type definition of concrete Space elements. Currently only the Space3D element as
    member of this substitution group is defined.
   </xsd:documentation>
  </xsd:annotation>
 </xsd:element>

 <xsd:complexType abstract="true" name="abstractSpaceType">
```

```
 <xsd:sequence>
  <xsd:element ref="db3d:Description" minOccurs="0"/>
  <xsd:element name="SpaceDefinition" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="ScalarOperator" type="xsd:float"/>
     <xsd:element name="SRID" type="xsd:integer"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="Constraints" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element maxOccurs="unbounded" minOccurs="0" name="SpaceConstraint"/>
     <xsd:element maxOccurs="unbounded" minOccurs="0" name="ThematicGroupConstraint">
      <xsd:complexType>
       <xsd:simpleContent>
        <xsd:extension base="xsd:string">
         <xsd:attribute name="nameTG" type="xsd:string"/>
        </xsd:extension>
       </xsd:simpleContent>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string"/>
 <xsd:attribute name="spaceID" type="xsd:positiveInteger"/>
</xsd:complexType>

<xsd:element name="Space3D" substitutionGroup="db3d:_Space">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="db3d:abstractSpaceType">
    <xsd:sequence>
     <xsd:element name="ThematicGroups">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="ThematicGroup">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element ref="db3d:Description"/>
           <xsd:element ref="db3d:RecordDefinition"/>
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:string"/>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
```

```
      </xsd:complexType>
     </xsd:element>
     <xsd:element ref="db3d:Object3DArray"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Object3D">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="db3d:TimeStamp"/>
   <xsd:element name="OID" type="xsd:string"/>
   <xsd:element maxOccurs="1" minOccurs="0" name="SpatialPart">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" ref="db3d:_SpatialObject3D"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element maxOccurs="1" minOccurs="0" name="ThematicPart">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element ref="db3d:Description" minOccurs="0"/>
      <xsd:element name="ThematicGroupMember" minOccurs="0">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element ref="db3d:Record"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
       </xsd:complexType>
      </xsd:element>
      <xsd:element name="GeometryThematic" minOccurs="0">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element ref="db3d:Table"/>
        </xsd:sequence>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="ID" type="xsd:ID" use="required"/>
 </xsd:complexType>
</xsd:element>

<xsd:element abstract="true" name="_SpatialObject3D" type="db3d:abstractSpatialObject3DType"/>
```

```
<xsd:complexType abstract="true" name="abstractSpatialObject3DType">
 <xsd:sequence>
  <xsd:element ref="db3d:MBB3D"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="Sample3D" substitutionGroup="db3d:_SpatialObject3D">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="db3d:abstractSpatialObject3DType">
    <xsd:sequence>
     <xsd:element name="Components">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="PointNet3DComp">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element ref="db3d:MBB3D"/>
           <xsd:element name="Points">
            <xsd:complexType>
             <xsd:sequence>
              <xsd:element minOccurs="0" maxOccurs="unbounded" ref="db3d:Point3D"/>
             </xsd:sequence>
            </xsd:complexType>
           </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="compID" type="xsd:int" use="required"/>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
       <xsd:attribute name="idCounter" type="xsd:int" use="required"/>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Curve3D" substitutionGroup="db3d:_SpatialObject3D">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:extension base="db3d:abstractSpatialObject3DType">
    <xsd:sequence>
     <xsd:element name="Components">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="SegmentNet3DComp">
```

```
          <xsd:complexType>
           <xsd:sequence>
            <xsd:element ref="db3d:MBB3D"/>
            <xsd:element name="Points">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="2" ref="db3d:Point3D"/>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Elements">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="1" name="SegmentElt3D">
                <xsd:complexType>
                 <xsd:sequence>
                  <xsd:element maxOccurs="2" minOccurs="2" ref="db3d:Vertex"/>
                  <xsd:element maxOccurs="2" minOccurs="0" ref="db3d:Neighbour"/>
                 </xsd:sequence>
                 <xsd:attribute name="ID" type="xsd:ID" use="required"/>
                </xsd:complexType>
               </xsd:element>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="0" ref="db3d:EntryElement"/>
           </xsd:sequence>
           <xsd:attribute name="compID" type="xsd:int" use="required"/>
          </xsd:complexType>
         </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="idCounter" type="xsd:int" use="required"/>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="Surface3D" substitutionGroup="db3d:_SpatialObject3D">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="db3d:abstractSpatialObject3DType">
     <xsd:sequence>
      <xsd:element name="Components">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" name="TriangleNet3DComp">
```

```
          <xsd:complexType>
           <xsd:sequence>
            <xsd:element ref="db3d:MBB3D"/>
            <xsd:element name="Points">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="3" ref="db3d:Point3D"/>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Elements">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="1" name="TriangleElt3D">
                <xsd:complexType>
                 <xsd:sequence>
                  <xsd:element maxOccurs="3" minOccurs="3" ref="db3d:Vertex"/>
                  <xsd:element maxOccurs="3" minOccurs="0" ref="db3d:Neighbour"/>
                 </xsd:sequence>
                 <xsd:attribute name="ID" type="xsd:ID" use="required"/>
                </xsd:complexType>
               </xsd:element>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="0" ref="db3d:EntryElement"/>
           </xsd:sequence>
           <xsd:attribute name="compID" type="xsd:int" use="required"/>
          </xsd:complexType>
         </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="idCounter" type="xsd:int" use="required"/>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="Volume3D" substitutionGroup="db3d:_SpatialObject3D">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="db3d:abstractSpatialObject3DType">
     <xsd:sequence>
      <xsd:element name="Components">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" name="TetrahedronNet3DComp">
```

```
          <xsd:complexType>
           <xsd:sequence>
            <xsd:element ref="db3d:MBB3D"/>
            <xsd:element name="Points">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="4" ref="db3d:Point3D"/>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Elements">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="1" name="TetrahedronElt3D">
                <xsd:complexType>
                 <xsd:sequence>
                  <xsd:element maxOccurs="4" minOccurs="4" ref="db3d:Vertex"/>
                  <xsd:element maxOccurs="4" minOccurs="0" ref="db3d:Neighbour"/>
                 </xsd:sequence>
                 <xsd:attribute name="ID" type="xsd:ID" use="required"/>
                </xsd:complexType>
               </xsd:element>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="0" ref="db3d:EntryElement"/>
           </xsd:sequence>
           <xsd:attribute name="compID" type="xsd:int" use="required"/>
          </xsd:complexType>
         </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="idCounter" type="xsd:int" use="required"/>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="Hull3D" substitutionGroup="db3d:_SpatialObject3D">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:extension base="db3d:abstractSpatialObject3DType">
     <xsd:sequence>
      <xsd:element name="Components">
       <xsd:complexType>
        <xsd:sequence>
         <xsd:element minOccurs="0" maxOccurs="unbounded" name="Hull3DComp">
```

```
          <xsd:complexType>
           <xsd:sequence>
            <xsd:element ref="db3d:MBB3D"/>
            <xsd:element name="Points">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="4" ref="db3d:Point3D"/>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Elements">
             <xsd:complexType>
              <xsd:sequence>
               <xsd:element maxOccurs="unbounded" minOccurs="1" name="TriangleElt3D">
                <xsd:complexType>
                 <xsd:sequence>
                  <xsd:element maxOccurs="3" minOccurs="3" ref="db3d:Vertex"/>
                  <xsd:element maxOccurs="3" minOccurs="0" ref="db3d:Neighbour"/>
                 </xsd:sequence>
                 <xsd:attribute name="ID" type="xsd:ID" use="required"/>
                </xsd:complexType>
               </xsd:element>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="0" ref="db3d:EntryElement"/>
           </xsd:sequence>
           <xsd:attribute name="compID" type="xsd:int" use="required"/>
          </xsd:complexType>
         </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="idCounter" type="xsd:int" use="required"/>
       </xsd:complexType>
      </xsd:element>
     </xsd:sequence>
    </xsd:extension>
   </xsd:complexContent>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="ResultObject3D">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="ScalarOperator" type="xsd:float"/>
    <xsd:element name="PerformedOperation" type="xsd:integer"/>
    <xsd:element name="LastObjectType" type="xsd:integer"/>
    <xsd:element name="LastObjectOID" type="xsd:string"/>
    <xsd:element name="ResultGeometry">
     <xsd:complexType>
```

```
<xsd:sequence>
 <xsd:element name="Points" minOccurs="0">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Point3D" maxOccurs="unbounded">
     <xsd:complexType mixed="true">
      <xsd:sequence/>
      <xsd:attribute name="ID" type="xsd:ID" use="required"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Segments" minOccurs="0">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Segment3D" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="Point3D" minOccurs="2" maxOccurs="2" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="ID" type="xsd:ID" use="required"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Triangles" minOccurs="0">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Triangle3D" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="Point3D" minOccurs="3" maxOccurs="3" type="xsd:string"/>
      </xsd:sequence>
      <xsd:attribute name="ID" type="xsd:ID" use="required"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Tetrahedrons" minOccurs="0">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Tetrahedron3D" maxOccurs="unbounded">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="Point3D" minOccurs="4" maxOccurs="4" type="xsd:string"/>
      </xsd:sequence>
```

```
          <xsd:attribute name="ID" type="xsd:ID" use="required"/>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
     <xsd:element name="Wireframes" minOccurs="0">
      <xsd:complexType>
       <xsd:sequence>
        <xsd:element name="Wireframe3D" maxOccurs="unbounded">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element name="Point3D" minOccurs="0"
             maxOccurs="unbounded" type="xsd:string"/>
           <xsd:element name="Segment3D" minOccurs="0"
             maxOccurs="unbounded">
            <xsd:complexType>
             <xsd:sequence>
              <xsd:element name="Point3D" minOccurs="2"
                maxOccurs="2" type="xsd:string"/>
             </xsd:sequence>
            </xsd:complexType>
           </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="ID" type="xsd:ID" use="required"/>
         </xsd:complexType>
        </xsd:element>
       </xsd:sequence>
      </xsd:complexType>
     </xsd:element>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="ResultThematic" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="ThematicGroup" minOccurs="0">
      <xsd:complexType>
       <xsd:attribute name="name" type="xsd:string"/>
      </xsd:complexType>
     </xsd:element>
     <xsd:element ref="db3d:Description" minOccurs="0"/>
     <xsd:element ref="db3d:RecordDefinition" minOccurs="0"/>
     <xsd:element ref="db3d:Record" minOccurs="0"/>
     <xsd:element ref="db3d:Table" minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
```

```
  <xsd:attribute name="ID" type="xsd:ID" use="required"/>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Table">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element minOccurs="1" maxOccurs="1" ref="db3d:RecordDefinition"/>
   <xsd:element minOccurs="0" maxOccurs="unbounded" ref="db3d:Record"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="RecordDefinition">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Classname" type="xsd:string"/>
   <xsd:element minOccurs="1" maxOccurs="unbounded" name="Column">
    <xsd:complexType mixed="true">
     <xsd:attribute name="type">
      <xsd:simpleType>
       <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Boolean"/>
        <xsd:enumeration value="Float"/>
        <xsd:enumeration value="Double"/>
        <xsd:enumeration value="Integer"/>
        <xsd:enumeration value="Long"/>
        <xsd:enumeration value="String"/>
        <xsd:enumeration value="Vector3D"/>
       </xsd:restriction>
      </xsd:simpleType>
     </xsd:attribute>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Record">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="ReferencedObject" minOccurs="0">
    <xsd:complexType>
     <xsd:attribute name="IDREF" type="xsd:IDREF" use="required"/>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="Values">
    <xsd:complexType>
     <xsd:sequence>
```

```
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
       <xsd:element name="Boolean" type="xsd:boolean"/>
       <xsd:element name="Float" type="xsd:float"/>
       <xsd:element name="Double" type="xsd:double"/>
       <xsd:element name="Integer" type="xsd:int"/>
       <xsd:element name="Long" type="xsd:long"/>
       <xsd:element name="String" type="xsd:string"/>
       <xsd:element name="Vector3D" type="xsd:string"/>
      </xsd:choice>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Point3D">
 <xsd:complexType>
  <xsd:annotation>
   <xsd:documentation>
    Point3D is a sequence from the coordinates x,y and z separated by an commata
   </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute fixed="." name="decimal" type="xsd:string"/>
    <xsd:attribute fixed="," name="cs" type="xsd:string"/>
    <xsd:attribute name="ID" type="xsd:ID" use="required"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>

<xsd:element name="MBB3D">
 <xsd:complexType>
  <xsd:attribute name="pmin" type="xsd:string"/>
  <xsd:attribute name="pmax" type="xsd:string"/>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Vertex">
 <xsd:complexType>
  <xsd:attribute name="IDREF" type="xsd:IDREF" use="required"/>
 </xsd:complexType>
</xsd:element>

<xsd:element name="Neighbour">
 <xsd:complexType>
  <xsd:attribute name="IDREF" type="xsd:IDREF" use="required"/>
```

```
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="EntryElement">
  <xsd:complexType>
   <xsd:attribute name="IDREF" type="xsd:IDREF" use="required"/>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="Description" type="xsd:string"/>
 <xsd:element name="TimeStamp" type="xsd:integer"/>

 <xsd:element name="Object3DArray">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="db3d:Object3D" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

 <xsd:element name="ResultObject3DArray">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element ref="db3d:ResultObject3D" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

</xsd:schema>
```