Master-Arbeit

Konzeption eines Transaktionsmodells für den WebDAV-Standard

Martin Jung
Hochschule Bonn-Rhein-Sieg

DLR

Hochschule
Bonn-Rhein-Sieg

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Master Thesis

im Studiengang

Master of Science in Computer Science

Konzeption eines Transaktionsmodells für den WebDAV-Standard

von

Martin Jung

Erstgutachter und Betreuer: Prof. Dr. Karl W. Neunast
Zweitgutachter: Prof. Dr. Jürgen Wirtgen
Hochschule Bonn-Rhein-Sieg

Betreuer: Dipl.-Math. Jens Rühmkorf
Betreuer: Dipl. Inf. (FH) Markus Litz
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Köln

Eingereicht am 24. Februar 2009

Danksagung

An dieser Stelle möchte ich allen danken, die zum Gelingen dieser Arbeit beigetragen haben.

Ich bedanke mich bei Herrn Prof. Dr. Karl W. Neunast für die umfassende und gute Betreuung meiner Master Thesis. Herrn Prof. Dr. Jürgen Wirtgen danke ich sehr herzlich für die Übernahme der Zweitprüfung.

Jens Rühmkorf und Markus Litz, meinen Betreuern beim DLR, verdanke ich aus vielen Gesprächen und Diskussionen wertvolle und wichtige Impulse zum Gelingen dieser Arbeit. Ich danke auch den Vorgesetzten sowie den weiteren Mitarbeitern in der Einrichtung Simulations- und Softwaretechnik des DLR. Die dortigen Bedingungen für die Erstellung dieser Arbeit waren hervorragend.

Mein besonderer Dank gilt meinem Korrekturleser Andreas Frank, für die vielen Stunden, die er mit der Korrektur meiner Master Thesis verbracht hat und durch konstruktive Kritik den Fortschritt der Arbeit begleitet hat.

Ich danke auch meiner Freundin Martina Dickopf, die mir stets mit sehr viel Geduld und Verständnis zur Seite steht.

Der größte Dank gebührt meinen Eltern, die mir diesen Weg ermöglicht haben und mir immer helfend mit Rat und Tat zur Seite stehen.

Erklärung

Name: Jung, Martin

Adresse: Maischeider Str. 1
56276 Stebach

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Martin Jung Stebach, den 24. Februar 2009

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das WebDAV-Protokoll (Web-based Distributed Authoring and Versioning) ermöglicht die Bearbeitung und Verwaltung von Dateien auf einem Web-Server. Dies stellt z.B. für räumlich und/oder zeitlich getrennt arbeitende Mitglieder eines Projektes eine einfache Möglichkeit zum Informationsaustausch und Bearbeiten gemeinsamer Dokumente dar. Aus technischer Sicht ist WebDAV eine Erweiterung des HTTP-Protokolls. Einige, zwar noch vorhandene, aber im Regelfall von Webservern nicht mehr unterstützte und fehlende Funktionen des HTTP-Protokolls, insbesondere die zur Verwaltung des schreibenden Zugriffs, werden nun durch das WebDAV-Protokoll (wieder) möglich und die De-facto-Limitierung beim HTTP-Protokoll auf rein lesende Zugriffe wird aufgehoben.

Durch die rasche Zunahme und den steigenden Verbreitungsgrad von WebDAV-basierten Anwendungen, wie etwa Dokumentenmanagementsystemen, steigen auch die Anforderungen an deren Zuverlässigkeit. Die voll umfassende Unterstützung von Transaktionen, d.h. die Zusammenfassung einer Menge von Verarbeitungsschritten zu einer logischen Einheit, würde hierzu einen wichtigen Beitrag leisten. Die für Transaktionen geforderten Eigenschaften, die gleichzeitig auch deren Hauptvorteile darstellen, werden durch das bekannte Akronym ACID beschrieben, welches für Atomarität (atomicity), Konsistenz (consistency), Isoliertheit (isolation) und Dauerhaftigkeit (durability) steht.

Für die beiden Transaktionseigenschaften Atomarität und Isoliertheit gibt es zurzeit keine standardkonforme WebDAV-Implementierung. Die Firma Microsoft bietet zwar mit ihrem Produkt Exchange Server 2007 eine Implementierung von Transaktionen über eine proprietäre Erweiterung des WebDAV-Protokolls an, nutzt bei der Umsetzung aber nicht die von der IETF (Internet Engineering Task Force) erarbeiteten abstrakten Konzepte. Zudem werden bei dieser Implementierung nicht alle ACID-Eigenschaften unterstützt. Da die Einhaltung von Standards aber ein wichtiges Kriterium zur Gewährleistung der In-

teroperabilität von verschiedenen Anwendungen unterschiedlicher Hersteller darstellt, ist die Standardkonformität bei der Konzeption des Transaktionsmodells im Rahmen dieser Arbeit ein zentraler Aspekt.

Zurzeit unterstützt das WebDAV-Protokoll allerdings nur die Punkte Konsistenz und Dauerhaftigkeit, eine komplette und vor allem standardkonforme Unterstützung der ACID-Eigenschaften von Transaktionen ist nicht gegeben. Es werden durch den WebDAV-Standard verschiedene, relativ abstrakte, Konzepte zur Verfügung gestellt. Im Rahmen dieser Arbeit wird evaluiert, inwieweit diese für die Realisierung von Atomarität und Isoliertheit genutzt, bzw. konkret ausgestaltet werden können.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, ein Transaktionsmodell zu konzipieren, dass heißt die modellhafte Beschreibung eines Transaktionsablaufes, welche für den WebDAV-Standard als solches noch nicht existiert. Dabei sollen die ACID-Eigenschaften voll umfänglich unterstützt werden.

Die Eigenschaften Atomarität und Isoliertheit stellen, wie bereits erwähnt, dabei die zu erarbeitenden Hauptpunkte dar. Die beiden anderen Eigenschaften, Konsistenz und Dauerhaftigkeit, werden bereits durch die meisten bestehenden WebDAV-Systeme standardkonform unterstützt und stehen aus diesem Grund nicht direkt im Fokus dieser Arbeit, werden aber natürlich trotzdem nicht außer Acht gelassen.

Da der Prozess der Standardisierung durch die IETF für das WebDAV-Protokoll mit seinen Erweiterungen noch nicht abgeschlossen ist, besteht die Möglichkeit, hierauf noch Einfluss zu nehmen und die konkrete Ausgestaltung des Standards mit zu erarbeiten. Deshalb soll im Rahmen dieser Arbeit eng mit der IETF zusammengearbeitet und die von der IETF zur Verfügung gestellten Kommunikationswege, wie IETF-Meetings und IETF-Mailinglisten, genutzt werden.

Bei der Konzeption des Transaktionsmodells sollen bereits vorhandene Standards und Konzepte so weit als möglich verwendet werden. Falls sich jedoch herausstellt, dass die bestehenden Standards an einigen Stellen nicht ausreichen, soll versucht werden, die im Rahmen dieser Arbeit erstellten Lösungen über die IETF in den noch laufenden Prozess der Standardisierung einfließen zu lassen.

1.3 Aufbau der Arbeit

Zu Beginn wird in Kapitel 2 kurz das DLR (Deutsche Zentrum für Luft- und Raumfahrt e.V.) vorgestellt. Die vorliegende Arbeit wurde dort in der DLR-Einrichtung Simulations- und Softwaretechnik erstellt. Die erarbeiteten Ergebnisse sollen in die Weiterentwicklung der beim DLR verwendeten Datenmanagementsoftware DataFinder einfließen. Diese wird ebenfalls in Kapitel 2 genauer beschrieben. Um die Hintergrundinformationen abzurunden, wird zum Ende des Kapitels noch die bereits im Motivationsteil erwähnte IETF und ihre Arbeitsweise vorgestellt.

Kapitel 3 befasst sich mit den verwendeten Technologien und Prinzipien, die den aktuellen Stand von Wissenschaft und Technik widerspiegeln. Dieses Kapitel soll dem Leser zum einen das notwendige theoretische Basiswissen vermitteln und zum anderen bestehende Implementierungen vorstellen, die dann als Grundlagen für weitere Betrachtungen und insbesondere für die in Kapitel 4 durchgeführte Anforderungsanalyse genutzt werden.

Auf Basis der sich aus der Analyse ergebenden Anforderungen wird dann in Kapitel 5 das entsprechende Transaktionsmodell entwickelt. Dabei wird genau geprüft, ob das vorhandene theoretische Grundgerüst des momentanen Standards ausreicht, um die konkreten Anforderungen an das Transaktionsmodell umsetzen zu können. In diesem Kapitel wird des Weiteren die genaue Arbeitsweise des Transaktionsmodells, insbesondere im Hinblick auf die Nebenläufigkeitskontrolle, vorgestellt und bewertet. Diese Ergebnisse dienen als Grundlage für die Diskussion mit der IETF über Erweiterungen bzw. Ergänzungen des WebDAV-Standards, welche im nachfolgenden Kapitel 6 genauer spezifiziert werden.

Um die, auf Basis des in dieser Arbeit entwickelten Transaktionsmodells, getroffene Entwurfsentscheidung, sowie die Machbarkeit der technischen Umsetzung zu verifizieren, wird im vorletzten Kapitel 7 dieser Arbeit eine prototypische Implementierung des Modells vorgestellt.

Zum Schluss werden im letzten Kapitel alle Ergebnisse zusammengefasst. Zudem werden gesammelte Erfahrungen und mögliche zukünftige Erweiterungen und Entwicklungen aufgezeigt.

2 Hintergrund

2.1 Das Deutsche Zentrum für Luft- und Raumfahrt e.V. im Überblick

Das Deutsche Zentrum für Luft- und Raumfahrt e.V. (DLR) ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt, Energie und Verkehr. Über die eigene Forschung hinaus ist das DLR als Raumfahrtagentur im Auftrag der Bundesregierung für die Planung und Umsetzung der deutschen Raumfahrtaktivitäten zuständig.

Das DLR beschäftigt an seinen 29 Instituten bzw. Test- und Betriebseinrichtungen ca. 5.300 Mitarbeiterinnen und Mitarbeiter und ist an 13 Standorten in Deutschland vertreten u.a. in Köln-Porz, Berlin-Adlershof, Bonn-Oberkassel, Braunschweig, Göttingen, Oberpfaffenhofen (bei München) und Stuttgart. Außerdem unterhält das DLR Außenbüros in Brüssel, Paris und Washington D.C..

Die Einrichtung Simulations- und Softwaretechnik (SISTEC) des DLR befindet sich an den Forschungsstandorten Köln-Porz und Braunschweig. Der Arbeitsschwerpunkt liegt im Bereich Software Engineering. Die Kernkompetenzen der Einrichtung sind:

- Verteilte Systeme und Grid Computing
- Innovative Software-Technologien
- Software-Qualitätssicherung

Neben diesen Kernkompetenzen ist SISTEC für die Einarbeitung in die neuen und modernen Softwaretechnologien und ihre Einführung in Kooperationsprojekten mit DLR-Instituten und externen Partnern zuständig.

2.2 Die Anwendung DataFinder

Die vom DLR in der Abteilung SISTEC entwickelte Software DataFinder[1] ist eine Anwendung zur Verwaltung von wissenschaftlich-technischen Daten. Seit Ende des Jahres 2008 ist sie als Open-Source-Software freigegeben.

Aufgabe des DataFinders ist es, Daten in geeigneter Weise mit Metadaten zu kennzeichnen und zu strukturieren. Dabei können entweder vorhandene oder aber auch neu erzeugte, zum Teil ungeordnete, Daten verwaltet werden. Ziel des DataFinder ist es u.a., mit Hilfe einer Suchfunktion in den Metadaten dafür zu sorgen, dass die gespeicherten Daten wiedergefunden werden. Das Metamodell zur Strukturierung der Daten ist dabei vom Administrator frei wählbar, so dass es im Hinblick auf die Art der Daten, die vom DataFinder verwaltet werden können, keinerlei Einschränkungen gibt.

Abbildung 2.1 zeigt die Client-Server-Architektur des DataFinders, die auf Serverseite zweigeteilt ist. Die Metadaten zur Strukturierung werden dabei auf einem speziellen Server abgelegt. Die eigentlichen Daten können entweder ebenfalls direkt auf diesem Server oder mittels verschiedener Speichertechnologien wie z.B. NFS oder GridFTP auf verteilten Servern gespeichert werden. Zudem können die Daten auch "offline" gespeichert werden, z.B. auf Tapes oder DVDs. In diesem Fall verwaltet der DataFinder nur die Datenstruktur, die dazugehörigen Metadaten und ein entsprechendes Inhaltsverzeichnis.

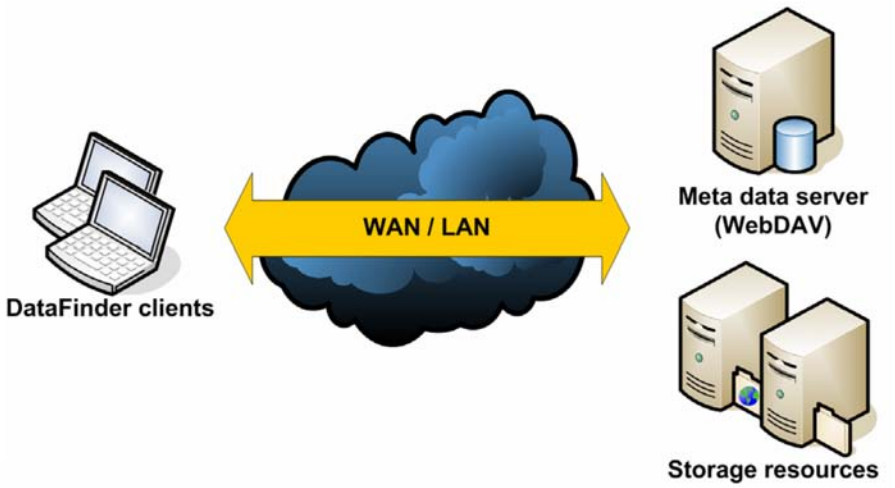

Abbildung 2.1: Client-Server-Architektur des DataFinders

[1] http://sourceforge.net/projects/datafinder/

Zur Verwaltung der Meta-Daten nutzt der DataFinder das WebDAV-Protokoll (Web-based Distributed Authoring and Versioning). WebDAV ist eine auf XML (eXtensible Markup Language) basierende Erweiterung des HTTP-Protokolls, die zusätzliche Möglichkeiten vorsieht, Dokumente nicht nur zu lesen, sondern auch zu modifizieren.

Dank der verwendeten offenen Standards ist die Software auf den verschiedensten Plattformen lauffähig. Einzige Voraussetzung ist, dass die entsprechenden Standards unterstützt werden. So kann z.B. zur Speicherung der Metadaten der kommerzielle Tamino WebDAV-Server[2] also auch der Open-Source-Server Catacomb[3] verwendet werden.

Auf Clientseite werden die Daten mit Hilfe einer Python-Applikation angezeigt, verwaltet und administriert. Dank der Verwendung von Python ist der Client ebenfalls auf allen Plattformen lauffähig, für die es eine Python-Laufzeitumgebung gibt[4].

2.3 Standardisierung durch die IETF

Einen Grund für den großen Erfolg des Internets stellt u. a. die Tatsache dar, dass die verwendeten Protokolle offene Standards und somit für jedermann zugänglich und verwendbar sind. Nur so konnte die Interoperabilität zwischen den einzelnen Kommunikationspartnern, die Produkte unterschiedlichster Hersteller verwenden, sichergestellt werden. Das TCP/IP-Schichtenmodell (siehe dazu auch Abschnitt 3.1 auf Seite 9) mit seinen Protokollen bildet die Grundlage des heutigen Internets. Sie werden zum größten Teil durch die Internet Engineering Task Force, kurz IETF, standardisiert.

Die IETF ist eine offene, internationale Freiwilligenvereinigung von Personen, zum größten Teil Netzwerktechniker, Hersteller, Netzbetreiber, Forscher oder professionelle Anwender, die sich mit der technischen Weiterentwicklung des TCP/IP-Protokollstapels und des Internets befassen. Da es sich ausdrücklich um eine offene Vereinigung handelt, steht sie jedem Interessierten zur Diskussion und Mitarbeit offen, es existiert keine förmliche Mitgliedschaft. Die IETF legt wichtige offene Standards, die sogenannten RFCs (Request for Comments, zu Deutsch: Forderung nach Kommentaren), fest. RFCs haben aus historischen Gründen zwar Empfehlungscharakter, faktisch jedoch sind sie normativ.

[2] http://www.softwareag.com/de/products/wm/tamino/default.asp

[3] http://catacomb.tigris.org/

[4] http://www.python.org

Die IETF arbeitet nach dem Leitspruch

„*We reject kings, presidents and voting. We believe in rough consensus and running code.*“

[Cla08]

Demnach bedarf es bei der Festlegung eines Standards oder bei einer Entscheidungsfindung keiner genauen Abstimmung, ein "grober Konsens" innerhalb der Arbeitsgruppe, dass heißt, es werden keine relevanten Einwände erhoben, ist ausreichend für eine Einigung.

Die IETF ist in die folgenden acht Bereiche unterteilt: Anwendungen, Internet-Dienste, Netzwerkmanagement, Echtzeitanwendung/Infrastruktur, Routing, Sicherheit, Transportdienste und Benutzerdienste. Diese Bereiche sind wiederum in Untergruppen aufgeteilt, aus denen sich Arbeitsgruppen, so genannte Working Groups, bilden, die gemeinsam ein Problem bewältigen. Die einzelnen Arbeitsgruppen kommunizieren dabei größten Teils über Mailing-Listen oder die dreimal im Jahr stattfindenden festen Treffen im Rahmen der IETF-Meetings.

Ziel jeder Arbeitsgruppe ist es, ein RFC zu erstellen, das den Status "Internet Standard" erhält. Nachdem ein RFC diesen Status erhalten hat, löst sich die Arbeitsgruppe im Regelfall wieder auf. Ein RFC kann innerhalb der IETF einen der folgenden Zustände einnehmen:

- Proposed Standard: Protokolle, die als Standard vorgeschlagen werden. Dies ist der erste Schritt zur Standardisierung des Protokolls
- Draft Standard: Bildet die Vorstufe zum Internet Standard. Hat ein Protokoll den Status Proposed Standard und gibt es mindestens zwei unabhängige und interoperable Implementierungen des Protokolls, erreicht es den Status Draft Standard. In dieser Phase können noch Modifikationen am Protokoll vorgenommen werden.
- Internet Standard bzw. Full Standard: Dies sind Protokolle, die als Standard für die Nutzung freigegeben sind.
- Informational Document: Diese RFCs können nahezu alles beinhalten, wie etwa Hinweise, Ideen oder Anleitungen.
- Best Current Practices (BCP): Bei diesen Dokumenten handelt es sich um administrative Anweisungen und Regeln.

- Experimental Protocols: Protokolle, die sich im experimentellen Status befinden. Dies stellt eine Vorstufe des Proposed Standards dar, die aber nicht zwingend vorgeschrieben ist.
- Historic Documents: RFCs, die durch neuere ersetzt wurden und deren Protokolle keine Anwendung mehr finden.

Als eigentlicher Standard werden nur Dokumente aus den ersten drei Kategorien (Proposed, Draft oder Internet Standard) durch die IETF akzeptiert und verstanden. Es gibt nur wenige RFCs und damit Protokolle, die den Status Internet Standard erreicht haben, z.B. Telnet und FTP. Selbst etablierte und viel genutzte Protokolle, z.B. das HTTP- oder das WebDAV-Protokoll, befinden sich im Status Proposed Standard und werden noch durch die entsprechende Arbeitsgruppe modifiziert und angepasst.

3 Stand von Wissenschaft und Technik

In diesem Kapitel soll der aktuelle Stand von Wissenschaft und Technik aufgezeigt werden. Dazu werden zunächst (Abschnitt 3.1) die Protokolle bzw. Standards vorgestellt, die in dieser Arbeit Verwendung finden. Insbesondere der Abschnitt zur DeltaV-Erweiterung (Abschnitt 3.2.2 auf Seite 17), der in dieser Arbeit aus dem entsprechenden RFC, welches sehr theoretisch und formell beschrieben ist, herausgearbeitet wurde, soll zur Dokumentation der Konzepte und Arbeitsweisen und als Grundlage für spätere Implementierungen dienen. Da sich die DeltaV-Erweiterung, wie auch bereits [KPSW04] beschreibt, als sehr komplex darstellt und es zurzeit auch noch keine Referenzimplementierung dieses Standards gibt, soll der entsprechende Abschnitt auch dazu beitragen, die zugrunde liegenden Konzepte anschaulich darzulegen und so die Komplexität zu reduzieren. Anschließend gibt Abschnitt 3.4 auf Seite 27 einen Überblick über Transaktionen im Allgemeinen. Zum Abschluss dieses Kapitels werden schließlich nicht-standardkonforme Transaktionsansätze in bestehenden Implementierungen betrachtet.

3.1 Grundlagen des HTTP-Protokolls

Das Hypertext Transfer Protocol (HTTP) ist ein Netzwerkprotokoll zur Übertragung von Daten. Ein Netzwerkprotokoll stellt eine Vereinbarung von Regeln und Formaten dar, die es durch ein Netzwerk verbundenen Kommunikationspartnern erlaubt, Daten auszutauschen. Da ein einzelnes Protokoll für alle dafür notwendigen Aufgaben zu komplex und unflexibel wäre, wurden die Aufgaben auf verschiedene Protokolle, wie etwa HTTP und SMTP, aufgeteilt. Die Protokolle sind dabei in Schichten angeordnet. Es existieren zwei nach [Tan02b] wichtige Referenzmodelle für die Anordnung der Protokolle und Aufteilung dieser Schichten. Zum einen das ISO/OSI-Referenzmodell [ISO96] und zum anderen das TCP/IP-Schichtenmodell (RFC1122) [RF1]. Das ISO/OSI-Modell findet im Gegensatz zum TCP/IP-Modell in der Praxis nur wenig Verwendung, es wird jedoch wegen seiner allgemei-

nen Gültigkeit stark im theoretischen Umfeld verwendet. Beiden Modellen gemeinsam ist, dass die jeweils höhere Schicht die Funktionen der darunter angeordneten Schicht nutzt. Die dabei entstehende Struktur wird auch als Protokollstapel bezeichnet [Tan02b]. Abbildung 3.1 zeigt die jeweiligen Modelle, sowie die Aufteilung von stark verbreiteten und bekannten Protokollbeispielen auf die jeweiligen Schichten. Die für diese Arbeit relevanten Protokolle, HTTP bzw. WebDAV, befinden sich in beiden Referenzmodellen jeweils in der höchsten Schicht, der sogenannten Anwendungs- bzw. Applikationsschicht.

<table>
<tr><th>ISO/OSI-Referenzmodell</th><th>TCP/IP Schichtenmodell</th><th>Protokollbeispiele</th></tr>
<tr><td>7 Anwendung</td><td rowspan="3">4 Anwendung</td><td rowspan="3">HTTP, WebDAV, FTP</td></tr>
<tr><td>6 Darstellung</td></tr>
<tr><td>5 Sitzung</td></tr>
<tr><td>4 Transport</td><td>3 Transport</td><td>TCP, UDP</td></tr>
<tr><td>3 Vermittlung</td><td>2 Vermittlung</td><td>IP,ICMP,IGMP</td></tr>
<tr><td>2 Sicherung</td><td rowspan="2">1 Netzwerkzugang</td><td rowspan="2">Ethernet, Token Ring, ATM</td></tr>
<tr><td>1 Bitübertragung</td></tr>
</table>

Abbildung 3.1: ISO/OSI-Referenzmodell und TCP/IP-Schichtenmodell

HTTP stellt eines der wichtigsten und am weitesten verbreiteten Protokolle des Internets dar. Es wurde 1989 von Tim Berners-Lee am CERN (Europäische Organisation für Kernforschung) entworfen. Heute wird es durch die IETF weiterentwickelt und betreut. Der aktuelle Stand des Protokolls, Version 1.1, ist im RFC 2616 als Proposed Standard festgelegt [RF2].

HTTP arbeitet auf einem Client-Server basierenden Nachrichten-Modell, bei dem Anfrage- (Request) und Antwortnachrichten (Response) zwischen Client und Server verschickt werden. Aus diesem Grund wird das Nachrichten-Modell auch Request-Response-Modell genannt. Ein Client, meist ein Browser, öffnet eine Verbindung und sendet eine Anfrage an einen HTTP-Server. Eine Client-Anfrage besteht dabei aus den folgenden drei Teilen:

- HTTP-Methode: Diese Methode bestimmt die gewünschte Aktion, die der Server ausführen soll. Zusätzlich enthält die Methode einen Verweis auf die Daten, die durch die Operation manipuliert bzw. gelesen werden sollen. In Listing 3.1 ist in der ersten Zeile ein Beispiel für die GET-Methode zu finden, die den Server anweist, eine bestimmte Datei an den Client auszuliefern.

  Die Daten, die der Client auf dem Server ansprechen kann, können aus beliebigen Inhalten, wie etwa HTML-Dateien, Bildern oder Videos bestehen. Sie werden als Ressource bezeichnet. Eine Ressource wird über einen sogenannten Uniform Resour-

ce Locator (URL) adressiert und benannt. Eine URL identifiziert eine Ressource eindeutig über das verwendete Netzwerkprotokoll und enthält als festen Bestandteil immer auch deren Speicherort. Die URL `http://dlr.de/TestDokument` identifiziert z.B. die Ressource `TestDokument` auf dem Server `dlr.de` über das HTTP-Protokoll.

- Verwendete Version: Ein Verweis auf die verwendete Version des HTTP-Protokolls. In Listing 3.1 ist am Ende der ersten Zeile zu sehen, dass die Version HTTP/1.1 verwendet wird.

- Spezielle Nachrichtenköpfe: Diese Nachrichtenköpfe, so genannte Header, bestehen aus Schlüssel/Wertpaaren, die beliebig zu einer Anforderung hinzugefügt werden können, wobei manche Schlüssel bei bestimmten Anfragen des Clients vorgegeben sind. In Listing 3.1 ist im Header, in der Zeile 2, der Schlüssel Host zu sehen.

Generell gilt, dass die folgenden HTTP- und DeltaV-Listings alle im Rahmen der Entwicklung des Prototypen erstellt und getestet wurden.

```
1 GET /TestDokument.html HTTP/1.1
2 Host: www.dlr.de
```

Listing 3.1: HTTP-Client Anfrage

Das HTTP-Protokoll definiert acht Methoden, wobei die, für diese Arbeit relevanten, im Folgenden näher vorgestellt werden. Für die anderen, hier nicht relevanten, Methoden TRACE und CONNECT sei auf die entsprechende Literatur in RFC2616 [RF2] sowie [GT02] verwiesen.

- DELETE: entfernt die angegebene Ressource auf dem Server

- GET: liefert die angegebene Ressource an den Client aus

- HEAD: weist den Server an, nur den HTTP-Header mit seinen Statusinformationen zu schicken

- OPTIONS: liefer die Methoden und Konzepte die von einem Server unterstützt werden

- POST: übermittelt Informationen an den Server, wie z.B. Formulardaten

- PUT: weist den Server an, die angegebenen Ressource unter Angabe des Ziels auf einen Web-Server hochzuladen.

Eine Antwort (Response) des Servers besteht aus den folgenden vier Teilen:

- Verwendete Version: Ein Verweis auf die verwendete Version des HTTP-Protokolls. In Listing 3.2 ist am Anfang der ersten Zeile die verwendete Version HTTP/1.1 zu sehen.
- Status: Dieser Status setzt sich aus einer dreistelligen Zahl, dem Statuscode, sowie einer kurzen textuellen Beschreibung hierzu zusammen. Sie zeigt dem Client an, ob seine Anfrage mit Erfolg ausgeführt werden konnte, oder ob ein Fehler aufgetreten ist. In Listing 3.2 zeigt am Ende der Zeile 1 der Statuscode mit dem Wert "200 OK" z.B. an, dass eine erfolgreiche Verarbeitung der Anfrage stattgefunden hat.
- Spezielle Nachrichtenköpfe: Diese Header haben den gleichen Aufbau wie die entsprechenden Client-Header.
- Daten: Die evtl. angeforderten Daten sind ebenfalls in der Antwort zu finden. In Listing 3.1 wurde die Ressource `TestDokument` angefordert, die in den Zeilen 6 bis 10 in Listing 3.2 zurückgeliefert wird.

```
1 HTTP/1.1 200 OK
2 Date: Wed, 12 Nov 2008 15:12:05 GMT
3 Content-Type: text/html
4 Content-Length: 2203
5
6 <html>
7    <body>
8    ...weiterer Dateiinhalt....
9    </body>
10 </html>
```

Listing 3.2: HTTP-Server Antwort

Bei HTTP handelt es sich um ein zustandsloses Protokoll, d.h. mehrere Anfragen desselben Benutzers werden getrennt voneinander behandelt. Insbesondere werden die Anfragen

ohne Bezug zu früheren Anfragen, auch des gleichen Benutzers, abgearbeitet, Sitzungsinformationen werden nicht ausgetauscht oder verwaltet. Erfolgt eine Anfrage eines Clients mittels des HTTP-Requests an den in der URL spezifizierten Web-Server, wird eine TCP-Verbindung aufgebaut, über die der Request erfolgt. Wenn der Web-Server die angeforderte Ressource zur Verfügung stellen kann, wird die Ressource über dieselbe Verbindung durch den HTTP-Response zurückgeschickt. Anschließend wird die Verbindung geschlossen. Müssen weitere Daten übertragen werden (z.B. Bilder oder Dateien), wird der gleiche Vorgang, insbesondere mit erneutem Verbindungsaufbau, wiederholt, bis die gewünschten Daten beim Client angekommen sind. Diese Eigenschaften, sowie die Tatsache, dass die Kommunikation nur vom Benutzer ausgehen kann, stellen besondere Anforderungen an die Entwicklung des Transaktionsmodells, da, wie bereits erwähnt, die Anfragen eines Benutzers als voneinander unabhängig durch den Server bearbeitet werden.

3.2 Grundlagen des WebDAV-Standards

Web-based Distributed Authoring and Versioning, kurz WebDAV, ist eine auf XML basierende Erweiterung des HTTP-Standards. Ziel von WebDAV ist es, Befehle und Konzepte zur Manipulation von Inhalten auf einem Web-Server zu bieten. So soll insbesondere eine Unterstützung für verteilte Zusammenarbeit, dem sogenannten Web-Authoring, durch den WebDAV-Standard ermöglicht werden. Die von HTTP in dieser Hinsicht angebotenen Methoden und Konzepte sind entweder unzureichend oder sie werden, wie etwa die DELETE – Methode zum Löschen einer Ressource, bei vielen Web-Servern nicht implementiert [Dus04].

WebDAV stellt ein verteiltes Dateisystem über das Netzwerk bereit. Die Architektur des WebDAV-Dateisystems entspricht dabei einer Client-Server-Architektur. Innerhalb dieser Client-Server-Architektur unterscheidet [TvS08] zwischen dem Remote-Access-Modell und dem Upload/Download-Modell. WebDAV arbeitet nach dem Upload/Download-Modell, bei dem ein Client lokal Änderungen an einer Ressource vornimmt, die er vorher von einem Server heruntergeladen hat. Im Gegensatz dazu wird bei der Verwendung des Remote-Access-Modells dem Client ein transparenter Zugriff auf die Dateien gewährt, wobei die Daten auf dem Server verbleiben. Dieses Modell wird u.a. von dem im Unix-Umfeld stark verbreiteten NFS (Network File System) der Firma Sun Microsystems verwendet (siehe Abbildung 3.2).

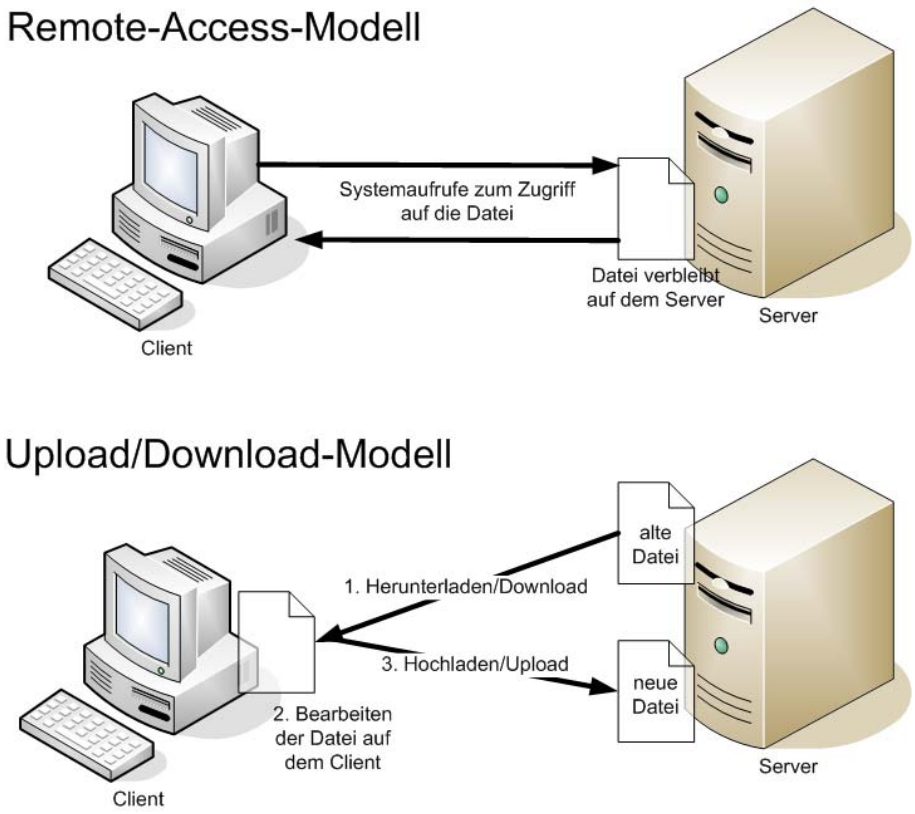

Abbildung 3.2: Remote-Access-Modell und Upload/Download Modell, angelehnt an [TvS08]

Initiiert wurde die Entwicklung des WebDAV-Standards 1996 durch die Bildung einer Arbeitsgruppe aus Mitgliedern der IETF und des World Wide Web Consortium (W3C) unter Führung von Jim Whitehead.

Insgesamt besteht WebDAV nicht aus einem einzelnen Standard, sondern es wurden mehrere Standards mit jeweils unterschiedlichen Schwerpunkten miteinander kombiniert. Die Basis bildet dabei RFC 4918 [RFCc], dass durch die IETF im Jahr 1999 als Proposed Standard herausgegeben wurde und in Abschnitt 3.2.1 vorgestellt wird.

Eine weitere wichtige Erweiterung im Bezug auf die Umsetzung der Transaktionen im Rahmen dieser Arbeit bildet die DeltaV-Erweiterung von WebDAV, auf die in Abschnitt 3.2.2 näher eingegangen wird. In Abschnitt 3.2.3 schließlich werden der Vollständigkeit halber die weiteren WebDAV-Erweiterungen, wie z.B. ACL zur Rechteverwaltung des Benutzers, aufgeführt.

Umgesetzt wird das WebDAV-Protokoll inzwischen von vielen Applikationen, allerdings selten in allen Teilen und Erweiterungen. So unterstützen populäre kommerzielle Pro-

gramme, wie etwa Microsoft Windows und Office oder Adobe Photoshop, den WebDAV-Basis-Standard. Weiterhin werden zahlreiche WebDAV-fähige Open-Source-Anwendungen angeboten, beispielsweise der KDE Desktop, DAVExplorer oder Subversion [Dus04].

3.2.1 WebDAV-Basis-Standard

WebDAV legt neben der Protokollspezifikation auch bestimmte Datenobjekte fest. Dies hat Auswirkungen sowohl auf den Server, der diese Datenobjekte über das Protokoll anbieten muss, als auch auf den jeweiligen Client, der die Daten bearbeiten möchte. Das Datenmodell von WebDAV basiert auf einer hierarchischen Baumstruktur, welche die folgenden zwei Objekte verwendet:

- Ressource: Der Begriff der Ressource wurde aus dem HTTP-Protokoll übernommen. Alle Objekte, die durch eine URL adressiert werden können, stellen eine Ressource dar. Eine Datei auf einem Web-Server ist z.B. eine solche Ressource.
- Collection: Eine Collection stellt eine Ressource mit zusätzlichen Eigenschaften dar. Eine Collection kann andere Ressourcen beinhalten. Somit kann eine Collection z.B. mit einem Ordner in einem Dateisystem verglichen werden. Durch die entsprechende Verknüpfung von Collections und Ressourcen kann nun eine Hierarchie bzw. Baumstruktur zur Organisation der Daten aufgebaut werden, um so z.B. ein komplettes Dateisystem abzubilden.

Zusätzlich zu diesen beiden Ressourcen bietet der Basis-Standard Ressourcen zur Kennzeichnung und Verwaltung von Benutzern und Gruppen. Diese finden jedoch in dieser Arbeit keine weitere Verwendung und werden deshalb nicht weiter betrachtet. Hierzu sei auf das entsprechenden RFC 3744[1] verwiesen.

WebDAV-Properties

Jede Ressource besitzt Attribute bzw. Eigenschaften, die so genannten Metadaten. Metadaten sind Daten, die Informationen über andere Daten enthalten. Diese werden im WebDAV-Protokoll als so genannte Properties (Eigenschaften) bezeichnet und werden in die folgenden zwei Gruppen unterteilt:

[1] http://www.ietf.org/rfc/rfc3744.txt

- Live Properties: Diese Eigenschaften werden durch den Server bzw. das System automatisch erfasst und verwaltet. Beispiele hierfür sind das Datum der letzten Änderung, sowie die Größe der Ressource.
- Dead Properties: Im Gegensatz zu den Live Properties werden die Dead Properties komplett vom Client verwaltet und überprüft. Der Server speichert diese Werte nur, kümmert sich aber nicht weiter um ihre Korrektheit. Die Werte können frei gewählt werden, wie z.B. der Titel oder Autor eines Dokumentes.

WebDAV-Methoden

Das Standard WebDAV-Protokoll erweitert die Methoden des HTTP-Protokolls um die folgenden neuen Methoden:

- COPY: Kopiert eine Ressource von einer URL zur einer anderen.
- MKCOL: Erstellt eine neue Collection.
- MOVE: Verschiebt eine Ressource von einer URL zu einer anderen
- PROPPATCH: Dient zum Löschen oder Ändern von Properties.
- PROPFIND: Mit dieser Methode können die Properties einer Ressource abgefragt werden. Außerdem dient sie dazu, sämtliche Ressourcen einer Collection in Erfahrung zu bringen.
- LOCK: Setzt einen Lock (Schreib–Sperre) auf einer Ressource.
- UNLOCK: Entfernt die Sperre wieder.

WebDAV-Sperren

Das Sperren und Freigeben von Ressourcen (Locking) stellt einen wichtigen Aspekt zur Fehlervermeidung bei der parallelen Verarbeitung von Ressourcen dar. Damit kann z.B. verhindert werden, dass sich Benutzer ihre Änderungen gegenseitig überschreiben. Im Folgenden werden die Mechanismen, die das WebDAV-Protokoll in diesem Bereich bietet, näher vorgestellt.

Das WebDAV-Protokoll bietet mit Hilfe der LOCK-Methode die Möglichkeit, den Zugriff auf eine Ressource einzuschränken. Beim Ausführen der LOCK-Methode kommen zwei Arten des Lockings von Ressourcen zur Anwendung:

- Exclusive Locks: Am häufigsten wird das Exclusive Locking verwendet. So ist sichergestellt, dass nur der Benutzer, der den Lock auf einer Ressource angelegt hat, auch eine Bearbeitung vornehmen kann. Wollen andere Benutzer diese Ressource bearbeiten, so müssen sie warten, bis der Besitzer des Locks dieses durch die UNLOCK–Methode wieder entfernt hat.
- Shared Locks: Das Shared Locking erlaubt den gleichzeitigen Zugriff von mehreren Benutzern auf eine Ressource. So kann der Zugriff für eine bestimmte Gruppe von Benutzern erlaubt werden. Für alle anderen Benutzer ist die Ressource jedoch weiterhin gesperrt. Unter Ausnutzung eventueller Absprachen zwischen den konkurrierenden und kooperierenden Benutzern ist so ein gleichzeitiges Bearbeiten einer Ressource möglich.

Realisiert werden diese Locks durch so genannte Sperrtokens, die der Client, sofern er eine Ressource sperren möchte, vom Server anfordern kann. Möchte der entsprechende Client nun das Dokument verändern, muss er den vorher erhaltenen Token zur Authentifizierung im Header seiner Anfrage mitschicken und erhält damit schreibenden Zugriff auf die entsprechende Ressource. Die beiden Locks verhindern dabei jedoch nur den schreibenden Zugriff auf die jeweilige Ressource, sind also so genannte Schreib-Sperren (write-locks). Das Lesen der Ressource ist somit durch andere Benutzer weiterhin möglich, da WebDAV die so genannten Lese-Sperren (read-locks), um auch das Lesen von Ressourcen zu verhindern, nicht unterstützt.

3.2.2 Versionierung von Ressourcen - DeltaV

Die Versionierung von Ressourcen hat die DeltaV Working Group, eine IETF-Arbeitsgruppe, ähnlich der für WebDAV, übernommen. Mit der Veröffentlichung des RFC3253 [RFCa] im März 2002 als Proposed Standard wurde die WebDAV-Spezifikation um Versionskontrolleigenschaften erweitert. Das folgende Zitat beschreibt sehr kompakt das Aufgabengebiet der Versionskontrolle:

"Versionskontrolle bezeichnet die Kunst, Anderungen von Informationen zu verwalten."
Ben Collins-Sussman in [CSFP06]

Mit Hilfe der DeltaV-Erweiterung können nun Änderungen von Ressourcen archiviert werden. Sinn und Zweck dieser Archivierung ist u.a. die Möglichkeit, vorherige Versions-

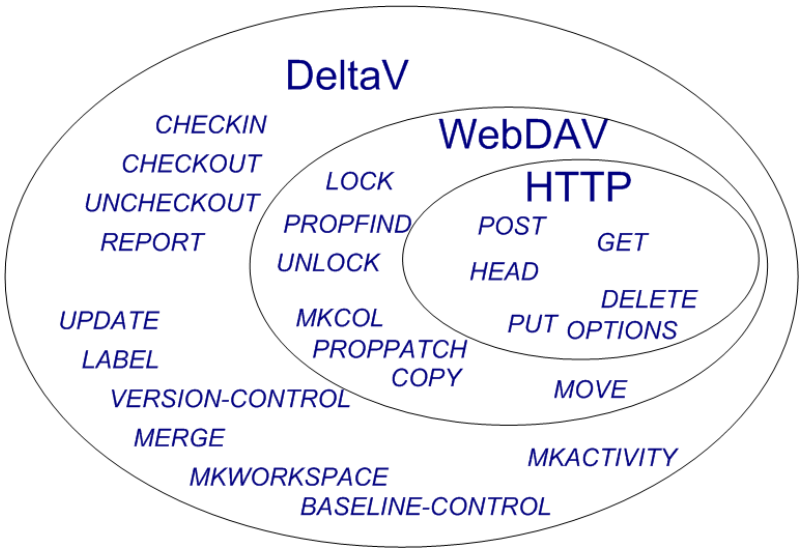

Abbildung 3.3: Methoden von HTTP, WebDAV und DeltaV

stände wiederherzustellen und Änderungen zurückzunehmen, was insbesondere im Bereich der Software-Entwicklung intensiv genutzt wird. Zusätzlich entsteht durch die Versionsverwaltung eine lückenlose Historie aller Änderungen.

DeltaV erweitert WebDAV um elf neue Methoden, sowie verschiedene Verarbeitungskonzepte. Abbildung 3.3 gibt einen Überblick über alle Methoden, die nun durch die Verwendung von HTTP, WebDAV und DeltaV zur Verfügung stehen.

Die Methoden und abstrakten Konzepte der DeltaV-Erweiterung, die für diese Arbeit relevant sind, werden in den folgenden Abschnitten detaillierter beschrieben. Für eine genaue Beschreibung der restlichen Methoden und Konzepte sei auf das entsprechende RFC [RFCa] sowie auf [Dus04] verwiesen.

DeltaV-Datenelemente

So wie die WebDAV-Spezifikation das HTTP-Protokoll um Konzepte erweitern, werden auch mit DeltaV Erweiterungen am bestehenden Datenmodell vorgenommen. Eine anfangs nicht unter Versionskontrolle stehende Ressource, eine sogenannte *Versionable Resource*, wird mit der Methode VERSION-CONTROL der Versionskontrolle unterworfen. Ist eine Ressource nun unter Versionskontrolle, wird sie als *Version Controlled Resource* (VCR) bezeichnet. Eine VCR besitzt eine feste URL für die jeweilige Ressource, über die alle Operationen zur Versionskontrolle durchgeführt werden können. Die URL der VCR verändert sich dabei nicht, egal wie oft die einzelne Ressource auch verändert wird. Bei jeder

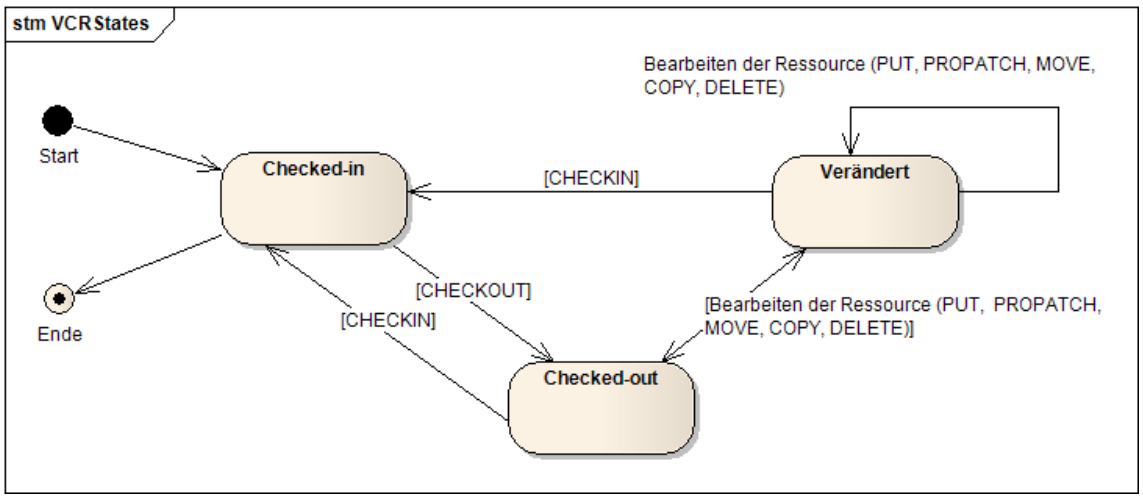

Abbildung 3.4: Zustände im Checkin/Checkout-Modell

Änderung an der VCR wird jeweils eine neue Version erstellt, diese neu erstellte Version wird dabei als *Version Resource* (VR) bezeichnet (nicht zu verwechseln mit VCR) und besitzt selbst auch wieder eine eindeutige URL. Diese URL repräsentiert eine unveränderbare Version der Ressource, mit allen Eigenschaften und Inhalten, zu einem bestimmten Zeitpunkt, was somit einem linearen Versionierungsmodell entspricht. In den nächsten beiden Abschnitten wird nun vorgestellt, wie neue Versionen erstellt werden und auf die jeweiligen Daten zugegriffen werden kann.

Checkin/Checkout-Modell

DeltaV arbeitet nach dem Checkin/Checkout-Modell. Wie der Name des Modells bereits andeutet, kann sich eine Ressource entweder im Zustand Checked-in oder im Zustand Checked-out befinden. Ressourcen können dabei nur geändert werden, wenn sie sich im Zustand Checked-out befinden. Durch die DeltaV-Methode CHECKOUT kann eine Ressource (VCR) in den Zustand Checked-out versetzt werden und es können dann Änderungen hieran vorgenommen werden. Dieser Vorgang wird auch als "auschecken" bezeichnet. Nachdem die Bearbeitung abgeschlossen ist, wird die Ressource mit der Methode CHECKIN in den Zustand Checked-in versetzt und es wird eine neue Version (VR) erstellt. Dieser Vorgang auch auch als "einchecken" bezeichnet. Mit der Methode UNCHECKOUT können Änderungen wieder verworfen werden und es wird keine neue Version erstellt. Abbildung 3.4 fasst die einzelnen Zustände noch einmal grafisch zusammen.

DeltaV-Datenorganisation

Befindet sich eine VCR im Zustand Checked-in, besitzt sie ein Live-Attribut mit dem Namen "DAV:checked-in". Dieses Attribut beinhaltet als Wert eine URL, die auf die aktuelle VR, die so genannte "Target Version", zeigt. Mit Hilfe dieses Attributes kann auf die aktuellste Version der Ressource zugegriffen werden.

Zusätzlich besitzt jede VR die beiden Live-Properties "predecessor-set" und "successor-set", mit deren Hilfe Beziehung zwischen den verschiedenen Versionen einer Ressource hergestellt werden. Das Attribut "predecessor-set" enthält die URLs der direkten Vorgänger der jeweiligen Version, deswegen wird dieses Attribut auch als Menge umgesetzt. Das Attribut "successor-set" beinhaltet alle Nachfolgeversionen der jeweiligen VR. Hier wird ebenfalls wieder mit einer Menge gearbeitet, da eine Version auch in mehrere Nachfolgeversionen einfließen kann. Mit Hilfe dieser beiden Attribute entsteht eine Art doppelt verkettete Liste, in der nach der gewünschten Version gesucht werden kann. Abbildung 3.5 verdeutlicht diesen Aufbau.

Die VCR `http://dlr.de/TestDokument` ist im Zustand Checked-in. Der Inhalt des Live-Properties "DAV:checked-in" enthält die URL der aktuellen Version, die VR `http://dlr.de/vr/TestDokument3`. Um nun auf eine ältere Version der Datei zuzugreifen, kann im "predecessor-set" der VR `http://dlr.de/vr/TestDokument3` nachgesehen werden, welche Version vorausging (in der Abbildung also die VR `http://dlr.de/vr/TestDokument2`) und die entsprechende URL ausgelesen werden.

DeltaV-Working-Ressource

Die DeltaV-Working Ressource stellt eine exakte Kopie einer bestehenden VCR dar, die vom Server in einen extra bereitgestellten privaten Arbeitsbereich kopiert wird. So hat der Benutzer die Möglichkeit, eine VCR zu bearbeiten, ohne dass die Ergebnisse der Bearbeitung für andere Benutzer sichtbar sind. Zusätzlich zu den bestehenden Properties der Quell-VCR erhält die neu erstellte Working-Ressource das Live-Property "DAV:auto-update", welches die URL der originalen VCR enthält. Eine Working-Ressource kann mit Hilfe der Option "D:apply-to-version" der CHECKOUT-Methode erzeugt werden. Listing 3.3 zeigt, wie eine Working-Ressource für eine VCR `TestDokument` angelegt wird.

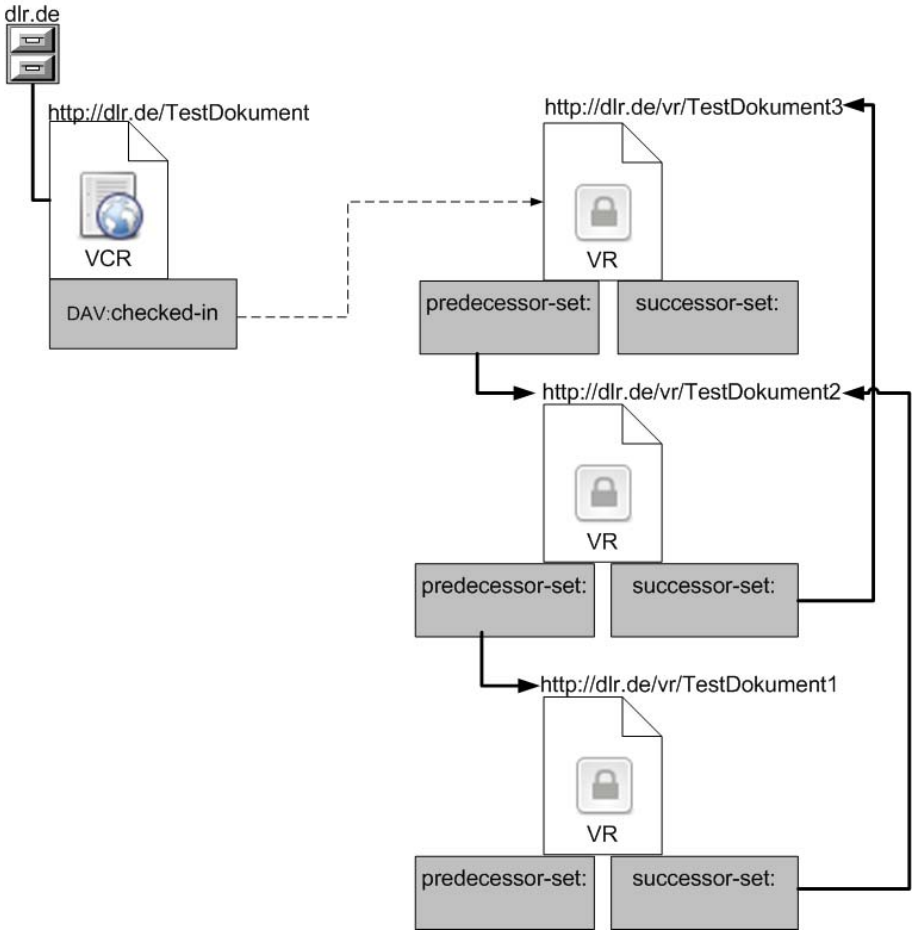

Abbildung 3.5: DeltaV-Datenorganisation

```
1 CHECKOUT /TestDokument HTTP/1.1
2 Host: dlr.de
3 Content-Type: text/xml; charset="utf-8"
4 Content-Length: 1234
5
6 <?xml version="1.0" encoding="utf-8" ?>
7 <D:checkout xmlns:D="DAV:">
8    <D:apply-to-version/>
9 </D:checkout>
```

Listing 3.3: Checkout einer VCR

In der Antwort des Servers wird die neue URL der Working-Ressource angegeben, mit der der Client dann arbeiten kann. Listing 3.4 zeigt die Antwort des Servers nach der Anfrage

aus Listing 3.3. Der eigentliche Speicherort der Working-Ressource auf dem Server kann dabei (vom Server) frei gewählt werden.

```
1 HTTP/1.1 201 Created
2 Location: http://dlr.de/WR/TestDokument
```

Listing 3.4: Serverantwort auf den Checkout

Abbildung 3.6 zeigt den Zustand des Servers nach der Ausführung der CHECKOUT-Methode in Listing 3.3. Die VCR `http://dlr.de/TestDokument` befindet sich im Zustand Checked-out und es wurde die Working-Ressource `http://dlr.de/WR/TestDokument` im Ordner WR auf dem Server angelegt. Das Attribut "DAV:auto-update" verwaltet die URL der ursprünglichen Quell-VCR.

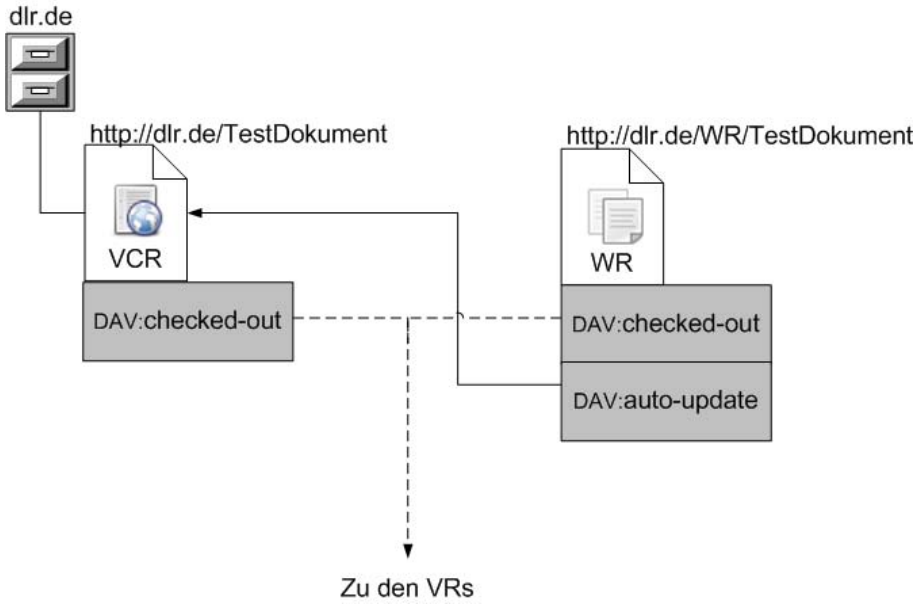

Abbildung 3.6: Zustand auf dem Server nach dem Checkout einer VCR

Wird nun auf einer Working-Ressource die Methode CHECKIN aufgerufen, liest der Server zunächst das Attribute "DAV:auto-update" aus, um zu erfahren, zu welcher ursprünglichen VCR die Working-Ressource gehört. Falls nicht schon geschehen, wird nun die originale VCR durch den Server ausgecheckt und anschließend mit den Daten der Working-Ressource aktualisiert. Nach erfolgreicher Aktualisierung wird die originale VCR wieder durch den Server eingecheckt und die Working-Ressource durch den Server gelöscht. Das bedeutet also, dass sich eine Working-Ressource nur im Zustand Checked-out befinden kann.

DeltaV-Activity-Ressource

Eine weitere Ressource, also ein mit Hilfe einer URL adressierbares Datenobjekt, um die das HTTP-Protkoll durch DeltaV erweitert wird, ist die so genannte Activity. Die Aufgabe der Activity-Ressource ist es, eine Aufgabe bzw. Aktivität zu allgemeinen Dokumentationszwecken oder auch zu Zwecken der Versionierung zu kennzeichnen. Dies geht so von statten, dass die zugehörigen Ressourcen der Aufgabe bzw. Aktivität einer Activity-Ressource zugeordnet werden. Dabei kann es sich sowohl um eine Einzelaufgabe eines Nutzers, als auch um die Gemeinschaftsaufgaben mehrerer Nutzer, die nicht notwendiger Weise verschieden sein müssen, handeln. So kann eine Activity z.B. dafür verwendet werden, dass Beheben eines Software-Fehlers (eines so genannten Bugfixes), in einem Entwicklungsprozess einer Software festzuhalten. Dazu werden die Dateien, die für die Behebung des Software-Fehlers benötigt werden, einer Activity "Bugfix" hinzugefügt.

Detailliert betrachtet kann die Activity-Ressource eine Menge von VCRs aufnehmen, die sich im Zustand Checked-out befinden. Der Begriff Menge sei hier an den mathematischen Begriff der Menge angelehnt, was bedeutet, dass keine doppelten VCRs auftreten dürfen. Die Activity-Ressource besitzt zur Verwaltung einer Menge das Live-Property "DAV:activity-checkout-set". In diesem Attribut stehen die URLs der VCRs, die der Activity zugewiesen wurden und sich im Zustand Checked-out befinden. Wie jede Ressource besitzt auch die Activity-Ressource eine eindeutige URL. Wird nun diese URL von einer CHECKIN-Methode aufgerufen, veranlasst dies den Server, für alle VCRs, die sich in der Menge "DAV:activity-checkout-set" befinden, die Methode CHECKIN aufzurufen. Alle durch die Methode CHECKIN erfolgreich bearbeiteten VCRs werden dann durch den Server automatisch aus dem Attribut "DAV:activity-checkout-set" gelöscht. Das Live-Property "DAV:activity-version-set" der Activity-Ressource nimmt alle bereits durch den Server erfolgreich abgearbeiteten VCRs mit ihrer neuen VR – URL, also der URL des neuen Versionsstandes, auf. Mit Hilfe der Methode MKACTIVITY und Angabe einer Ziel-URL kann der Client durch den Server eine Activity unter der entsprechend angegebenen URL anlegen lassen. Listing 3.5 zeigt, wie die Activity actUserA auf dem Server `dlr.de` im Verzeichnis /act angelegt wird.

```
1 MKACTIVITY /act/actUserA HTTP/1.1
2 Host: dlr.de
3 Content-Length: 0
```

Listing 3.5: Anlegen einer Activity

Wird nun eine VCR mit Hilfe der Methode CHECKOUT ausgecheckt, kann der Methode als Option die URL einer Activity-Ressource mitgegeben werden, zu der die VCR hinzugefügt werden soll. Der Server muss dann automatisch dafür sorgen, dass die URL der ausgecheckten VCR in die Menge "DAV:activity-checkout-set" der entsprechenden Activity aufgenommen wird. Listing 3.6 zeigt in den Zeilen 8 bis 10, wie die VCR `TestFooDokument` der Activity `http://dlr.de/act/actUserA` hinzugefügt wird.

```
1 CHECKOUT TestFooDokument HTTP/1.1 Host: dlr.de
2 Content-Type: text/xml; charset="utf-8"
3 Content-Length: 1234
4
5 <?xml version="1.0" encoding="utf-8" ?>
6 <D:checkout xmlns:D="DAV:">
7     <D:activity-set>
8       <D:href>http://dlr.de/act/actUserA</D:href>
9    </D:activity-set>
10 </D:checkout>
```

Listing 3.6: Checkout einer VCR zu einer Activity

Abbildung 3.7 verdeutlicht die Situation auf dem Server, wenn zusätzlich zu der VCR `TestFooDokument` auch die VCR `TestBarDokument` ausgecheckt und der Activity `http://dlr.de/act/actUserA` hinzugefügt wurde.

Abbildung 3.8 zeigt den Zustand auf dem Server, nachdem die Activity erfolgreich eingecheckt und der Server dadurch angewiesen wurde, auch die beiden VCRs `TestFooDokument` und `TestBarDokument` einzuchecken. Die Menge "DAV:activity-checkout-set" wurde geleert und die neu erstellten VRs sind in der Menge „DAV:activity-version-set” eingetragen worden. Des Weiteren bekommen durch den Checkin-Prozess auch die jeweilige Live-Properties "DAV:checked-in" der beiden VCRs die neuen URL-Adressen der neuen Version, d.h. die VR, eingestellt.

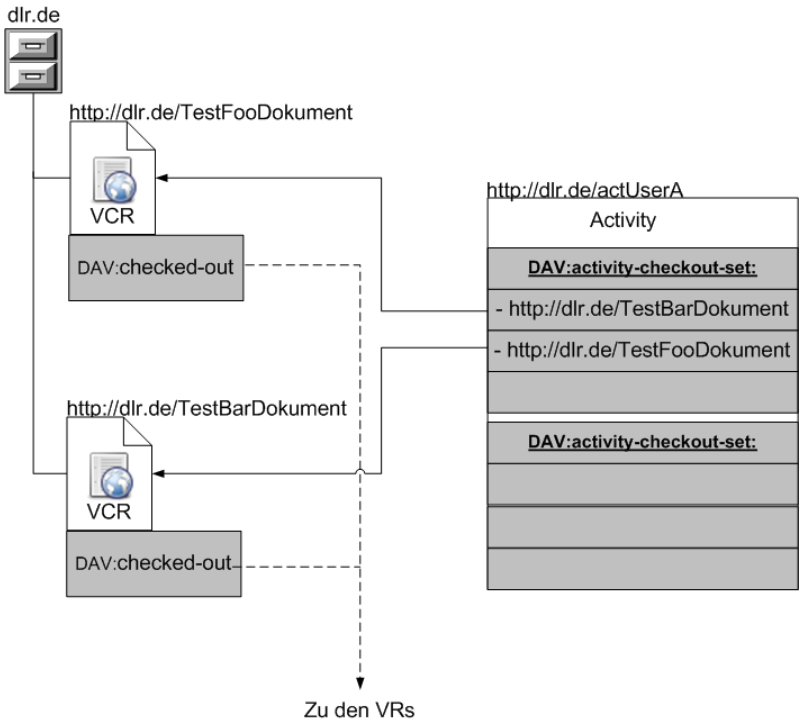

Abbildung 3.7: Zustand auf dem Server nach dem Checkout einer VCR zu einer Activity

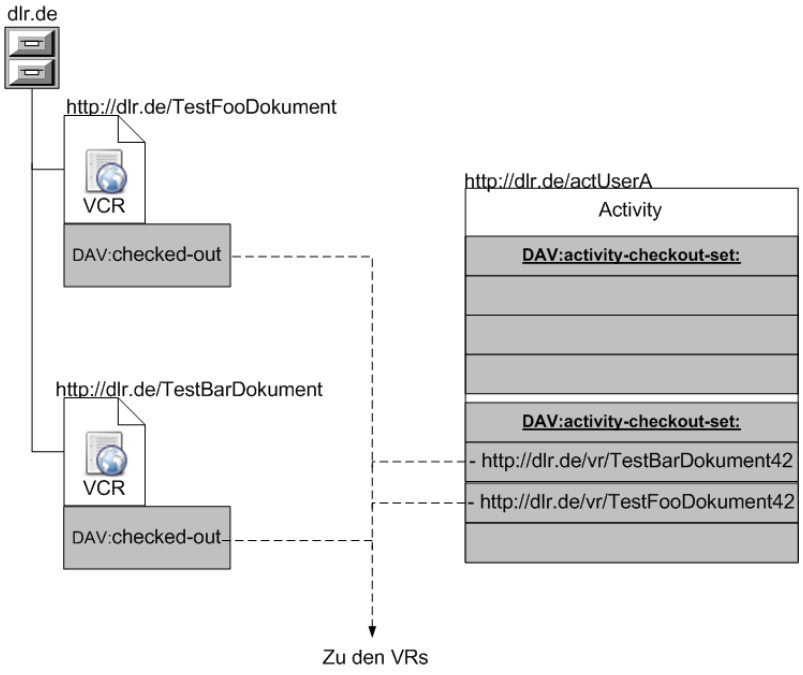

Abbildung 3.8: Zustand auf dem Server nach einem Checkin der Activity-Ressource

3.2.3 Weitere WebDAV-Standards

Im Folgenden werden die weiteren WebDAV-Standards, neben DeltaV, aus Gründen der Vollständigkeit aufgeführt, auch wenn sie in dieser Arbeit keine direkte Verwendung finden.

- Access Control Protocol –ACP: Das Access Control Protocol (ACP) erlaubt es Clients, Zugriffsbeschränkungen des Servers zu lesen und zu bearbeiten. Es ist in RFC 3744[2] als Proposed Standard definiert.

- Calendaring Extension – CalDAV: Die WebDAV Calendaring Extension (CalDAV) beschreibt einen auf dem iCalendar-Format basierenden Zugriff auf Kalender- und Terminplaner-Ressourcen. Es ist in RFC 4791[3] als Proposed Standard definiert.

- DAV Searching and Locating – DASL: DASL ist eine Erweiterung des WebDAV-Protokolls, die das Durchsuchen und Auffinden von Ressourcen ermöglicht. Es ist in RFC 5323[4] als Proposed Standard definiert.

- Datatype Properties: Dieser Standard beschreibt eine Menge von Datentypen, die WebDAV-Clients und -Server verwenden können. Es ist in RFC 4316[5] als Experimental definiert.

- Mount: Bei dieser Empfehlung handelt sich um keinen Standard. Diese Empfehlung soll Client- und Browser-Entwicklern helfen, ein einheitliches Vorgehen im Umgang mit WebDAV-Ressourcen zu erreichen. Es ist ein Informational RFC, beschrieben in RFC 4709[6].

- Ordered Collections: Das WebDAV Ordered Collections Protocol beschreibt, wie Datensammlungen serverseitig geordnet abgelegt werden können. Es ist in RFC 3648[7] als Proposed Standard definiert.

- Quota: Die Definition von WebDAV Quotas dient der serverseitigen Umsetzung von Beschränkungen des zur Verfügung stehenden Speicherplatzes. Es ist als Proposed Standard in RFC 4331[8] definiert.

[2] http://www.ietf.org/rfc/rfc3744.txt
[3] http://www.ietf.org/rfc/rfc4719.txt
[4] http://www.ietf.org/rfc/rfc5323.txt
[5] http://www.ietf.org/rfc/rfc4316.txt
[6] http://www.ietf.org/rfc/rfc4709.txt
[7] http://www.ietf.org/rfc/rfc3648.txt
[8] http://www.ietf.org/rfc/rfc4331.txt

- Redirect: Der WebDAV Redirect ermöglicht dem Client das Anlegen und Bearbeiten von Weiterleitungen. Es ist RFC 4437[9] als Experimental definiert.

3.3 Extensible Markup Language (XML)

Die "eXtensible Markup Language", kurz XML, ist eine durch das World-Wide-Web-Konsortium (W3C) eingeführte Sprache zur strukturierten Auszeichnung von Daten. XML wurde aus der Standard Generalized Markup Language (SGML) abgeleitet und ist sowohl für Menschen als auch Maschinen lesbar. Die Regeln für den Aufbau von XML-Dokumenten sind streng festgelegt, um so gerade beim Datenaustausch oder bei der Schnittstellendefinition zu verhindern, dass Daten falsch interpretiert werden. Die Struktur eines XML-Dokumentes entspricht einer Baumstruktur. Als Knoten kommen Elemente und Attribute in Frage, es existiert nur ein Wurzelelement als Ausgangsknoten. Wenn ein XML-Dokument diese beiden Anforderungen erfüllt, wird es als "wohlgeformt" bezeichnet. Mit Hilfe einer so genannten Schema-Definition lassen sich XML-Dokumente strukturell und inhaltlich einschränken, wodurch anwendungsspezifische Sprachen definiert werden können. Von diesen Schema-Definitionen gibt es zurzeit zwei Ausprägungen, die Document Type Definition (DTD) und das XML-Schema. Entspricht ein Dokument einer der beiden Schema-Definitionen und ist es "wohlgeformt", gilt es als "valide" [Von07].

3.4 Einführung in Transaktionen

Transaktionen haben in der Informatik eine lange Tradition, ihre Anfänge gehen bis weit in die 60er Jahre des letzten Jahrhunderts zurück. Erstmalig wurden Transaktionen bei der Entwicklung des Flugbuchungssystems SABRE[10] (Akronym für Semi-Automatic Business Research Environment) der American Airlines durch die Firma IBM verwendet und stellen bis heute ein wichtiges Forschungsfeld in der Informatik dar. Wegen der Größe des Themengebietes kann dieser Abschnitt nur einen Kurzüberblick über die für diese Arbeit relevanten Teile liefern. Für einen detaillierten Überblick sei auf [BN97] verwiesen.

Es gibt eine Vielzahl von Definitionen für den Begriff Transaktion. Diese Arbeit verwendet die Definition von Jim Gray und Andreas Reuter [GR93]. Jim Gray hat für seine Forschungen im Bereich der Transaktionen u. a. im Jahre 1998 den Turing Award erhal-

[9] http://www.ietf.org/rfc/rfc4437.txt
[10] http://www.sabretravelnetwork.com/

ten. Transaktionen stellen gemäß seiner Definition eine Menge von Operationen dar, die als logische Einheit betrachtet werden, und die die Eigenschaften Atomarität (atomicity), Konsistenz (consistency), Isolation (isolation) und Dauerhaftigkeit (durability) besitzen. Diese vier Eigenschaften bilden das bekannte ACID-Prinzip und werden im folgenden Abschnitt genauer vorgestellt. Darauf aufbauend werden dann im folgenden Abschnitt Transaktionen anhand ihres Aufbaus in verschiedene Transaktionsklassen aufgeteilt.

3.4.1 ACID-Eigenschaften

- Atomarität bedeutet, dass entweder alle Operationen einer Transaktion erfolgreich durchgeführt werden, oder aber keine. Aufgrund dieses „Alles oder Nichts“-Prinzips bleibt z. B. im Falle eines Absturzes während der Abarbeitung einer Transaktion das System unverändert.

- Konsistenz heißt, dass nach Abarbeitung einer Transaktion ein konsistenter, also widerspruchsfreier und in sich stimmiger Systemzustand vorliegt, der alle definierten Integritätsbedingungen erfüllt. Voraussetzung ist, dass auch vor Beginn der Transaktion ein solcher konsistenter Systemzustand vorlag.

- Die Sicherstellung der Isoliertheit sorgt dafür, dass sich die in Ausführung befindlichen Transaktionen nicht gegenseitig beeinflussen oder behindern, also jeweils isoliert voneinander ablaufen.

- Dauerhaftigkeit bedeutet, dass die durch die erfolgreiche Ausführung der Transaktion erzielten Resultate dauerhaft im System verbleiben und z.B. auch im Falle eines Absturzes nicht verloren gehen.

3.4.2 Klassifikation von Transaktionen

Transaktionen können anhand ihres Aufbaus in verschiedene Klassen eingeteilt werden.

Im Folgenden wird mit der Klasse der flachen Transaktionen das Grundmodell klassischer Transaktionen vorgestellt, dass in der Praxis eine sehr weite Verbreitung findet. Diese Klasse ist jedoch mit ihren strikten Eigenschaften für manche Anwendungen und Prozesse ungeeignet, weil sich z.B. bei sehr großen Transaktionen evtl. Performance- und Durchsatzprobleme durch lang andauernde Sperren ergeben. Deshalb haben sich aus diesem Grundmodell weitere Transaktionsklassen gebildet. Mit den Klassen der verteilten und

verschachtelten Transaktionen werden zwei weitere, nach [TvS03] wichtige und weit verbreitete Klassen vorgestellt. Mit der Klasse der langlaufenden Transaktionen wird schließlich eine, insbesondere im Web-Umfeld sehr häufig verwendete, Klasse vorgestellt. Gray und Reuter liefern in [GR93] einen ausführlichen Überblick über diese und weitere, im Rahmen dieser Arbeit aber nicht näher betrachteten Transaktionsklassen.

- Flache Transaktionen (Flat Transactions): Ein Modell, das die ACID-Eigenschaften strikt erfüllt. Sie stellen die einfachste und am weitesten verbreitete Art von Transaktionen dar. Flache Transaktionen wurden für kurze Buchungsoperationen entwickelt, wie sie beispielsweise im Bankensektor, oder bei Flugbuchungen, auftreten. Wie der Name bereits vermuten lässt, besitzen flache Transaktionen keine Hierarchien, die Abhängigkeiten zu anderen Transaktionen herstellen würden.

- Verschachtelte Transaktionen (Chained Transactions): Verschachtelte Transaktionen besitzen eine Hierarchie, in der eine Eltern-Transaktion eine oder mehrere Kind-Transaktionen enthalten kann. Die Eltern-Transaktion ist dabei nur erfolgreich, wenn auch alle Kind-Transaktionen erfolgreich waren. Durch dieses Konzept ist es möglich, große Transaktionsvorgänge in verschiedene Teilaufgaben zu gliedern. Dabei ergibt sich allerdings das Problem, dass eine Kind-Transaktion erfolgreich beendet worden ist, während eine weitere Kind-Transaktion abbricht. In diesem Fall muss die Eltern-Transaktion dafür sorgen, dass die Änderungen der bereits erfolgreich gelaufenen Kind-Transaktion rückgängig gemacht werden. Dies stellt streng genommen eine Verletzung der ACID-Eigenschaft Dauerhaftigkeit dar, da die Resultate einer erfolgreich beendeten Transaktion, hier der Kind-Transaktion, nicht dauerhaft im System verbleiben, sondern wieder zurückgenommen werden. Durch die erhöhte Komplexität erfordern verschachtelte Transaktionen meistens einen erhöhten administrativen Aufwand.

- Verteilte Transaktionen bieten die Möglichkeit, Teiltransaktionen verteilt auf mehreren Maschinen und / oder auch auf verschiedenen Systemen durchführen zu können. Im Normalfall ist es so, dass es bei der Ausführung einer verteilten Transaktion einen Koordinator und einen oder mehrere Teilnehmer gibt.

- Langlaufende Transaktionen (Long-Lived Transactions): Langlaufende Transaktionen sind Transaktionen, die, gemessen an anderen Transaktionen, überdurchschnitt-

lich lange andauern. Die verlängerte Dauer kann dabei verschiedene Gründe haben. So kann z.B. die Transaktion aus einer sehr großen Anzahl von Operationen bestehen (z.B. die Aktualisierung aller Artikel einer Lagerverwaltung) oder es sind Personen in die Abarbeitung einer Transaktion integriert. So kann es im Bereich des Dokumentenmanagements, z.B. bei der Entwicklung einer Entwurfszeichnung, zu einer Bearbeitung und damit zu Transaktionszeiten von Stunden, Tagen oder sogar Monaten kommen. Insbesondere im Web-Umfeld spielen langlaufende Transaktionen eine wichtige Rolle. Ein Beispiel hierfür sind Datenänderungen eines Benutzers in einem Onlineshop, evtl. über mehrere Web-Seiten hinweg und ggf. auch mit Bearbeitungspausen, die gesammelt und erst durch eine abschließende Bestätigung gespeichert werden. Im Web-Umfeld werden diese Transaktionen auch Web-Transaktionen genannt.

3.5 Bestehende nicht-standardkonforme Implementierungen von Transaktionsansätzen

Der WebDAV-Standard setzt sich aus einem Basisstandard und verschiedenen Erweiterungen zusammen (siehe Abschnitt 3.2.1). Der Basisstandard wird dabei von einer Vielzahl von Software-Produkten standardkonform unterstützt. Die jeweiligen Erweiterungen hingegen werden meistens nur teilweise und nicht vollständig von unterschiedlichen Softwareprodukten umgesetzt. In [Dus04] sowie unter [web09] existiert eine Übersicht, welche Produkte den WebDAV-Standard in welchem Umfang realisieren. Die beiden im Folgenden exemplarisch vorgestellten Produkte Microsoft Exchange Server 2007 sowie Subversion sind auf Basis dieser beiden Quellen sowie aufgrund einer Empfehlung (siehe Anlage B) von Geoffry Glemm, einem der Hauptautoren des DeltaV-Standards, in dieser Arbeit ausgewählt worden. Es sind die beiden Produkte im WebDAV-Umfeld, die in einem gewissen Umfang bereits transaktionale Eigenschaften über den WebDAV-Standard realisieren.

3.5.1 Microsoft Exchange Server 2007

Die Software Exchange Server 2007 der Firma Microsoft ist eine so genannte kollaborative Software oder auch Groupware genannt. Sie ermöglicht es einer Gruppe von Benutzern, über zeitliche und/oder räumliche Trennung hinweg zusammenzuarbeiten. Die Hauptfunktionen der Software sind dabei E-Mail-, Kalender- und Aufgabenfunktionen.

Exchange ermöglicht es, mit Hilfe von WebDAV auf eine Vielzahl von Applikationselementen zuzugreifen. Der Zugriff erfolgt dabei über das so genannte Microsoft Web Storage System, welches das Ziel verfolgt, einen einheitlichen Zugriff auf Ressourcen über unterschiedliche Protokolle, wie etwa WebDAV, IMAP oder POP3, zu ermöglichen. Exchange bietet dabei erheblich mehr Funktionalitäten als im WebDAV-Standard dokumentiert und festgehalten [Dus04]. Ein Großteil der Funktionen ist dabei im Mircosoft Developer Network, kurz MSDN, dokumentiert, welches als eine Art Online-Bibliothek für Artikel und Referenzen, insbesondere zur Softwareentwicklung, mit und für Produkte der Firma Microsoft [Mic09a] zur Verfügung steht. Von diesen Erweiterungen sind insbesondere zwei Ansätze für diese Arbeit relevant, die gewisse Transaktionseigenschaften aufweisen. Zum einen die so genannten Batch-Methods und zum anderen die so genannten Units of Work, die in den beiden folgenden Abschnitten detaillierter vorgestellt werden.

Batch-Methods

Ein sehr einfacher Ansatz, um gleiche Operationen auf mehreren Ressourcen zu gruppieren und mit einem Methodenaufruf durchzuführen, sind die von Microsoft als Batch-Methods bezeichneten Methoden. So ist es etwa mit der dazu neu eingeführten WebDAV-Methode BDELETE möglich, eine Menge von Ressourcen mit einem Methodenaufruf zu löschen. Standardkonform müsste für jede Ressource einzeln die DELETE-Methode ausgeführt werden. Neben dieser BDELETE-Methoden wurden noch vier weitere, analog umgesetzte, Methoden zum Kopieren (BCOPY), Verschieben (BMOVE), Properties lesen (BPROPFIND) und Properties setzen (BPROPPATCH) eingeführt [Mic09b].

Mit diesen neuen Methoden lässt sich die ACID-Eigenschaft der Atomarität teilweise erfüllen und es können gleiche Methoden zusammen gruppiert und dauerhaft gespeichert werden (ACID-Eigenschaft der Dauerhaftigkeit), jedoch keine verschiedenartigen, also beispielsweise eine Kopier- mit einer Verschiebeoperation.

Eine wichtige Funktion, die zur Umsetzung der vollen Atomaritätseigenschaft bei diesen neuen Methoden fehlt, ist, dass im Fehlerfall alle durchgeführten Aktionen zurückgenommen werden.

Die ACID-Eigenschaft der Isolation, sowie die damit verbundene Nebenläufigkeitskontrolle, wird nicht durch den Server unterstützt.

Units of Work

Microsoft bietet innerhalb von Exchange Server 2007 eine eigene Definition von Transaktionen über den WebDAV-Standard. Mit ihnen können so genannte Units of Work definiert werden. Diese ermöglichen es, die ACID-Eigenschaft der Atomarität umzusetzen und eine Menge von Dateioperationen zu kennzeichnen und gemeinsam zu verarbeiten. Dabei können die Dateioperationen abschließend abgeschlossen, also commited, oder durch den Benutzer zurückgenommen, also rollbacked, werden.

Zur Umsetzung wurde dazu die bestehende LOCK-Methode des WebDAV-Standards erweitert. Der Client führt dabei die LOCK-Methode mit der neu eingeführten Option "DAV:Lock type: transaction" aus, die den Start einer Transaktion kennzeichnet. Als Antwort auf die Lock-Methode erhält der Client einen Lock-Token, der seine Transaktion repräsentiert. Nun kann der Client mit Hilfe des Lock-Tokens, den er als Option in den Header jeder Operation einfügt, kennzeichnen, dass die jeweilige Operation innerhalb dieser Transaktion durchgeführt werden soll. Mit Hilfe der Methode UNLOCK und einer entsprechenden Option kann die Transaktion anschließend beendet oder abgebrochen werden. Die Dokumentation [Mic09c] sagt jedoch nichts darüber aus, ob die durchgeführten Operationen für andere Benutzer bereits während der Bearbeitung, also vor dem Abschluss der Transaktion, sichtbar sind. Ausdrücklich wird in der Dokumentation darauf hingewiesen, dass nur die ACID-Eigenschaften der Atomarität und der Dauerhaftigkeit durch die Transaktion unterstützt werden. Der Client ist selbst dafür verantwortlich, dass die Eigenschaft der Isolation und die damit verbundene Nebenläufigkeitskontrolle für konkurrierenden Zugriff eingehalten wird. Es werden auch von Serverseite keine direkten Konzepte zur Nebenläufigkeitskontrolle unterstützt.

Kombiniert man die im vorherigen Abschnitt vorgestellten Batch-Updates mit den Units of Work, können diese nun ebenfalls die Atomarität voll umfänglich unterstützen, d.h. die diesbezügliche Einschränkung der Batch-Updates wird aufgehoben.

Zusammenfassend lässt sich feststellen, dass der Vorteil dieser beiden vorgestellten An-

sätze darin besteht, dass insbesondere Kommunikations- und Bearbeitungsaufwand eingespart werden kann. Allerdings werden die ACID-Eigenschaften nicht vollständig unterstützt. So fehlen insbesondere Ansätze zur Unterstützung der Isolation und der damit verbundenen Nebenläufigkeitskontrolle. Zusammen mit der Einschränkung, dass nur gleiche Batch-Update-Methoden miteinander gruppiert werden können und der Tatsache, dass diese Methoden nicht standardkonform umgesetzt wurden, sind dies die Hauptargumente gegen diesen Transaktionsansatz.

3.5.2 Subversion

Subversion (oder kurz SVN) ist eine Open-Source-Software zur Versionskontrolle (siehe Abschnitt 3.2.2) von Daten, Verzeichnissen und den zugehörigen Metadaten. Die Software wird auch als Repository bezeichnet. Die Entwicklung von Subversion wurde Anfang 2000 gestartet und nach etwa 4 Jahren Entwicklungszeit konnte die erste offizielle und stabile Version 1.0 veröffentlicht werden [CSFP06].

Subversion ist ein so genanntes zentralisiertes Versionskontrollsystem, welches auf einer zentralen Architektur mit einem Server beruht. Dieser Server ist für die Verwaltung und Bereitstellung der Daten verantwortlich. Für den Zugriff auf diesen zentralen Server unterstützt Subversion verschiedene Netzwerkprotokolle (siehe Kapitel 3.1), unter anderem HTTP und WebDAV mit seinen Erweiterungen, insbesondere DeltaV. Da Subversion Transaktionen zum Verändern des zentralen Datenbestands verwendet, soll diese Umsetzung im Folgenden näher betrachtet werden.

Subversion arbeitet mit einem so genannten clientseitigen Workspace. Das bedeutet, dass sich der Benutzer die Daten aus dem zentralen Repository auf den Client kopiert (auscheckt). Auf diesen Kopien werden nun die gewünschten Änderungen des Benutzers vorgenommen. Hat er seine Änderungen durchgeführt und möchte seine Arbeit abschließen, führt er einen so genannten Commit aus, bei dem die lokalen Daten wieder auf den Server zurück kopiert (eingecheckt) und vorgenommene Änderungen protokolliert und festgehalten werden. Diese Aktualisierung wird von Subversion als Transaktion realisiert. Dabei werden die ACID-Eigenschaften voll umfänglich unterstützt. Diese Unterstützung gilt jedoch nicht allgemein für alle Funktionen, sondern nur speziell für die Aktualisierung.

Zur Umsetzung verwendet Subversion teilweise die in Abschnitt 3.2.2 vorgestellte Activity-Ressource. Jedoch ist auch hier die Umsetzung in einigen Teilen nicht standardkonform. So werden Methoden und Optionen verwendet, die im WebDAV-Standard nicht definiert

sind. Aus diesem Grund wird auf der offiziellen Projektseite [Sub09] davor gewarnt, Subversion als WebDAV- bzw. DeltaV–Client oder -Server zu verwenden. Einen Grund für die Diskrepanz zum DeltaV-Standard stellt die Tatsache dar, dass eine erste Version des DeltaV-Standards erst 2002 veröffentlich wurde. Zu dieser Zeit war jedoch schon die erste Beta-Version von Subversion im Einsatz (Ende 2001) [CSFP06]. Für die Zukunft ist jedoch geplant, eine möglichst standardkonforme Implementierung zu schaffen [Sub09].

In Kapitel 7 wird noch einmal näher auf Subversion eingegangen, da für den Prototypen einige Ansätze, insbesondere die transaktionale Umsetzung des Aktualisierungsvorgangs, zur Umsetzung des Transaktionsmodells verwendet werden konnten.

Subversion liefert bereits einige gute Ansätze, Probleme stellen aber die fehlende Standardkonformität und der Zwang, die Daten immer in den clientseitigen Workspace herunterladen zu müssen, dar. Außerdem ist die Beschränkung der transaktionalen Unterstützung auf den Vorgang der Aktualisierung ein Problem. Hier wäre die Erweiterung auf jedwede Art von WebDAV-Methoden wünschenswert.

4 Anforderungen an das Transaktionsmodell

In diesem Kapitel werden die Anforderungen an das Modell erarbeitet und festgelegt. Auf deren Basis wird dann in Kapitel 5 das eigentliche Transaktionsmodell entwickelt und vorgestellt.

Die Notwendigkeit einer detaillierten Anforderungsanalyse untermauert auch die folgende Aussage von Julian Reschke, einem der Mitautoren des WebDAV-Standards, welche in einer Diskussion auf der IETF-Mailingliste zum Thema "Umsetzung von Transaktionen mit Hilfe von WebDAV" fiel.

"..and please don't say: I want to do JDBC over HTTP ... " [Res03]

Dies unterstreicht noch einmal, dass man die Anforderungen und transaktionalen Konzepte aus dem Datenbankbereich (hier JDBC) nicht einfach eins zu eins auf HTTP bzw. WebDAV abbilden kann. Diese Aussage verdeutlicht auch die Notwendigkeit einer umfassenden Anforderungsanalyse, um sich einerseits von den Anforderungen aus dem Datenbankbereich abzugrenzen und andererseits zu untersuchen, in wieweit die bestehenden Transaktionskonzepte aus dem Datenbankbereich übertragen werden können.

Die Struktur dieser Anforderungsanalyse orientiert sich an dem von der IEEE (Institute of Electrical and Electronic Engineers) festgelegten Standard 830-1998 "Software Requirement Specification (SRS)". Die im Rahmen dieser Arbeit erstellte Anforderungsanalyse selbst teilt sich dabei nach einer kurzen Einleitung zur Festlegung und Abgrenzungen der Ziele inhaltlich in zwei Teile. Im ersten Teil(Abschnitt 4.1) werden die funktionalen Anforderungen diskutiert und festgelegt. Im zweiten Abschnitt 4.2 werden dementsprechend die nicht-funktionalen Anforderungen erarbeitet. Insbesondere die Diskussion der Anforderungen soll dazu beitragen, eine Abgrenzung zu den Anforderungen im Datenbankumfeld zu schaffen.

Die Anforderungen an das Transaktionsmodell ergeben sich zum größten Teil aus den in Kapitel 3.4 beschriebenen ACID-Eigenschaften. Weitere Anforderungen kommen aus dem Bereich des Dokumentenmanagements, insbesondere auf Basis der Software DataFinder (siehe Kapitel 2.2), sowie des Versionskontrollsystems Subversion (Abschnitt 3.5.2). Aber auch die bereits auf der IETF-Mailingliste geführten Diskussionen dienen als Quelle für die Anforderungsanalyse.

Dieses Kapitel beinhaltet alle Anforderungen, die zur Realisierung eines Transaktionsmodells für den WebDAV-Standard notwendig sind. Mit Hilfe des zu erstellenden Transaktionsmodells soll Anwendungsentwicklern, die eine Implementierung des Modells verwenden, ein vereinfachter und erweiterter Zugriff auf WebDAV-Systeme ermöglicht werden. Das Modell soll Methoden, Zustände und Konzepte definieren, um eine Menge von WebDAV-Dateioperationen zusammenzufassen, die innerhalb einer Transaktion verarbeiten werden und dabei die in Kapitel 3.4.1 beschriebenen ACID-Eigenschaften unterstützen.

Über Schnittstellen des Modells, die zugesicherte Eigenschaften erfüllen, soll der Entwickler transaktionsbasiert eine Menge von Dateioperationen nach dem "Alles oder Nichts"-Prinzip durchführen können. Das Modell soll dabei konkurrierende Zugriffe und Fehler, die während der Bearbeitung einer Transaktion (etwa durch Prozessfehler oder Computerausfälle) entstehen, erkennen und dem Entwickler entsprechende Abfragen, um darauf reagieren zu können, ermöglichen. Um diese Anforderungen umzusetzen, müssen die Transaktionsbefehle und Zustände des Modells, die bei der Verarbeitung entstehen können, auf den WebDAV-Standard mit seinen abstrakten Konzepten übertragen werden. Das Modell soll bis auf die Auswahl des Protokolls, hier sind HTTP und WebDAV mit seinen Erweiterungen im Rahmen der Aufgabenstellung fest vorgegeben, frei von festgelegten Implementierungen und technischen Voraussetzungen sein. So soll ein möglichst generisches und standardkonformes Modell entwickelt werden, mit dem Transaktionen abgebildet werden können, dass später sowohl auf Server- wie auch auf Client-Seite entsprechend implementiert und verwendet werden kann.

4.1 Funktionale Anforderungen an das Transaktionsmodell

In diesem Abschnitt werden die funktionalen Anforderungen diskutiert und festgelegt. Die konkreten Anforderungen, die sich aus den folgenden Betrachtungen ergeben, werden mit dem Kürzel "FA" und einer fortlaufenden Nummer beschrieben, um sie im weiteren Verlauf dieser Arbeit referenzieren zu können.

Anforderungen an die Transaktionsschnittstelle FA1 bis FA14

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, soll das Transaktionsmodell aufzeigen, wie mit Hilfe des WebDAV-Standards eine multifunktionale Schnittstelle für Transaktionen bereitgestellt werden kann. Im vorliegenden Abschnitt werden nun die Funktionen dieser Schnittstelle festgelegt und beschrieben. Die Schnittstelle des Modells muss, um einen kompletten Transaktionsablauf abzubilden, mindestens die folgenden elementaren Funktionen einer Transaktion auf den WebDAV-Standard abbilden können:

- Allgemeine Transaktionsbefehle:
  - BEGIN_TRANSACTION: Start (FA1)
  - END_TRANSACTION: Ende (FA2)
  - ABORT_TRANSACTION: Abbrechen und Zurücknahme aller Änderungen (FA3)
- Operationen auf Ressourcen bzw. Collections
  - READ_RESOURCE: lesen (FA4)
  - WRITE_RESOURCE: speichern (FA5)
  - CREATE_RESOURCE: anlegen (FA6)
  - MOVE_RESOURCE: verschieben (FA7)
  - COPY_RESOURCE: kopieren (FA8)
  - RENAME_RESOURCE: umbenennen (FA9)
  - DELETE_RESOURCE: löschen (FA10)
- Operationen auf Properties:
  - READ_PROPERTY: lesen (FA11)
  - CREATE_ PROPERTY: anlegen (FA12)
  - CHANGE_ PROPERTY: modifizieren (FA13)
  - DELETE_ PROPERTY: löschen (FA14)

ACID-Eigenschaften FA15 bis FA18

Nachdem in Kapitel 3.4.1 die ACID-Eigenschaften grundlegend vorgestellt wurden, sollen diese nun weiter spezifiziert und an die Bedürfnisse, die sich aus den Zielsetzungen dieser Arbeit ergeben, angepasst werden. Insbesondere wird dabei auf die Wechselwirkungen der einzelnen Eigenschaften miteinander und die daraus resultierenden Fehlerquellen eingegangen. Die Grenzen zwischen den einzelnen ACID-Eigenschaften sind dabei oft fließend, sodass sich in der Literatur verschiedene Zuordnungen von Anforderungen zu Eigenschaften finden lassen. Die vorliegende Arbeit orientiert sich bei dieser Zuordnung an [BN97], da es sich hierbei um ein weit verbreitetes und etabliertes Standardwerk handelt.

Atomarität

Die Atomarität zählt zu den wichtigsten Eigenschaften einer Transaktion. Entweder werden alle Teiloperationen erfolgreich durchgeführt oder aber keine. Aus dieser Grundforderung ergeben sich weitere Anforderungen, die im Folgenden vorgestellt werden.

Das Modell soll Mechanismen zur Sicherstellung der so genannten "failure atomicity" unterstützen (FA15). Dies bedeutet, dass auch beim Auftreten von Fehlern, beispielsweise durch einen Serverausfall, die Atomarität gesichert ist. In einem solchen Fehlerfall sollen die Transaktionsschritte aller nicht erfolgreich beendeten Transaktionen (zu einem Zeitpunkt können n Transaktionen aktiv sein) zurückgenommen werden (Rollback). Durch dieses Verhalten kann unter anderem die Eigenschaft der Konsistenz gesichert werden, damit so dass System von einem konsistenten Zustand in den nächsten überführt werden kann. Da in der Realität eine Vielzahl von Fehlern auftreten, werden im Folgenden die verschiedenen Fehler in sogenannten Fehlerklassen zusammengefasst und die nach [Sch03] wichtigsten vorgestellt.

- Transaktionsfehler: Fehler, bei denen die Transaktion durch den Server abgebrochen werden muss. Ursachen hierfür können beispielsweise Konflikte mit anderen Transaktionen sein.
- Systemfehler: Treten auf, wenn einzelne Operationen einer Transaktion nicht durchgeführt werden können, z.B. weil es durch einen Speicherüberlauf zum Verlust von Daten im Hauptspeicher kommt.

- Speicherfehler: Treten auf, wenn es zu Lesefehlern auf Sekundärspeichern, wie etwa der Festplatte, kommt.

Da das Modell und das Protokoll auf stark verteilten Systemen zum Einsatz kommen sollen, werden die bereits genannten Fehlerklassen noch um Kommunikationsfehler erweitert (nach [TvS03]).

- Kommunikationsfehler: Fehler oder Probleme bei der Kommunikation zwischen Client und Server. Diese Kommunikationsfehler werden von [Tan03VerSys] noch in die folgenden Unterklassen unterteilt:
  - Der Client findet den Server nicht.
  - Die Nachrichten von Client zu Server, bzw. umgekehrt, gehen verloren.
  - Der Server bzw. Client stürzt während der Bearbeitung ab.

Der Schwerpunkt des vorliegenden Modells soll auf der Vermeidung bzw., wenn dies nicht möglich ist, auf der Erkennung von Transaktions- (FA15.1) sowie Kommunikationsfehlern (FA15.2) liegen. System- und Speicherfehler sind zu stark mit einer konkreten Hard- und Softwarearchitektur verbunden und werden deshalb in dieser Arbeit nicht betrachtet.

Neben der gerade vorgestellten "failure atomicity" ist es nach [CDK05] eine sich aus der Eigenschaft der Atomarität ergebende direkte Konsequenz, dass nach Ablauf einer erfolgreichen Transaktion das Ergebnis dauerhaft gespeichert wird, also auch die ACID-Eigenschaft der Dauerhaftigkeit gegeben ist.

Konsistenz

Die Konsistenz bezeichnet allgemein die Widerspruchsfreiheit der Daten bzw. des Systems. Diese Widerspruchsfreiheit wird durch die so genannten Konsistenz-Policies festgelegt, die nur Konsistenz-erhaltende Transaktionen erlauben. So könnte eine einfache Policy z.B. festlegen, dass alle Datenelemente eine eindeutige ID besitzen müssen, um so Dubletten zu verhindern.

Für die Sicherstellung der Konsistenz sind jedoch, anders als bei Atomarität, Dauerhaftigkeit und Isolation, der Applikationsentwickler selbst und die jeweiligen technischen Prozesse der Implementierung des Transaktionsmodells verantwortlich. Durch seine Programmierung kann ein Entwickler zum einen beliebige Konsistenzbedingungen, also Konsistenz-Policies, festlegen und zum anderen fast immer auch eine Möglichkeit finden, gegen diese zu verstoßen.

Des Weiteren ist es unmöglich, absolut alle Konsistenzbedingungen, die sich aus verschiedensten Branchen und Anwendungsgebieten ergeben, allgemeingültig festzulegen. So hat ein Versionskontrollsystem z.B. völlig andere Konsistenzanforderungen als ein Dokumentenmanagementsystem. Beide sollen trotz dieser Unterschiede jedoch Transaktionen auf Basis des WebDAV-Standards nutzen können. Ziel dieses Modells soll es deshalb sein, dass der Entwickler sich auf die Eigenschaften Isolation, Atomarität und Dauerhaftigkeit verlassen kann und sich konzeptionell nur um die Konsistenz des Gesamtsystems selbst kümmern muss. Aus diesem Blickwinkel sieht es auch [BN97], der die Eigenschaft Konsistenz zu streng für Transaktionen hält. Deshalb ergeben sich aus der Eigenschaft Konsistenz keine neuen Anforderungen an das Modell.

Isolation

Das Modell soll sicherstellen, dass sich parallel ablaufende Transaktionen nicht gegenseitig beeinflussen oder behindern, also jeweils isoliert voneinander ablaufen. Um dies zu erreichen, soll das Modell geeignete Konzepte zur Nebenläufigkeitssteuerung unterstützen (FA16). Die Nebenläufigkeitssteuerung hat das Ziel, möglichst viele Transaktionen parallel ablaufen zu lassen und sicherzustellen, dass der Datenbestand seinen konsistenten Status beibehält. Dabei werden die Operationen der parallel ablaufenden Transaktionen so eingeplant, dass das Endergebnis so aussieht, als wären die einzelnen Transaktionen sequentiell abgelaufen [TvS03]. Zusätzlich dazu wird durch das Konzept der Serialisierbarkeit sichergestellt, dass die geplanten Ausführungen zu keinem Konflikt führen. Ein solcher Konflikt entsteht, wenn zwei Operationen auf die gleiche Ressource zugreifen und dabei mindestens eine der beiden Operationen eine Schreiboperation ist. Die verschiedenen Algorithmen zur Umsetzung der Nebenläufigkeitskontrolle unterscheiden sich insbesondere dadurch, wie Lese- und Schreiboperationen miteinander synchronisiert werden [Sch03]. Daraus ergeben sich die beiden folgenden Klassen von Algorithmen, die beide durch das in dieser Arbeit erstellte Modell unterstützt werden sollen:

- Optimistische Verfahren: Auch verifizierende Verfahren genannt. Bei diesen Verfahren wird erst zum Ende der Transaktion überprüft, ob die Operationen synchronisierbar und damit konfliktfrei waren (FA16.1).

- Pessimistische Verfahren: Auch präventive Verfahren genannt. Dort werden die Operationen synchronisiert, also Konflikte verhindert, bevor die Operationen ausgeführt werden (FA16.2).

Eine dritte Klasse von Verfahren stellen die so genannten Zeitstempelverfahren dar, die den Zugriff auf die Datenobjekte in der chronologischen Reihenfolge der jeweiligen Transaktion erlauben. Dies bedeutet, dass die Transaktion mit dem frühesten Zeitstempel den Vorrang hat. Nach [Sch03] haben letztere Verfahren in der Praxis jedoch so gut wie keine Bedeutung mehr. Zudem fallen sie von ihrer Arbeitsweise her nach [Sch03] in die Kategorie der pessimistischen Verfahren. Aus diesem Grund werden diese Verfahren im Folgenden nicht weiter betrachtet.

Da die Ergebnisse einer Transaktion erst nach erfolgreichem Abschluss aller Teiloperationen der Transaktion sichtbar sein dürfen, ergeben sich weiterhin eine Vielzahl von Bedingungen zum Schutz vor Fehlern, die durch die Nebenläufigkeitskontrolle des Systems sichergestellt werden müssen. Im Folgenden werden die vier Grundtypen von Fehlern (nach [Sch03]), die sich bei der parallelen Verarbeitung von Transaktionen ergeben, vorgestellt. Dabei sollen die beiden Grundtypen "Verlorengegangene Änderungen" (FA17) sowie "Nicht freigegebene Änderungen" (FA18), durch das Modell erkannt bzw. verhindert werden. Die beiden Fehler "Inkonsistente Analyse" und "Phantom-Problem" sind stark mit der Arbeitsweise relationaler Datenbanksysteme verbunden und können deshalb bei der Verwendung von WebDAV nicht auftreten. Sie werden nur der Vollständigkeit halber aufgeführt.

- Verlorengegangene Änderungen: Auch als Lost-Updates-Problem bekannt. Mehrere Transaktionen modifizieren parallel denselben Datensatz / dieselben Datensätze und nach Ablauf der Transaktionen sind nur die Änderungen einer Transaktion übernommen worden, da sich die Transaktionen gegenseitig ihre Modifikationen überschrieben haben. Abbildung 4.1 zeigt ein Beispiel für verlorengegangene Änderungen. Beide Transaktionen lesen die Variable Foo ein, jedoch werden nach Ablauf der Transaktion TA2 nur die Änderungen dieser Transaktion gespeichert. Die Änderungen von Transaktion TA1 sind verloren.

- Nicht freigegebene Änderungen: Auch als Dirty-Read-Problem bekannt. Es werden Daten einer noch nicht abgeschlossenen Transaktion (TA1) von einer anderen Transaktion (TA2) verarbeitet. Zu einem Fehlerfall kommt es, wenn die Änderungen der

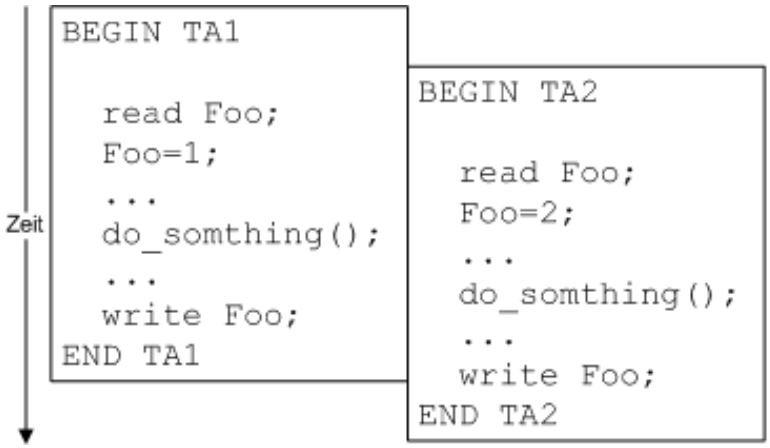

Abbildung 4.1: Beispiel für verlorengegangene Änderungen, angelehnt an [Sch03]

Ursprungstransaktion TA1 zurückgenommen werden. TA2 arbeitet nach der Rücknahme der Transaktion TA1 auf ungültigen Daten. In Abbildung 4.2 bearbeitet TA1 die Variable Foo und das System veröffentlicht bereits das Ergebnis der Schreiboperation, bevor die Transaktion abgeschlossen ist. TA2 liest und verarbeitet den durch TA1 aktualisierten Wert der Variablen Foo. Nun tritt in TA1 ein Fehler auf und die Änderungen der Transaktion werden zurückgenommen. TA2 arbeitet jetzt auf einem ungültigen Wert der Variablen.

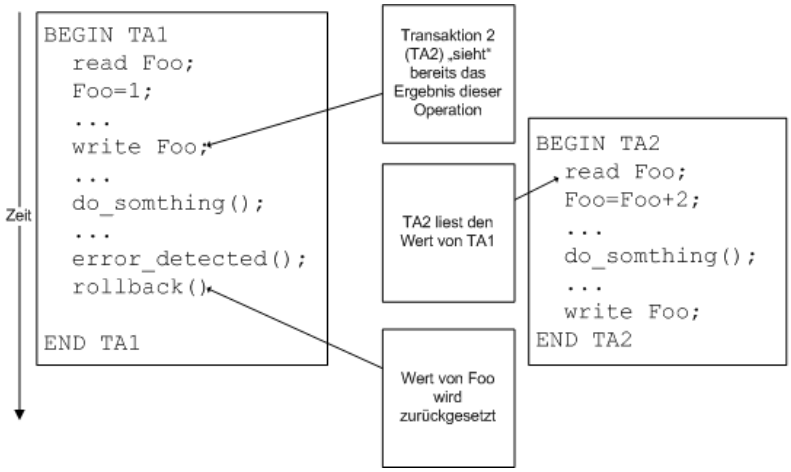

Abbildung 4.2: Beispiel für nicht freigegebene Änderungen, angelehnt an [BN97]

- Inkonsistente Analyse: Auch als Inconsistent-Analysis- oder Incorrect-Summary-Problem bekannt. Dieses Problem tritt auf, wenn eine Transaktion TA1 Berechnungen auf einer Teilmenge von Datensätzen durchführt, die während der Berechnung von

einer anderen Transaktion TA2 verändert werden. Da WebDAV keine Berechnungs- und / oder Teilmengenoperationen zur Verfügung stellt, ist diese Gruppe von Problemen für das Modell nicht relevant.

- Phantom Problem: Suchkriterien treffen während einer Transaktion auf unterschiedliche Datensätze zu, parallel dazu fügt jedoch eine andere Transaktion, die Suchkriterien betreffende, Datensätze hinzu, entfernt oder verändert sie. Dieses Problem tritt bei der Verwendung von WebDAV ebenfalls nicht auf, da das WebDAV-Protokoll keine Operationen auf Suchergebnismengen zur Verfügung stellt.

Dauerhaftigkeit

Nach Abschluss einer erfolgreichen Transaktion muss die Eigenschaft Dauerhaftigkeit dafür sorgen, dass die Ergebnisse persistent gespeichert werden, unabhängig von eventuellen Hard- oder Softwarefehlern, die nach Abschluss der Transaktion noch auftreten können. Diese Transaktionseigenschaft ist ebenfalls zu stark mit einer konkreten Hard- und Softwarearchitektur sowie Implementierung verbunden und wird deshalb nicht weiter betrachtet.

Eindeutige Zuordnung zwischen Client und Server

Da das WebDAV- und das HTTP-Protokoll verbindungs- und zustandslos arbeiten (siehe Kapitel 3.1) und somit keine feste und direkte Verbindung zwischen Client und Server besteht, muss die Möglichkeit geschaffen werden, transaktionsübergreifend eine eindeutige Zuordnung zwischen Client und Server (Session-Management) zu ermöglichen (FA19).

4.2 Nicht-funktionale Anforderungen an das Transaktionsmodell

Im folgenden Abschnitt werden die nicht-funktionalen Anforderungen diskutiert und festgelegt. Die Anforderungen legen fest, welche Eigenschaften und Qualitätsansprüche an das Transaktionsmodell gestellt werden. Sie werden mit dem Kürzel "NFA" und einer fortlaufenden Nummer beschrieben, um sie im weiteren Verlauf dieser Arbeit referenzieren zu können.

Standardkonformität NFA1

Das Modell soll möglichst konform mit den Standards der IETF (siehe 2.3) sein. Das Modell soll dabei auf dem HTTP-Protokoll (RFC 2616) und seinen entsprechenden Erweiterungen WebDAV (RFC 3253) und DeltaV (RFC 3253) basieren.

Kommunikationsaufwand NFA2

Der Kommunikationsaufwand in Bezug auf Anzahl und Größe der Nachrichten, die zwischen Client und Server ausgetauscht werden, soll möglichst gering gehalten werden, um u.a. Netzwerkressourcen zu schonen und auch Client-Rechnern mit geringer Bandbreite den Zugang zum System zu ermöglichen.

Effiziente Unterstützung langer Transaktionen NFA3

Im Bereich des Web-Authorings können Bearbeitungsprozesse lange andauern (siehe Abschnitt 3.4.2). Aus diesem Grund soll das Transaktionsmodell Transaktionen aus der Klasse der langen Transaktionen (siehe Kapitel 3.4.2) effizient unterstützen.

Die komplexeren Transaktionsklassen, wie z.B. verteilte oder verschachtelte Transaktionen, werden im Rahmen dieser Arbeit nicht betrachtet, da diese aus Architekturgründen nicht über das WebDAV-Protokoll abgebildet werden können.

5 Entwurf des Transaktionsmodells auf Basis des WebDAV-Standards

Bei der Entwicklung und Umsetzung des Transaktionsmodells wurde nach der Methodologie zur Entwicklung von Transaktionsmodellen nach [WV02] vorgegangen, da es sich um ein etabliertes und weit verbreitetes Verfahren handelt. Diese Methodologie setzt sich aus fünf Schritten zusammen:

1. Bestimmung der elementaren, also unteilbaren, Operationen des Systems.
2. Modellierung von Transaktionen, die sich aus den unteilbaren Operationen zusammensetzen und dabei die ACID-Eigenschaften erfüllen.
3. Bestimmung von Ablaufplänen (sog. schedules), damit eine Vielzahl von Transaktionen parallel ausgeführt werden kann.
4. Festlegen von korrekten Ablaufplänen, die die ACID-Eigenschaften nicht verletzen.
5. Implementierung des Transaktionsmodells.

In diesem Kapitel werden zunächst die Punkte 1 und 2 erarbeitet. Dazu werden im Abschnitt 5.1 die unteilbaren Operationen festgelegt. Anschließend wird im nächsten Abschnitt 5.2 ein Grundmodell entwickelt. Dieses Grundmodell bietet die wesentlichen Funktionen zur Kennzeichnung von Start, Ende und Abbruch einer Transaktion sowie die Funktionen zum Lesen und Erzeugen von Ressourcen. Ausgehend von diesem Grundmodell werden passende Modelle zur Umsetzung einer optimistischen (Abschnitt 5.3) und einer pessimistischen Nebenläufigkeitskontrolle (Abschnitt 5.4) vorgestellt.

Es wird weiterhin aufgezeigt, wie bei einer Implementierung des Modells die Punkte 3 und 4 sichergestellt werden können. Diese beiden Punkte sind natürlich eng miteinander verknüpft, die Trennung wirkt auf den ersten Blick etwas künstlich, in der Methodologie wird aber explizit diese Aufteilung empfohlen. Dazu werden in den beiden Ausprägungen

des Grundmodells jeweils transaktionssichere Funktion, zur Umsetzung der in den Anforderungen festgelegten Operationen vorgestellt. Des Weiteren wird aufgezeigt, wie Konflikte erkannt und/oder verhindert werden können. Der fünfte Punkt, die Implementierung, wird in Kapitel 7 vorgestellt.

Ein wichtiges Ziel bei der Entwicklung des Modells ist es, die Transaktionen, im Gegensatz zu vorhandenen Modellen und ihren jeweiligen Implementierungen (siehe Abschnitt 3.5), möglichst standardkonform abzubilden. Neben der Literatur zum Thema Transaktionen wurden deshalb auch die verschiedenen RFCs des HTTP- und WebDAV-Standards, sowie bestehende, nicht-standardkonforme Implementierungen (siehe Kap. 3.5) intensiv analysiert, um herauszufinden, ob eine standardkonforme Abbildung überhaupt möglich ist.

Mit dem DeltaV-Standard wurde eine WebDAV-Erweiterung gefunden, die genügend Konzepte vorweist, um sie als Basis für ein größtenteils standardkonformes Transaktionsmodell verwenden zu können. In Abschnitt 5.5 wird auf die Beurteilung des Modells durch die IETF bezüglich seiner Standardkonformität eingegangen. Es wird gezeigt, welche Sonderfälle durch den DeltaV-Standard noch nicht abgedeckt werden. Da sich WebDAV, sowie dessen Erweiterungen, noch im Standardisierungsprozess befinden (siehe Abschnitt 2.3), besteht hier jedoch die Möglichkeit der Einflussnahme. In Kapitel 6 werden deshalb Vorschläge zur Erweiterung des bestehenden DeltaV-Standards gemacht, sowie die sich daraus ergebenden neuen Möglichkeiten vorgestellt, mit denen alle Anforderungen umgesetzt werden können.

Zum Abschluss dieses Kapitels werden das Grundmodell sowie die beiden Nebenläufigkeitskonzepte bewertet. Diese Bewertung soll die Entscheidungsgrundlage für eine prototypische Implementierung des Transaktionsmodells bilden, die in Kapitel 7 vorgestellt wird.

5.1 Bestimmung der unteilbaren Operationen des Transaktionsmodells

Im Folgenden werden zunächst die unteilbaren HTTP- bzw. WebDAV-Methoden aufgeführt. Diese werden innerhalb einer Transaktion dazu genutzt, höherwertige und transaktionssichere Funktionen zu bilden. Die Bezeichnungen unteilbar bzw. atomar sind dabei nur als logisch unteilbar bzw. atomar zu verstehen, d.h. die HTTP- bzw. WebDAV-Methoden

stellen sich für den Aufrufer der Methoden als unteilbar bzw. atomar dar, sie können aber physisch durchaus aus mehreren Operationen bestehen [GR93]. Die technische Implementierung in den darunterliegenden Schichten des TCP/IP-Schichtenmodells bleibt von der Anforderung der Unteilbarkeit unberührt. Im Folgenden werden die für dieses Modell als logisch unteilbar betrachteten Operationen aufgelistet:

- HTTP-Methoden
  - PUT
  - GET
  - DELETE
  - OPTIONS
- WebDAV-Methoden
  - LOCK
  - UNLOCK
  - MKCOL
  - COPY
  - MOVE
  - PROPFIND
  - PROPPATCH
- DeltaV-Methoden
  - CHECKOUT
  - CHECKIN
  - UNCHECKOUT
  - MKACTIVITY
  - VERSON-CONTROL

Eine detaillierte Beschreibung der oben aufgeführten Methoden befindet sich in Kapitel 3.

5.2 Grundmodell

Das in diesem Abschnitt vorgestellte Grundmodell bildet den Rahmen einer Transaktion ab. Es werden dazu die in der Anforderungsanalyse (Kapitel 4) festgelegten grundlegen-

den Transaktionsbefehle mit Hilfe des WebDAV-Standards umgesetzt. Diese Transaktionsbefehle beinhalten die Kennzeichnung von Start (Begin), Ende (Commit) sowie Abbruch (Abort) einer Transaktion. Des Weiteren wird das Grundmodell dazu genutzt, die Gemeinsamkeiten der beiden Modellausprägungen zur Umsetzung der pessimistischen bzw. der optimistischen Nebenläufigkeitskontrolle darzustellen. Die Unterschiede und Eigenschaften dieser beiden Varianten werden dann in jeweils separaten Abschnitten behandelt (siehe Abschnitt 5.3 und 5.4). Eine weitere Zielsetzung des Grundmodells ist es, ein Konstrukt einzuführen, welches der Zustandslosigkeit des HTTP-Protokolls dahin gehend begegnet, dass mit ihrer Hilfe eine Benutzersitzung mit ihren transaktionsspezifischen Informationen über eine Anfrage hinaus erhalten bleibt.

5.2.1 Start, Ende und Abbruch einer Transaktion auf den WebDAV-Standard abbilden

Mit Hilfe einer Activity-Ressource wird auf dem Server eine Aufgabe gekennzeichnet (siehe Abschnitt 3.2.2). In diesem Transaktions-Modell werden Transaktionen als spezielle Aufgaben betrachtet. Die Activity-Ressource bildet die Grundlage zur Kennzeichnung von Start, Ende und Abbruch einer Transaktionen, welches im Folgenden spezifiziert und vorgestellt wird.

Start einer Transaktion kennzeichnen - TXN_Start

Der Start einer Transaktion, welches im Folgenden als Funktion TXN_Start bezeichnet wird, geschieht implizit mit dem Anlegen einer Activity durch die MKACTIVITY-Methode. Der Client ist dabei für das Festlegen des Speicherortes der Activity, also der URL, auf dem Server verantwortlich. Der Client weiß jedoch nicht, in welchem Bereich des Servers Activity-Ressourcen angelegt werden dürfen. Deshalb muss der Client mit Hilfe der OPTIONS-Methode (siehe Kap. 3.1) diesen Speicherort zuvor in Erfahrung bringen. Dazu gibt er der OPTIONS-Methode die Option "D:activity-collection-set" mit, die den Server anweist, in seiner Antwort den Speicherbereich für Activity-Ressourcen zurückzugeben. Listing 5.1 zeigt eine entsprechende Anfrage, in der in Zeile 8 explizit die Information über den Speicherort der Activity-Ressource angefordert wird.

```
1 OPTIONS / HTTP/1.1
2 Host: dlr.de
```

```
3 Content-Type: text/xml
4 charset="utf-8"
5
6 <?xml version="1.0" encoding="utf-8"?>
7 <D:options xmlns:D="DAV:">
8   <D:activity-collection-set/>
9 </D:options>
```

Listing 5.1: Ausführung der OPTIONS-Methode

Listing 5.2 zeigt die entsprechende Serverantwort. Sie enthält zunächst in den Zeilen 6 bis 12 die für die Options-Methode typischen Informationen zur Kennzeichnung der unterstützten Methoden und Features des Servers. Zusätzlich dazu enthält die Antwort aber auch in Zeile 17 die Information zum Speicherort von Activity-Ressourcen.

```
1 HTTP/1.1 200 OK
2 Date: Tue, 15 Apr 2008 13:19:52 GMT
3 Content-Length: 185
4 Content-Type: text/xml; charset="utf-8"
5 Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4 mod_jk2/2.0.4 PHP↩
      /4.3.10-16 mod_perl/1.999.21 Perl/v5.8.4
6 DAV: 1
7 DAV: version-control,checkout,working-resource
8 DAV: merge,baseline,activity,version-controlled-collection
9 DAV: <http://apache.org/dav/propset/fs/1>
10 MS-Author-Via: DAV
11 Allow: OPTIONS,GET,HEAD,POST,DELETE,TRACE,PROPFIND,PROPPATCH,COPY,MOVE,CHECKOUT
12 Via: 1.1 pdedlrbirli2 (NetCache NetApp/6.1.1)
13
14 <?xml version="1.0" encoding="utf-8"?>
15 <D:options-response xmlns:D="DAV:">
16  <D:activity-collection-set>
17    <D:href>/act</D:href>
18  </D:activity-collection-set>
19 </D:options-response>
```

Listing 5.2: Serverantwort auf die OPTIONS-Anfrage

Der Client hat nun vom Server erfahren, in welchem Bereich er Activity-Ressourcen anlegen darf, in dem Beispiel aus Listing 5.2 also unter `dlr.de/act`. Der Client muss nun dafür sorgen, dass seine Activity und damit seine Transaktion in diesem Bereich eindeutig gekennzeichnet ist.

Leider ist im DeltaV-Standard nicht vorgesehen, dass der Server die komplette Activity-URL, also Speicherort plus Namen, erzeugt, um so evtl. auftretende doppelte Benennungen von Activities von vornherein zu verhindern. Eine mögliche Implementierung, um die doppelte Benennungen von Activity-Ressourcen möglichst auszuschließen, wäre, dass sich der Client für jede Transaktion einen Universally Unique Identifier, kurz UUID, erzeugt. Die UUID stellt dabei ein Standardverfahren dar, um mit Hilfe von festgelegten Algorithmen 16 Byte lange eindeutige Identifikatoren zu erstellen[1]. Der Vorteil dieses Verfahrens ist unter anderem der, dass diese Identifikatoren ohne eine zentrale Kontrollinstanz erzeugt werden können. In der Praxis werden sie oft in verteilten Systemen, wie etwa beim Konzept des Entfernten Prozeduraufrufes (Remote Procedure Call, kurz RPC) verwendet. Das UUID-Verfahren ist durch die IETF als Proposed – Standard im RFC 4122 festgelegt[2], in dem auch die zur Erzeugung notwendigen Algorithmen aufgeführt werden. Listing 5.3 zeigt noch einmal zusammenfassend, wie der Start einer Transaktion implizit durch das Anlegen einer Activity-Ressource mit eindeutiger UUID durch den Client mit Hilfe der MKACTIVITY-Methode ausgeführt wird.

```
1 MKACTIVITY /act/5d585fab-05b3-1d41-95fe-f76556624ab2 HTTP/1.1
2 Host: dlr.de
3 Content-Length: 0
```

Listing 5.3: Anlegen einer Activity-Resource mit Hilfe einer UUID

Die Activity-Ressource wird in diesem Modell auch dazu genutzt, die Zustandslosigkeit des HTTP–Protokolls zu überwinden, weil nun Client und Server ihre jeweiligen Transaktionen mit Hilfe der erzeugten eindeutigen Activity über eine einzelne Anfrage hinweg erhalten können.

Abbildung 5.1 zeigt in einem UML 2.0 Sequenzdiagramm noch einmal den vollständigen Initialisierungsprozess, der zwischen Client und Server ablaufen muss, um eine Transaktion zu starten.

[1] Die Kollisionswahrscheinlichkeit geht wegen der 2^{128} möglichen Schlüssel gegen Null.
[2] http://tools.ietf.org/html/rfc4122

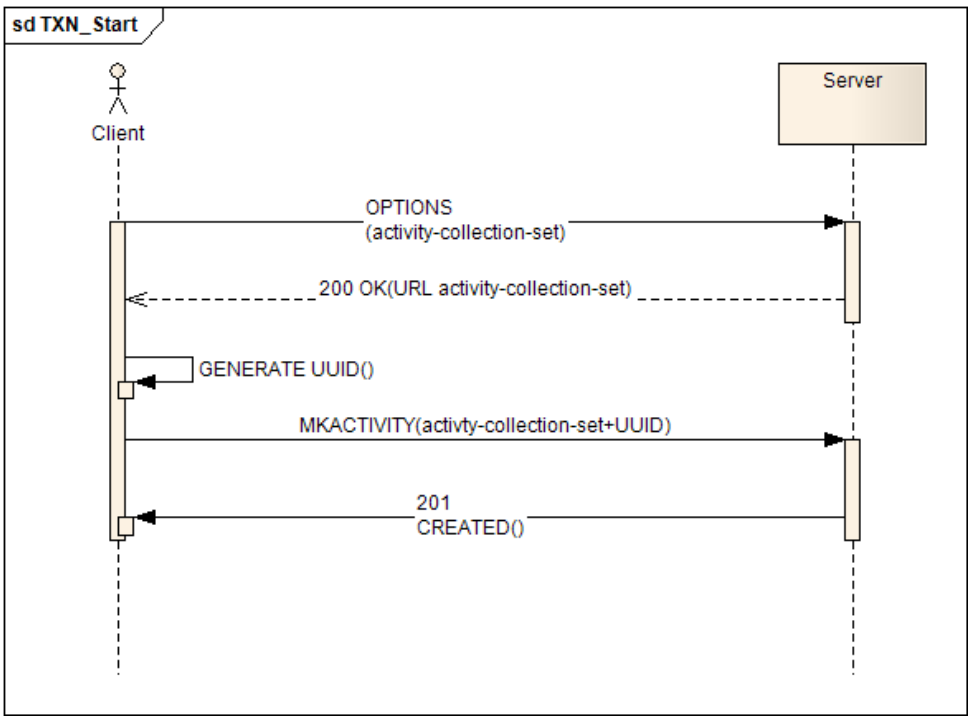

Abbildung 5.1: Befehlsfolge zum Start einer Transaktion

Ende einer Transaktion kennzeichnen – TXN_END

Das Ende einer Transaktion (im Folgenden als TXN_END bezeichnet), kennzeichnet der Client dadurch, dass er die Methode CHECKIN mit der entsprechenden URL der Activity-Ressource, die seine Transaktion repräsentiert, aufruft. Über den Statuscode (siehe Kapitel 3.1), den der Server als Antwort auf die CHECKIN-Methode des Clients zurückliefert, erfährt der Client, ob die Transaktion erfolgreich verarbeitet wurde. Zudem hat der Client mit Hilfe der PROPFIND-Methode die Möglichkeit, die beiden Live-Properties "DAV:activity-checkout-set" [3] sowie "DAV:activity-version-set" [4] der Activity-Ressource abzufragen.

Nach erfolgreicher Verarbeitung der Transaktion kann der Client mit Hilfe der DELETE-Methode die Activity-Ressource vom Server löschen, oder sie für spätere Transaktionen wiederverwenden. Welche dieser beiden Möglichkeiten gewählt wird, ist einer konkreten Implementierung freigestellt.

[3] In diesem Attribut stehen die URLs der VCRs, die der Activity zugewiesen wurden und sich im Zustand Checked-out befinden.

[4] In diesem Attribut stehen alle bereits erfolgreich eingecheckten VCRs mit ihrer neuen VR–URL.

Abbruch einer Transaktion kennzeichnen

Eine Transaktion kann zum einen von der Client- und zum anderen auch von der Server-Seite aus abgebrochen werden. Der Client kann eine Transaktion mit Hilfe der Methode UNCHECKOUT und Angabe der URL der Activity-Ressource, die seine entsprechende Transaktion repräsentiert, abbrechen. Listing 5.4 zeigt den Aufruf der UNCHECKOUT-Methode für den Abbruch der in Listing 5.3 angelegten Activity-Ressource.

```
1 UNCHECKOUT /act/5d585fab-05b3-1d41-95fe-f76556624ab2 HTTP/1.1
2 Host: dlr.de
3 Content-Length: 0
```

Listing 5.4: Abbruch einer Transaktion

Der Server muss nun alle Änderungen, die im Rahmen der Transaktion durchgeführt wurden, zurücknehmen. Dazu ruft er für alle VCRs, deren URLs sich im Live-Property "DAV:activity-checkout-set" befinden, die Methode UNCHECKOUT auf.

Der Server kann den Abbruch einer Transaktion, je nach Bedarf, zu einem beliebigen Zeitpunkt durchführen. Der Server kann den Client jedoch über den Abbruch erst informieren, wenn der Client wieder Kontakt mit ihm aufnimmt. Diese Einschränkung entsteht dadurch, dass durch die Verwendung des Request–Response-Modells die Kommunikation nur durch den Client initiiert werden kann. Der Server kann über entsprechende Statuscodes und Meldungen, mit denen er auf den Request des Clients antwortet, signalisieren, dass die Transaktion abgebrochen wurde.

5.2.2 Erkennung von Netzwerkfehlern auf Basis des WebDAV-Standards

Das HTTP-Protokoll sowie alle entsprechenden WebDAV-Erweiterungen, sind auf Schicht 4 (Anwendungsschicht), des TCP/IP-Schichtenmodells angeordnet (siehe auch Abschnitt 3.1). Es werden in dieser Schicht die Dienste der darunterliegenden Transportschicht verwendet. Diese stellt die beiden Protokolle TCP und IP zur Verfügung. HTTP nutzt dabei das TCP-Protokoll, welches sicherstellt, dass sich Client und Server auf eine abgesicherte Kommunikation verlassen können. So müssen sich Client und Server nicht um doppelte, verlorene oder verzögerte Nachrichten kümmern [Tan00]. Beim Auftreten von Fehlern benachrichtigt das TCP-Protokoll die darüberliegende Schicht, so dass diese auf den Fehler reagieren kann.

5.2.3 Ressourcen lesen

In diesem Modell wird zwischen zwei unterschiedlichen Leseoperationen unterschieden. Die einen, in diesem Modell als READ_N-Operationen bezeichnet, sind einfache Leseoperationen, die nicht Basis einer späteren Veränderung des Datenbestandes durch eine nachfolgende Schreiboperation sind. Bedingt durch diese Eigenschaft müssen solche Operationen nicht durch eine Transaktion abgesichert werden, es kann hier also Verwaltungsaufwand reduziert werden.

Alle anderen Leseoperationen, die hier als READ_TA-Operationen bezeichnet werden, haben indirekten Einfluss auf den Datenbestand, da sie Basis einer späteren Veränderung des Datenbestandes durch eine nachfolgende Schreiboperation sind. Ein Beispiel für eine solche READ_TA-Operation ist etwa das Einlesen der Quelldaten bei einem Kopiervorgang. Um u. a. sicherzustellen, dass diese Daten beim Beenden der Transaktion noch gültig, also die Quelldaten nicht bereits schon durch eine andere Transaktion verändert wurden, muss diese Leseoperation transaktionssicher durchgeführt werden.

Im Folgenden wird nun detailliert auf die READ_N-Operation eingegangen. Die Durchführung transaktionssicherer Leseoperationen mit Hilfe der READ_TA-Operationen wird jeweils in den beiden Modellausprägungen vorgestellt.

Soll auf Basis der gelesenen Daten keine Änderung am Datenbestand vorgenommen werden, können diese Leseoperationen direkt und ohne besondere Vorbedingungen mit den dafür vorgesehenen WebDAV-Methoden durchgeführt werden, da diese Leseoperationen, die Datenintegrität des Datenbestandes nicht verletzen können. Im Folgenden werden die relevanten WebDAV-Methoden zum Lesen noch einmal genauer betrachtet:

- Dateiinhalt lesen: Um den Dateiinhalt einer Ressource zu lesen, kann, wie bereits im Grundlagenkapitel beschrieben, mit Hilfe der GET-Methode auf die entsprechende URL der VCR zugegriffen werden. In der Antwort des Servers wird dann der gesamte Dateiinhalt der aktuellsten VR zurückgeliefert, abhängig davon, welche VR im Attribut CHECKED-IN bzw. CHECKED-OUT der VCR angegeben ist. Nachdem die Übertragung auf den Client abgeschlossen ist, kann der Client den Dateiinhalt mit einem entsprechenden Programm auslesen.

- Properties lesen: Mit Hilfe der PROPFIND-Methode können die entsprechenden Dead- bzw. Live-Properties einer Ressource ausgelesen werden. Der PROPFIND-

Methode wird die URL der entsprechenden VCR übergeben und als Antwort erhält der Client die in seiner Anfrage angeforderten Attribute der aktuellsten VR zurück.

5.2.4 Ressourcen erzeugen – TXN_RESOURCE_CREATE

Wie bereits in Kapitel 3.1 vorgestellt, werden mit Hilfe der Methoden PUT und VERSION-CONTROL neue Ressourcen dem System hinzugefügt. Im bestehenden DeltaV-Standard fehlt der Methode VERSION-CONTROL jedoch zur Zeit die Funktionalität zur Zurücknahme dieser Aktion. So kann die Teilanforderung der Atomarität, die besagt, dass im Fehlerfall alle Änderungen einer Transaktion zurück zu nehmen sind, nicht direkt umgesetzt werden. Im Folgenden wird nun dargelegt, wie es dennoch möglich ist, Ressourcen transaktionssicher anzulegen. Zunächst wird mit Hilfe der PUT-Methode eine leere Ressource auf dem Server angelegt[5]. Dazu wird die PUT-Methode mit Angabe der Ziel-URL, jedoch ohne Dateiinhalt, aufgerufen. Anschließend wird diese Ressource der Versionskontrolle mit Hilfe der Methode VERSION-CONTROL unterworfen. Somit steht auf dem Server nun eine leere Ressource ohne Dateiinhalt und Properties zur Verfügung.

Nachdem die Ressource nun der Versionskontrolle untersteht, können alle Änderungen an der Ressource, wie in den folgenden Abschnitten beschrieben, transaktionssicher durchgeführt werden.

Kapitel 6 geht darauf ein, wie es mit Hilfe minimaler Modifikationen des momentanen Standards möglich ist, diesen Mehraufwand zu vermeiden und trotzdem eine vollständige Unterstützung der Atomarität zu erreichen.

5.3 Abbildung der optimistischen Nebenläufigkeitskontrolle auf den WebDAV-Standard

Nachdem in den vorangegangenen Abschnitten die Grundlagen zur Kennzeichnung von Start, Ende und Abbruch einer Transaktion definiert wurden, wird nun die Arbeitsweise des Modells vorgestellt, wenn mit Hilfe einer optimistischen Nebenläufigkeitskontrolle transaktionssichere Operationen auf dem Datenbestand ausgeführt werden.

Die Verfahren zur optimistischen Nebenläufigkeitskontrolle arbeiten grundsätzlich nach dem gleichen Prinzip. Nach dem Start einer Transaktion werden Kopien der zu verändernden Datensätze für diese Transaktion erzeugt. Auf diesen Kopien werden dann die

[5] Ähnlich zu dem POSIX-Programm touch im Unix-Umfeld

gewünschten Änderungen durchgeführt. Erst nach dem Abschluss der Transaktion werden sie in den originalen Datenbestand übernommen [Sch03]. Nach Kung und Robinson, die 1981 in [KR81] die optimistische Nebenläufigkeitskontrolle vorgestellt haben, durchläuft das Verfahren beim Verändern des Datenbestandes drei Phasen. Diese besitzen nach [Sch03] bis heute Gültigkeit.

1. Ausführungsphase (read phase): Nach dem Start einer Transaktion werden die zu verändernden Datenobjekte aus dem Datenbestand herausgelesen und in einen privaten Arbeitsbereich, den so genannten Workspace kopiert. In diesem Arbeitsbereich werden die Änderungen an den kopierten Datenobjekten vorgenommen. In der Literatur findet sich auch die Bezeichnung Lesephase für diese erste Verarbeitungsstufe. Dieser Name ist jedoch irreführend, da in dieser Phase auch explizit geschrieben wird. Aus diesem Grund wird im Verlauf dieser Arbeit die Bezeichung Ausführungsphase nach [Dad96] verwendet.

2. Validierungsphase (validation phase): Diese Phase beginnt, wenn der Benutzer die Transaktion abschließt. Der Server muss hier prüfen, ob die gewünschten Änderungen des Datenbestands innerhalb der Transaktion serialisierbar sind, also keinen Konflikt mit anderen nebenläufigen Transaktionen auslösen.

3. Schreibphase (write phase): Nach Abschluss einer erfolgreichen Validierungsphase werden die Änderungen an den Kopien in den ursprünglichen Datenbestand übernommen. Ist auch diese Phase erfolgreich, gilt die gesamte Transaktion als erfolgreich abgeschlossen.

In den folgenden Abschnitten wird nun aufgezeigt, wie diese drei Phasen mit Hilfe des Transaktionsmodells auf den WebDAV-Standard abgebildet werden können. Die Umsetzung der Ausführungsphase wird in Abschnitt 5.3.1 vorgestellt, dort werden auf Basis der festgelegten atomaren Operationen höherwertige und transaktionssichere Funktionen definiert. Diese Funktionen ermöglichen das Kopieren und Bearbeiten in einem privaten Arbeitbereich. In Abschnitt 5.3.2 und 5.3.3 wird die Umsetzung der Validierungs- bzw. der Schreibphase vorgestellt und aufgezeigt, wie Fehler und Konflikte im Modell erkannt werden können.

5.3.1 Erstellung transaktionssicherer Funktionen auf Basis des WebDAV-Standards - Ausführungsphase

In diesem Abschnitt wird festgelegt, wie das Transaktionsmodell die Phase eins der optimistischen Nebenläufigkeitskontrolle, die Ausführungsphase, umsetzt. Dazu wird auf Basis des WebDAV-Standards aufgezeigt, wie die darin definierten Konzepte und Methoden genutzt werden können, um transaktionssichere Funktionen zu erstellen. Die im Folgenden definierten Funktionen können nach dem Start einer Transaktion mit Hilfe der Funktion TXN_Start (siehe Abschnitt 5.2.1) zum transaktionssicheren Arbeiten mit dem Datenbestand verwendet werden. Zur Umsetzung werden dazu die in Kapitel 3.2.2 vorgestellten DeltaV-Konzepte Activity- bzw. Working-Ressource miteinander kombiniert. Die Working-Ressourcen stellen Kopien der zu bearbeitenden Ressourcen dar. Die Activity-Ressourcen werden dazu verwendet, diese Kopien der jeweiligen Transaktion zuzuordnen. In den nächsten Abschnitten wird dieses Zusammenspiel detaillierter betrachtet.

Hier sei erwähnt, dass beim Entwurf des Modells die Kopien der Datenobjekte (also der VCRs) bewusst auf Serverseite abgelegt wurden. Somit basiert das in dieser Arbeit beschriebene Transaktionsmodell auf einem so genannten serverseitigen Workspace-Modell. Eine andere Möglichkeit zur Ablage der Kopien stellt das clientseitige Workspace-Modell dar. Bei diesem Modell werden die Kopien der entsprechenden Datenobjekte auf Client-Seite abgelegt und bearbeitet. Das serverseitige Workspace-Modell bietet insbesondere bei strukturellen Änderungen, z.B. beim Verschieben von Ordnerstrukturen, Vorteile, da hier nicht, wie beim clientseitigen Modell, die Daten zunächst auf den Client kopiert werden müssen. Das in Abschnitt 3.5.2 vorgestellte Programm Subversion verwendet beispielsweise ein solches clientseitiges Workspace-Modell.

Erstellung der Kopien und Zuweisung zur Transaktion – TXN_OPT_CHECKOUT

Im Folgenden wird nun vorgestellt, wie der private Arbeitsbereich des Benutzers mit den entsprechenden Arbeitskopien zur Umsetzung der Ausführungsphase auf den WebDAV-Standard abgebildet wird.

Mit Hilfe der CHECKOUT-Methode werden dazu die DeltaV-Konzepte Activity- bzw. Working-Ressource miteinander kombiniert. Eine Kopie einer originalen VCR wird mit Hilfe der Option "D:apply-to-version" erstellt (siehe Kapitel 3.2.2), die der Methode CHECKOUT übergeben wird. Diese Option erzeugt eine so genannte Working-Ressource, al-

so eine Kopie der originalen VCR, in einem (server-seitigen) privaten Arbeitsbereich. Mit Hilfe der Option "D:activity-set" der CHECKOUT-Methode wird diese Kopie einer entsprechenden Transaktion, also einer Activity, zugeordnet. Dieses Vorgehen wird im Folgenden mit der Funktion TXN_OPT_CHECKOUT zusammengefasst. Listing 5.5 zeigt beispielhaft die Durchführung eines TXN_OPT_CHECKOUTs, bei dem eine Kopie angelegt und einer Activity zugeordnet wird. Die erste Zeile gibt an, dass die VCR `TestFooDokument` ausgecheckt wird. In Zeile 7 wird mit Hilfe der Option "D:apply-to-version" eine entsprechende Kopie der originalen VCR, also eine Working-Ressource, angelegt. In den Zeilen 8 bis 10 wird durch die Option "DAV:activity-set" diese Kopie der Activity `http://dlr.de/act/5d585fab-05b3-1d41-95fe-f76556624ab22` und damit der Transaktion aus Listing 5.3 zugeordnet.

```
1  CHECKOUT TestFooDokument HTTP/1.1
2  Host: dlr.de
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: 1234
5
6  <?xml version="1.0" encoding="utf-8" ?>
7  <D:checkout xmlns:D="DAV:">
8     <D:apply-to-version/>
9     <D:activity-set>
10       <D:href>http://dlr.de/act/5d585fab-05b3-1d41-95fe-f76556624ab2</D:href>
11    </D:activity-set>
12 </D:checkout>
```

Listing 5.5: Checkout einer Ressource und Hinzufügen zu einer Activity

Die Antwort des Servers auf diesen Checkout beinhaltet nun die URL der Working-Ressource, die zum Ändern verwendet werden kann. Listing 5.6 zeigt eine Antwort auf die in Listing 5.5 dargestellte Anfrage und speziell in Zeile 2 die entsprechende URL der Working-Ressource.

```
1  HTTP/1.1 201 Created
2  Location: http://dlr.de/WR/TestFooDokument
```

Listing 5.6: Serverantwort auf den Checkout einer Ressource

Abbildung 5.2 zeigt zusammenfassend noch einmal den Zustand, der nach der Anwendung der Funktion TXN_OPT_CHECKOUT auf die beiden VCRs `TestFooDokument` und `TestBarDokument` vorliegt. So wurden zwei entsprechende Working-Ressourcen angelegt und der Activity `actUserA` zugewiesen. Die URLs der Working-Ressourcen wurden zudem dem Attribut "DAV:activity-checkout-set" der Activity hinzugefügt. Das Attribut "DAV:auto-update" der beiden Working-Ressourcen zeigt auf die jeweilige originale VCR.

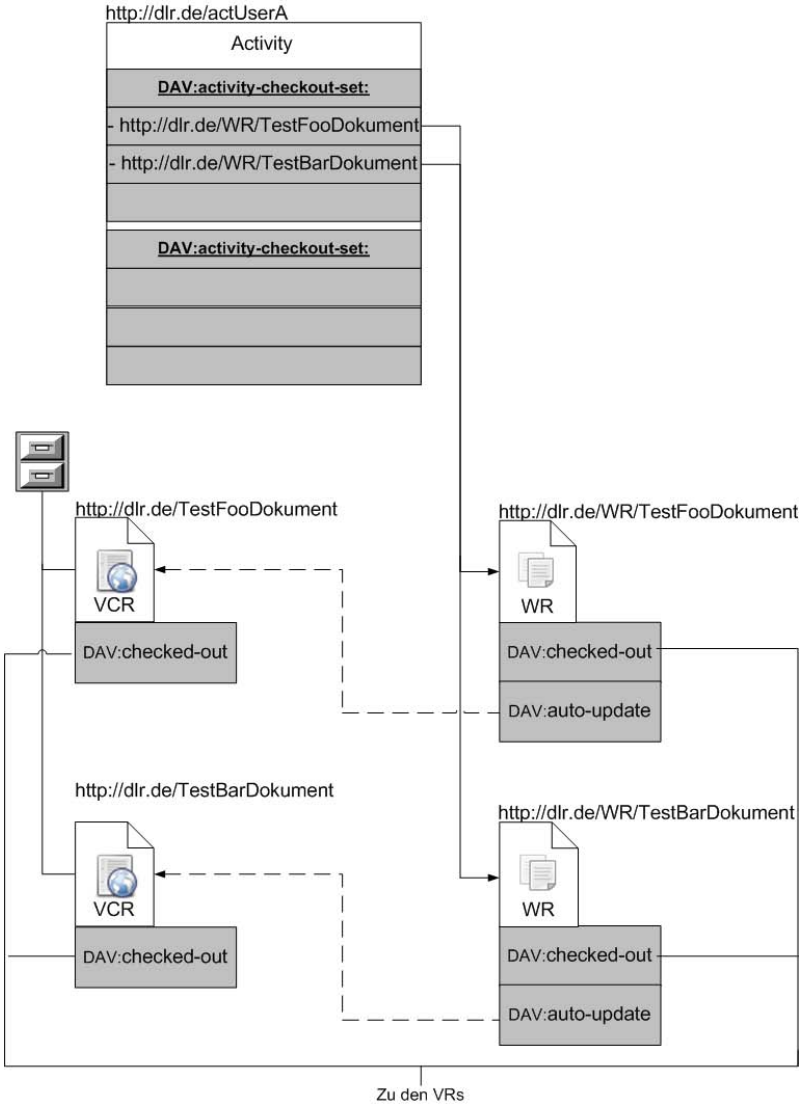

Abbildung 5.2: Zustand auf dem Server nach dem Checkout zweier Ressourcen

Um die durchgeführten Änderungen auf den jeweiligen Kopien dauerhaft zu speichern, muss die Transaktion mit dem Aufruf der Funktion TXN_END beendet werden (siehe Abschnitt 5.2.1). Dies bewirkt, dass der originale Datenbestand mit den Daten der jeweiligen Kopie aktualisiert wird. Um die Verbindung zwischen der Working-Ressource und

der ursprünglichen Original-VCR herzustellen, liest der Server das Attribut "auto-update" der Working-Ressource aus, welches auf die ursprüngliche URL der VCR verweist (siehe Abbildung 5.2). In den folgenden Abschnitten wird nun detailliert vorgestellt, wie die Bearbeitung der Kopien durchgeführt wird.

Ressourcen transaktionssicher lesen - TXN_OPT_RESOURCE_READ

Um Leseoperationen transaktionssicher durchführen zu können, also um z.B. beim Beenden einer Transaktion festzustellen, ob sich die Ressource seit dem letzten Zugriff verändert hat, muss die Ressource mit Hilfe der Funktion TXN_OPT_CHECKOUT (siehe Abschnitt 5.3.1) ausgecheckt werden. Die eigentlichen transaktionssicheren Leseoperationen arbeiten dann auf der entsprechenden Working-Ressource-Kopie. Sie können, analog zum READ_N-Verfahren (siehe 5.2.3) mit den dafür vorgesehenen WebDAV-Methoden GET und PROPFIND durchgeführt werden.

Ressourcen bearbeiten – TXN_OPT_RESOURCE_EDIT

Allen in diesem Abschnitt zur Modifikation einer Ressource vorgestellten Funktionen ist gemein, dass, bevor die Bearbeitung einer Ressource durchgeführt werden kann, diese mit der Funktion TXN_OPT_CHECKOUT ausgecheckt und damit eine entsprechende Kopie der originalen Ressource angelegt werden musste (siehe Abschnitt 5.3.1). Im Folgenden werden die relevanten WebDAV-Methoden GET, PUT und PROPPATCH zum Bearbeiten noch einmal genauer betrachtet:

- Um den Inhalt einer Datei zu verändern, muss diese erst auf den Client übertragen werden, da das WebDAV-Dateisystem nach dem Upload/Download-Modell arbeitet (siehe Abschnitt 3.2.1). Mit Hilfe einer GET-Anfrage, die die URL der Kopie enthält, kann der Client die Datei beim Server anfordern und lokal abspeichern. Änderungen an der Datei werden lokal vorgenommen. Die PUT-Methode speichert den veränderten Inhalt unter Angabe der URL der Working-Ressource-Kopie auf dem Server.

- Dateiinhalt überschreiben: Mit Hilfe der PUT-Methode kann der neue Dateiinhalt direkt unter Angabe der entsprechenden URL der Working-Ressource-Kopie gespeichert werden.

- Properties verändern, hinzufügen und löschen: Zum Ändern, Hinzufügen und Löschen von Properties muss die Ressource nicht mit Hilfe der GET-Methode zum

Client übertragen werden. Die PROPPATCH-Methode erlaubt das direkte Ändern, Hinzufügen und Löschen von Properties unter Angabe der URL der entsprechenden Working-Ressource-Kopie. Es sei hier erwähnt, dass manche Live-Properties, wie etwa das "creationdate", durch den Server geschützt sind und nicht durch den Client verändert werden können.

Ressourcen löschen – TXN_OPT_RESOURCE_DEL

Beim Löschen einer VCR lässt der DeltaV-Standard einem Entwickler viel Freiraum. So ist diese Methode zum einen optional, d.h. ein Server muss diese Methode nicht implementieren, und zum anderen bzgl. ihres Verhaltens nicht genau spezifiziert (siehe dazu auch [Dus04]).

In dem im Rahmen dieser Arbeit entwickelten Modell wird darum vorgeschrieben, dass eine VCR nach dem Checkin/Checkout-Modell nur dann gelöscht werden kann, wenn sich die VCR im Zustand Checked-out befindet. Die Ressource muss also entsprechend mit Hilfe der Funktion TXN_OPT_CHECKOUT ausgecheckt werden und kann anschließend mit Hilfe der DELETE-Methode unter Angabe der URL der entsprechenden Working-Resource-Kopie gelöscht werden. Auf die genauere Spezifikation des Standards wird in Kapitel 6 eingegangen.

Ressourcen kopieren - TXN_OPT_RESOURCE_COPY

Zum Kopieren von Ressourcen wird die WebDAV-Methode COPY verwendet. Die Ausführung der COPY-Methode ist dabei äquivalent zu der Ausführung der folgenden vier Befehle:

- GET-Methode mit der URL der Quell-Ressource zum Einlesen der Daten
- PROPFIND-Methode mit der URL zum Einlesen der Proptertys
- PUT-Methode mit dem Inhalt der Quell-Daten zum Schreiben der Daten an den neuen Zielort (Ziel-URL)
- PROPPATCH-Methode zum Setzen der Properties der Quell-Daten an dem neuen Zielort

Da das Übertragen auf den Client jedoch unnötig ist, kann die COPY-Methode mit Angabe der Quell- und der Ziel-URL dafür sorgen, dass der Server das Kopieren direkt übernimmt.

Durch die Nutzung des DeltaV-Standards kann bei der Verwendung der COPY-Methode eine Unterscheidung in vier Fälle vorgenommen werden. Diese Fallunterscheidung ist notwendig, da der DeltaV-Standard zwischen VCR und Ressourcen, die nicht unter Versionskontrolle stehen, unterscheidet. In [Dus04] wurden diese Fälle wie folgt als Use Cases bzw. Anwendungsfälle beschrieben:

- UC1: Quelle und Ziel sind beide VCRs
- UC2: Quelle ist eine VCR – Ziel besteht schon, ist aber nicht unter Versionskontrolle, also keine VCR
- UC3: Quelle besteht schon, ist aber nicht unter Versionskontrolle – Ziel ist eine VCR
- UC4: Quelle ist eine VCR - Ziel existiert nicht

Im Folgenden wird aufgezeigt, wie diese Use Cases im Transaktionsmodell behandelt werden. Ziel ist es dabei, die Use Cases UC2 bis UC4 auf den Use Case UC1 abzubilden, da dieser die folgenden, zusätzlich zum WebDAV-Standard geltenden, Vorbedingungen erfüllt.

- VB1: Das Ziel muss eine VCR sein.
- VB2: Die Quelle muss eine VCR.

Die Bedingungen VB1 und VB2 sind nötig, da das in dieser Arbeit entwickelte Transaktionsmodell grundsätzlich nur mit VCRs arbeitet.

Wie bereits beschrieben, werden diese Vorbedingungen bereits durch den Anwendungsfall UC1 erfüllt. Für die Anwendungsfälle UC2 und UC3 wird die Ziel-Ressource (UC2) bzw. Quell-Ressource (UC3) mit Hilfe der Methode VERSION-CONTROL, unter Angabe der entsprechenden URL, der Versionskontrolle unterstellt. Für den Fall UC4 ist ein vorheriges Anlegen der Ziel-Ressource mit Hilfe der in Abschnitt 5.2 festgelegten Funktion TXN_CREATE_RESOURCE notwendig.

Somit erfüllen alle Anwendungsfälle die Vorbedingungen VB1 und VB2 und die COPY-Methode, die im Weiteren mit der Funktion TXN_OPT_RESOURCE_COPY bezeichnet wird, kann wie folgt ausgeführt werden.

- Die Ziel-VCR wird mit der Funktion TXN_OPT_CHECKOUT ausgecheckt[6].
- Die Quell-VCR wird mit der Funktion TXN_OPT_CHECKOUT ausgecheckt[7].
- Im letzten Schritt kann der Client nun die ursprüngliche COPY-Methode (siehe Abschnitt 3.1) ausführen, jedoch werden die ursprünglichen URLs der Ziel- und der Quell-VCR durch die URLs der in den beiden vorherigen Schritten angelegten Working-Ressource-Kopien ersetzt.

Abbildung 5.3 fasst das algorithmische Vorgehen zur Umsetung der Funktion TXN_OPT_COPY mit Hilfe eines UML 2.0 Sequenzdiagramms noch einmal zusammen. Dort wird zunächst in den Alternativabläufen alt1 bis alt3 sichergestellt, dass Quelle und Ziel eine VCR sind. Anschließend werden sie entsprechend mit der Funktion TXN_OPT_CHECKOUT ausgecheckt. Danach wird die eigentliche COPY-Methode auf den Working-Ressource-Kopien durchgeführt. Nachdem der Server die Anweisung zum Kopieren in Form der COPY-Methode erhalten hat, führt er serverseitig die entsprechenden Befehle aus, die notwendig sind, um das Endergebnis so aussehen zu lassen, als würde der Server eine GET- und eine PROPFIND-Methode auf die Quellressource ausführen und anschließend diese Daten per PUT- und PROPPATCH-Methode auf das angegebene Ziel übertragen.

Ressourcen verschieben – TXN_OPT_RESOURCE_MOVE

Zum Verschieben von Ressourcen wird die WebDAV-Methode MOVE verwendet. Sie verhält sich im Prinzip wie die COPY-Methode, nur dass hier zusätzlich die Quelle-Ressource beim Abschluss des Vorganges gelöscht wird. Da das Herunterladen auf den Client auch hier unnötig ist, sorgt die MOVE-Methode durch Angabe der Quell- und der Ziel-URL dafür, dass der Server alle nötigen Operationen direkt durchführt.

Für die Quell- und die Zielressource ergeben sich die gleichen Fallunterscheidungen wie im vorherigen Abschnitt zur Umsetzung der Funktion TXN_OPT_RESOURCE_COPY. O.B.d.A.[8] seien im Folgenden die Bedingungen VB1 und VB2 (Quelle und Ziel sind eine VCR) vorausgesetzt.

[6] Es wird eine Working-Ressource-Kopie der Ziel-VCR angelegt und einer entsprechenden Transaktion zugeordnet.

[7] Es wird eine Working-Ressource-Kopie der Quell-VCR angelegt und einer entsprechenden Transaktion zugeordnet.

[8] Ohne Beschränkung der Allgemeinheit

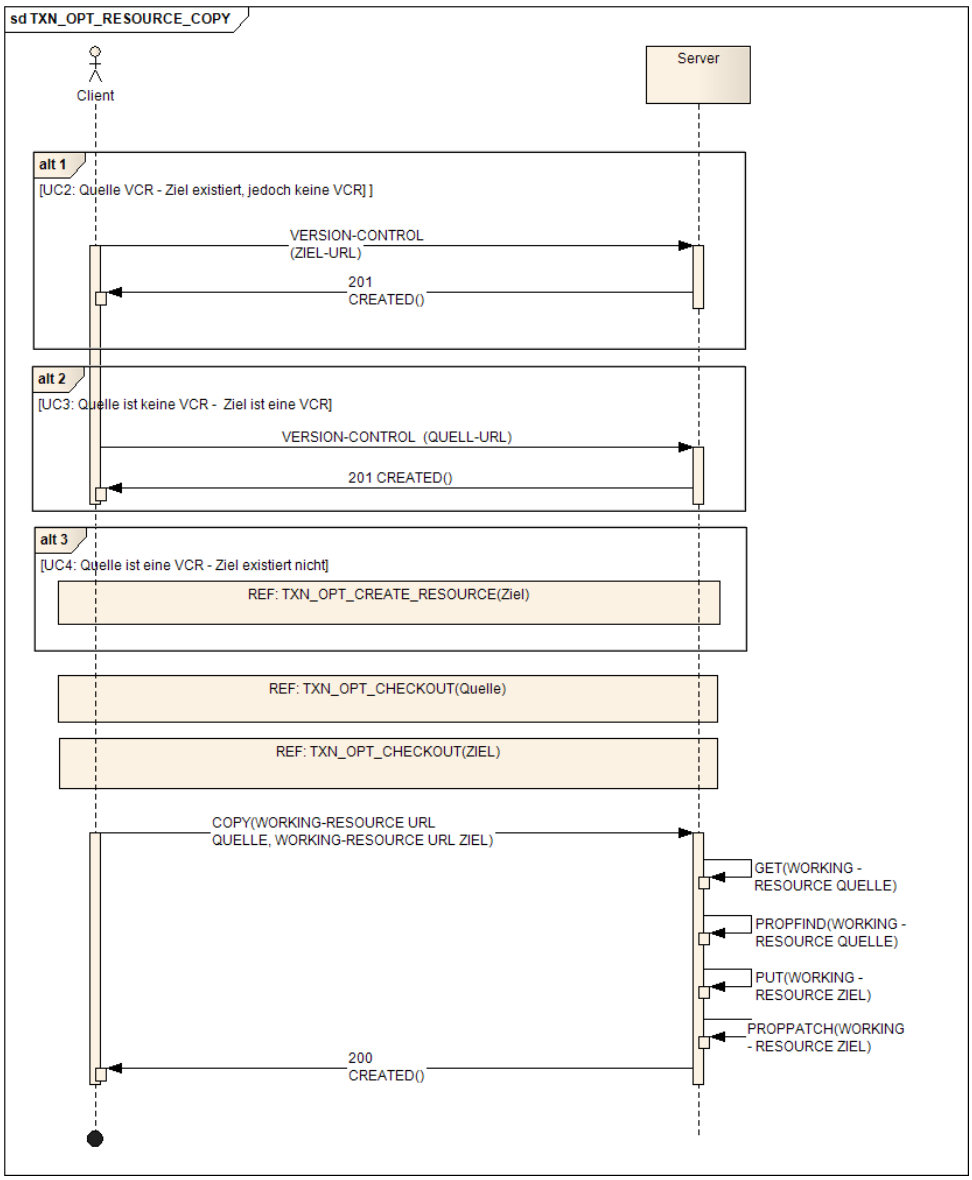

Abbildung 5.3: Befehlsfolge zum Kopieren einer Ressource

Die algorithmische Vorgehensweise für TXN_OPT_RESOURCE_MOVE ist in enger Anlehnung an TXN_OPT_RESOURCE_COPY realisiert. Der einzige Unterschied besteht darin, dass im letzten Schritt die MOVE-Methode, im Gegensatz zur COPY-Methode, mit den entsprechenden Ziel- und Quell-Working-Ressource-Kopien aufgerufen wird.

Abbildung 5.4 fasst hier, ebenfalls in einem UML 2.0 Sequenzdiagramm, die Aktionen, die zur transaktionssicheren Ausführung der Funktion TXN_OPT_RESOURCE_MOVE in diesem Modell notwendig sind, zusammen. Dabei entsprechen die Alternativabläufe alt1 bis alt3 der Funktion TXN_OPT_RESOURCE_COPY. Sie dienen auch hier dazu, die

Bedingungen VB1 und VB2 zu erfüllen. Anschließend werden die Quell- und die Ziel-Ressource mit Hilfe der Funktion TXN_OPT_CHECKOUT ausgecheckt und einer Transaktion zugewiesen. Ist dies erfolgreich, kann der Client die eigentliche MOVE-Methode unter Angabe der URLs von original Ziel- und Quell-Working-Ressource-Kopie ausführen.

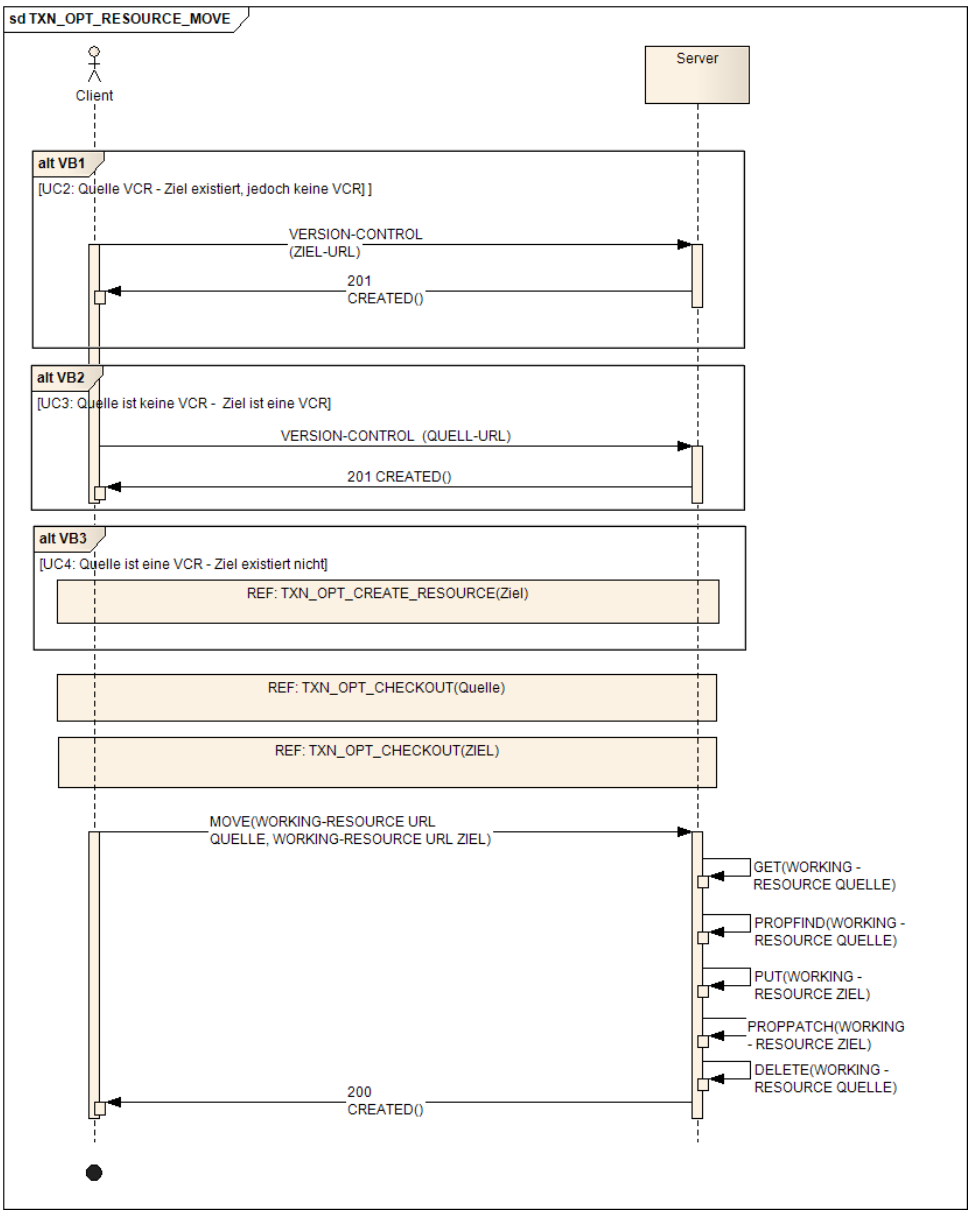

Abbildung 5.4: Befehlsfolge zum Verschieben einer Ressource

5.3.2 Sicherstellung der Serialisierung der Dateioperationen – Validierungsphase

In diesem Abschnitt wird vorgestellt, wie das Transaktionsmodell die Phase zwei der optimistischen Nebenläufigkeitskontrolle, die Validierungsphase, auf den WebDAV-Standard abbildet. Diese Phase beginnt, wenn der Benutzer die Transaktion mit Hilfe der Funktion TXN_END beendet (siehe Abschnitt 5.2). Nun muss der Server überprüfen, ob die gewünschten Änderungen der Transaktion, die auf den Kopien der originalen VCRs durchgeführt wurden, serialisierbar sind, also keinen Konflikt mit nebenläufigen Transaktionen auslösen. [KR81] legt dar, welche Bedingungen innerhalb der Validierungsphase erfüllt sein müssen, damit die Schritte einer Transaktion serialisierbar sind. Im Folgenden werden diese Bedingungen aufgeführt. Dabei seien die Transaktionen $T_1, T_2, \ldots, T_N$ konkurrierend ablaufende Transaktionen. Jeder Transaktion wird zu Beginn der Ausführungsphase ein Zeitstempel $time(T_j)$ zugeordnet. Dieser Zeitstempel wird angelegt, wenn die erste Ressource von der Transaktion angefordert wird. Die Menge der Elemente, die innerhalb der Ausführungsphase verändert wurden, wird mit $WE(T_j)$, gekennzeichnet (WE als Abkürzung für write element). Entsprechend werden die innerhalb der Transaktion zwar mit den Funktionen TXN_OPT_RESOURCE_READ gelesenen bzw. mit TXN_OPT_CHECKOUT ausgecheckten, jedoch nicht veränderten, Elemente mit $RE(T_j)$ gekennzeichnet (RE als Abkürzung für read element).

Anschließend wird überprüft, ob die abzuschließende Transaktion T_j sich nicht mit einer bereits abgeschlossenen oder gerade ebenfalls laufenden Transaktion T_i im Konflikt befindet. Um T_j erfolgreich abzuschließen, muss die Transaktion mindestens eine der folgenden drei Bedingungen erfüllen.

- B1: Für $1 \leq i \neq j \leq n$ gilt: Transaktion T_i ist bereits abgeschlossen, hat also ihre Schreibphase (Phase 3) beendet, bevor die Transaktion T_j ihre Ausführungsphase (Phase 1) beginnt.

- B2: Für $1 \leq i \neq j \leq n$ gilt: Die Elemente, die von T_i geschrieben werden, werden nicht von T_j gelesen. Es gilt also $WE(T_i) \cap RE(T_j) = \emptyset$. Des Weiteren muss T_i ihre Schreibphase (Phase 3) beendet haben, bevor T_j die ihre beginnt.

- B3: Für $1 \leq i \neq j \leq n$ gilt: Die Elemente, die von T_i geschrieben werden, werden nicht von T_j gelesen oder geschrieben. Es gilt also $WE(T_i) \cap (RE(T_j) \cup WE(T_j)) = \emptyset$.

Des Weiteren muss T_i ihre Ausführungsphase vor derjenigen von T_j beendet haben.

Abbildung 5.5 zeigt, zu welchen Zeitpunkten bei der Abarbeitung der einzelnen Phasen der Transaktionen die Bedingungen B1 bis B3 frühestens überprüft werden können.

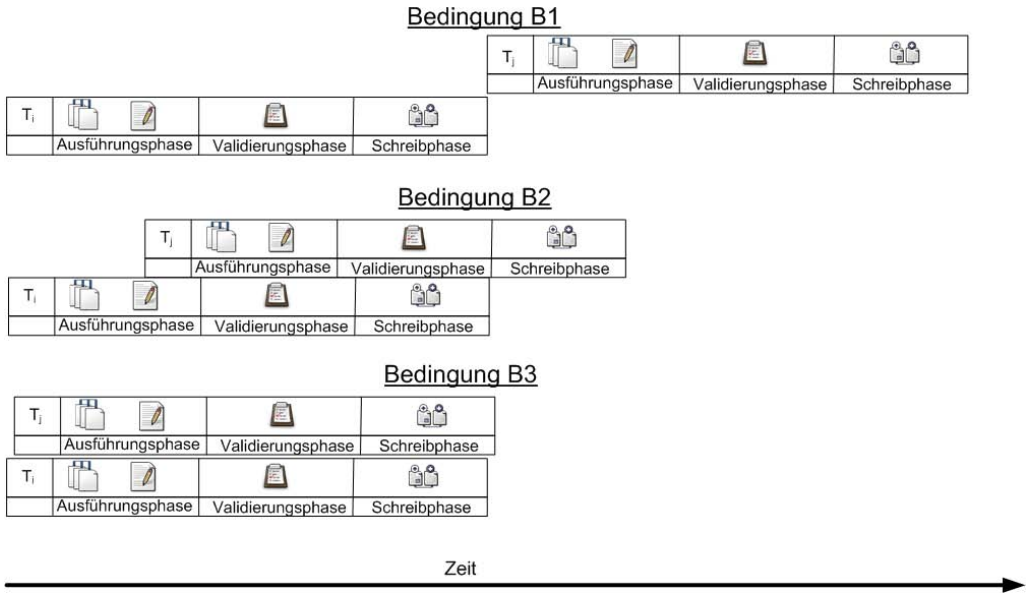

Abbildung 5.5: Zeitliche Überprüfung der Bedingungen zur Serialisierung, angelehnt an [KR81]

Wie bereits beschrieben, muss nur eine der drei Bedingungen zum erfolgreichen Abschluss von T_j erfüllt sein. In [Dad96] wird aufgezeigt, wie die Bedingungen B1 bis B3 auf die folgende Bedingung B_OPT reduziert werden können. Demnach muss für eine erfolgreich zu validierende Transaktion T_j gelten:

- B_OPT: Für $1 \leq i \neq j \leq n$ gilt: Die Elemente, die von T_i geschrieben werden, werden nicht von T_j gelesen. Es gilt also $WE(Ti) \cap RE(T_j) = \emptyset$. Des Weiteren muss T_i ihre Schreibphase (Phase 3) beendet haben, bevor T_j die Validierungsphase beginnt.

Das bedeutet, dass die Bedingung B_OPT in der Validierungsphase von T_j überprüft, ob T_j inkonsistente bzw. veraltete Daten eingelesen und/oder verarbeitet hat. Um dies umzusetzen, wird überprüft, ob eine andere Transaktion T_i mit $1 \leq i \neq j \leq n$, die während der Ausführungsphase von T_j in ihre Schreibphase übergegangen ist, Elemente verändert hat, die von T_j in der Ausführungsphase eingelesen wurden. Abbildung 5.6 zeigt analog zu 5.5, ab wann die Bedingung B_OPT frühestens überprüft werden kann.

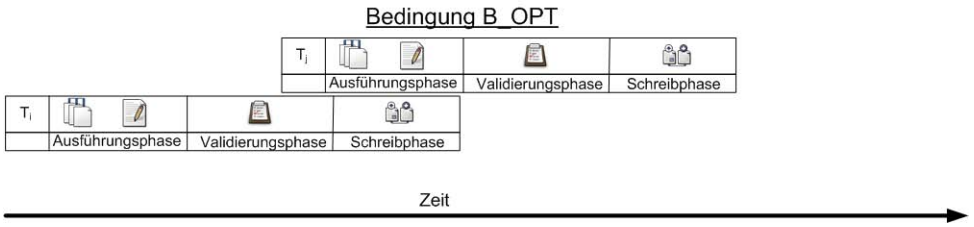

Abbildung 5.6: Zeitliche Überprüfung der Bedingung B_OPT

Im nächsten Abschnitt soll nun beispielhaft gezeigt werden, wie die vorhandenen Konzepte von WebDAV dazu genutzt werden können, die Bedingung B_OPT zu überprüfen. Dabei sei angemerkt, dass der vorgestellte Ansatz nur ein Vorschlag zur Implementierung ist, um aufzuzeigen, welche Hilfsmittel der Standard zur Überprüfung der Serialisierbarkeit bietet. Es ist dem Entwickler freigestellt, eine davon abweichende, seinen Bedürfnissen optimal angepasste Strategie zu nutzen. Einzige Voraussetzung, die der Entwickler sicherstellen muss, ist die, dass die Transaktionen zum erfolgreichen Abschluss eine der Bedingungen B1 bis B3 bzw. B_OPT erfüllen und dass das im Abschnitt 5.2 festgelegte Verhalten durch den Server sichergestellt ist.

Überprüfung der Serialisierbarkeit mit Hilfe von WebDAV

Im Folgenden wird nun beispielhaft gezeigt, wie die im vorherigen Abschnitt vorgestellte Bedingung B_OPT mit Hilfe des WebDAV-Standards überprüft werden kann.

Bedingt durch das Upload/Download-Modell des WebDAV-Datenmodells (siehe Abschnitt 3.1) sowie der in diesem Transaktionsmodell definierten Verhaltensweisen, muss zur Veränderung des Datenbestandes jede Ressource gelesen werden, bevor sie verändert werden kann. Dadurch enthalten die Lese- bzw. Schreibmengen einer Transaktion $RE(T_i)$ bzw. $WE(T_i)$ immer die gleichen Datenelemente. Dabei ist es unerheblich, ob das Lesen explizit durch den Client, z.B. beim Ausführen der Funktion TXN_OPT_RESOURCE_READ oder implizit durch den Server bei der Ausführung der Funktion TXN_OPT_RESOURCE_MOVE erfolgt. Aus diesem Grund soll hier nicht mehr zwischen Lesemenge $RE(T_i)$ und Schreibmenge $WE(T_i)$ innerhalb einer Transaktion unterschieden werden. Stattdessen wird mit einer neuen Menge CHK(T_i) gearbeitet (CHK steht dabei für Checked-out), es gilt $CHK(T_i) = RE(T_i) \cup WE(T_i)$. Diese Menge $CHK(T_i)$ beinhaltet, übertragen auf den WebDAV-Standard, also alle Ressourcen, die einer Transaktion mit Hilfe der Funktion TXN_OPT_CHECKOUT zugeordnet wurden.

Auf Basis dieser neuen Menge $CHK(T_i)$ wird die im vorherigen Abschnitt vorgestellte Bedingung B_OPT zur Validierung von T_j angepasst:

- CHKB_OPT - Anpassung von B_OPT: Für $1 \leq i \neq j \leq n$ gilt, dass die Elemente, die von T_i geschrieben oder gelesen werden, nicht von T_j gelesen oder geschrieben werden. Es gilt also $(WE(T_i) \cup RE(T_i)) \cap (WE(T_j) \cup RE(T_j)) = \emptyset$ und somit $CHK(T_i) \cap CHK(T_j) = \emptyset$. Des Weiteren muss T_i ihre Schreibphase (Phase 3) beendet haben, bevor T_j ihre Validierungsphase beginnt.

Da die zu überprüfende Menge der Datenelemente vergrößert wurde, stellt die Verwendung der Menge $CHK(T_i)$ somit eine Verschärfung der bestehenden Bedingung B_OPT dar.

Zur Umsetzung der Bedingung CHKB_OPT wird nun ausgenutzt, dass, wie bereits in Kapitel 3.2.2 beschrieben, die DeltaV-Erweiterung versionsbasiert arbeitet. Jedes Auschecken einer VCR, gefolgt von einem Einchecken, erzeugt eine neue VR und damit eine neue Version der Ressource. Die Kopie einer VCR, also die Working-Ressource, „merkt“ sich nun, von welcher Version (VR) sie ursprünglich erzeugt wurde. Beim Einchecken, wenn also der Inhalt der Kopie auf die ursprüngliche VCR übertragen wird, wird zuvor die Versionsnummer der Kopie mit der Versionsnummer der originalen VCR verglichen. Sind die beiden Versionsnummern identisch, hat es in der Zwischenzeit, also seit der Erstellung der Kopien, keine Änderungen an der originalen VCR gegeben. Die originale VCR kann nun mit den Daten der Working-Ressource-Kopie aktualisiert werden. Sind die Versionsnummern hingegen unterschiedlich, wurde die VCR bereits durch eine andere Transaktion bearbeitet und aktualisiert. Dies bedeutet, dass ein Konflikt vorliegt, die Daten dieser Transaktion ungültig sind und die Transaktion abgebrochen werden muss.

Abbildung 5.7 fasst dieses Vorgehen anhand eines Beispiels in einem UML 2.0 Sequenzdiagramm zusammen. Die Benutzer Alice und Bob möchten beide die VCR `TestDokument` bearbeiten. Dazu fügt Alice mit Hilfe der Funktion TXN_OPT_CHECKOUT die VCR ihrer vorher erstellten Transaktion ActUUIDAlice hinzu. Der Server erzeugt eine private Working-Ressource-Kopie der originalen VCR für Alice und antwortet ihr mit der URL der Kopie. Bob führt ebenfalls diese Schritte aus und erhält entsprechend auch eine private Kopie. Beide Kopien basieren dabei auf der Versionsnummer 42. Nun führen Alice und Bob, getrennt voneinander, Änderungen an ihren jeweiligen Kopien mit den Methoden GET, PUT, PROPPATCH usw. durch. Im Sequenzdiagramm wird dies durch einen

Loop-Interaktionsrahmen dargestellt. Nachdem Alice die Bearbeitung abgeschlossen hat, beendet sie ihre Transaktion mit dem Aufruf der Funktion TXN_END auf der Activity ActUUIDAlice. Der Server wird damit angewiesen, auf allen Working-Ressourcen, die dieser Activity zugeordnet sind, ebenfalls die Methode CHECKIN aufzurufen. Beim Einchecken der Working-Ressourcen vergleicht der Server nun die Versionsnummer der Working-Ressource mit der Versionsnummer der ursprünglichen VCR. Diese sind gleich (beide 42) und so werden die Daten der originalen VCR aktualisiert und es wird eine neue Version, die 43., erstellt.

Nun hat Bob ebenfalls seine Bearbeitung abgeschlossen und möchte seine Änderungen an der Ressource abspeichern. Er führt ebenfalls die Funktion TXN_END auf seiner Activity ActUUIDBob aus. Beim Einchecken und dem nachfolgenden Vergleich der Versionsnummern zwischen Working-Ressource und originaler VCR wird jedoch durch den Server festgestellt, dass die Versionsnummern unterschiedlich sind. Die Working-Ressource hat die Version 42 und die originale VCR die Version 43. Somit schlägt der Checkin der Working-Ressource fehl und die gesamte Transaktion von Bob wird zurückgenommen, indem der Server die Methode UNCHECKOUT auf der Activity von Bob aufruft. Somit wurde die Transaktion von Alice erfolgreich abgeschlossen, die von Bob hingegen nicht.

Die Transaktion, die von Alice durchgeführt wurde, war erfolgreich, weil sie die Bedingung CHKB_OPT erfüllte, es gab hier keine vorangegangene Transaktion, die aktiv war, bevor Alice mit ihrer Transaktion die Validierungsphase erreichte.

Bobs Transaktion hingegen erfüllte die Bedingung CHKB_OPT nicht, da zur Validierungszeit von Bobs Transaktion Alice ihre Schreibphase schon beendet hatte und Alice und Bob beide in ihren jeweiligen Mengen $CHK(Alice)$ bzw. $CHK(Bob)$ die VCR `Testdokument` hatten, also $CHK(Alice) \cap CHK(Bob) \neq \emptyset$ ist.

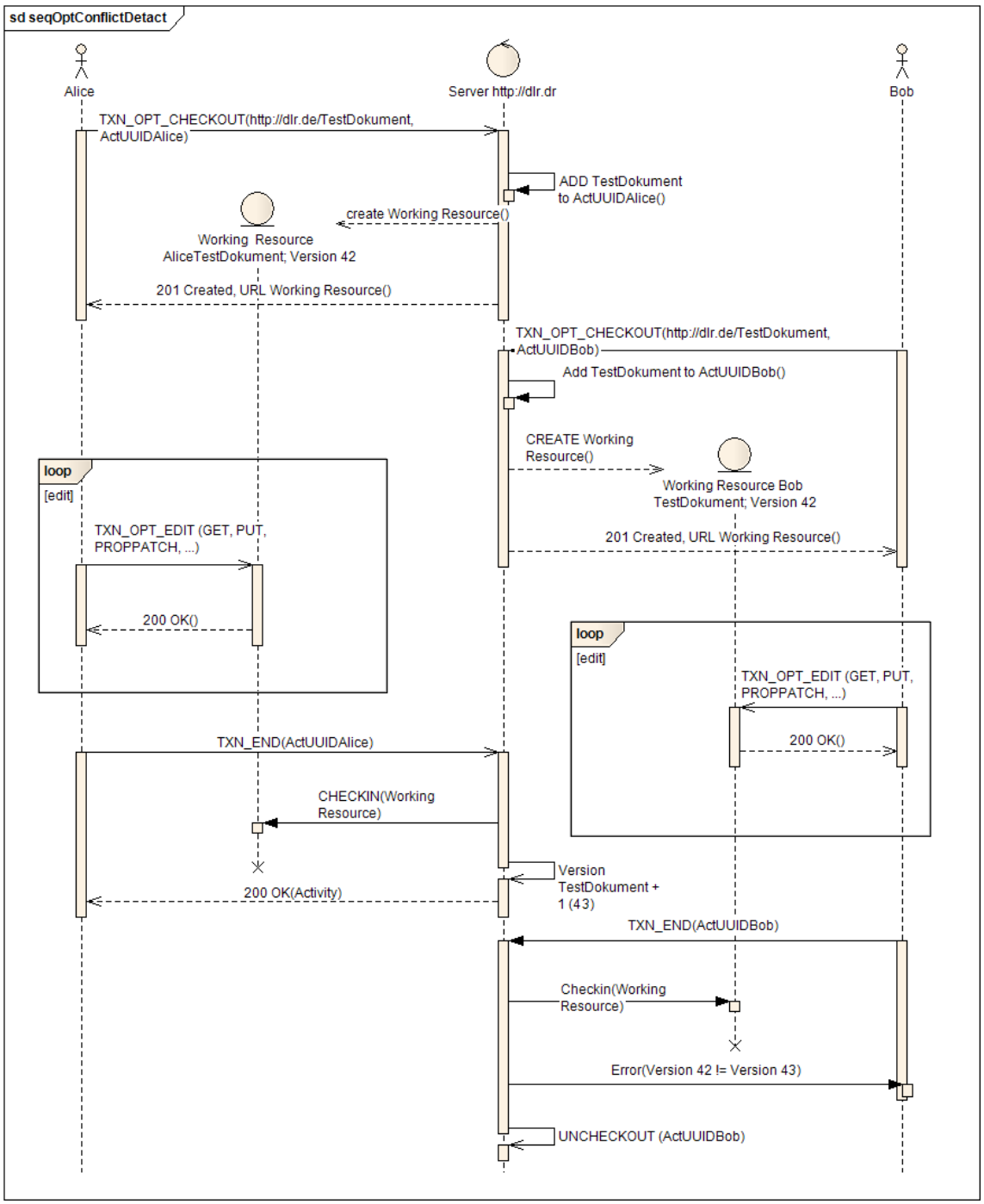

Abbildung 5.7: Überprüfung der Serialisierbarkeit

Im Folgenden wird nun beschrieben, mit welchen WebDAV-Attributen und -Mechanismen die Versionsnummern von Kopie und Original technisch überprüft werden können. Der WebDAV-Standard sieht leider kein explizites Live-Property "Versionsnummer" vor. Zur Umsetzung wird deshalb ausgenutzt, dass die Kopie einer originalen VCR, also die Working-Ressource, wie bereits beschrieben, auch die Live-Properties der Original-VCR erhält. Somit besitzen VCR und Working-Ressource-Kopie beide das Attribut "DAV:checked-out", dass die URL der aktuellsten VR der VCR enthält. Abbildung 5.8 fasst die Situation, bezogen auf das Beispiel von Alice und Bob, auf dem Server zusammen. Dabei ist in der Abbildung die Situation aus Sicht von Alice zu sehen, nachdem sie die VCR `TestDokument` ausgecheckt hat. Die Abbildung zeigt, dass das Attribut "DAV:checked-out" der originalen VCR `TestDokument` als auch die private Working-Ressource jeweils auf die VR `TestDokument42` verweisen.

Wird die Working-Ressource nun wieder eingecheckt, kann der Server die beiden URLs aus den jeweiligen Attributen "DAV:checked-out" vergleichen. Zeigen beide URLs auf die gleiche VR, also auf die gleiche Version, hat sich seit dem Erzeugen der Working-Ressource-Kopien nichts an der Original-VCR geändert. Wie in Abbildung 5.8 zu sehen, zeigen VCR und Working-Ressource auf die gleiche VR und somit besteht zwischen den Transaktionen kein Konflikt bezüglich dieser Ressource.

Hätte sich dagegen seit dem Auschecken der VCR etwas geändert, wäre also die VCR in der Zwischenzeit durch eine andere Transaktion bearbeitet worden, würden die Attribute "DAV:checked-out" bzw. "DAV:checked-in" der VCR auf eine andere VR-URL bzw. Version zeigen als das Attribut "DAV:checked-out" der Working-Ressource-Kopie.

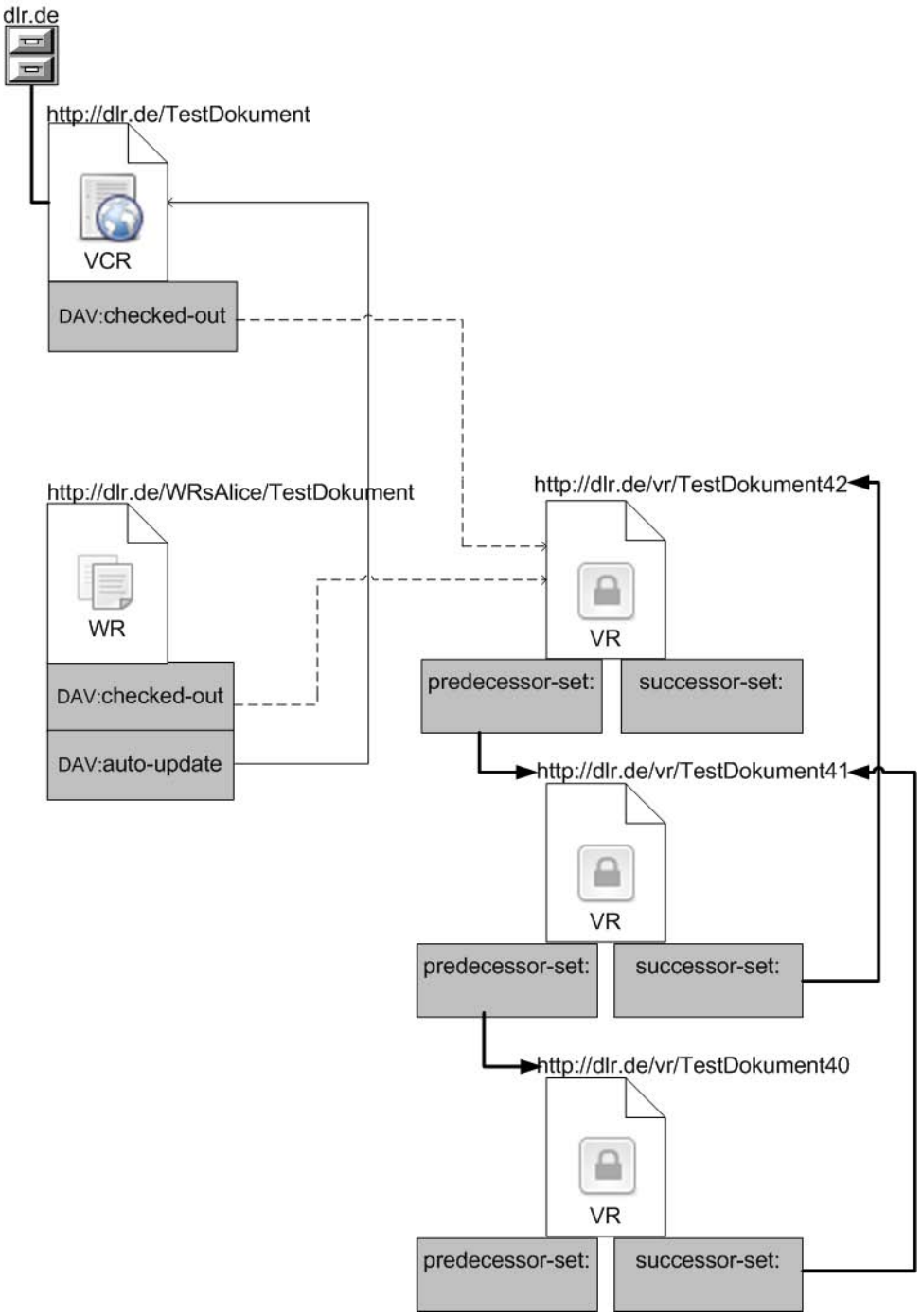

Abbildung 5.8: Situation auf dem Server, nachdem Alice die VCR TestDokument ausgeckeckt hat

Abbildung 5.9 zeigt die Situation aus dem gerade verwendeten Beispiel zum Zeitpunkt als Alice ihre Transaktion erfolgreich beendet hat, Bob sich jedoch noch in der Ausführungsphase befindet. Alice hat die VCR `TestDokument` erfolgreich eingecheckt und es wurde nach erfolgreichem Abschluss der Validierungs- und Schreibphase eine neue VR `TestDokument43` erstellt, deren URL das Attribut "DAV:checked-in" enthält. Bobs Working-Ressource-Kopie (in der Abbildung rot umrandet) zeigt jedoch noch auf die veraltete VR `TestDokument42`. Möchte nun im nächsten Schritt Bob seine Transaktion beenden, schlägt das Einchecken der Working-Ressource und damit auch seine Transaktion fehl, da das Attribut "DAV:checked-out" seiner Working-Ressource auf eine andere VR zeigt, als das Attribut "DAV:checked-in" der originalen VCR.

Die Überprüfung anhand von Versionsnummern entspricht dabei dem Architekturmuster Optimistic Offline Lock. Architekturmuster sind bewährte Lösungsvorschläge und Schemata für bestimmte Probleme. Ein Muster bietet dem Entwickler somit die Möglichkeit, mit einem erprobten Konzept jeweils eine Klasse von Problemen, unabhängig von einer bestimmten Programmiersprache, in der Entwicklung zu lösen. Das Optimistic Offline Lock Architekturmuster wird im Standardwerk für Entwurfsmuster zur Entwicklung von großen Software-Systemen "Patterns für Enterprise Application-Architekturen" von Martin Fowler vorgestellt [Fow03].

Ein einfacherer Ansatz zur Lösung des gerade beschriebenen Problems wäre die Prüfung, ob sich eine VCR im Zustand Checked-out bzw. Checked-in befindet. Im Detail betrachtet sieht dieser Ansatz vor, dass beim Einchecken einer VCR durch eine Transaktion zunächst überprüft wird, ob die Ressource, die zuvor bereits ausgecheckt wurde, sich wieder im Zustand Checked-in befindet, also schon von einer anderen Transaktion verändert und eingecheckt wurde. Dieser Ansatz zur Prüfung der Bedingung CHKB_OPT kann jedoch bei näherem Betrachten nicht verwendet werden, da er zu Problemen führt, wenn mehr als zwei konkurrierende Transaktionen aktiv sind. Für zwei Transaktionen liefert der Ansatz noch ein richtiges Ergebnis, bei drei konkurrierenden Transaktionen kann es hingegen schon zu folgendem Fehlerfall kommen. Angenommen die Transaktionen T_i und T_j haben zum Zeitpunkt t_0 die VCR `TestDokument` ausgecheckt. Die Transaktion T_i nimmt nun Änderungen an ihrer Kopie vor und checkt die Änderungen zum Zeitpunkt t_{0+1} wieder ein, die VCR wird also in den Zustand Checked-in versetzt. Eine weiter Transaktion T_p checkt nun zum Zeitpunkt $t0 + 2$ die Ressource wieder aus, so dass sich die VCR wieder im Zustand Checked-out befindet. Zum Zeitpunkt $t0 + 3$ checkt die Transaktion T_j ihre

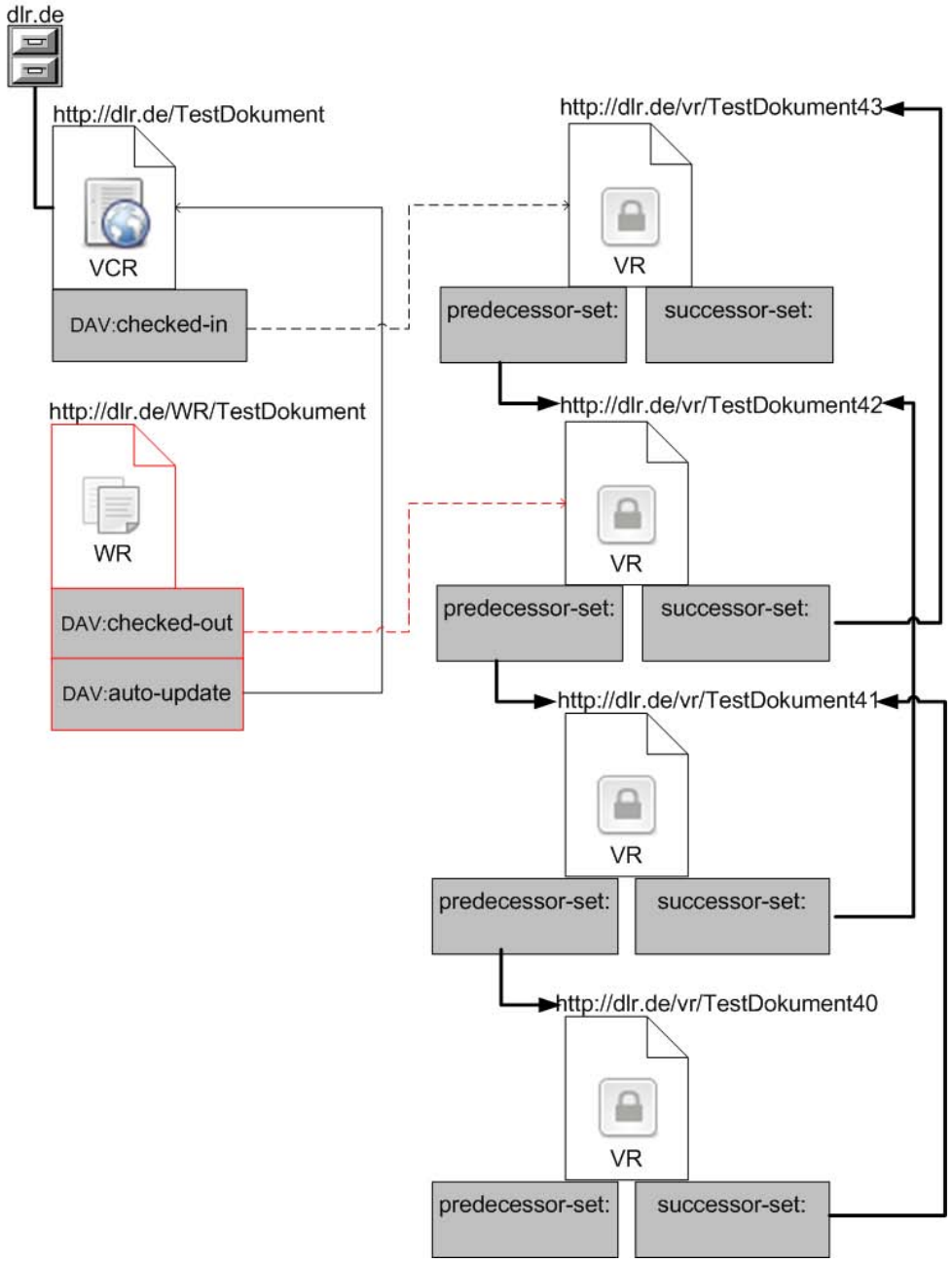

Abbildung 5.9: Situation auf dem Server, nachdem Bob seine VCR einchecken möchte

Änderungen ein. Sie sieht, dass sich die VCR im Zustand Checked-out befindet, geht also davon aus, dass es seit ihrem Checkout keine Änderungen an der Ressource gegeben hat. Hier liegt nun der Fehler, da T_i die Ressource bereits geändert hat, T_j dies aber nicht erfährt. Beim Einchecken werden nun die Ergebnisse von T_i überschrieben und es kommt zu einem Lost-Update-Problem.

5.3.3 Abbildung der Schreibphase auf den WebDAV-Standard

Nachdem eine Transaktion erfolgreich validiert wurde, kann nun die Phase drei, die Schreibphase, durchgeführt werden. Der Server muss hierzu für alle Working-Ressourcen, die sich in der Menge "DAV:activity-checkout-set" befinden, die folgenden Schritte durchführen:

- Eine neue VR mit dem Inhalt der Working-Ressource-Kopie anlegen.
- Die VCR in den Zustand Checked-in setzen und in das Attribut "DAV:checked-in" die URL der neu erstellten VR einstellen.

Ist die Menge "DAV:activity-checkout-set" erfolgreich abgearbeitet, ist damit auch die gesamte Transaktion erfolgreich abgeschlossen und dem Client wird im letzten Schritt mit der Antwort "200 OK" die erfolgreiche Abarbeitung der Transaktion signalisiert.

5.4 Abbildung der pessimistischen Nebenläufigkeitskontrolle auf den WebDAV-Standard

Auf Basis des Grundmodells, welches in Abschnitt 5.2 vorgestellt und in dem die Grundlagen zur Kennzeichnung von Start, Ende und Abbruch einer Transaktion beschrieben wurden, soll nun die zweite Ausprägung des Modells vorgestellt werden. Diese beschreibt das Vorgehen bei Verwendung einer pessimistischen Nebenläufigkeitskontrolle, die bereits in Abschnitt 3.4 vorgestellt wurde.

Die meisten Umsetzungen einer pessimistischen Nebenläufigkeitskontrolle arbeiten dabei mit so genannten Sperrverfahren [Sch03], die den Zugriff auf eine Ressource im Vorhinein mit entsprechenden Sperren bzw. Locks beschränken und somit Konflikte möglichst verhindern[9]. In den so genannten Sperrprotokollen wird festgelegt, wie die entsprechenden Sperren gesetzt und freigeben werden müssen, damit eine Transaktion serialisierbar ist und keine Konflikte mit anderen Transaktionen auslöst.

Im Folgenden werden nun anhand eines ausgewählten Sperrprotokolls die Mechanismen und Konzepte vorgestellt, die WebDAV zurzeit zur Unterstützung der pessimistischen Nebenläufigkeitskontrolle bietet. Als Sperrprotokoll wird dabei das sogenannte Zwei-Phasen-Sperrprotokoll verwendet, welches im nächsten Abschnitt kurz vorgestellt wird. Es wurde

[9] Im Gegensatz zu der im vorherigen Abschnitt vorgestellten optimistischen Nebenläufigkeitskontrolle, bei der der Schwerpunkt auf der Konflikterkennung liegt.

ausgewählt, da es das nach [WV02] meistgenutzte Protokoll im kommerziellen Datenbankumfeld, dem Ursprung der Transaktionsverarbeitung, darstellt. Hier sei noch erwähnt, dass bei Verwendung dieses Sperrprotokolls alle Konzepte und Mechanismen zum Einsatz kommen, die der WebDAV-Standard im Bereich der Sperrprotokolle bietet.

Der bestehende WebDAV-Standard sieht zur Zeit keine Möglichkeit vor, dass der Client das vom Server unterstützte Sperrprotokoll ermitteln kann. Aus diesem Grund wird in Kapitel 6 ein entsprechender Vorschlag zur Ergänzung des WebDAV-Standards vorgestellt, um so eine Interaktion zwischen unterschiedlichen Client- und Serversystemen zu unterstützen.

5.4.1 Grundlagen des Zwei-Phasen-Sperrprotokolls

Das Zwei-Phasen-Sperrprotokoll, auch 2PL genannt (engl. two-phase locking protocol), wurde erstmals von [GRA78] vorgestellt. Das 2PL-Protokoll tritt in zwei Varianten auf, beide Varianten fordern:

- 2PL1: Eine Transaktion muss Objekte vor dem Zugriff sperren.

Die strikte Variante, dass so genannte strict 2PL, fordert zusätzlich:

- 2PL2a: Alle von einer Transaktion gehaltenen Sperren werden zum Ende der Transaktion wieder freigegeben.

Dagegen fordert die abgemilderte Version des 2PL-Protokolls die Bedingung:

- 2PL2b: Wenn eine Transaktion eine Sperre freigibt, darf sie danach keine zusätzlichen Sperren mehr anfordern. Es kann jedoch bereits vor Ende der Transaktion mit der Freigabe gesperrter Objekte begonnen werden.

In diesem Protokoll steht die Bedingung 2PL1 für die erste Phase, die so genannte Wachstumsphase, in der eine Transaktion ihre benötigten Sperren anfordert. Die zweite Phase, welche durch die Bedingungen 2PL2a bzw. 2PL2b beschrieben wird, ist die entsprechende Freigabephase.

Da die abgemilderte Variante mit der Bedingung 2PL2b deutlich mehr Verwaltungsaufwand erfordert, erhält die strikte Variante mit der Bedingung 2PL2a in der Praxis sehr viel häufiger den Vorzug [Sch03].

5.4.2 Abbildung des strikten Zwei-Phasen-Sperrprotokolls auf den WebDAV-Standard

In diesem Absatz soll nun vorgestellt werden, wie die beiden Phasen des Zwei-Phasen-Sperrprotokolls und das entsprechende Bearbeiten der Ressourcen auf den WebDAV-Standard abgebildet werden können.

Wegen des geringeren Verwaltungsaufwands wird hier die Variante des strikten Zwei-Phasen-Sperrens umgesetzt.

Abbildung der Wachstumsphase auf den WebDAV-Standard – TXN_PES_CHECKOUT

In der Wachstumsphase wird die Bedingung 2PL1 des 2PL-Protokolls umgesetzt. Dazu müssen alle von der Transaktion benötigten Datenobjekte gesperrt werden. Wie bereits in Kapitel 3.2.1 erwähnt, unterstützt der WebDAV-Standard jedoch nur write-locks, die den schreibenden Zugriff auf die jeweilige Ressource verhindern. Das Lesen von Ressourcen ist somit weithin auch bei gesetzter Sperre möglich. In Kapitel 6 wird ein Vorschlag zur Erweiterung des WebDAV-Standards um read-locks vorgestellt.

Damit der Client seine Ressourcen sperren kann, muss er die gewünschte VCR mit Hilfe der LOCK-Methode unter Angabe der URL sperren. Kann der Client die Ressource erfolgreich sperren, erhält er vom Server die Antwort: "200 OK" und im Body der Antwortnachricht den entsprechenden eindeutigen LOCK-Token, den er zum Bearbeiten der Ressource nutzen muss (siehe Abschnitt 3.2.1). Erhält der Client den Lock nicht, muss er warten und es zu einem späteren Zeitpunkt noch einmal versuchen. Da das Auschecken einer VCR auch eine Veränderung der Ressource darstellt, kann nur der Client, der die Ressource vorher gesperrt hat, diese auch auschecken und seiner Transaktion hinzufügen.

Das algorithmische Vorgehen zum Sperren einer Ressource und Hinzufügen zu einer Transaktion, welches im Folgenden als Funktion TXN_PES_CHECKOUT bezeichnet wird, besteht zusammengefasst aus den Folgenden beiden Befehlen:

- LOCK-Methode mit entsprechender Angabe der URL der Ressource.
- CHECKOUT-Methode mit der entsprechenden URL als Parameter und als Option den entsprechenden Lock-Token, sowie die URL der Activity.

Listing 5.7 zeigt ein Auschecken der VCR `TestDokument` mit Hilfe der CHECKOUT-

Methode und das Hinzufügen zu einer Activity, nachdem die Ressource zuvor entsprechend gelockt wurde. In Zeile 5 ist der Lock-Token zu sehen, der es überhaupt erst erlaubt, die VCR `TestDokument` auszuchecken.

```
1  CHECKOUT TestDokument HTTP/1.1
2  Host: dlr.de
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: 1234
5  If: (<locktoken:e71d4fae-5dec-22d6-fea5-00a0c91e6>)
6
7  <?xml version="1.0" encoding="utf-8" ?>
8  <D:checkout xmlns:D="DAV:">
9      <D:activity-set>
10       <D:href>/act/5d585fab-05b3-1d41-95fe-f76556624ab2</D:href>
11    </D:activity-set>
12 </D:checkout>
```

Listing 5.7: Auschecken der VCR TestDokument mit Hilfe der CHECKOUT-Methode

Abbildung 5.10 stellt die Situation auf dem Server noch einmal grafisch dar. Die VCR `Testdokument` ist gelockt und befindest sich im Zustand Checked-out. Die URL der VCR wurde dem Attribut "DAV:activity-checkout-set" der entsprechenden Activity-Ressource hinzugefügt.

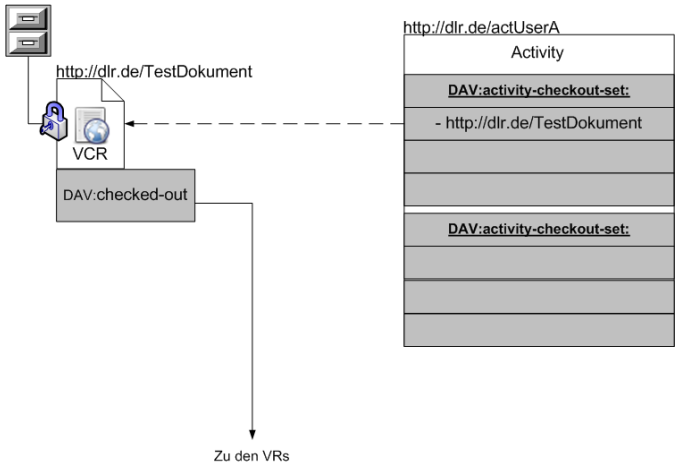

Abbildung 5.10: Situation auf dem Server nach dem Ausführen der Funktion TXN_PES-_CHECKOUT

Ressourcen transaktionssicher lesen mit der Funktion TXN_PES_RESOURCE_READ

Um mit Hilfe der pessimistischen Nebenläufigkeitskontrolle Leseoperationen explizit transaktionssicher durchführen zu können, muss die Ressource mit Hilfe der Funktion TXN_PES_CHECKOUT gelockt und einer Transaktion hinzugefügt werden. Dies stellt sicher, dass die Ressource zur Laufzeit der eigenen Transaktion nicht durch andere Transaktionen verändert wird. Die Funktion TXN_PES_RESOURCE_READ lässt sich also durch die Ausführung der beiden folgenden Funktionen bzw. Methoden auf den WebDAV-Standard abbilden:

- TXN_PES_CHECKOUT
- Ausführen der WebDAV-Methoden GET und PROPFIND zum transaktionssicheren Lesen der Ressource.

Ressourcen bearbeiten – TXN_PES_RESOURCE_EDIT

Um Ressourcen zu verändern, muss die entsprechende VCR ebenfalls mit Hilfe der Funktion TXN_PES_CHECKOUT gesperrt und einer Transaktion zugewiesen werden. Ist dies erfolgreich durchgeführt worden, kann die Ressource mit den WebDAV-Methoden GET, PUT und PROPPATCH bearbeitet werden.

Im Gegensatz zum optimistischen Verfahren werden hier die Änderungen nicht auf einer privaten Kopie durchgeführt, sondern es wird direkt auf der originalen URL der VCR gearbeitet. Der Server muss trotzdem sicherstellen, dass er mit geeigneten Mitteln die Änderungen an der Ressource erst nach erfolgreichem Ablauf der Transaktion sichtbar macht. Die entsprechende Implementierung ist hier dem jeweiligen Entwickler bei der Realisierung des Modells freigestellt. Er muss nur sicherstellen, dass das hier modellhaft beschriebene Verhalten nach außen für den Client korrekt eingehalten wird.

Ressourcen löschen – TXN_PES_RESOURCE_DEL

Das Löschen einer VCR ist im DeltaV-Standard, wie bereits in Abschnitt 5.3 zur optimistischen Nebenläufigkeit dargelegt, nicht genau spezifiziert. Es soll deshalb in diesem Transaktionsmodell gelten, dass eine VCR nach dem Checkin/Checkout-Modell nur dann gelöscht werden kann, wenn sich die VCR im Zustand Checked-out befindet. Dies bedeutet, dass die Ressource mit der Funktion TXN_PES_CHECKOUT gesperrt und einer Transaktion

zugewiesen wurde. Erst dann kann die VCR mit Hilfe der DELETE-Methode gelöscht werden. Dieses Vorgehen wird im Folgenden mit der Funktion TXN_PES_RESOURCE_DEL zusammengefasst.

Ressourcen kopieren – TXN_PES_RESOURCE_COPY

Zum Kopieren von Ressourcen wird die WebDAV-Methode COPY verwendet, deren Verhalten bereits in Abschnitt 5.3 vorgestellt wurde. Bei der Verwendung dieser Methode im pessimistischen Modell existieren ebenfalls die im Abschnitt 5.3.1 vorgestellten Anwendungsfälle UC1 bis UC4. Auch die Vorgehensweise zur Erfüllung der Vorbedingungen VB1 und VB2, die festlegen, dass Quelle und Ziel beide jeweils eine VCR sind, werden entsprechend aus dem optimistischen Modell übernommen.

Sind die Vorbedingungen erfüllt, kann das transaktionssichere Kopieren, im folgenden TXN_PES_RESOURCE_COPY genannt, ausgeführt werden.

Abbildung 5.11 fasst in einem UML 2.0 Sequenzdiagramm die Aktionen, die zur transaktionssicheren Ausführung der COPY-Methode in diesem Modell notwendig sind, zusammen. Dabei entsprechen die Alternativabläufe alt1 bis alt3 der Funktion TXN_PES-_RESOURCE_COPY im optimistischen Fall. Sie dienen auch hier dazu, die Bedingungen VB1 und VB2 zu erfüllen. Anschließend werden die für die pessimistische Nebenläufigkeit typischen LOCK-Methoden mit Hilfe der Funktion TXN_PES_CHECKOUT auf Quell- und Ziel-VCR ausgeführt. Konnte der Client dieses erfolgreich umsetzen, kann er die eigentliche COPY-Methode unter Angabe der originalen Quell- und Ziel-VCRs ausführen.

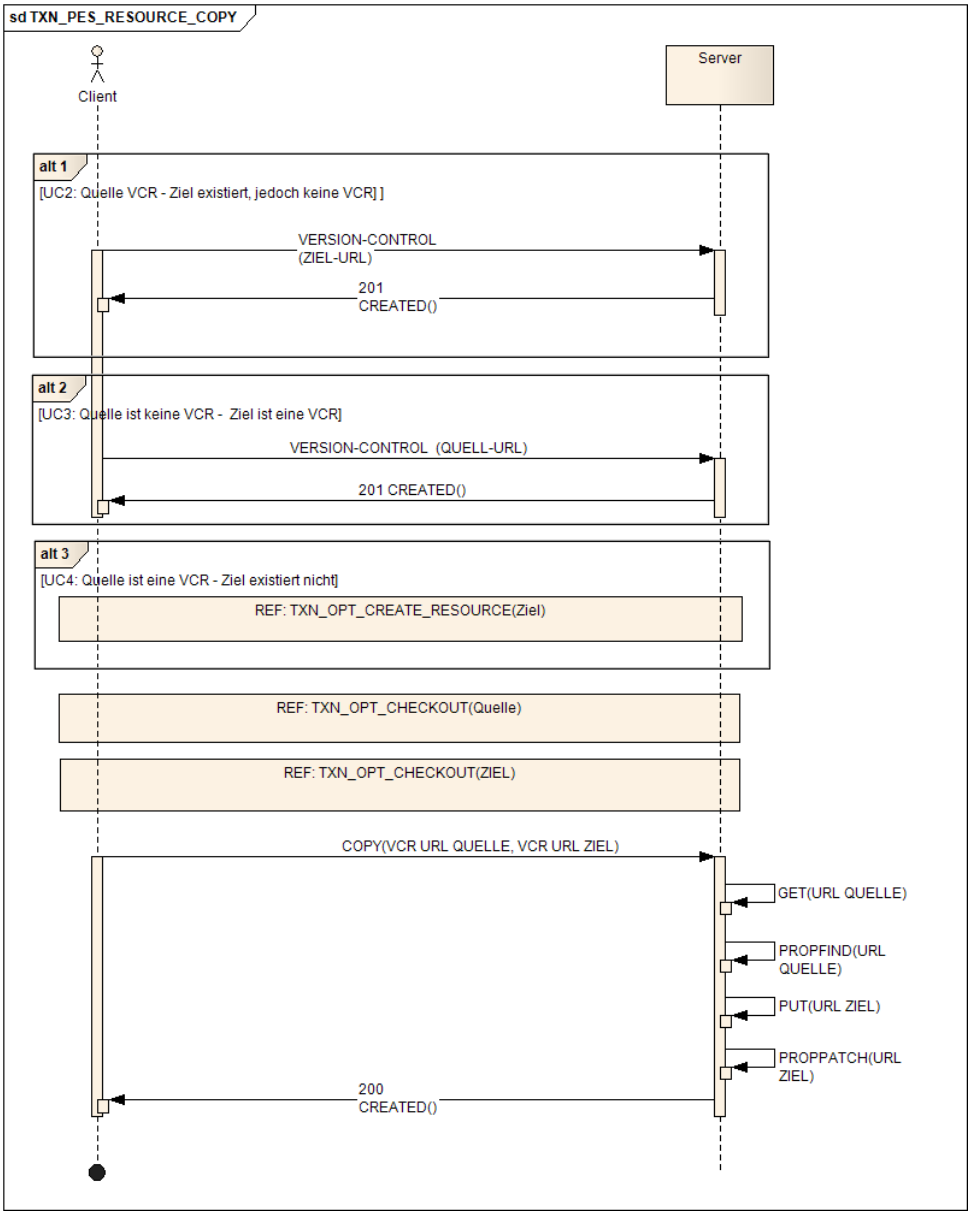

Abbildung 5.11: Ausführung der Funktion TXN_PES_RESOURCE_COPY

Ressourcen verschieben TXN_PES_RESOURCE_MOVE

Zum Verschieben von Ressourcen wird die WebDAV-Methode MOVE verwendet (siehe Abschnitt 3.1). Sie verhält sich identisch zur COPY-Methode, nur dass hier zusätzlich die Quelle-Ressource beim Abschluss des Vorganges gelöscht wird.

Für die Quell- und die Zielressource ergeben sich die gleichen Fallunterscheidungen wie im vorherigen Abschnitt zur Umsetzung der Funktion TXN_OPT_RESOURCE_COPY. O.B.d.A. seien im Folgenden die Bedingungen VB1 und VB2 (Quelle und Ziel sind eine VCR) vorausgesetzt.

Die algorithmische Vorgehensweise für TXN_OPT_RESOURCE_MOVE ist dabei ebenfalls in enger Anlehnung an die Funktion TXN_PES_RESOURCE_COPY realisiert. Der einzige Unterschied besteht darin, dass im letzten Schritt die MOVE-Methode im Gegensatz zur COPY-Methode mit den entsprechenden Ziel- und Quell-URLs der originalen VCR aufgerufen wird.

Abbildung 5.12 fasst hier die Aktionen, die zur transaktionssicheren Ausführung der Funktion TXN_PES_RESOURCE_MOVE in diesem Modell notwendig sind, zusammen. Die Alternativabläufe alt1 bis alt3 entsprechen der Funktion TXN_PES_RESOURCE-_COPY. Sie dienen dazu, die Bedingungen VB1 und VB2 zu erfüllen. Anschließend werden die für die pessimistische Nebenläufigkeit typischen LOCK-Methoden mit Hilfe der Funktion TXN_PES_CHECKOUT auf Quell- und Ziel-VCR ausgeführt. Konnte der Client dieses erfolgreich abschließen, kann er die eigentliche MOVE-Methode unter Angabe der originalen Quell- und Ziel-VCRs ausführen.

Umsetzung der Freigabephase und Beenden der Transaktion – TXN_PES_END

Um die Transaktion abzuschließen, muss der Client die CHECKIN-Methode unter Angabe der URL seiner Activity ausführen und seine Transaktion beenden (siehe 5.2). Ist die CHECKIN-Methode erfolgreich ausgeführt, kann der Client nun mit Hilfe der UNLOCK-Methode die Sperren auf seinen Ressourcen wieder aufheben. Die Funktion TXN_PES-_END umfasst somit zunächst das Aufrufen der Funktion TXN_END, sowie das anschließende Freigeben aller Ressourcen mit Hilfe der UNLOCK-Methode.

Abbildung 5.13 zeigt zusammenfassend die möglichen Zustände, die eine VCR in diesem Modell bei der Verwendung der pessimistischen Nebenläufigkeitskontrolle annehmen kann. Wie in der Abbildung zu sehen, ist das Sperren der Ressource mit Hilfe der LOCK-Methode

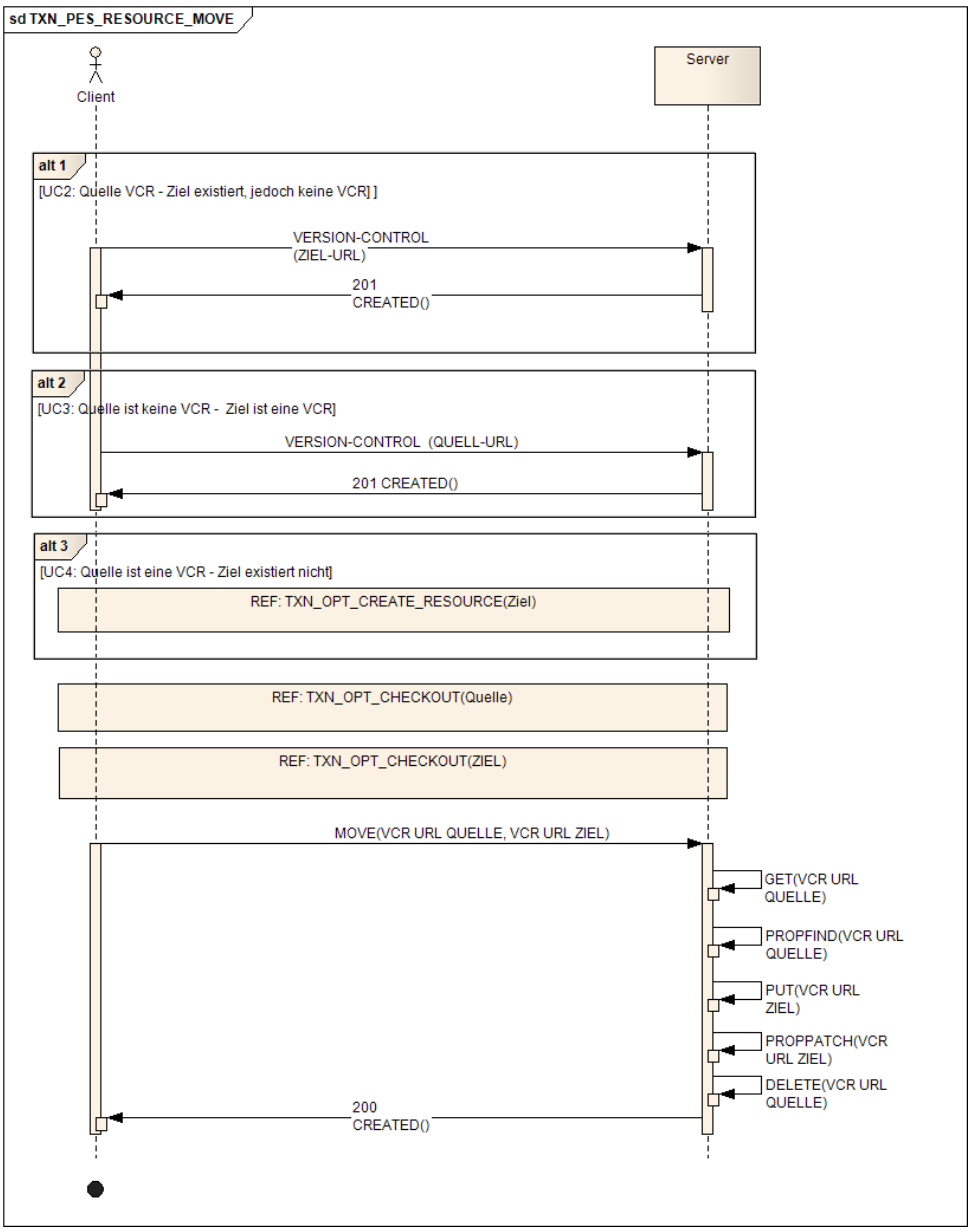

Abbildung 5.12: Ausführung der Funktion TXN_PES_RESOURCE_MOVE

sowie das anschließende Auschecken mit Hilfe der CHECKOUT-Methode keine atomare Einheit. Dies hat zur Folge, dass der Client bei der Ausführung dieser Aktion unterbrochen werden kann und es so zu gesperrten Ressourcen kommt, die keiner Transaktion mehr zugeordnet sind. Zu einer ähnlichen Situation kann es auch beim Einchecken und Freigeben der jeweiligen Ressource kommen. Für diese Aktionen sind auch wieder zwei Methoden, nämlich CHECKIN und UNLOCK, notwendig, wodurch ebenfalls keine atomare Einheit entsteht. In Kapitel 6 wird deshalb ein Vorschlag zur Erweiterung der CHECKOUT- so-

wie der CHECKIN-Methode, vorgestellt, um die jeweiligen Aktionen als atomare Einheit durchführen zu können.

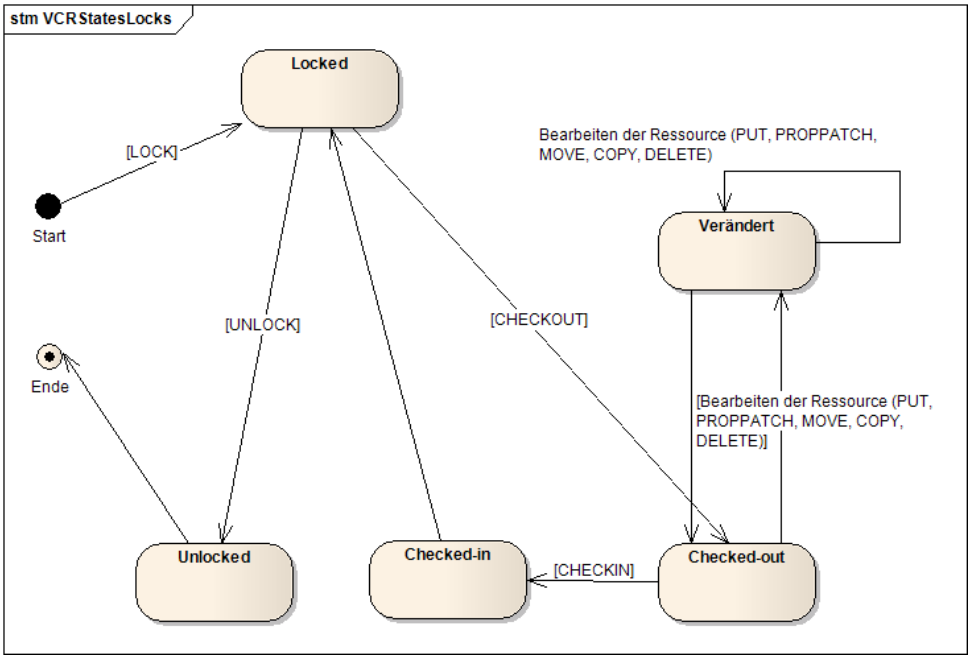

Abbildung 5.13: Mögliche Zustände, die eine VCR einnehmen kann

5.5 Zusammenarbeit mit der IETF bei der Modellentwicklung

Um die Standardkonformität des Modells sicherzustellen und um verschiedene Ansätze mit Fachleuten diskutieren zu können, wurde bei der Entwicklung des Modells eng mit der IETF[10] zusammengearbeitet. Ein weiterer Grund für die enge Zusammenarbeit war, dass Änderungswünsche am bestehenden Standard momentan noch berücksichtigt werden können, da sich die in dieser Arbeit verwendeten Protokolle alle in der noch nicht beendeten Standardisierungsphase befinden.

Wie bereits in Kapitel 2 beschrieben, sind Mailinglisten und die IETF-Meetings die bevorzugten Kommunikationskanäle innerhalb der IETF. Zu Beginn dieser Arbeit wurde deshalb in den einschlägigen Mailinglisten nach bereits bestehenden Ansätzen zur Umsetzung von Transaktionen mit Hilfe des WebDAV-Standards gesucht. Nach dieser Recherche, sowie einer E-Mail-Anfrage an die Mailinglisten des WebDAV-Basis- und des DeltaV-Standards stellte sich heraus, dass noch keine standardkonformen Ansätze realisiert wurden. Es zeigte

[10] Deren Aufgaben und Arbeitsweisen bereits in Kapitel 2.3 vorgestellt wurden

sich aber, dass es seitens der IETF ein großes Interesse an einer Erarbeitung des Themas gibt.

Der Entwurf eines ersten Transaktionsmodells und dessen Abbildung auf den WebDAV-Standard wurde deshalb auf der IETF-Mailingliste vorgestellt und diskutiert. Unter anderem auch aufgrund der Diskussion wurde das Modell weiterentwickelt, bis der in dieser Arbeit vorgestellte letzte Stand schließlich durch die IETF als zulässiges und sinnvolles Modell zur Realisierung von Transaktionen mittels WebDAV bestätigt wurde[11]. Nach Abschluss der Diskussion wurde das Modell schließlich auch auf der 72. IETF-Conference in Dublin, Irland, vorgestellt[12]. Dort wurde der Vorschlag gemacht, die Ergebnisse dieser Arbeit in einem Informational RFC zusammenzufassen und dem bestehenden WebDAV-Standard hinzuzufügen.

Die Modellausprägung zur Umsetzung der pessimistischen Nebenläufigkeitskontrolle wurde erst nach der 72. IETF-Conference entwickelt und bedarf, wie in Kapitel 6 zu sehen, einiger Erweiterungen des bestehenden Standards. Um die damit verbundenen Diskussionen zu bündeln und zu vereinfachen, ist geplant, diese Ausprägung des Transaktionsmodells auf dem nächsten IETF-Meeting im Juli 2009 in Stockholm, Schweden, vorzustellen.

5.6 Bewertung des Transaktionsmodells und Auswahl eines Nebenläufigkeitskonzeptes

In diesem Abschnitt sollen die vorher dargestellten Konzepte zur Umsetzung einer optimistischen bzw. pessimistischen Nebenläufigkeitskontrolle einander kritisch gegenübergestellt und bewertet werden. Nach Fowler kann keine allgemeingültige Aussage darüber getroffen werden, welches Nebenläufigkeitskonzept und somit auch welche Modellausprägung für eine spätere Implementierung die beste Wahl darstellt. Eine optimale Auswahl ist immer von den individuellen Anforderungen und Nebenbedingungen des jeweiligen Projekts abhängig [Fow03]. Deshalb werden in den folgenden Abschnitten die Anforderungen und Entscheidungskriterien aufgezeigt, anhand derer eine entsprechende Auswahl einer Mo-

[11] Anhang A zeigt einen Auszug der Diskussion, die das standardkonform in dieser Arbeit entwickelte Transaktionsmodell wiedergibt. An der Diskussion (siehe auch `http://lists.w3.org/Archives/Public/ietf-dav-versioning/2008AprJun/0003.html`) war unter anderem mit Geoffrey M. Clemm einer der Hauptautoren des aktuellen DeltaV-Standards beteiligt.

[12] `http://www.ietf.org/meetings/72/index.html`

dellausprägung für das in Kapitel 2.2 vorgestellte Datenmanagementsystem DataFinder getroffen wurde.

5.6.1 Umsetzung der funktionalen Anforderungen

Beide Modellausprägungen erfüllen die in Abschnitt 4.1 festgelegten funktionalen Anforderungen. Diese können also keinen Beitrag zur Entscheidungsfindung liefern. Dennoch soll aus Gründen der Vollständigkeit hier die Umsetzung der funktionalen Anforderungen betrachtet werden.

Wie in Abschnitt 5.2 aufgezeigt, erfüllt das Grundmodell, auf dem beide Ausprägungen basieren, die Schnittstellenanforderungen zur Kennzeichnung von Start, Ende und Abbruch einer Transaktion und somit die Anforderungen FA1 bis FA3. Die Anforderung Kommunikations- und Transaktionsfehlern zu erkennen (Anforderung FA15), wird ebenfalls durch das Grundmodell abgedeckt. Auch die eindeutige Zuordnung zwischen Client und Server(Anforderung FA19), stellt das Grundmodell sicher.

Die weiteren funktionalen Anforderungen, die im Folgenden behandelt werden, werden nicht durch das Grundmodell, sondern zum einen durch das Modell der optimistischen Nebenläufigkeitskontrolle (siehe Abschnitt 5.3) und zum anderen durch das Modell der pessimistischen Nebenläufigkeitskontrolle (siehe Abschnitt 5.4) erfüllt.

Die Schnittstellen zum Zugriff auf den Datenbestand sind Bestandteil der Anforderungen FA4 bis FA14, deren Umsetzung in Abschnitt 5.3 und 5.4 vorgestellt wurde. In diesen Abschnitten wurde entwickelt, wie die in dieser Arbeit betrachteten ACID-Eigenschaften Atomarität (FA15) und Isolation (FA16) umgesetzt werden können. Es wurde aufgezeigt, wie sichergestellt werden kann, dass verloren gegangene (Anforderung FA16.1) und/oder nicht freigegebene Änderungen (Anforderung FA16.2) erkannt bzw. verhindert werden können.

5.6.2 Umsetzung der nicht-funktionalen Anforderungen

In diesem Abschnitt soll die individuelle Umsetzung der nicht-funktionalen Anforderungen durch die beiden Modellausprägungen betrachtet und bewertet werden. Die Anforderung NFA1 zur standardkonformen Abbildung des Transaktionsmodells stellt ein wichtiges Ziel dieser Arbeit dar. Hieraus resultiert die - von Begin an gesuchte - enge Zusammenarbeit mit der IETF. Wie in Abschnitt 5.4, sowie zusammengefasst in Kapitel 6, zu sehen, kann die pessimistische Nebenläufigkeitskontrolle nicht ohne eine Erweiterung des Standards ver-

wendet werden. Die optimistische Nebenläufigkeitskontrolle hingegen ist, bis auf eine geringe Einschränkung beim Anlegen von Ressourcen (zur Behebung siehe Erweiterungsvorschlag in Kapitel 6), vollständig konform mit dem zurzeit bestehenden WebDAV-Standard möglich. Die Einhaltung von Standards stellt ein wichtiges Kriterium zur Interoperabilität und Konnektivität mit anderen Softwaresystemen dar. Insbesondere im Web-Umfeld spielt dieser Aspekt eine große Rolle. Aber auch andere Dienste, mit denen Informationen, insbesondere in heterogenen Umgebungen, ausgetauscht werden, hängen davon ab, dass Daten und Prozesse standardkonform sind. Durch die Einhaltung werden Verarbeitungs- und Übertragungsfehler verringert, welche eine Zusammenarbeit sonst stören könnten. Weiterhin reduziert die Einhaltung und Nutzung von standardisierten Protokollen und Formaten die Fehleranfälligkeit und erleichtert die Fehlersuche. Im WebDAV-Bereich haben sich zwei Open-Source-Testprodukte etabliert, welche die Leistungsfähigkeit und Standardkonformität von WebDAV-Systemen untersuchen. Die Software Litmus WebDAV Server Protocol Compliance Test Suite[13] ist darauf spezialisiert, die Konformität eines Servers, bezogen auf den WebDAV-Standard, zu ermitteln. Es wird dabei die Konformität nach RFC 2518 (Standard WebDAV) und die nach RFC 3744 (Access Control Protocol) überprüft. Die Software Prestan[14] dagegen wird mit dem Ziel eingesetzt, die Geschwindigkeit einzelner WebDAV-Servermethoden zu ermitteln. Zurzeit unterstützt das System nur die Methoden des WebDAV-Basisstandards aus RFC 2518. Mit diesen beiden Werkzeugen wird unter anderem der Verbreitung des WebDAV-Standards in der Praxis Vorschub geleistet. Nur ein von Implementierungen eingehaltener Standard kann Interoperabilität garantieren. Das zweite Programm zielt auf den Praxiseinsatz ab. Performance-Kennzahlen sind für die Entwicklung unerlässlich, um im späteren Produktiveinsatz auch unter hohen Lastbedingungen ein akzeptables Antwortverhalten zu erreichen.

Der Test des im Rahmen dieser Arbeit entwickelten Prototypen kann noch nicht durch diese Produkte überprüft werden, da die Unterstützung der DeltaV-Erweiterung noch nicht gegeben ist. Da es sich aber jeweils um Open-Source-Produkte handelt, ist eine Erweiterung durchaus denkbar. Im Rahmen des 72. IETF-Meetings wurde eine entsprechende Anregung an die zuständige Community gegeben. Damit ließe sich die Erfüllung von NFA1 unabhängig überprüfen.

[13] http://www.webdav.org/neon/litmus/
[14] http://sourceforge.net/projects/prestan/

NFA2 fordert einen möglichst geringen Kommunikationsaufwand. Durch das Fehlen der Lock-Anforderungen bei der optimistischen Nebenläufigkeitskontrolle wird diese Anforderung besser umgesetzt. Bei der pessimistischen Nebenläufigkeitskontrolle wird dagegen mit Locks gearbeitet. Das Warten auf die entsprechende Freigabe der Locks kann mit Hilfe von WebDAV nur über so genanntes Polling gelöst werden, d.h. der Client muss immer wieder beim Server anfragen, ob die Ressource nun verfügbar ist. Dadurch entsteht, im Vergleich zur optimistischen Nebenläufigkeitskontrolle, ein sehr viel höherer Kommunikationsaufwand.

Lange Transaktionen werden durch die optimistische Nebenläufigkeitskontrolle nur dann effektiv unterstützt (NFA3), wenn Konflikte eher selten auftreten. Im folgenden Abschnitt werden die entsprechenden Konsequenzen, falls Konflikte doch auftreten, vorgestellt.

5.6.3 Konfliktverhalten

Nach Fowler stellen das Konfliktverhalten sowie der im nachfolgenden Abschnitt diskutierte Implementierungsaufwand, die Hauptkriterien zur Auswahl eines Nebenläufigkeitskonzeptes dar [Fow03]. Bei der optimistischen Nebenläufigkeitskontrolle liegt der Kern auf der Konflikterkennung, während er bei der pessimistischen auf der Konfliktvermeidung liegt. Wie in Abschnitt 5.3 zu sehen, steht der optimistischen Nebenläufigkeitskontrolle als einziges Mittel zur Synchronisation das Zurücksetzen der gesamten konfliktverursachenden Transaktion zur Verfügung. Die pessimistische Nebenläufigkeitskontrolle hingegen lässt in den meisten Fällen die konfliktverursachende Transaktion mit entsprechenden Sperren warten und setzt nur in Ausnahmefällen eine Transaktion zurück. Die optimistische Nebenläufigkeitskontrolle erlaubt somit eine maximale Parallelverarbeitung, weil kein Prozess je auf eine Sperre warten muss [TvS03]. Jedoch ist dieser Gewinn nur von Vorteil, wenn zum einen wenig konfliktverursachende Transaktionen auftreten und zum anderen die Kosten für das Zurücksetzten und das Wiederaufnehmen der fehlgeschlagenen Transaktion nicht zu hoch sind.

5.6.4 Implementierungsaufwand

Nach Fowler gehören die Aufgaben zur Lösung von Nebenläufigkeitsproblemen zu den komplexesten Programmieraufgaben überhaupt. Weil es zum einen sehr schwierig ist, die Programme zuverlässig zu testen und zum anderen sind im Betrieb Fehler oft schwer zu reproduzieren [Fow03].

Fowler führt an, dass eine optimistische Nebenläufigkeitskontrolle einfacher zu implementieren ist, als eine pessimistische [Fow03]. Insbesondere können bei einem optimistischen Verfahren keine Deadlocks[15] auftreten. Zur Lösung des Deadlock-Problems gibt es eine Vielzahl von Ansätzen. Für eine detaillierte Vorstellung sei auf [Sch03] verwiesen.

Ein weiteres Problem, welches durch die Sperren der pessimistischen Nebenläufigkeitskontrolle herbeigeführt wird, ist das Problem der verwaisten Sperren. Dies sind Sperren, die der Client-Rechner, etwa durch einen Absturz, nicht aufgehoben hat. Zur Lösung dieses Problems gibt es noch keinen allgemeingültigen Lösungsweg. Aus diesem Grund sieht der WebDAV-Standard vor, das Sperren bei Bedarf durch den Server entfernt werden können[TvS08]. Hierzu muss natürlich auf dem Server ein entsprechender Algorithmus mit passenden Erkennungskriterien implementiert sein.

Das nächste Problem, welches sowohl die optimistische als auch die pessimistische Nebenläufigkeitskontrolle betrifft, ist das Problem der so genannten Livelocks oder auch Starvations genannt. Im Gegensatz zu den Deadlocks wird mit Livelocks ein Zustand beschrieben, bei dem Transaktionen zwar aktiv sind und versuchen, Operationen auszuführen, jedoch nicht erfolgreich zu Ende kommen. Dieses Problem tritt auf, wenn beispielsweise Sperren unfair zugeteilt werden (im pessimistischen Fall) oder immer wieder die gleiche Transaktion abgebrochen wird (im optimistischen Fall). Livelocks sind meistens unangenehmer als Deadlocks, da sie zum einen schwer zu entdecken sind und sich zum anderen negativ auf den Systemdurchsatz auswirken [WV02].

Es sei hier noch erwähnt, dass die optimistische Nebenläufigkeitskontrolle zwar ohne einen expliziten Locking-Mechanismus auskommt, jedoch in der nach [WV02] so genannten "critical section", welche die Validierungs- und Schreibphase umfasst, je nach Umsetzung ein erhöhter Verwaltungs- und somit auch Implementierungsaufwand entstehen kann. Des Weiteren bedeutet die Erstellung der Arbeitskopien einen nicht zu vernachlässigenden Aufwand. In [Unl94] werden verschiedene Algorithmen zur Umsetzung vorgestellt und bewertet.

Die gerade vorgestellten Probleme stellen nur einen Ausschnitt der Schwierigkeiten dar, die bei einer konkreten Implementierung gelöst werden müssen. Das folgende Zitat aus Fowler fasst das Ganze sehr gut zusammen:

[15] Nach [CDK05] ist ein Deadlock ein Zustand, in dem jedes Mitglied einer Gruppe von Transaktionen darauf wartet, dass ein anderes Mitglied eine Sperre aufhebt, welches aber selbst wiederum auf eine Sperre wartet. Die Mitglieder der Gruppe von Transaktionen halten sich also gegenseitig auf.

"Keiner der beiden Ansätze ist vollkommen frei von Problemen, tatsächlich können sie durch diese Techniken leicht Probleme erzeugen, die fast so schwer sind, wie die grundlegenden Nebenläufigkeitsprobleme, die sie eigentlich zu lösen versuchen." [Fow03]

5.6.5 Abschließende Auswahl eines Nebenläufigkeitskonzeptes

Anhand der in den vorherigen Abschnitten diskutierten Kriterien und möglichen Probleme und Aufwände soll nun eine passende Modellausprägung für eine erste prototypische Implementierung ausgewählt werden.

Anhand der funktionalen Anforderungen lassen sich keine passenden Entscheidungen treffen, da sie, wie bereits erwähnt, von beiden Modellausprägungen vollständig umgesetzt werden. Bei den nicht-funktionalen Anforderungen hat die optimistische Nebenläufigkeitskontrolle jedoch in allen Bereichen klare Vorteile. Insbesondere die Einhaltung der Standards stellt ein starkes Argument zur Auswahl der optimistischen Nebenläufigkeitskontrolle dar.

Nach [Fow03] sollte im Hinblick auf den Implementierungsaufwand nach Möglichkeit ebenfalls der optimistischen Nebenläufigkeitskontrolle der Vorzug gegeben werden, wenn es die Rahmenbedingungen zulassen. Da im Projekt DataFinder (siehe Abschnitt 2.2) von einem geringen Konfliktaufkommen ausgegangen und somit der Hauptnachteil der optimistischen Nebenläufigkeitskontrolle entkräftet wird, ist dies gegeben.

Letztendlich haben insbesondere die Punkte Implementierungsaufwand, Konfliktverhalten, sowie die Standardkonformität zur Entscheidung für einen Prototypen auf Basis der optimistischen Nebenläufigkeitskontrolle geführt. Kapitel 7 stellt diesen vor.

6 Erweiterungsvorschläge für den bestehenden WebDAV-Standard

In diesem Kapitel werden die, in den vorangegangenen Kapiteln erarbeiteten und aufgezeigten, Erweiterungsvorschläge für den WebDAV- bzw. den DeltaV-Standard zusammengefasst. Da sich, wie bereits in Kapitel 3.1 beschrieben, die beiden Protokolle im Standardisierungsprozess befinden, besteht hier noch die Möglichkeit der Einflussnahme. Es werden sowohl die Erweiterungsvorschläge bzgl. des Grundmodells, die über die bereits mit der IETF abgestimmten Maßnahmen hinaus gehen, als auch die Konzepte zur Erweiterung des Standards bzgl. der pessimistischen Nebenläufigkeitskontrolle zusammengefasst.

Die einzelnen Vorschläge werden, um sie referenzierbar zu machen, mit einem eindeutigen Bezeichner (dem Kürzel "VO" für Vorschlag plus einer laufenden Nummer) versehen. Des Weiteren werden sie gewichtet, um zu verdeutlichen, welche Relevanz der jeweilige Erweiterungsvorschlag nach Meinung des Verfassers für die Umsetzung des Modells und seiner entsprechenden Ausprägung hat. VO bedeutet dabei, dass die Erweiterung absolut notwendig ist, um die Funktionalität zu ermöglichen. VO* drückt aus, dass die Erweiterung als optional anzusehen ist. Der Aufbau der Vorschläge orientiert sich dabei an den bestehenden RFCs des WebDAV-Standards, bei denen, neben einer Beschreibung des entsprechenden Konzeptes oder der Methode, nach Bedarf auch noch Vor- bzw. Nachbedingungen (im RFC pre- bzw. postconditions genannt) definiert werden.

6.1 Erweiterungen für das Grundmodell

VO1*: Löschen einer Activity nach einem Checkin

Wie bereits in Abschnitt 5.2.1 beschrieben, wäre es wünschenswert, dass eine Activity-Ressource nach einem erfolgreichen Einchecken bzw. nach dem Abschluss der Transaktion automatisch durch den Server gelöscht wird. Bei dem bisherigen Ansatz muss die Ressource

nach erfolgreichem Abschluss der Transaktion durch den Benutzer mit Hilfe der DELETE-Methode explizit entfernt werden. Dies hat zur Folge, dass Einchecken und Löschen der Activity nicht als atomare Operation durchgeführt werden können. Um dies zu erreichen, wäre eine Erweiterung der CHECKIN-Methode von Vorteil, die im Folgenden vorgestellt wird.

- **Vorbedingung:** Es muss eine Activity-Ressource eingecheckt werden.
- **Ausführung:** Als Option wird der CHECKIN-Methode der Parameter "DAV:delete-activity-on-success" mitgegeben
- **Nachbedingung:** Nach erfolgreichem Einchecken der Activity-Ressource soll diese gelöscht und dem Client mit einer Antwort "200 OK" signalisiert werden, dass die Verarbeitung erfolgreich abgeschlossen wurde. Sollte es zu einem Fehler beim Einchecken der Activity-Ressource kommen, soll die entsprechende Activity nicht gelöscht und dem Client mit einem entsprechenden Fehlercode signalisiert werden, dass die Verarbeitung fehlgeschlagen ist.

VO2*: Eindeutige Activities durch den Server vergeben

Mit diesem Vorschlag soll die MKACTIVITY-Methode zum Anlegen einer Activity so erweitert werden, dass der Server dem Client automatisch eine freie und eindeutige Activity-Ressource anlegt. So kann für den Client Kommunikations- und Berechnungsaufwand eingespart werden. Die Anfrage des Clients an den Server, an welchem Speicherplatz mit Hilfe der OPTIONS-Methode Activities angelegt werden dürfen und die Berechnung der eindeutigen UUID durch den Client können entfallen.

- **Ausführung:** Mit Hilfe der Option "DAV:auto-create" wird der Server angewiesen, automatisch eine eindeutige Activity-Ressource anzulegen.
- **Nachbedingung:** Nach erfolgreichem Anlegen einer eindeutigen Activity-Ressource wird dem Client in der Antwort die URL der entsprechenden Ressource mitgeteilt.

VO3*: Transaktionssicheres Anlegen einer Ressource

Um den in Abschnitt 5.2.4 beschriebenen Workaround zum transaktionssicheren Anlegen von Ressourcen einsparen zu können, soll im Folgenden die Methode VERSION-CONTROL erweitert werden.

- **Vorbedingung:** Es muss eine Activity-Ressource angelegt sein und die zu erzeugende VCR darf noch nicht existieren.

- **Ausführung:** Die Methode VERSON-CONTROL wird um die Option "DAV:add-to-activity" erweitert. Dieser Option wird als Parameter die gültige URL einer entsprechenden Activity-Ressource übergeben. Der Server wird mit dieser neuen Option angewiesen, sofern die entsprechende VCR noch nicht existiert, eine leere VCR anzulegen, diese in den Zustand Checked-out zu setzen und sie der angegebenen Activity-Ressource hinzuzufügen. Das Attribut "DAV:checked-in" soll dabei keine URL einer VR enthalten. Dies soll dem Server signalisieren, dass die Ressource erst nach erfolgreichem Abschluss der Transaktion, nachdem sie entsprechend initialisiert und bearbeitet wurde, für andere Transaktionen bzw. Nutzer auf dem Server sichtbar und verfügbar sein soll. Wird die VCR eingecheckt, wird eine neue Version erstellt, die nun für alle sichtbar und verwendbar ist. Wird auf der VCR jedoch die Methode UNCHECKOUT aufgerufen, wird diese entsprechend vom Server wieder gelöscht.

VO4*: Definition der DELETE-Methode zum Löschen einer VCR

Wie in Abschnitt 5.3.1 beschrieben, ist eine exaktere Definition der DELETE-Methode zum Löschen einer VCR nötig. Diese Definition soll dann das für dieses Modell geforderte Verhalten gewährleisten, so dass eine VCR nur gelöscht werden kann, wenn sie sich im Zustand Checked-out befindet.

Wenn nun die DELETE-Methode auf einer VCR aufgerufen wird, muss sich der Server dies merken, um ggf. zur Sicherstellung der Konsistenz keine weiteren Zugriffe mehr auf diese zu erlauben. Wird die VCR wieder eingecheckt, wird sie durch den Server gelöscht.

6.2 Erweiterungen zur Umsetzung der pessimistischen Nebenläufigkeitskontrolle

VO5: Abfrage des Sperrprotokolls

Die pessimistische Nebenläufigkeitskontrolle arbeitet zur Sicherstellung der ACID-Eigenschaften mit so genannten Sperrprotokollen (siehe Abschnitt 5.4). Damit der Client vom Server erfahren kann, welches Sperrprotokoll unterstützt wird, soll die OPTIONS-Methode um die Abfrage des Sperrprotokolls mit Hilfe der Option "DAV:lock-protocol" erweitert

werden. In der Antwort des Servers erfährt der Client damit, welches Sperrprotokoll unterstützt wird. Um das Sperrprotokoll eindeutig zu kennzeichnen, müssten die unterstützten Sperrprotokolle auch wieder in einem RFC festgehalten werden. So wäre etwa für das Zwei-Phasen-Sperrprotokoll die Bezeichnung "2PL", sowie für binäres Sperren "BINL" möglich.

VO6*: Einführung von Readlocks

Um weitere Sperrprotokolle sowie eine bessere Unterscheidung zwischen gelesenen und geschriebenen Datenelementen zu ermöglichen, sollen neben den bereits vorhandenen Writelocks, die ein Schreiben der Ressource verhindern, nun auch Readlocks eingeführt werden, die ein Lesen und Schreiben der Ressource verhindern, wenn der Benutzer nicht den dazugehörigen Lock-Token besitzt. Um dies umzusetzen, soll die Option "DAV:lock-type" der LOCK-Methode um den Typ Read erweitert werden.

VO7*: CHECKOUT und LOCK als atomare Einheit

Um das Auschecken und Sperren einer VCR gemeinsam als atomare Einheit durchzuführen, soll die CHECKOUT-Methode um eine entsprechende Option erweitert werden. Diese Option soll es ermöglichen, bereits mit Hilfe der CHECKOUT-Methode die Ressource zu sperren. Dazu kann der CHECKOUT-Methode die Option "DAV:lock-type" hinzugefügt werden, die signalisieren soll, dass die entsprechende VCR gesperrt werden soll. Der Wert, den diese Option hat, signalisiert dem Server dann, ob die Ressource mit einem Read- oder Writelock versehen werden soll.

VO8*: CHECKIN und UNLOCK als atomare Einheit

Um das Einchecken und Aufheben der Sperre einer VCR ebenfalls gemeinsam als atomare Einheit durchzuführen, soll die CHECKIN-Methode erweitert werden. Mit Hilfe der Option "DAV:free-locks" soll dem Server signalisiert werden, dass dieser, nach einem erfolgreichen Einchecken der Ressource, diese freigeben und die entsprechenden Locks entfernen soll.

7 Prototypische Implementierung auf Basis der optimistischen Nebenläufigkeitskontrolle

Dieses Kapitel beschreibt die technische Realisierung des zuvor entwickelten Transaktionsmodells mit optimistischer Nebenläufigkeitskontrolle. Dazu werden zunächst in den folgenden beiden Abschnitten die Erweitungen von zwei bestehenden WebDAV-Implementierungen vorgestellt, auf deren Basis die eigentliche prototypische Implementierung des Modells durchgeführt wird. Zum Abschluss dieses Kapitels wird auf die durchgeführten Tests eingegangen.

Wie [KPSW04] beschreibt, existiert zurzeit keine Referenzimplementierung des DeltaV-Standards. Aus diesem Grund ist für die prototypische Umsetzung eigener Entwicklungsaufwand nötig. Im Rahmen dieser Arbeit wurden dazu die Open-Source-Projekte Apache Jackrabbit (in der Version 1.5.2) auf Client-Seite, sowie Subversion (in der Version 1.5.5) auf Server-Seite zunächst um die fehlenden DeltaV-Konzepte standardkonform erweitert.

Auf Basis dieser angepassten Software-Bibliotheken ist es dann im nächsten Schritt möglich, das im vorherigen Abschnitt theoretisch entwickelte Transaktionsmodell auch praktisch zu implementieren. Es wurden dabei bewusst zwei Produkte unterschiedlicher Hersteller ausgewählt. Die IETF fordert, dass es mindestens zwei unabhängige und interoperable Implementierungen eines Protokolls geben muss, damit es akzeptiert wird.

Subversion (siehe Abschnitt 3.5.2) wurde dabei auf Empfehlung von Geoffry Glemm, einem der Autoren des DeltaV-Standards, ausgewählt (siehe Anlage B), da Subversion bereits einen Teil der Konzepte und Datenobjekte unterstützt, die im Transaktionsmodell Verwendung finden. Apache Jackrabbit, welches im nächsten Abschnitt noch detaillierter vorgestellt wird, wurde u.a. auf Empfehlung von Julian Reschke ausgewählt, der ebenfalls zu den WebDAV-Autoren gehört und aktiv am Apache-Jackrabbit-Projekt mitarbeitet.

Die beiden Implementierungen müssen neben dem WebDAV-Basisstandard, der von beiden Programmen vollständig unterstützt wird, zur Umsetzung des Transaktionsmodells noch die folgenden DeltaV-Konzepte unterstützen, die teilweise noch standardkonform erarbeitet bzw. erweitert werden mussten:

- VCR- und VR-Ressourcen
- Activity-Ressourcen
- Working-Ressourcen
- CHECKIN- und CHECKOUT-Methode

7.1 Clientseitige Erweiterung des Apache-Jackrabbit-Projekt

Um die notwendigen Fähigkeiten zur Transaktionsunterstützung auf Client-Seite zu erhalten, wurde Apache Jackrabbit[1](programmiert in Java) entsprechend erweitert. Jackrabbit ist eine Implementierung des JCR-Standards (Java Content Repository API, dokumentiert im Java Specification Request, JSR 170[2]). Ziel von Jackrabbit bzw. des JCR-Standards allgemein ist es, Datenquellen eines Dokumentenmanagementsystems unter einer einheitlichen API, aber über verschiedene Netzwerkprotokolle, unter anderem WebDAV, anzusprechen. So soll ein möglichst einfacher Zugang zur Verwaltung, Ablage und Bearbeitung von Inhalten auf einem Server bereitgestellt werden.

Der entsprechende WebDAV-Clientzugriff auf den Server wird durch das Bibliothekspaket "org.apache.jackrabbit.webdav.client.methods" zur Verfügung gestellt. In diesem Paket sind die einzelnen WebDAV-Methoden als Klassen realisiert. Die entsprechende Parametrisierung der einzelnen Methoden, z.B. mit den Angaben zur URL, wird dabei über den Konstruktor der jeweiligen Klasse vorgenommen.

Wie in der zum Paket gehörenden Todo-Liste, in der wünschenswerte Erweiterungen aufgeführt sind, zu sehen ist, fehlt es jedoch der Implementierung der CHECKOUT-Methode an der Möglichkeit, Optionen zu setzen. Aus diesem Grund wurde in der vorliegenden Arbeit die Klasse CheckoutMethode erweitert. Dazu wurden, nach den allgemeinen Projektrichtlinien des Jackrabbit-Projekts, der Klasse CheckoutMethode weitere Konstruktoren hinzugefügt, über die die entsprechenden Optionen gesetzt werden können. Diese Optionen

[1] jackrabbit.apache.org

[2] http://jcp.org/en/jsr/detail?id=170

werden weiterverarbeitet und in Form eines Requests an den Server weitergeleitet. Über die neu erstellten Konstruktoren können nun neben der URL der Ressource zusätzliche Angaben darüber gemacht werden, ob die Ressource einer angegebenen Activity hinzugefügt und/oder ob eine Working-Ressource-Kopie erstellt werden soll.

Neben der Erweiterung der CHECKOUT-Methode, welche dem Projekt[3] zur Verfügung gestellt wurde, wurde auch die Dokumentation des Benutzerhandbuchs aktualisiert[4]. Dort wurde eine Einführung zum Umgang mit der Software-Bibliothek, bezogen auf WebDAV, unter dem Titel „Using the WebDAV-Client API linked with standard WebDAV-Servers“ verfasst.

7.2 Serverseitige Erweiterung des Subversion-Projekts

Auf Server-Seite wurde das Versionsmanagementsystem Subversion, entwickelt in der Programmiersprache C, um die für diese Arbeit benötigten Transaktionskonzepte erweitert. Subversion unterstützt viele aber nicht alle in der Einleitung zu diesem Kapitel genannten DeltaV-Konzepte.

Bei den ersten Tests der Funktionalitäten stellte sich heraus, dass Subversion den Checkin-Vorgang von Activities nicht standardkonform unterstützt. So werden hier die Activities nicht mit der dafür vorgesehenen Methode CHECKIN, sondern mit der Methode MERGE eingecheckt. Im Paket "/mod_dav_svn" wurden die Dateien "activity.c" sowie "mod_dav_svn.c" angepasst, um das Einchecken von Activities mit Hilfe der CHECKIN-Methode zu ermöglichen.

Eine weitere fehlende Funktionalität war das Auschecken von VCRs. Subversion unterstützt zurzeit nur das Auschecken von VRs. Da eine Analyse des Datenmodells von Subversion gezeigt hat, dass der benötigte Zeitaufwand für eine entsprechende Erweiterung den Rahmen dieser Arbeit sprengen würde, wurde ein Workaround für die prototypische Implementierung gewählt. Dazu wurde die fehlende Funktionalität clientseitig durch eine Erweiterung der Funktionen TXN_OPT_CHECKOUT umgesetzt. Die Funktion wurde dahingehend erweitert, dass bevor die CHECKOUT-Methode ausgeführt wird, zunächst mit Hilfe der PROPFIND-Methode, unter Angabe der URL der VCR, die Live-Properties "DAV:checked-out" bzw. "DAV:checked-in" abgefragt werden, um die aktuellste VR zu er-

[3] unter `https://issues.apache.org/jira/browse/JCR-1732`
[4] unter http://wiki.apache.org/jackrabbit/WebDAV

mitteln (siehe Abschnitt 3.2.2). Anschließend kann diese VR dann mit Hilfe der Funktion TXN_OPT_CHECKOUT ausgecheckt und einer Activity hinzugefügt werden.

7.3 Implementierung des Transaktionsmodells mit optimistischer Nebenläufigkeitskontrolle

Auf Basis der in den vorangegangenen Abschnitten erweiterten Software-Bibliotheken wurde eine prototypische Implementierung des Transaktionsmodells mit optimistischer Nebenläufigkeitskontrolle erstellt. Ziel der technischen Realisierung ist es, die Machbarkeit der entwickelten Konzepte zu zeigen, also einen so genannten Proof-of-Concept zu erbringen.

Abbildung 7.1 stellt dazu die wichtigsten in dieser Arbeit entwickelten Java-Klassen als UML 2.0 Klassendiagramm, mit einer Beschränkung auf die für das Transaktionsmodell wesentlichen Methoden und Variablen, dar.

Eine Instanz der Klasse "DAVConnection" stellt dem Benutzer die Verbindung zum WebDAV-Server zur Verfügung. Dazu muss der Nutzer bei der Instanziierung der Klasse die notwendigen Angaben über die URL des Servers, den Benutzernamen, sowie ggf. das Passwort übergeben. Wurde die Klasse erfolgreich instanziiert, kann über die Methode „getTransaction“ ein Objekt der Klasse „DAVTransaction“ bezogen werden. Diese Klasse stellt dem Benutzer die Funktionen des Transaktionsmodells, wie sie in Abschnitt 5.3 definiert wurden, zur Verfügung. Mit Hilfe der Methode "txn_start" kann eine Transaktion gestartet werden. Nachdem dies erfolgt ist, können über die weiteren Methoden Zugriffe und Veränderungen des Datenbestandes durchgeführt werden. Die Klasse TXNResource repräsentiert dabei eine Ressource mit ihrer eindeutigen URL, den jeweiligen Properties, die mit ihren Schlüssel-Wert-Paaren über eine HashMap abgebildet werden, sowie dem eigentlichen Dateiinhalt.

Die Klasse TXNStatus schließlich kapselt die Eigenschaften einer entsprechenden Server-Antwort. Mit Hilfe dieser Klasse kann der Benutzer den Statuscode, die Kurzbeschreibung des Statuscodes, sowie die komplette Antwort des Servers abfragen.

Um nun z.B. eine Ressource transaktionssicher zu lesen, kann der Benutzer, nachdem er mit Hilfe der Methode "txn_start" die Transaktion gestartet hat, mit Hilfe der Methode "txn_opt_read" diese vom Server einlesen. Als Antwort erhält er eine Instanz der Klasse TXNResource. Auf ihr kann der Benutzer dann die Änderungen durchführen, in dem er die entsprechenden Getter- und Setter-Methoden aufruft. Über die Methode "txn_save",

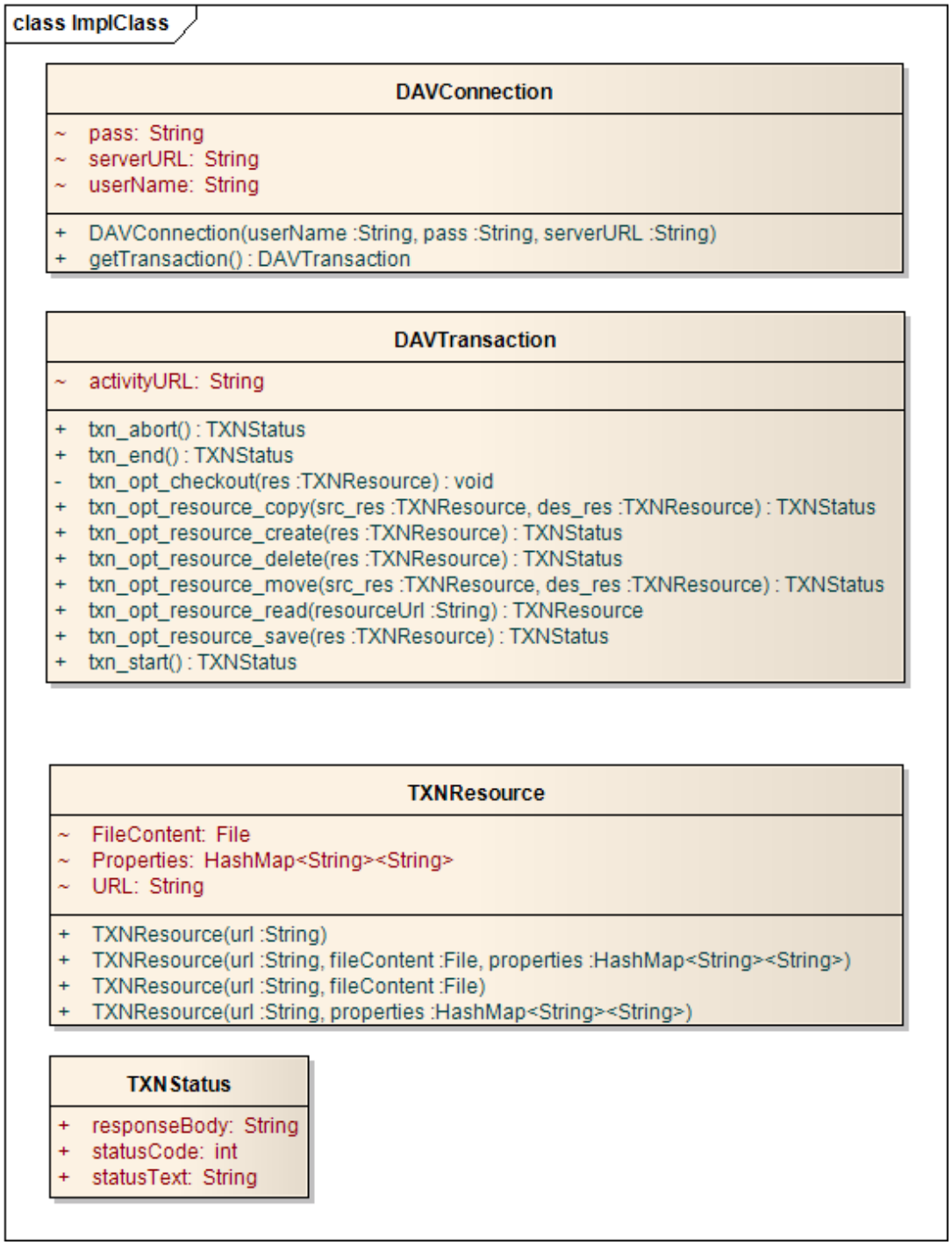

Abbildung 7.1: Klassendiagramm der prototypischen Implementierung

der die Ressource übergeben wird, werden die Änderungen temporär gespeichert[5]. Zum Abschließen der Transaktion muss der Benutzer die Methode "txn_end" aufrufen. Der Server überprüft dazu, ob die durchgeführten Änderungen konfliktfrei verarbeitet werden

[5]In der entsprechenden Working-Ressource-Kopie auf dem Server.

können. Ist dies der Fall, werden sie dauerhaft gespeichert und stehen nun allen Benutzern zur Verfügung.

7.4 Test des Prototypen

In diesem Abschnitt werden zum Abschluss des Kapitels zunächst die Testumgebung und anschließend die Vorgehensweise bei der Durchführung der Tests beschrieben.

7.4.1 Testumgebung

Zur Ausführung der Tests wurde folgende Testumgebung aufgebaut. Auf Serverseite wurde, basierend auf der Linux-Distribution Debian[6] (Version 4.0) der Apache Web-Server[7] (Version 2.2.11) installiert. Die im Rahmen dieser Arbeit angepasste Version von Subversion wurde als Modul in den Apache Web-Server eingebunden.

Auf Client-Seite wurde die in dieser Arbeit angepasste WebDAV-Bibliothek des Jackrabbit-Projekts verwendet. Die Tests wurden jeweils auf einem PC mit Windows XP, auf dem auch mit Hilfe der Entwicklungsumgebung Eclipse[8] die Software implementiert wurde, sowie auf einem PC mit der Linux-Distribution Ubuntu[9] (Version 8.04) als Betriebssystem, durchgeführt. Auf beiden Rechnern stand ein Java Runtime Environment[10] in der Version 1.6.0_05 zur Verfügung.

7.4.2 Testdurchführung

Die einzelnen Funktionen des Grundmodells und der optimistischen Ausprägung der Nebenläufigkeitskontrolle, sowie die jeweiligen Erweiterungen von Subversion und Jackrabbit wurden bereits während der Entwicklung intensiv getestet. Dazu wurden so genannte Unit-Tests, auch Komponententests oder Modultests genannt, entwickelt. Diese wurden dann in der im vorherigen Abschnitt beschriebenen Testumgebung ausgeführt.

Unit-Tests stellen eine Möglichkeit dar, eine automatisierte Testumgebung zu definieren und jede Änderung am Quelltext daraufhin zu testen, ob die Funktionalität der einzelnen Komponenten (Units) weiterhin gegeben ist. Das Test-Framework JUnit[11] wur-

[6] http://www.debian.org/
[7] http://httpd.apache.org/
[8] http://www.eclipse.org/
[9] http://www.ubuntu.com/
[10] http://www.java.sun.com/
[11] http://www.junit.org/

de speziell für die Umsetzung von Unit-Tests auf Basis der Programmiersprache Java entwickelt[Lin05].

In den Unit-Tests wurden zunächst die einzelnen Funktionen getestet, die in einer entsprechenden Testklasse zusammengefasst worden sind. Dazu wurde insbesondere das Verhalten der Klasse DAVTransaction getestet, mit der dann stückweise das gesamte Projekt überprüft wurde. Durch die so abgebildeten Szenarien wurden auch die Transaktionseigenschaften, gemäß der in Kapitel 4.1 festgelegten Anforderungen, mit gezielt konfliktverursachenden Operationen getestet.

Wie Fowler bemerkt, ist es sehr aufwendig, nebenläufige Anwendungen vollständig zu testen[Fow03]. Bei der in dieser Arbeit durchgeführten Implementierungen handelt es sich um einen Prototypen. Es wurden nicht alle theoretisch denkbaren Testfälle berücksichtigt. Insbesondere das Verhalten des Servers bei extremen Timing- bzw. Lastsituationen ist zu testen. Als Beispiel sei etwa der gleichzeitige Start sehr vieler Validierungsphasen als kritisches Szenario genannt. Weitere Szenarien sind in der Literatur etwa unter [Unl94] zu finden.

8 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Transaktionsmodell für den WebDAV-Standard erarbeitet. Das Modell ermöglicht es, eine Menge von Dateioperationen transaktionsbasiert durchzuführen.

Das Modell unterstützt dabei zur Sicherstellung der Serialisierbarkeit sowohl optimistische als auch pessimistische Verfahren. Die Unterstützung des optimistischen Verfahrens wurde dabei von der IETF als zulässiges und sinnvolles Vorgehen zur Realisierung von Transaktionen mittels WebDAV bestätigt. Für die pessimistischen Verfahren wurde im Rahmen dieser Arbeit aufgezeigt, wie die bestehenden Konzepte des WebDAV-Standards erweitert werden müssen, um dies ebenfalls umsetzen zu können.

Um die getroffene Entwurfsentscheidung zu verifizieren, wurde eine prototypische Implementierung des Modells vorgenommen. Hierbei wurde, nach einer entsprechenden Evaluierung und Bewertung, die optimistische Nebenläufigkeitskontrolle umgesetzt. Clientseitig setzt die Implementierung auf der Jackrabit-Library auf, die serverseitige Implementierung verwendet als Grundlage den WebDAV-Server von Subversion.

Auf dem IETF-Meeting 2008 in Dublin wurde das Modell mit optimistischer Nebenläufigkeitskontrolle vorgestellt. Dort wurde der Vorschlag gemacht, die Ergebnisse dieser Arbeit in einem Informational RFC zu veröffentlichen.

Darüber hinaus ist geplant, die notwendigen Erweiterungen zur Umsetzung der pessimistischen Nebenläufigkeitskontrolle auf dem nächsten IETF-Meeting im Juli 2009 in Stockholm, Schweden, vorzustellen. So soll nach einer entsprechenden Diskussion eine abgestimmte und akzeptierte Erweiterung des Standards erreicht werden. Hierzu hat es bereits erste Gespräche mit der zuständigen Working Group gegeben.

Die im Rahmen dieser Arbeit erstellten Ergebnisse fließen in die Weiterentwicklung der im DLR eingesetzten WebDAV-Anwendungen ein. Clientseitig sind das die Anwendung DataFinder sowie die Python WebDAV-Library; serverseitig ist die Umsetzung des DeltaV-Standards für den WebDAV-Server Catacomb geplant.

Darüber hinaus erscheint es sinnvoll, die bereits begonnene Erweiterung der Jackrabbit-Library fortzuführen, um so zu einer Java WebDAV-Client-Library zu gelangen, die zumindest den DeltaV-Standard vollständig unterstützt.

Es ist geplant, weitere Lösungen für das Themengebiet der Erkennung von Nebenläufigkeitsproblemen – wie etwa die der Dead- oder Livelock-Erkennung – zu erarbeiten.

A Auszug aus der Diskussion auf der IETF-Mailingliste

```
1
2
3 That should be fine.  Note though that it only works against servers that ↩
      support working resources ...
4
5 Cheers,
6 Geoff
7
8
9 ietf-dav-versioning-request@w3.org wrote on 07/04/2008 07:31:39 AM:
10
11 >
12 > So here is my next approach to detect conflicts on server-side;-)
13 >
14 > Preconditions:
15 > - the server allows checkout-fork
16 > - checkin-fork will always be forbidden.
17 >
18 > The client creates a private activity and does a "checkout" of his
19 > files to the activity. Furthermore the client adds a "apply-to-
20 > version" element inside the checkout request. This turns the VCR in
21 > a "Working Resource" so that every client has its own working copy.
22 >
23 > Now we come to the conflict detection:
24 >
25 > When the client "checkin" the activity, all his private working
26 > resources will be automatically checked-in by the server. When
```

```
27 > someone else already changed and checked in a VCR that is in our
28 > activity (a conflict) the server will see, that the predecessor-set
29 > to what our working resources points to, already has a successor. In
30 > this situation the server normally has to do a fork. Checkin-fork is
31 > forbidden by the server so the server return an errors message.
32 >
33 > Is this a possible approach?
34 >
35 > Thanks for your answers :-)
36 >
37 >   Martin
```

B Empfehlung von Geoffrey M Clemm zur Nutzung von Subversion

```
1 Hi Martin,
2
3 I believe the Subversion implementation uses working resources.
4 But note that there are very few implementations of DeltaV (and even the ↩
      Subversion implementation is incomplete), so you wouldn't want to write a ↩
      client using DeltaV unless you were either also building a server, or have ↩
      confirmed that the server you care about supports DeltaV.
5
6 Cheers,
7 Geoff
8
9 <M.Jung@dlr.de>
10
11 07/17/2008 04:45 AM
12 To Geoffrey M Clemm/Lexington/IBM@IBMUS
13
14 Subject  AW: AW: Conflict detection in DeltaV using ServerSide Workspace
15
16 Hello Geoff,
17
18 Thanks a lot for your help. Do you know any server that supports working ↩
      resources?
19 Or can I use my first approach with serverside workspace when the server allows ↩
      checkout-fork and I user a private serverside workspace for every user?
20
21 Cheers,
22 Martin
```

Literaturverzeichnis

[AIS77] Christopher Alexander, Sara Ishikawa und Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, 1977.

[BK91] Naser S. Barghouti und Gail E. Kaiser. *Concurrency control in advanced database applications. ACM Comput. Surv.*, 23(3): S. 269–317, 1991.

[BK01] Arthur J. Bernstein und Michael Kifer. *Databases and Transaction Processing: An Application-Oriented Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0201708728.

[BN97] Philip Bernstein und Eric Newcomer. *Principles of transaction processing: for the systems professional*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-415-4.

[CDK05] G.F. Coulouris, J. Dollimore und T. Kindberg. *Distributed Systems: Concepts And Design*. Addison Wesley Longman, 2005.

[Cla08] David Clark. *The Tao of IETF: A Novice's Guide to the Internet Engineering Task Force*. `http://www.ietf.org/tao.html`, 2008. [Online; 27-Jan-2009].

[CSFP06] Ben Collins-Sussman, Brian W. Fitzpatrick und C. Michael Pilato. *Versionskontrolle mit Subversion*. O'Reilly Media, 2006. ISBN 978-3-89721-460-6.

[Dad96] Peter Dadam. *Verteilte Datenbanken und Client/Server–Systeme*. Springer-Verlag, 1996.

[Dus04] Lisa Dusseault. *WebDAV Next-Generation Collaborative Web Authoring*. Prentice Hall, New Jersey, 2004.

[EJB08] *Java Platform, Enterprise Edition (Java EE) Enterprise JavaBeans Technology, http://java.sun.com/products/ejb/*. SUN, Feb. 2008.

[Emm03] W. Emmerich. *Konstruktion von verteilten Objekten.* dpunkt Verlag, Heidelberg, 2003.

[EN03] Ramez A. Elmasri und Shankrant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0805317554.

[Fow03] Martin Fowler. *Patterns für Enterprise Application-Architekturen.* mitp-Verlag, Bonn, 2003.

[Gar99] Felix C. Gartner. *Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Computing Surveys*, 31: S. 1–26, 1999.

[GHJV01] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, München, 5. Aufl., März 2001.

[GR93] Jim Gray und Andreas Reuter. *Transaction Processing: Concepts and Technique.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[GRA78] J GRAY. *Notes on Data Base Operating Systems. Operating Systems, An Advanced Course*, 6: S. 393–481, 1978.

[GT02] David Gourley und Brian Totty. *HTTP: The Definitive Guide.* O'Reilly Media, 2002.

[ISO96] ISO. *Interconnection: Basic Reference Model.* International Organization for Standardization (ISO), International Electrotechnical Committee (IEC), Genf, 1996.

[KH07] Markus Kehle und Robert Hien. *Hibernate und die Java Persistence API.* entickler.press, Paderborn, 2007.

[KPSW04] Sunghun Kim, Kai Pan, Elias Sinderson und E. James Whitehead. *Architecture and Data Model of a WebDAV-based Collaborative System.* In *Proceedings of 2004 Collaborative Technologies Symposium (CTS'04*, S. 48–55. 2004.

[KR81] H. T. Kung und John T. Robinson. *On optimistic methods for concurrency control. ACM Transactions on Database Systems*, 6: S. 213–226, 1981.

[Lin05] Johannes Link. *Softwaretests mit JUnit.* Dpunkt Verlag, 2005.

[Lit07] Markus Litz. *Standardizing Batch methods.* Techn. Ber., IETF, 2007. Online am 18.12.2008.

[LMP04a] Mark Little, Jon Maron und Greg Pavlik. *Java Transaction Processing: Design and Implementation.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 013035290X.

[LMP04b] Mark Little, Jon Maron und Greg Pavlik. *Java Transaction Processing: Design and Implementation.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 013035290X.

[Mic09a] Microsoft. *MSDN – Das Microsoft Developer Network.* `http://msdn.microsoft.com`, 2009. [Online; 27-Jan-2009].

[Mic09b] Microsoft. *WebDAV Batch Methods.* `http://msdn.microsoft.com/en-us/library/aa579405.aspx`, 2009. [Online; 27-Jan-2009].

[Mic09c] Microsoft. *WebDAV Transactions.* `http://msdn.microsoft.com/en-us/library/aa579443.aspx`, 2009. [Online; 27-Jan-2009].

[OHE96] Robert Orfali, Dan Harkey und Jeri Edwards. *The essential client/-server survival guide (2nd ed.).* John Wiley & Sons, Inc., New York, NY, USA, 1996. ISBN 0-471-15325-7.

[OV99] M. Tamer Ozsu und P. Valduriez. *Principles of distributed database systems.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999. ISBN 0-13-691643-0.

[Pap86] Christos Papadimitriou. *The theory of database concurrency control.* Computer Science Press, Inc., New York, NY, USA, 1986. ISBN 0-88175-027-1.

[Poh08] Klaus Pohl. *Requirements Engineering, Grundlagen Prinzipien, Techniken.* dpunkt.verlag, Heidelberg, DE, 2008. ISBN 978-3-89864-550-8.

[QC99] Klaus Quibeldey-Cirkel. *Entwurfsmuster: Design Patterns in der objektorientierten Softwaretechnik (Gebundene Ausgabe).* Springer-Verlag, Berlin, 1999.

[Res03] Julian Reschke. *Interest in standardizing Batch methods?* `http://lists.w3.org/Archives/Public/w3c-dist-auth/2003JulSep/0030.html`, 2003. [Online; 27-Jan-2009].

[RF1] RF1122. *TCP/IP RFC 1122 – Requirements for Internet Hosts – Communication Layers*. http://www.ietf.org/rfc/rfc1122.txt. [Online; 27-Jan-2009].

[RF2] RF2616. *RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1*. http://www.ietf.org/rfc/rfc2616.txt. [Online; 27-Jan-2009].

[RFCa] RFC3253. *RFC 3253 – Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)*. http://www.ietf.org/rfc/rfc3253.txt. [Online; 27-Jan-2009].

[RFCb] RFC3744. *RFC 3744 – Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol)*. http://www.ietf.org/rfc/rfc3744.txt. [Online; 27-Jan-2009].

[RFCc] RFC4918. *RFC 4918 – HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. http://www.ietf.org/rfc/rfc4918.txt. [Online; 27-Jan-2009].

[Sch03] M. Schneider. *Implementierungskonzepte für Datenbanksysteme*. Springer, 2003.

[SSRB02] Douglas Schmidt, Michael Stal, Hans Rohnert und Frank Buschmann. *Pattern-orientierte Software-Architektur. Muster für nebenläufige und vernetzte Objekte*. dpunkt.verlag, 1. Aufl., 2002. ISBN 3898641422.

[Sub09] Subversion. *Use of WebDAV in Subversion*. http://subversion.tigris.org/webdav-usage.html, 2009. [Online; 27-Jan-2009].

[Sum07] S. Sumathi, S. Esakkirajan. *Fundamentals of Relational Database Management Systems*. Springer, Berlin, 2007.

[SUN08] *Sun Microsystems Inc, Supported System Configuration, http://java.sun.com/javase/6/webnotes/install/system-configurations.html*. SUN, Feb. 2008.

[Tan00] Andrew S. Tanenbaum. *Computernetzwerke*. Pearson Studium, 2000.

[Tan02a] Andrew Tanenbaum. *Moderne Betriebssysteme*. Pearson Studium, München, 2002.

[Tan02b] Andrew S. Tanenbaum. *Computer networks: 2nd edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002. ISBN 0130384887.

[TvS03] Andrew Tanenbaum und Martin van Steen. *Verteilte Systeme, Grundlagen und Paradigmen.* Pearson Studium, München, 2003.

[TvS08] Andrew Tanenbaum und Martin van Steen. *Verteilte Systeme, Prinzipien und Paradigmen.* Pearson Studium, München, 2008.

[Unl94] Rainer Unland. *Optimistic Concurrency Control Revisited.* Inst. für Wirtschaftsinformatik, 1994 - Arbeitsberichte des Instituts für Wirtschaftsinformatik., Münster, 1994.

[Von07] Helmut: Vonhoegen. *Einstieg in XML – Grundlagen, Praxis, Referenzen; 4. Auflage.* Galileo Press, 2007.

[web09] webdav.org. *WebDAV Projects.* `http://webdav.org/projects/`, 2009. [Online; 27-Jan-2009].

[WV02] Gerhard Weikum und Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. ISBN 1-55860-508-8.