Bewertung zweier Ansätze zur verteilten Revisionskontrolle über das Internet

Diplomarbeit am Institut für Programmstrukturen und Datenorganisation

Jürgen Reuter

Betreuer:
Prof. Dr. Walter F. Tichy
James J. Hunt

Fakultät für Informatik
Universität Karlsruhe

20. August 2000

Ich versichere hiermit, daß ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 20.08.2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Einhergehend mit der zunehmenden Verbreitung und Bedeutung von Netzwerken wie dem Internet vollzieht sich derzeit ein vielschichtiger Wandel in den Strukturen der weltweiten Ökonomie, der unter dem Schlagwort der Globalisierung eine breite Aufmerksamkeit auf sich zieht. Dieser Wandel umfaßt gleichermaßen die voranschreitende sozioökonomische wie auch technische Vernetzung zum Zwecke einer koordinierten Umsetzung global verteilter Abläufe in Forschung und Industrie, aber auch im privaten Bereich. Arbeitsvorgänge werden durch Arbeitsgruppen (*Teams*) verrichtet, deren Mitglieder häufig räumlich getrennt agieren. Die räumliche Trennung der Mitglieder bedingt eine räumliche Verteilung der Arbeitsabläufe und erfordert somit zwangsläufig die Errichtung von Maßnahmen zur Kooperation und Koordination.

Bereits bei Entwicklungstätigkeiten einzelner und lokal agierender Personen entstehen Arbeitsprodukte, die sich über die Zeit entwickeln und zu deren Organisation und Verwaltung Revisionskontrollsysteme eingesetzt werden. Bei Zusammenarbeit räumlich getrennt agierender Entwickler einer Arbeitsgruppe ist die zur Kooperation und Koordination notwendige Kommunikation aufgrund der räumlichen Trennung zusätzlich eingeschränkt. In diesem Falle ist die Organisation und Verwaltung der Arbeitsprodukte etwa in der Form eines gemeinsamen Datenbestandes von noch größerer Bedeutung. Hierin begründet sich der Wunsch nach einem global zugreifbarem Revisionskontrollsystem.

Die global verfügbare, standardisierte Infrastruktur des Internet als heterogenem Netzwerk bietet die Voraussetzungen, die für die Errichtung solcher Revisionskontrollsysteme erforderlich sind. Die Flexibilität des Internet ermöglicht durch das Vorhandensein zahlreicher offener Protokollstandards vielfältige Ansätze zur konkreten Realisierung. Dies legt die Bewertung verschiedener Ansätze zur verteilten Revisionskontrolle nahe, von denen zwei in dieser Arbeit betrachtet werden.

1.1 Revisionskontrolle

Revisionskontrolle bezeichnet die Disziplin zur Organisation und Verwaltung von Arbeitsprodukten, die sich über die Zeit entwickeln. Revisionskontrolle ist überall dort von Bedeutung, wo Entwurfsaufgaben zur Erstellung und häufigen Änderung von Datenbeständen führen. Ihre Aufgabe besteht darin, Revisionen der Datenbestände und ihre Abhängigkeiten untereinander zu verfolgen. Ein einfaches und von den meisten Revisionskontrollsystemen verwendetes Modell zur Revisionskontrolle ist das *Checkin/Checkout-Modell.*

Die Idee des Checkin/Checkout-Modells ist es, Momentaufnahmen vom Zustand sich ändernder Daten zu benutzerdefinierbaren Zeitpunkten zu erfassen, zu archivieren und zu verwalten, um die Änderungen zurückverfolgen zu können. Eine solche Momentaufnahme wird als *Revision* bezeichnet. Die Revisionierung von Daten erfolgt üblicherweise auf der Basis von Dateien eines Dateisystems, die unter die Kontrolle einer Revisionsverwaltung gestellt werden. Wird eine Datei unter Revisionskontrolle gestellt, so wird der augenblickliche Zustand der Datei als *initiale Revision* archiviert. Durch *Ausbuchen* (*checkout*) einer Revision wird der Zustand einer Datei zum Zeitpunkt der zugehörigen Momentaufnahme wiederhergestellt. Dabei kann gegebenenfalls eine geplante Weiterentwicklung der Datei vorgemerkt werden. Durch das *Einbuchen* (*checkin*) der geänderten Datei wird eine neue Momentaufnahme erfaßt und als *Nachfolgerevision* archiviert. Wird dieselbe Revision mehrfach zur Weiterentwicklung ausgebucht, so entstehen mehrere Nachfolgerevisionen. Die Menge aller Revisionen bildet dann einen *Revisionsbaum* mit Verästelungen. Häufig besteht der Wunsch, Verästelungen wieder zusammenzuführen. Dies erfordert die *Verschmelzung* (*merge*) von Revisionen durch spezielle Operationen. Die Menge aller Revisionen bildet daher im allgemeinen Fall einen *Revisionsgraphen.* Versionskontrolle schließt die Verwaltung des Revisionsgraphen mit ein. Dies umfaßt die Speicherung und Verwaltung der Revisionen einschließlich ihrer Vorgänger- beziehungsweise Nachfolgerrelationen, aber auch die Speicherung und Verwaltung zusätzlicher Daten über die Revisionen selbst wie Erstellungszeitpunkt, Autor oder Kommentar. Mit diesen zusätzlichen Daten läßt sich die *Historie* der revisionierten Daten in der Art eines Logbuches protokollieren.

Das Checkin/Checkout-Modell ist zugleich Grundlage weiterer, komplexerer Modelle, die für die folgende Betrachtung jedoch nicht weiter von Interesse sind.

1.2 Unterstützung verteilter Entwicklung

Verteilte Revisionskontrolle erweitert die Disziplin der Revisionskontrolle um die Möglichkeit ihrer räumlich verteilten Anwendung. Sie ist dort von Bedeutung, wo revisionierte Datenbestände parallel an räumlich getrennten Orten entwickelt werden. Ihre Aufgabe besteht darin, entfernte Zugriffe auf die revisionierten Da-

tenbestände zu ermöglichen und konkurrierende Zugriffe zu koordinieren. *Verteilte Revisionskontrolle über das Internet* macht sich das Internet als globales heterogenes Rechnernetz zur Realisierung einer verteilten Revisionskontrolle zunutze. Über das Internet wird es möglich, von jedem Ort aus weltweit auf revisionierte Datenbestände zugreifen zu können. Die bestehende Infrastruktur des Internet bietet eine reiche und flexible Basis zur Integration einer verteilten Revisionskontrolle.

1.3 Aufgabenstellung

Ziel dieser Arbeit ist die Bewertung zweier Ansätze zur verteilten Revisionskontrolle über das Internet. Die Bewertung stützt sich auf die Analyse zweier Systeme, deren Implementierung teilweise Bestandteil dieser Arbeit ist. Im Zentrum des Interesses steht die Bewertung der beiden Ansätze zur Verteilung. Im Sinne einer guten Vergleichsbasis ist es daher zweckmäßig, wenn die übrigen Bestandteile der beiden Systeme identisch sind. Die beiden betrachteten Systeme gewährleisten dieses Ziel, indem sie einerseits auf demselben lokalen Revisionskontrollsystem RCE als Backend aufsetzen und andererseits mit VRCE dieselbe graphische Benutzeroberfläche als Frontend verwenden, über die nachfolgend ein kurzer Überblick gegeben werden soll. Im eigentlichen Interesse dieser Arbeit stehen die dazwischenliegenden Schichten der beiden Systeme, die die Verteilung realisieren.

1.4 RCE

Die beiden betrachteten Systeme basieren auf RCE als lokalem Revisionskontrollsystem. *RCE* (*Revision Control Engine*)[4] als Nachfolger von RCS[3] ist ein in der Programmiersprache C geschriebenes, programmierbares Werkzeug zur Revisionskontrolle, welches Revisionen beliebiger Anzahl auf Dateibasis in Archiven speichert und verwaltet. Es führt selbsttätig ein Logbuch zu jedem Archiv, speichert, extrahiert und identifiziert selbständig Revisionen und stellt in Ansätzen Mechanismen zur Konfigurationskontrolle zur Verfügung. Die Verwendung von RCE ist nicht auf Textdateien beschränkt, sondern es werden jegliche Datentypen einschließlich binärer Formate unterstützt. Durch die Verwendung von Differenzen aufeinanderfolgender Revisionen zu ihrer Speicherung werden beachtliche Einsparungen beim benötigten Speicherplatz erzielt. Die anwendungsorientierte, auf der Programmiersprache C basierende Programmierschnittstelle von RCE ermöglicht eine einfache Einbindung der Funktionalität von RCE in eigene Anwendungen. Eine vergleichende Analyse von RCE mit anderen Werkzeugen zur Revisionskontrolle findet sich in der Diplomarbeit zu CME[5].

Die Java-Schnittstelle zu RCE baut auf der C-Schnittstelle auf und

bietet mit ihren drei Klassen `durasoft.rce.RCE`, `durasoft.rce.Archive` und `durasoft.Revision` im wesentlichen dieselbe Funktionalität wie die C-Schnittstelle. Die in der C-Schnittstelle über die Deskriptoren `RCEArchHandle` und `RCERevHandle` referenzierten Archive und Revisionen werden in der Java-Schnittstelle als eigenständige Objekte modelliert; die verbleibende dritte Klasse faßt weitere, von Archiven und Revisionen unabhängige Funktionen zusammen. C-spezifische Datentypen wie `(char *)` werden auf entsprechende Java-Datentypen wie `java.lang.String` umgesetzt. Abbildung 1.1 gibt einen Überblick über die Java-Schnittstelle. Außer RCE selbst wurden alle übrigen Bestandteile der beiden in dieser Arbeit betrachteten Implementierungen in Java entwickelt und setzen daher auf der Java-Schnittstelle von RCE auf. Mit „RCE“ ist daher, wenn nicht ausdrücklich anders angegeben, im folgenden stets die Java-Schnittstelle von RCE gemeint.

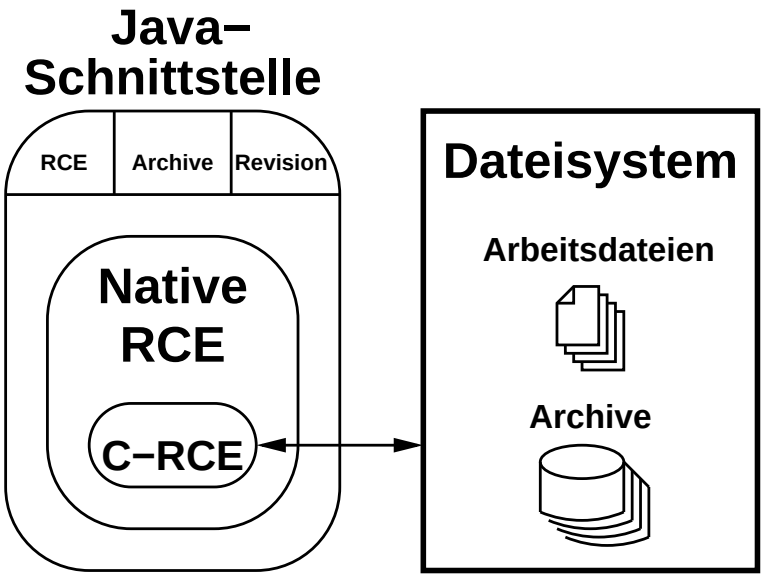

Abbildung 1.1: Java-Schnittstelle von RCE

1.5 VRCE

Mit *Visual RCE* oder kurz *VRCE* wird eine graphische Benutzeroberfläche in Java zu RCE bezeichnet, die alle wesentlichen Funktionen von RCE umsetzt (vgl. Abbildung 1.2). Sie operiert nicht direkt auf der Java-Schnittstelle von RCE, sondern trennt alle anwendungsspezifischen Aspekte von der reinen Funktionalität des zugrundeliegenden Revisionskontrollsystems mittels einer Schnittstelle, die im folgenden als *Repository-Schnittstelle* bezeichnet wird. Die Implementierung von VRCE schließt eine Implementierung der Repository-Schnittstelle ein, die die Funktionen der Repository-Schnittstelle direkt auf die Java-Schnittstelle umsetzt.

Im Falle der lokalen Anwendung von VRCE ergibt sich daher die in Abbildung 1.3 Architekturübersicht.

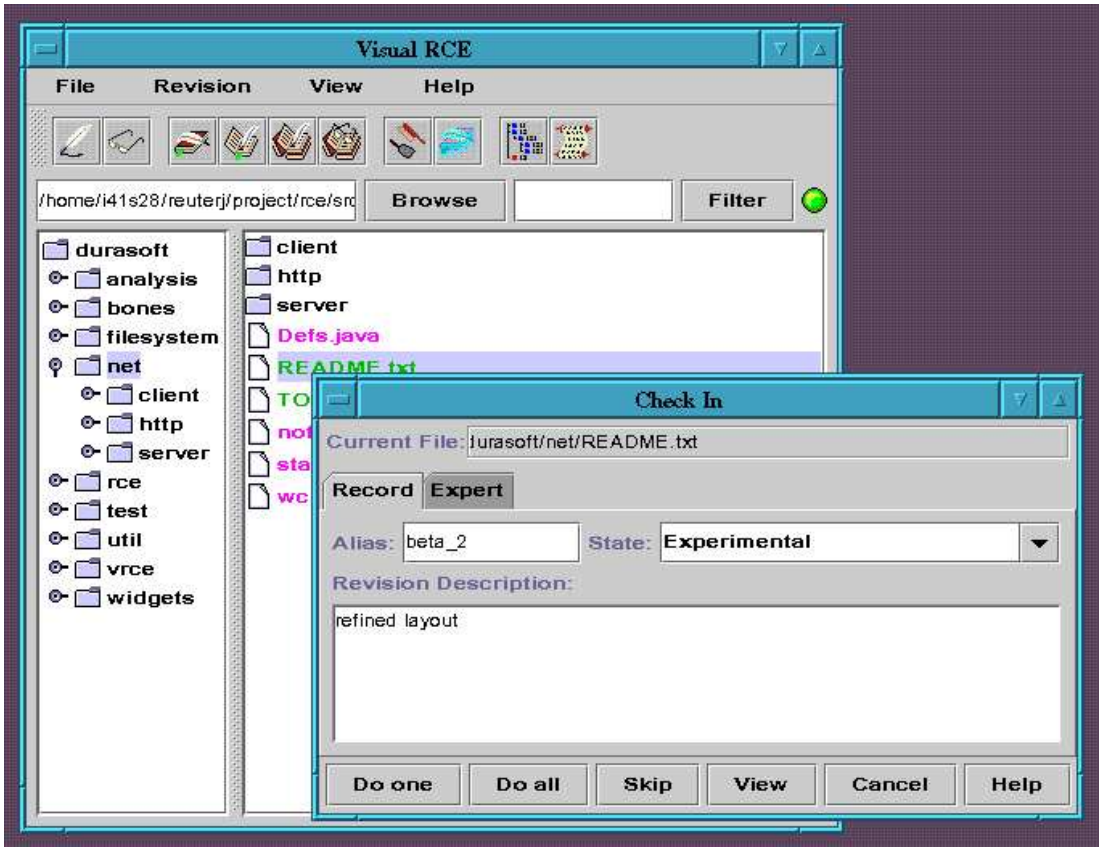

Abbildung 1.2: Graphische Benutzeroberfläche VRCE

1.5.1 Repository-Schnittstelle

Die von VRCE verwendete Repository-Schnittstelle ist zwar spezifisch auf RCE zugeschnitten, unterscheidet sich von der RCE-Schnittstelle aber in einigen wesentlichen Aspekten. Die RCE-Schnittstelle enthält feinkörnige, einfach gehaltene Dienstprimitive wie beispielsweise solche zum Setzen oder Lesen einzelner Archivattribute. Diese Dienstprimitive können mit Hilfe der Transaktionsprozeduren der RCE-Schnittstelle zu beliebig komplexen Transaktionen zusammengesetzt werden. Die Repository-Schnittstelle hingegen ist speziell auf VRCE abgestimmt. Daher konzentriert sie sich auf die von VRCE benötigten, auf höherer Ebene angesiedelten Funktionen. Hierzu zählen das gebündelte Setzen oder Lesen von Attributen oder das Einbuchen oder Ausbuchen einer Menge von Dateien mit einem einzigen Methodenaufruf. Neben der Bündelung bietet die Repository-Schnittstelle auch spezielle höhere Funktionen wie das Zwischenspeichern (*checkpoint*), das sich auf Funktionen wie das Einbuchen und anschließende Wiederaus-

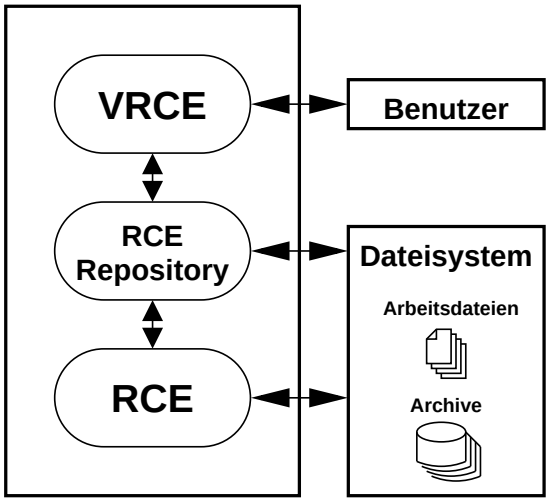

Abbildung 1.3: Architekturübersicht zur lokalen Anwendung von VRCE

buchen zurückführen läßt. Auf Transaktionsprozeduren und -semantik verzichtet die Repository-Schnittstelle, da die Granularität ihrer Funktionen hinreichend grob ist. Die Repository-Schnittstelle bietet somit eine an die Bedürfnisse von VRCE abgestimmte und mit Hinblick auf die Austauschbarkeit der darunterliegenden Kommunikation mit dem Revisionskontrollsystem entworfene Sammlung von Dienstprozeduren.

Kapitel 2

VRCE via RMI

VRCE via RMI verwendet im wesentlichen dieselbe Implementierung der Repository-Schnittstelle wie das eingangs beschriebene lokale VRCE mit dem Unterschied, daß anstelle einer lokalen eine entfernte Instanz von RCE als Ausgangsbasis verwendet wird. Damit erweitert VRCE via RMI die Java-Schnittstelle von RCE[4] um die Möglichkeit, die Verwaltung von Revisionsdaten in Archiven räumlich getrennt von der Erstellung und Verwendung der Daten durchzuführen. Dies geschieht zudem in einer für den Anwender der Schnittstelle transparenten Weise. Die Transparenz soll die räumliche Trennung vor dem Anwender so weit wie möglich verbergen, so daß eine auf der Java-Schnittstelle aufsetzende und für den lokalen Fall entwickelte Anwendung mit nur minimalen Änderungen auch im verteilten Fall läuft. Die Erweiterung der Java-Schnittstelle von RCE basiert auf der Verwendung entfernter Methodenaufrufe, die mittels des in Java standardmäßig enthaltenen Paketes RMI (*remote method invocation*) realisiert wurde.

Abbildung 2.1 zeigt die sich ergebende Architektur als Übersicht.

2.1 Fernbenutzungsmodell von RMI

Die konkrete Realisierung der entfernten Methodenaufraufe durch das Java-Paket RMI entspricht im wesentlichen den herkömmlichen Konzepten der Architektur von Rechnerverbundsystemen. In solchen Systemen erschließt sich dem Aufrufer mit dem entfernten Methodenaufruf die Möglichkeit zur Fernbenutzung einer Leistung. Zu einem solchen Aufruf gehören im allgemeinen Fall

- die Übermittlung von Parametern der aufrufenden an die aufgerufene Methode,
- die eigentliche Auslösung des entfernten Aufrufes sowie
- die Rückübermittlung des Rückgabewertes der aufgerufenen Methode.

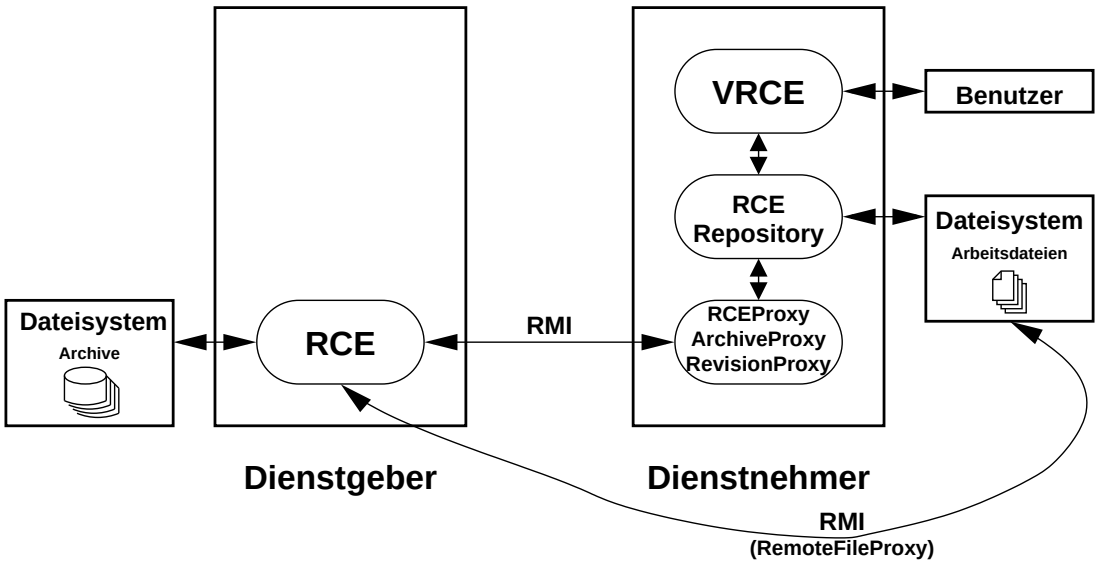

Abbildung 2.1: Architekturübersicht zu VRCE via RMI

Die als Parameter oder Rückgabewert übermittelten Daten können selbst wiederum Objekte mit Instanzenvariablen und aufrufbaren Instanzenmethoden sein. Dies eröffnet zwei Möglichkeiten zur Realisierung der Übermittlung eines entfernten Objektes:

- Das zu übermittelnde Objekt wird als reale Kopie gesendet (*Wertübergabe*), oder
- anstelle des eigentlichen Objektes wird ein Stellvertreterobjekt als Referenz übermittelt (*Referenzübergabe*).

Bei der Wertübergabe führen Änderungen an der Kopie beim Empfänger nicht zu Änderungen des originalen Objektes. Methodenaufrufe auf der Kopie werden lokal beim Empfänger ausgeführt. Hierzu muß der Kode der Objektklasse dem Empfänger bekannt sein. Weil als Parameter ein Objekt einer Unterklasse der in der Signatur angegebenen Klasse angegeben werden kann, ist der Kode der Objektklasse dem Empfänger im allgemeinen nicht bekannt. Daher muß bei der Übergabe gegebenfalls außer den Daten des Objektes auch der Kode der zugehörigen Klasse übermittelt werden. Ein Vorteil dieses dynamischen Ladens von Kode besteht in der Möglichkeit, zur Laufzeit neue Objekttypen in die Kommunikation zwischen Dienstgeber und Dienstnehmer einzuführen.

Methodenaufrufe auf einem als Referenz übermittelten Stellvertreterobjekt führen zu entfernten Methodenaufrufen auf dem originalen Objekt. Die Stellvertreterobjekte werden als Stummel (*stub*) realisiert. Ein Stummel stellt alle als öffentlich deklarierten Methoden des referenzierten Objektes stellvertretend

bereit. Er vertritt in Verbindung mit der übermittelten Referenz damit gewissermaßen das reale Objekt, welches sich an einem entferntem Ort befindet. Ruft der Empfänger eine Instanzenmethode dieses Stummels auf, so leitet der Stummel seine Aufrufparameter an das vertretene Objekte weiter, löst den entsprechenden Methodenaufruf beim vertretenen Objekt aus, empfängt gegebenenfalls den Rückgabewert des Aufrufes und reicht ihn an den Aufrufer des Stummels weiter. Ähnlich wie bei der Wertübergabe muß gegebenfalls auch hier Kode übermittelt werden. Um keinen Unterschied zwischen der Verwendung von Stummeln und den durch sie repräsentierten Objekten machen zu müssen, implementieren Stummel und repräsentiertes Objekt eine gemeinsame Schnittstelle. Diese Stellvertreterschnittstelle legt zugleich den Funktionsumfang des entfernten Objektes fest. Hinter einem Parameter, der eine Stellvertreterschnittstelle implementiert, kann sich dann je nach Bedarf entweder eine Referenz auf ein entferntes Objekt in Gestalt eines Stummels oder aber eine lokale Objektinstanz verbergen. Diese Transparenz ermöglicht eine räumliche Trennung zwischen dem Ort des Methodenaufrufes und dem der Methodenausführung weitgehend ohne Änderung einer bestehenden Implementierung. Das Konzept der Referenzübergabe bedingt eine symmetrische Kommunikationsbeziehung zwischen Dienstgeber und Dienstnehmer: übermittelt der Dienstnehmer ein aus seiner Sicht lokales Objekt dem Dienstgeber als Referenz, so führt der Aufruf einer Instanzenmethode des übermittelten Parameters durch den Dienstgeber über die entsprechende Methode des dienstgeberseitigen Stummels zur Ausführung der Methode am Ursprungsort des Objektes auf Dienstnehmerseite. Durch Speicherung eines Stummels kann der Dienstgeber effektiv jederzeit Methodenaufrufe auf Dienstnehmerseite auslösen; das Fernbenutzungsmodell zwischen Dienstgeber und Dienstnehmer ist daher im Prinzip symmetrisch.

RMI unterstützt sowohl Wert- wie auch Referenzübergabe. Bei der Wertübergabe wird das zu übermittelnde Objekt beim Sender serialisiert, übermittelt und beim Empfänger durch Deserialisierung eine Kopie des Objektes erstellt. Standardmäßig werden alle Instanzenvariablen des zu übermittelnden Objektes in die Serialisierung einbezogen, sofern sie nicht als transient gekennzeichnet sind. Handelt es sich bei diesen Variablen selbst um Objekte, so werden auch diese serialisiert. Hierzu müssen alle beteiligten Klassen die Schnittstelle `java.util.Serializable` implementieren. Objekte, die als Referenz übermittelt werden sollen, müssen außer der Stellvertreterschnittstelle auch die Schnittstelle `java.rmi.Remote` implementieren. Zur automatischen Erzeugung des dazugehörigen Stummel eines solchen Objektes enthält RMI das Werkzeug `rmic`. Muß, wie geschildert, dem Empfänger eines Objektes der Kode der zugehörigen Klasse oder des zugehörigen Stummels übermittelt werden, so führt RMI diese Übermittlung selbsttätig durch. Wegen der angesprochenen Symmetrie kann eine solche Übermittlungen in beiden Richtungen notwendig werden. Um die Kommunikation zwischen Dienstgeber und Dienstnehmer in Gang zu setzen, sieht RMI das Konzept der Registratur (*RMI registry*) vor. Die Registratur als TCP/IP-

basierter Namensgeberdienst des Dienstgebers ermöglicht dem Dienstnehmer das explizite Anfordern eines Stellvertreterobjektes unter Angabe des Namens, unter dem es in die Registratur vom Dienstgeber eingetragen wurde. Ein auf diese Weise erhaltenes Stellvertreterobjekt ist als *Zugriffsobjekt* Ausgangspunkt für weitere entfernte Methodenaufrufe. Abbildung 2.2 veranschaulicht beispielhaft die Kommunikation über RMI.

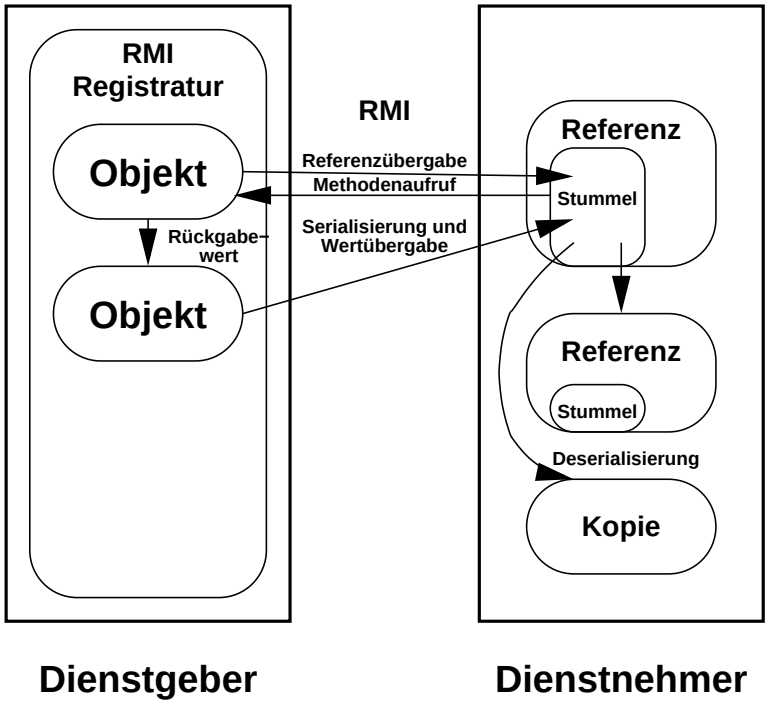

Abbildung 2.2: Typische Kommunikation über RMI

Ergänzend sei erwähnt, daß es neben RMI als Alternative mittlerweile auch eine CORBA-basierte Architektur zur Fernbenutzung gibt, die ebenfalls standardmäßig in Java als Paket `org.omg` enthalten ist. Während RMI speziell auf die Möglichkeiten von Java zugeschnitten ist und daher viel Spielraum zur Verbesserung von Laufzeit- und Speicherverhalten bietet, spricht für CORBA der Vorteil der Plattform- und vor allem Sprachenunabhängigkeit des Objektmodells mittels der Schnittstellendefinitionssprache IDL.

2.2 Modellierung der Java-Schnittstelle von RCE über RMI

RMI übernimmt, wie erläutert, alle Maßnahmen, die zur Realisierung entfernter Methodenaufrufe notwendig sind. Es bleibt zu modellieren, welche Objekte der Java-Schnittstelle in der Kommunikation über RMI zwischen Dienstgeber und

Dienstnehmer als Wert und welche als Referenz übergeben werden sollen. Es gibt drei Arten von Objekten, die als Referenz übergeben werden müssen:

- ein Zugriffsobjekt, welches eine Fabrikmethode für den Zugriff auf die unter RCE relevanten Objekte enthält,
- die RCE-Objekte, die die Funktionalität der Java-Schnittstelle von RCE zur Verfügung stellen, und
- Dateiobjekte, die den Zugriff auf entfernte Dateien ermöglichen.

Alle anderen über die Java-Schnittstelle transportierten Objekte wie `java.lang.String` sind als serialisierbar gekennzeichnet und werden von RMI stets als Wert übermittelt.

2.2.1 Zugriffsobjekt

Die Kommunikation über RMI wird, wie beschrieben, vom Dienstnehmer durch Anfordern eines Zugriffsobjektes von der Registratur in Gang gesetzt. Dieses muß mindestens eine Fabrikmethode enthalten, die den Zugriff auf alle Objekte der Java-Schnittstelle von RCE ermöglicht.

2.2.2 RCE-Objekte

Die über die Klasse `durasoft.rce.Archive` zugreifbaren RCE-Archive sollen bei VRCE via RMI auf einem Dienstgeber zentral gespeichert werden, um von einem beliebigen, in dasselbe Netzwerk eingebundenen Dienstnehmer aus auf sie Zugriff erhalten zu können. Dies bedingt die Deklaration von Archiven als entfernte Objekte, die demnach über RMI nur in der Form einer Referenz, nicht jedoch als Wert übergeben werden dürfen. Denn die Übergabe als Wert bedeutete die Übermittlung einer Kopie des Archivs an den Dienstnehmer anstelle einer Referenz, über die der Dienstnehmer auf das auf dem Dienstgeber gespeicherte Archiv zugreifen möchte.

Die Klasse `durasoft.rce.Revision` operiert ebenfalls auf RCE-Archiven auf dem Dienstgeber. Die Überlegungen zu `durasoft.rce.Archive` gelten daher in analoger Weise auch für Revisionsobjekte. Deshalb deklariert RCE via RMI auch diese Klasse als entfernt.

Zu beachten ist ferner, daß auch der Aufruf einer der statischen Methoden der Klasse `durasoft.rce.RCE` im allgemeinen ortsabhängig ist. So können einerseits Eigenheiten des darunterliegenden Systems relevant sein, beispielsweise bei der Methode `RCE.filePathConvert()`, die in Abhängigkeit vom Dateisegmenttrennzeichen (meist „/“ oder „\“) verschiedene Resultate liefern kann. Zum anderen liefern die Fabrikmethoden Archiv- oder Revisionsobjekte, deren Entstehungsort dem des Ausführungsortes der Fabrikmethode entspricht. Die Ausführung

der Methoden dieser Klasse muß demnach am selben Ort geschehen, an dem sich RCE-Archive befinden, also beim Dienstgeber. Daher muß auch diese Klasse als entfernt deklariert werden. Der naheliegende Gedanke, `durasoft.rce.RCE` zugleich als Zugriffsobjekt zu verwenden, ist allerdings nicht umsetzbar. Denn `durasoft.rce.RCE` enthält als Instanzenvariable den Benutzernamen, in dessen Auftrag Zugriffe auf Archive erfolgen. Daher muß für jeden Benutzer eine eigene Instanz dieser Klasse erzeugt werden.

VRCE via RMI deklariert somit die drei Klassen `durasoft.rce.Archiv`, `durasoft.rce.Revision` und `durasoft.rce.RCE` als entfernt, so daß sie über RMI stets als Referenz übermittelt werden.

2.2.3 Dateiobjekte

Daten werden nicht nur als Java-Objekte im Arbeitsspeicher gehalten, sondern auch in Form von Dateien permanent gespeichert. Im Falle eines Revisionskontrollsystems sind als Dateien vor allem die Archive und Arbeitsdateien relevant. RCE selbst wurde ursprünglich als lokale Anwendung entwickelt. Wird eine Datei aus einem RCE-Archiv auf einem Dienstgeber ausgebucht, so wird eine Arbeitsdatei im lokalen Dateisystem des Dienstgebers erzeugt. Wenn der Dienstnehmer, der die Ausbuchung veranlaßt hat, sich nicht dasselbe Dateisystem mit dem Dienstgeber teilt, muß er daher auf andere Weise auf die Datei zugreifen, beispielsweise im Sinne eines entfernten Objektes via RMI.

Dateien sind keine Java-Objekte; ihre Verwaltung erfolgt durch das Dateisystem des Dienstgebers und somit außerhalb des von der virtuellen Maschine von Java und RMI kontrollierten Bereiches. Zur Behandlung von Zugriffen auf entfernte Dateien müssen daher besondere Maßnahmen getroffen werden. Insbesondere Dateideskriptoren sind auf einem entfernten Rechner ohne besondere Maßnahmen nicht verwendbar. Dementsprechend zählt die Klasse `java.io.FileDescriptor` zu den seltenen Standardklassen unter Java, die weder die Schnittstelle `java.io.Serializable` implementieren, noch als entfernt deklariert sind und sich daher für den Transport via RMI nicht eignen.

Die Klasse `java.io.File` ist serialisierbar, jedoch nicht als entfernt deklariert. Daher werden Objekte dieses Typs bei der Übermittlung via RMI grundsätzlich als Wert übergeben. Die hierbei erfolgende Serialisierung umfaßt aber nicht den Inhalt der zugehörigen Datei, sondern im wesentlichen nur eine Zeichenkette zur Speicherung ihres Dateipfades. Der Empfänger erhält somit eine lokale Kopie des Objektes. Methoden wie `File.delete()` oder `File.canWrite()` beziehen sich dann stets auf das Dateisystem der Maschine, auf der sie aufgerufen werden, und nicht auf das Dateisystem der Maschine, von der das Objekt ursprünglich stammt. Um diesen Mangel zu beseitigen, deklariert VRCE via RMI hierzu `durasoft.rce.client.RemoteFile` als Unterklasse von `java.io.File`, die als entfernt deklariert wird, um bei der Übertragung via RMI als Referenz übergeben zu werden.

Die Schnittstelle von `java.io.File` respektive `durasoft.rce.client.RemoteFile` unterstützt zwar Operationen wie `File.delete()` oder `File.mkdir()` zur Manipulation von Dateien, bietet jedoch keine weitergehenden Möglichkeiten zum Editieren des zugehörigen Dateiinhaltes. Aber auch ein auf dem Dienstnehmer eingesetzter Texteditor, der nichts von Referenzen unter RMI weiß, benötigt zwingend eine lokale Kopie der beim Dienstgeber liegenden Datei, um auf ihr arbeiten zu können. Auch aus Effizienzgründen bei häufigen Änderungen an der Datei sowie aus Verfügbarkeitsgründen bei schlechter Netzanbindung ist eine lokale Kopie wünschenswert. Hierzu muß die gesamte Datei zum Dienstnehmer übertragen werden, um dort bearbeitet werden zu können und schließlich zum Dienstgeber zur Einbuchung zurück übermittelt zu werden. Für die Übertragung der Dateiinhalte enthält `durasoft.rce.client.RemoteFile` daher zusätzliche Methoden, die von VRCE via RMI allerdings explizit aufgerufen werden müssen. `RemoteFile` bietet somit alle Voraussetzungen für die Bearbeitung entfernter Dateien.

2.3 Architektur von Dienstgeber und Dienstnehmer

Die eigentliche Kommunikation zwischen Dienstgeber und Dienstnehmer übernimmt bereits RMI und braucht daher nicht gesondert implementiert zu werden. Die wesentliche Aufgabe des Dienstgebers von VRCE via RMI ist die Bereitstellung einer Registratur, um die Kommunikation via RMI in Gang zu setzen. Der Dienstgeber startet hierzu die RMI-Registratur und fügt als einzigen Eintrag eine Instanz der Klasse `durasoft.rce.RCEPortal` in sie ein. Dieses Portal übernimmt die Rolle des Zugriffsobjektes und zugleich die Authentifikation des Dienstnehmers. Hierzu werden die beim Methodenaufruf durch den Dienstnehmer übermittelten Parameter `name` und `password` durch den Dienstgeber überprüft. Im Erfolgsfalle wird der Parameter `name` an die zu erzeugende Instanz von `durasoft.rce.RCE` als Benutzername weitergereicht.

Auf der Seite des Dienstnehmers muß zum Aufbau der Kommunikation über den Registratur-Dienst eine Referenz auf das Portal angefordert werden. Über dessen Fabrikmethode kann dann eine entfernte Instanz der Klasse `durasoft.rce.RCE` erzeugt werden, auf die der Dienstnehmer eine Referenz erhält. Diese Referenz dient sodann als Ausgangspunkt zur Verwendung der gesamten Java-Schnittstelle von RCE über RMI.

Kapitel 3

VRCE via Delta-V

Während VRCE via RMI einen proprietären, auf RMI aufsetzenden Ansatz zur verteilten Revisionskontrolle auf der Basis von RCE darstellt, spiegelt Delta-V Bemühungen wider, zum gleichen Zwecke einen hersteller- und plattformübergreifenden Standard zu schaffen, der auf bestehenden Standards des WWW aufbaut. Einen detaillierteren Einblick in die hier nur grob skizzierten Aspekte der Einordnung, Ziele und Struktur von Delta-V gibt ein Artikel von E. WHITEHEAD[31].

Schon sehr früh in seiner noch kurzen Geschichte bestand ein Interesse, Revisionskontrolle über das World Wide Web (WWW) zu unterstützen. Vorausgehende Arbeiten verschiedener Autoren zeigten jedoch schon bald, daß das Ziel, verteilte Revisionskontrolle allein mit den bestehenden Möglichkeiten des WWW auf der Basis bestehender Dienstgeber und Dienstnehmer zu realisieren, zu architektonisch unbefriedigenden Lösungen führt. Zwei dieser Arbeiten ([20], [21]) sind im Zusammenhang mit der dieser Diplomarbeit vorausgehenden Studienarbeit entstanden.

Wesentlicher Grund für die architektonisch unbefriedigenden Lösungen waren Beschränkungen in der Erweiterbarkeit der Dienstgeber und der als Dienstnehmer eingesetzten Browsersoftware. Zwar boten einige dieser Dienstnehmer proprietäre Schnittstellen zur Erweiterung der Funktionalität an. Doch um eine von der Implementierung des Dienstnehmers unabhängige Lösung beispielsweise zum Einbuchen einer Datei zu realisieren, war ein umständlicher Umweg über Hilfsapplikationen (*helper applications*) notwendig, um die Datei am Dienstnehmer vorbei an den Dienstgeber zu schicken. Weitere Probleme bestanden darin, dabei zugleich die Aktualität der vom Dienstnehmer angezeigten WWW-Seite zu gewährleisten. Auch die standardisierten Dienstgeberschnittstellen wie das *Common Gateway Interface* (*CGI*) erwiesen sich als nur bedingt hilfreich, weil die erforderlichen Erweiterungen eine viel engere Koppelung zum Dienstgeber benötigten, als die Schnittstellen boten.

Als Konsequenz aus diesen Erfahrungen entstanden Bestrebungen, die Infrastruktur des WWW in Gestalt des HTTP-Protokolls[39] selbst zu erweitern, um verteilte Revisionskontrolle auf der Basis des WWW in standardisierter Form und

ohne die beschriebenen Nachteile zu ermöglichen. *WWW Distributed Authoring and Versioning* (*WebDAV*)[43], eine Erweiterung des HTTP-Protokolls, geht in diese Richtung. Aber obgleich der Titel dieses Protokolls die Unterstützung von Revisionskontrolle bereits ankündigt, erweitert es die Infrastruktur des WWW vorerst nur um einige für die Revisionskontrolle wesentliche Bestandteile. Die eigentliche Spezifikation der verteilten Revisionskontrolle über das WWW verwirklicht Delta-V[51]. Die Entwicklung von Delta-V als zukünftigem Internet-Standard erfolgt – wie bei solchen Protokollen üblich – durch eine für Internet-Teilnehmer offen zugängliche Arbeitsgruppe der Internet Engineering Task Force (IETF)[48].

Um eine Bewertung von VRCE via Delta-V zu ermöglichen, mußte eine Implementierung von Grund auf entwickelt werden, da sich das diesem Ansatz zugrundeliegende Protokoll Delta-V derzeit (August 2000) noch immer in Entwicklung befindet und eine Referenzimplementierung in absehbarer Zeit nicht zu erwarten ist. Die im Rahmen dieser Arbeit in der Programmiersprache *Java*[18] geschaffene Implementierung spiegelt den Stand des Delta-V-Protokolls in der Entwurfsversion 04.5 vom Mai 2000 wider. Diese kann als die erste im wesentlichen implementierbare Protokollversion angesehen werden. Bei der Implementierung zeigte sich noch ein gravierender Mangel in Version 04.5, der eine getreue Implementierung unmöglich machte. Durch eine geringfügige Abweichung der Implementierung vom Protokoll ließ sich dieser Mangel jedoch beseitigen. Die aktuelle Protokollversion 07 vom August 2000 bringt gegenüber der Version 04.5 nochmals erhebliche Änderungen mit sich; dabei wurde unter anderem auch der genannte Mangel beseitigt. Aufgrund des zeitlich gesetzten Rahmens kann diese Diplomarbeit jedoch nicht näher auf diese neueste Version eingehen, sondern richtet ihr Augenmerk auf Version 04.5.

Aufgrund der noch nicht abgeschlossenen Entwicklung von Delta-V muß die Implementierung als unvollständig und vorläufig vorbehaltlich einer noch ausstehenden, endgültigen Spezifikation von Delta-V aufgefaßt werden. Ferner beschränkt sich die hier betrachtete, innerhalb eines begrenzten Zeitrahmens entstandene Implementierung angesichts des großen Umfanges einer vollständigen Implementierung auf diejenigen Aspekte, die für den Vergleich zu VRCE via RMI notwendig erscheinen. Selbst unter dieser Beschränkung erreicht die Größe des solcherart durchgeführten Projektes einen Umfang von mehr als 40000 Zeilen Quelltext. Selbst nach Abzug von Leerzeilen, Kommentare u.ä. zur Bestimmung des Maßes Lines of Code (LOC) dürfte, dem Skriptum zur Vorlesung Softwaretechnik folgend (vgl. Abb. 3.1), die Größe der Implementierung der eines Projektes mittleren Umfangs entsprechen. Dieser Umfang erklärt sich nicht alleine durch Delta-V, sondern vor allem durch die breite Infrastruktur, auf der Delta-V aufbaut.

Gemäß der Dienstgeber/Dienstnehmer-Architektur des Protokolls gliedert sich die Gesamtarchitektur in Dienstgeber- und Dienstnehmerseite. Auf Dienstnehmerseite wurde auf VRCE zurückgegriffen. Die dienstnehmerseitige Imple-

	Programmierer	Dauer	Größe	Beispiel
Trivial	1	1-4 Wochen	500 LOC	
Klein	1	1-12 Monate	1-5 KLOC	
Mittel	2 - 5	1-2 Jahre	5-50 KLOC	Compiler
Groß	5 - 100	2-3 Jahre	50-100 KLOC	Betriebssystem
Sehr groß	100 - 1000	3-5 Jahre	1 MLOC	
Äußerst groß	2000 - 5000	5-10 Jahre	1-10 MLOC	SDI (Vermittlungssystem)

Abbildung 3.1: Projektumfang (Nach: Skriptum zur Vorlesung Softwaretechnik, WS93/94, Universität Karlsruhe)

mentierung beschränkt sich daher auf die Implementierung der Repository-Schnittstelle von VRCE, die als Brücke zwischen VRCE als RCE-Anwendung und Delta-V als Protokoll zur Kommunikation mit dem Dienstgeber dient. Der Dienstgeber basiert wiederum auf RCE als Revisionskontrollsystem. Abbildung 3.2 zeigt die sich ergebende Architektur als Übersicht.

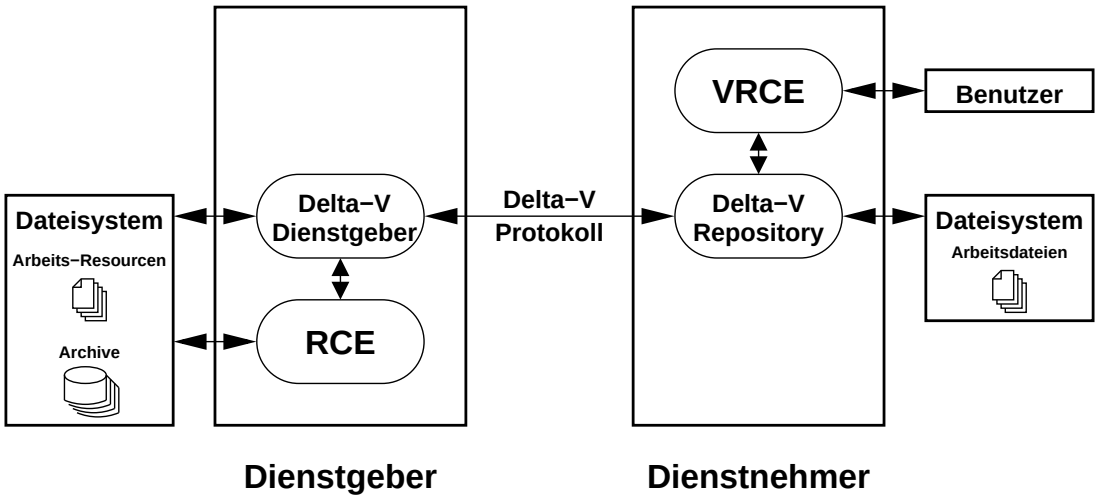

Abbildung 3.2: Architekturübersicht zu VRCE via Delta-V

3.1 Fernbenutzungsmodell von Delta-V

Delta-V ist eine Erweiterung des WebDAV-Protokolls, welches selbst eine Erweiterung des HTTP-Protokolls darstellt. Zum Verständnis des Fernbenutzungsmodells von Delta-V ist daher eine Kenntnis der Fernbenutzungsmodelle von WebDAV und HTTP erforderlich. In der Kommunikation zwischen Dienstgeber und Dienstnehmer unter Delta-V spielt XML eine Schlüsselrolle als Kommunikationssprache.

3.1.1 HTTP

Mit *Hypertext Transfer Protocol* (*HTTP*)[39] wird ein Protokoll des Internet auf Anwendungsebene für verteilte, kollaborative Informationssysteme bezeichnet, dessen Bedeutung in seiner Verbreitung, Plattformunabhängigkeit sowie der zahlreichen verfügbaren Anwendungen und der mit diesen gesammelten Erfahrungen liegt.

Die Fernbenutzung erfolgt unter HTTP nach einem Anfrage/Antwort-Paradigma. Die Anfrage enthält neben einer Anfragezeile mit Anfragemethode, Resourcenbezeichner (*URI*) und Protokollversion eine Nachricht gemäß dem MIME-Standard[36]. Die Nachricht selbst gliedert sich in einen Nachrichtenkopf mit Metadaten über die Nachricht und einen gegebenenfalls leeren Nachrichtenkörper mit den eigentlichen Nutzdaten. Die Antwort des Dienstgebers enthält neben einer Statuszeile mit Protokollversion, Statuskode und Statusbeschreibung ebenfalls eine Nachricht nach MIME-Standard. Abbildung 3.3 zeigt eine typische Anfrage eines Dienstnehmers zum Erhalt eines Dokuments und eine mögliche Antwort des Dienstgebers, der in diesem Beispiel ein HTML-Dokument zurückliefert.

```
>>Anfrage

GET /users/index.html HTTP/1.1
Host: www.foo.bar
Content-Length: 0

>>Antwort

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 50

<HTML>
  <BODY>
    Willkommen!
  </BODY>
</HTML>
```

Abbildung 3.3: Typische Kommunikation unter HTTP

Im Sinne eines Fernbenutzungsmodells kann man den Namen der HTTP-Methode als Namen einer Fernbenutzungsmethode und die Gesamtheit aus Anfrage-URI, Nachrichtenkopf und Nachrichtenkörper als Parameter eines Methodenaufrufes auffassen. Die Parameter können auch zur Diversifizierung von HTTP-Methoden verwendet werden, wenn die Anzahl an HTTP-Methodennamen gering gehalten werden soll, etwa um Namenskollisionen mit zukünftigen Erweiterungen des HTTP-Protokolls vorzubeugen.

Das HTTP-Protokoll selbst sieht mannigfaltige Möglichkeiten zur Erweiterung vor. Hierzu zählen beispielsweise die Definition neuer sowie die Erweiterung bestehender HTTP-Methoden und Nachrichtenkopfzeilen. Über die Kopfzeilen lassen sich Informationen insbesondere zur Kodierung und Parametrisierung des Nachrichtenkörpers transportieren. Beispiele für die Definition neuer Kopfzeilen und HTTP-Methoden finden sich in der nachfolgenden Beschreibung von WebDAV und Delta-V.

Das von HTTP verwendete Fernbenutzungsmodell ist insofern als asymmetrisch zu bezeichnen, als die Auslösung einer Kommunikation stets durch den Dienstnehmer in Form einer Anfrage erfolgt, die der Dienstgeber in einer Antwort erwidert; die Auslösung einer Kommunikation durch den Dienstgeber etwa zur Benachrichtigung eines Dienstnehmers über eine Änderung des Zustandes des Dienstgebers ist nicht vorgesehen.

3.1.2 XML

Mit *Extensible Markup Language*[27], kurz *XML*, wird eine Klasse von Objekten beschrieben, die als *XML-Dokumente* bezeichnet werden. XML beschreibt auch zum Teil das Verhalten von Software, die XML-Dokumente bearbeitet. Als Anwendung von SGML[25] sind XML-Dokumente stets SGML-konform. XML-Dokumente bestehen aus *Entitäten*, die zeichenorientierte Benutzerdaten gleichermaßen wie Strukturdaten *markup* enthalten können. Die Strukturdaten gliedern ein XML-Dokument in *XML-Elemente* und bestimmen somit die Speicheranordnung und logische Struktur des Dokumentes. Durch Angabe einer Dokumententypdeklaration (*Document Type Declaration*, *DTD*) besteht die Möglichkeit, XML-Dokumenten Einschränkungen hinsichtlich der Speicheranordnung und logischen Struktur aufzuerlegen.

Die XML-Spezifikation definiert zwei Stufen zur Korrektheitsprüfung von XML-Dokumenten: die Wohlgeformtheit und die Gültigkeit. Ein XML-Dokument ist *wohlgeformt* (*well-formed*), wenn – stark vereinfachend ausgedrückt – die syntaktische Korrektheit des Dokuments in etwa so weit garantiert ist, wie sie ohne Kenntnis der Dokumententypdeklaration für das gegebene Dokument möglich ist. Um *gültig* (*valid*) zu sein, muß ein XML-Dokument darüberhinaus als deutlich stärkere Zusicherung syntaktisch korrekt im Sinne der DTD sein.

Namensräume in XML sollen Kollisionen bei der Bezeichnung von XML-Elementen vermeiden helfen. Da Bezeichner von Namensräumen mitunter lang sein können, werden sie im Dokument einmalig deklariert und an den betreffenden XML-Elementen durch einen bei der Deklaration angegebenen, üblicherweise kurzen Präfix referenziert. Abbildung 3.4 zeigt ein typisches XML-Dokument. Es deklariert die beiden Namensräume `urn:loc.gov:books` und `urn:ISBN:0-395-36341-6`, ordnet ihnen die Präfixe `bk` und `isbn` zu und verwendet diese Präfixe für die Elemente `book`, `title` und `number`.

```
<?xml version="1.0"?>
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

Abbildung 3.4: Typisches XML-Dokument mit Namensraum-Deklarationen

3.1.3 WebDAV

WebDAV[43], oder kurz DAV (*Distributed Authoring and Versioning*), ist ein Protokollstandard, der das Ergebnis von Bemühungen widerspiegelt, HTTP mit dem Ziel zu erweitern, Autorenschaft im World Wide Web unter Nutzung von HTTP räumlich verteilt durchführen zu können. Zwar bietet bereits HTTP selbst rudimentäre Funktionen wie die HTTP-Methoden `GET`, `PUT` und `DELETE` für den Dokumentenaustausch zwischen Dienstgeber und Dienstnehmer und das Löschen von Resourcen. Bei verteilter Autorenschaft kann das unkoordinierte Zusammentreffen von Aktualisierungen mittels `PUT` allerdings zum ungewollten Überschreiben von Dokumenten und somit zum Verlust von Daten führen (*Problem der verlorengegangenen Aktualisierung*, *lost update problem*). Daher sind Mechanismen zur Koordinierung von Zugriffen auf Resourcen notwendig. Auf diesen Mechanismen aufbauend läßt sich ferner eine verteilte Revisionsverwaltung realisieren, wie sie von Delta-V definiert wird.

Desweiteren bieten aktuelle Dienstgeberimplementierungen zahlreiche Konfigurationsmöglichkeiten, die häufig nur durch Änderungen der Konfigurationsdateien des Dienstgebers selbst gesteuert werden können und – schlimmstenfalls – erst nach einem Neustart des Dienstgebers wirksam werden. Wünschenswert wäre daher die Möglichkeit, Änderungen an der Konfiguration eines Dienstgebers in standardisierter Form ebenfalls über HTTP abwickeln zu können. Zu den wichtigsten dieser Konfigurationsmerkmale zählen die Verwaltung von Zugriffsrechten auf geschützte Bereiche des Namensraums des Dienstgebers sowie die Einrichtung von Umleitungen (*redirections*) und Bindegliedern (*bindings*). Bei einer Umleitung beantwortet der Dienstgeber eine Anfrage mit der Aufforderung an den Dienstnehmer, dieselbe Anfrage unter einer anders benannten Resource zu wiederholen. Bei einem Bindeglied wird die Anfrage dienstgeberintern an eine andere Resource weitergeleitet, ohne daß der Dienstnehmer hierüber benachrichtigt wird.

WebDAV selbst definiert eine Infrastruktur, auf der aufbauend weitere Protokolle die Umsetzung der angestrebten Ziele verfolgen. Delta-V ist eines dieser Protokolle. Weitere, wie Delta-V ebenfalls noch in der Entwicklung befindliche Protokolle, setzen die Konfiguration des Dienstgebers durch entfernte Zugriffe um. Dazu zählen das Protokoll *WebDAV Bindings*[53], welches Bindeglieder be-

handelt, daneben *WebDAV Redirect Reference Resources*[52], das Umleitungen über einen neuen Resourcentyp zu definieren versucht, sowie das *WebDAV ACLs Protocol*[44], welches Zugriffskontrolle auf der Basis von Zugriffslisten realisiert. Einen tieferen Einblick in die Entwurfsüberlegungen zu WebDAV und der darauf aufbauenden Protokolle finden sich in einem eigenen, informellen RFC[41]. Die von WebDAV eingeführte Infrastruktur umfaßt unter anderem die Konzepte der Sammlung von Resourcen und der Verwaltung von Eigenschaften von Resourcen, die Organisation und Verwaltung von Namensräumen, das Konzept der Sperre auf einer Resource sowie die Definition von Mindestanforderungen zur Authentifizierung von Dienstnehmern.

3.1.3.1 Sammlungen

Ein Resourcenbezeichner wie `http://www.foo.bar/abc/def/ghi.html` bezeichnet unter HTTP die Resource mit dem Pfad `/abc/def/ghi.html` auf dem Wirtsrechner `www.foo.bar`. Geschah bereits unter HTTP/1.0 die Interpretation des Pfades bei den meisten Dienstgeberimplementierungen in der Art eines Dateipfades eines hierarchischen Dateisystems, so war dies eine zwar durchaus mögliche und auch übliche Interpretation. Das HTTP-Protokoll selbst macht bezüglich der Interpretation eines solches Pfades jedoch keinerlei Aussagen.

Erst WebDAV führt mit dem Prinzip der Sammlung (*collection*) als spezielle Resource das Konzept eines hierarchischen Namensraumes ein, ähnlich dem Konzept der Dateipfade unter einem hierarchischen Dateisystems aus Dateien und Verzeichnissen als spezielle Dateien. Der Begriff der Sammlung ist essentiell zur Definition der auf Sammlungen operierenden und nachfolgend beschriebenen Methoden wie `PROPFIND` und `MKCOL`. Die Wahl der Bezeichnungen Resource und Sammlung abstrahiert von der konkreten Implementierung des hierarchischen Namensraums auf der Basis eines Dateisystems; anstelle eines Dateisystems kann beispielsweise genausogut eine Datenbank die Speicherung und Verwaltung der Resourcen und Sammlungen übernehmen.

3.1.3.2 Eigenschaften

Ein schwerer Mangel bestehender HTTP-Dienstgeber ohne WebDAV-Unterstützung besteht im Fehlen von über HTTP konfigurierbarer Metadaten über Resourcen. Ein prominentes Beispiel hierfür ist die Nachrichtenkopfzeile `Content-Type`, die bei Übermittlung eines Dokumentes Typinformation zu dessen Inhalt bereitstellen soll. Ein Browser beispielsweise wertet diese Information aus, um zu entscheiden, ob ein empfangenes Dokument als Text, HTML oder Bild angezeigt werden soll, oder ob, etwa beim Herunterladen eines ausführbaren Programmes, der Benutzer aufgefordert werden soll, einen Pfad zur Speicherung des empfangenen Dokumenten einzugeben. Die Typinformation, die als `Content-Type` übermittelt wird, wird vom Dienstgeber beispielsweise

anhand der Namensendung des Pfades der Resource bestimmt; die Zuordnung von Namensendung zu Typinformation wird üblicherweise durch Konfiguration des Dienstgebers bestimmt. Ein falsch bestimmter Typ kann beispielsweise dazu führen, daß ein Browser den Binärkode eines Programmes anzeigt, anstatt den Benutzer zur Angabe eines Pfades zur Speicherung aufzufordern. Zwar kann beim Aufladen einer Resource auf einen Dienstgeber mittels `PUT` dem Dienstgeber mit der gleichen Kopfzeile die Typinformation geliefert werden; der Dienstgeber muß dann aber solche Metadaten in irgendeiner Form verwalten.

Das Konzept der Eigenschaften von WebDAV bemüht sich, die Verwaltung der Metadaten zu standardisieren. Jede Resource unter WebDAV, einschließlich der Sammlungen, muß die Verwendung von Eigenschaften unterstützen. WebDAV definiert eine Eigenschaft als ein XML-Element beliebigen XML-konformen Inhaltes; der Name des XML-Elementes wird zur Bezeichnung der Eigenschaft verwendet. Neben einigen vordefinierten Eigenschaften, die eine zu WebDAV konforme Implementierung zwingend unterstützen muß, wird die Unterstützung beliebiger benutzerdefinierbarer Eigenschaften empfohlen. Da gerade bei benutzerdefinierten Eigenschaften die Gefahr von Namenskollisionen besteht, sieht WebDAV die Verwendung des Konzeptes der Namensräume von XML[28] bei der Bezeichnung von Eigenschaften vor. Die von WebDAV eingeführten HTTP-Methoden `PROPPATCH` sowie `PROPFIND` dienen zum Setzen, Ändern oder Entfernen sowie zum Abfragen von Eigenschaften. Das Abfragen von Eigenschaften umfaßt neben dem gezielten Abfragen durch Angabe des Suchschlüssels auch die Möglichkeit, alle Eigenschaften einer Resource oder eine Aufzählung der Suchschlüssel aller vorhandenen Eigenschaften zu erfragen. Durch Angabe der von WebDAV eingeführten Kopfzeile `Depth` läßt sich bestimmen, ob die Abfrage von Eigenschaften sich auf eine einzelne Resource (`Depth: 0`), auf alle Resourcen einer Sammlung (`Depth: 1`), oder rekursiv auf allen Resourcen der Namenshierarchie unterhalb einer Sammlung (`Depth: infinity`) bezieht. Dies ermöglicht insbesondere auch die Exploration des Namensraumes eines Dienstgebers.

Abbildung 3.5 zeigt ein typisches Beispiel zur Abfrage aller Eigenschaften einer einzelnen Resource. Neben Eigenschaften des mit dem Präfix `A:` abgekürzten Namensraumes `DAV:` enthält die Resource aus dem Beispiel auch einige benutzerdefinierte Eigenschaften eines mit `RCE:` bezeichneten Namensraumes, der mit dem Präfix `B:` abgekürzt wird.

```
>>Anfrage

PROPFIND /dfs/foo.java HTTP/1.1
Host: Boston.ira.uka.de:4711
Content-Length: 0

>>Antwort

HTTP/1.0 207 Multi-Status
Connection: close
```

```
Content-Type: text/xml
Content-Length: 1418
Date: Tue, 15 Aug 2000 14:00:33 GMT
Server: JWebDAV-Server/0.3 mod_http/0.3 mod_dav/0.1 mod_deltav/0.1

<?xml version="1.0"?>
<A:multistatus xmlns:A="DAV:">
  <A:response>
    <A:href>http://Boston.ira.uka.de:4711/dfs/foo.java</A:href>
    <A:propstat>
      <A:prop xmlns:B="RCE:">
        <A:auto-version>F</A:auto-version>
        <A:working-resource-id-set />
        <A:getcontenttype>text/plain</A:getcontenttype>
        <A:displayname>/dfs/foo.java</A:displayname>
        <A:resourcetype>
          <A:versioned-resource />
        </A:resourcetype>
        <A:getlastmodified>Mon, 17 Jul 2000 14:53:22 GMT</A:getlastmodified>
        <A:defaulttarget>revision-id 1.2</A:defaulttarget>
        <A:initial-revision>
          <A:href>http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.1</A:href>
        </A:initial-revision>
        <B:access-list>reuter(RWX),jjh(RW),schullig(R)</B:access-list>
        <B:arch-keys>ON0</B:arch-keys>
        <A:creationdate>2000-07-17T16:53:22.0+00.00</A:creationdate>
        <A:linear>F</A:linear>
        <A:revision-set>
          <A:href>http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.1</A:href>
          <A:href>http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.2</A:href>
        </A:revision-set>
        <A:getcontentlength>2052</A:getcontentlength>
      </A:prop>
      <A:status>HTTP/1.0 200 OK</A:status>
    </A:propstat>
  </A:response>
</A:multistatus>
```

Abbildung 3.5: Typische Anwendung von `PROPFIND` zur Abfrage aller Eigenschaften einer Resource

WebDAV fordert die Atomizität der Methode `PROPPATCH` in dem Sinne, daß im Fehlerfalle der Zustand des Dienstgebers von der Bearbeitung der Anfrage unberührt bleiben muß. Praktisch angewendet bedeutet dies, daß, wenn das Ändern einer einzelnen Eigenschaft fehlschlägt, alle zu diesem Zeitpunkt bereits erfolgreich durchgeführten Modifikationen an Eigenschaften verworfen werden müssen. Dies läuft auf die Bereitstellung einer Transaktionsverwaltung hinaus; vielfache Änderungen von Eigenschaften können bei einer transaktionsbasierten Verwal-

tung in einer einzigen Operation festgeschrieben oder verworfen werden.

Zu den von WebDAV vordefinierten Eigenschaften gehören unter anderem `DAV:contenttype` zur Lösung des eingangs beschriebenen Problems und `DAV:creationdate` zur Speicherung des Erzeugungszeitpunktes einer Resource. Die Eigenschaft `DAV:resourcetype` zur Beschreibung des Resourcentyps ermöglicht dem Dienstnehmer im Zusammenspiel mit den auf WebDAV aufbauenden Protokollen das Erkennen von Sammlungen, Bindegliedern, Umleitungen und versionierten Resourcen. Weitere vordefinierte Eigenschaften werden zur Verwaltung der nachfolgend beschriebenen Sperren verwendet.

3.1.3.3 Organisation und Verwaltung von Namensräumen

WebDAV spezifiziert desweiteren die HTTP-Methode `MKCOL` zum Erzeugen von Sammlungen. Gelöscht werden können Sammlungen gegebenenfalls mit dem bereits in HTTP definierten Befehl `DELETE`. Zum Kopieren oder Verschieben einzelner Resourcen oder ganzer Hierarchien von Resourcen definiert WebDAV die HTTP-Methoden `COPY` und `MOVE`.

3.1.3.4 Sperren

Um das Problem des eingangs geschilderten ungewollten Überschreibens zu lösen, führt WebDAV die Möglichkeit ein, Resourcen mit einer Sperre zu versehen. Auf der Basis solcher Sperren lassen sich vielfältige Mechanismen zur Synchronisation konkurrierender Zugriffe bis hin zu einer ausgereiften Transaktionsverwaltung aufbauen. Zur Realisierung führt WebDAV die HTTP-Methoden `LOCK` und `UNLOCK` ein, mit denen Sperren gesetzt und wieder gelöscht werden können. Der Status einer Sperre kann über die spezielle Eigenschaft `DAV:lockdiscovery` ermittelt werden. Durch Setzen einer Sperre auf eine nicht-existente Resource (*null resource*) kann ein Resourcenbezeichner reserviert werden. WebDAV sieht die Verwendung verschiedener, auch benutzerdefinierbarer Sperren vor und definiert selbst eine Schreibsperre. Das Konzept der Sperren unter WebDAV gestaltet sich verhältnismäßig komplex; allein gut 20% des Protokolls befassen sich direkt mit Sperren. Die Unterstützung von Sperren bleibt einer Implementierung von WebDAV freigestellt.

3.1.3.5 Authentifizierung

Der Authentifizierung kommt unter WebDAV eine besondere Bedeutung zu, da wegen der vielfältigen Möglichkeiten zur Manipulation von Resourcen ein Zugriffsschutz erforderlich ist. Ferner erfordert die Anwendung und Verwaltung der Sperren die Authentifizierung des Zugreifers. Das HTTP-Protokoll sieht Digest Authentication[40] als optionalen Authentifizierungsmechanismus vor. WebDAV definiert die Unterstützung von Digest Authentication als Erfordernis für jede

konforme Implementation des Protokolls, es sei denn, die Verbindung zwischen Dienstgeber und Dienstnehmer kann als sicher gelten.

3.1.4 Delta-V

Delta-V ist eine derzeit noch in der Entwicklung befindliche Erweiterung von WebDAV zur Spezifikation verteilter Versionskontrolle über das WWW. Wichtige Ziele sind dabei die Unterstützung von revisionierten Daten beliebigen Typs, Internationalisierung, starke Authentifizierung und sichere Datenübermittlung; ferner soll auch WWW-Dienstnehmern, die Delta-V nicht unterstützen, weitreichende Teilhabe ermöglicht werden. Das Delta-V-Protokoll führt hierzu ein Versionsmodell ein und definiert auf WebDAV aufbauend neue Methoden, Kopfzeilen Statuskodes und Eigenschaften.

Die Einbettung von Delta-V in WebDAV respektive HTTP zielt auf Plattformunabhängigkeit der Implementierungen von Dienstgeber und Dienstnehmer und somit auf ein abstraktes, standardisiertes Versionsmodell als zentralen Kern eines jeden Revisionskontroll- oder Konfigurationsmanagementsystems, um mit möglichst vielen der zahlreichen bereits bestehenden Systeme verträglich zu sein. Diese Verträglichkeit läßt sich mit den folgenden zwei Aspekten umreißen, die zur späteren Bewertung von Delta-V als wesentlich erscheinen.

Einerseits ist es für eine allgemeine Akzeptanz von Delta-V unumgänglich, die Funktionalität dieses Protokolls insofern zu abstrahieren und minimieren, daß möglichst viele der bestehenden Systeme vom einfachen Revisionskontrollsystem bis hin zum komplexen Konfigurationsmanagementsystem als Basis zur Implementierung eines Delta-V-Dienstgebers verwendet werden können. Dies erfordert, daß Delta-V als Schnittstelle einen möglichst allgemein gehaltenen, kleinen Kern von Funktionen definieren muß, der sich auf möglichst viele der bestehenden Systeme abbilden läßt. Andererseits muß Delta-V eine hinreichend umfangreiche Funktionalität bieten, damit Dienstnehmer darauf aufbauend auch komplexere Revisionskontrollsysteme und Konfigurationsmanagementsysteme implementieren können.

Es sei angemerkt daß den Diskussionsbeiträgen zur Delta-V-Arbeitsgruppe[48] zu entnehmen ist, daß die beteiligten Autoren sich der genannten Aspekte offenbar bewußt sind und eine Verträglichkeit zu möglichst vielen bestehenden Systemen anstreben. Zumindest ein Bemühen um Verträglichkeit von Delta-V zum System Rational ClearCase[9] darf angesichts der Tatsache unterstellt werden, daß einer der beiden Autoren als Mitarbeiter von Rational die Entwicklung von Delta-V maßgeblich beeinflußt. Aber auch die Verträglichkeit zum System Perforce, welches Verästelungen von Revisionen vor dem Anwender verbirgt und somit scheinbar ein eigenwilliges Versionsmodell verwendet, ist nach Auskunft einer der Autoren gegeben. Die Verträglichkeit mit RCE wird in einem nachfolgenden Abschnitt untersucht werden.

Um die Vielfalt an Systemen vom einfachen Revisionskontrollsystem bis zum

komplexen Konfigurationsmanagementsystem bedienen zu können, ist Delta-V in zwei Stufen definiert. Die erste, als *basic versioning* bezeichnete Stufe, enthält alle Erfordernisse, die ein Delta-V-konformer Dienstgeber minimal implementieren muß. Eine zweite, als *advanced versioning* bezeichnete Stufe, beschreibt eine Vielzahl optionaler Dienste bis hin zur Konfigurationskontrollmechanismen, die der Dienstgeber zusätzlich anbieten kann.

Die in dieser Arbeit betrachtete Entwurfsversion 04.5 von Delta-V bemüht sich, alle Aspekte der reinen Versionskontrolle von denen der optionalen Erweiterungen zu trennen. Aus Gründen der zeitlichen Beschränkung einerseits, wie auch wegen des Funktionsumfanges von RCE als Versionsplattform andererseits, welcher im wesentlichen dem der reinen Versionskontrolle entspricht, wird im folgenden nur die reine Versionskontrolle von Delta-V betrachtet. Sie unterstützt die Versionierung weitgehend voneinander unabhängiger Resourcen und erlaubt Autoren, Revisionen einer Resource zu erzeugen, mit symbolischen Namen zu versehen und auf sie zuzugreifen. Die Einbettung von Delta-V in WebDAV respektive HTTP ermöglicht mit Einschränkungen auch solchen Dienstnehmern die Versionierung, die Delta-V nicht explizit unterstützen.

Delta-V verwendet ein Versionsmodell nach dem Paradigma des Ein- und Ausbuchens einer Resource, das mit dem folgenden typischen Szenario kurz erläutert werden möge. Eine (*versionierbare Resource*) ist eine Resource, die unter Revisionskontrolle gestellt werden kann. Dies kann beispielsweise eine mit der HTTP-Methode `PUT` zum Dienstgeber übertragene Datei sein. Eine versionierbare Resource wird mit der unter Delta-V eingeführten HTTP-Methode `VERSION` in eine *versionierte Resource* umgewandelt; sie enthält die vormalige versionierbare Resource als initiale Revision. Mit der ebenfalls in Delta-V eingeführten HTTP-Methode `CHECKOUT` wird im Namensraum des Dienstgebers eine *Arbeits-Resource* erzeugt, die eine Kopie einer Revision der versionierten Resource enthält. Ein Dienstnehmer kann wie jeder gewöhnliche Browser mit der HTTP-Methode `GET` die Arbeits-Resource vom Dienstgeber herunterladen, um Änderungen an ihr durchzuführen. Die geänderte Arbeits-Resource kann mit `PUT` wieder zurück auf den Dienstgeber aufgespielt werden. Ein anschließendes `CHECKIN` auf der versionierten Resource mit der Arbeits-Resource als Parameter führt zum Einbuchen der geänderten Arbeits-Resource als neue Revision in die versionierte Resource. Es bleibt festzuhalten, daß die Methoden `CHECKIN` und `CHECKOUT` auf Resourcen operieren, die sich auf dem Dienstgeber befinden; für den Transport der Resourcen zwischen Dienstgeber und Dienstnehmer zeichnet der Dienstnehmer verantwortlich, etwa durch Anwendung der HTTP-Methoden `GET` beziehungsweise `PUT`.

Immer wenn durch Umwandeln einer versionierbaren Resource in eine versionierte Resource oder Einbuchen einer Arbeits-Resource eine neuen Revision entsteht, wird für diese vom Dienstgeber ein *dauerhafter Resourcenbezeichner* (*stable URL*) erzeugt. Über diesen Resourcenbezeichner kann beispielsweise mittels `GET` oder `PROPPATCH` auf die Revision wie auf eine eigenständige Resource zugegriffen werden. Der Resourcenbezeichner wird mit dem Attribut dauerhaft

beschrieben um auszudrücken, daß selbst die Umbenennung oder Verschiebung der zugehörigen versionierten Resource etwa mittels `MOVE` nicht zu einer Änderung des dauerhaften Resourcenbezeichners führt. Es sei ferner angemerkt, daß Resourcenbezeichner generell über die jeweilige versionierte Resource hinausgehend global eindeutige Bezeichner sind.

Delta-V verwendet das Konzept der Eigenschaften von WebDAV, um spezielle Eigenschaften der unter Delta-V neu eingeführten Resourcentypen versionierbare Resource, versionierte Resource, Revision und Arbeits-Resource zu spezifizieren. Sie umfassen im wesentlichen die aus der Sicht eines Dienstnehmers notwendigen Metadaten der beteiligten Resourcentypen. Die Eigenschaften `DAV:author` und `DAV:comment` zur Angabe von Autor einer Resource und Kommentar zu einer Resource werden auf allen genannten Resourcentypen definiert. Die auf Revisionen definierten Eigenschaften `DAV:revision-id`, `DAV:revision`, `DAV:predecessor-set`, `DAV:successor-set` und `DAV:working-resource-id-set` zusammen mit der auf versionierten Resourcen definierten Eigenschaft `DAV:revision-set` und der Eigenschaft `DAV:predecessor-set` von Arbeits-Resourcen ermöglichen dem Dienstnehmer das Auffinden beziehungsweise die Navigation durch alle Revisionen und Arbeits-Resourcen einer versionierten Resource. Bei Revisionen werden zusätzlich in den Eigenschaften `DAV:label-set` und `DAV:checkin-date` Namensaliase und Zeitpunkt des Einbuchens gespeichert.

Mit der in Delta-V neu eingeführten HTTP-Methode `REPORT` können, ähnlich wie mit `PROPFIND`, Eigenschaften von Resourcen abgefragt werden. Während sich bei `PROPFIND` die Menge der zu betrachtenden Resourcen allerdings aus der Hierarchie der Resourcen gemäß ihrer Resourcenbezeichner und der Angabe der Baumtiefe über die HTTP-Kopfzeile `Depth` ergibt, basiert bei `REPORT` die Ermittlung der zu betrachtenden Resourcen auf der Auswertung von Eigenschaften von Resourcen, die Mengen von Resourcenbezeichnern definieren. Das wichtigste Beispiel hierfür ist die Eigenschaft `DAV:revision-set` einer versionierten Resource, hinter der sich eine Menge dauerhafter Resourcenbezeichner verbirgt, die zu den Revisionen der versionierten Resource führen. Eine einzelne Anwendung des `REPORT`-Befehls auf diese Eigenschaft führt daher effektiv zum selben Ergebnis wie die Anwendung des Befehls `PROPFIND` auf alle in dieser Eigenschaft aufgezählten Revisionen. Daher eignet sich `REPORT` speziell zur gebündelten Abfrage aller für die Erstellung eines Logbuchs einer versionierten Resource benötigten Eigenschaften in einer einzigen Anfrage; dies erklärt zudem die Bezeichnung dieser Methode. In ähnlicher Weise lassen sich durch Anwendung von `REPORT` auf die Eigenschaft `DAV:working-resource-id-set` in einer einzigen Anfrage Daten zu allen Arbeits-Resourcen einer Revision erhalten.

Abbildung 3.6 zeigt eine typische Anfrage bei Verwendung der Methode `REPORT` und die zugehörige Antwort. Das Beispiel beschränkt sich auf das Erfragen der beiden Eigenschaften `DAV:revision-id` und `DAV:creationdate` bei Anwendung auf eine versionierte Resource mit den

beiden Revisionen `http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.1` und `http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.2`, um den Umfang der Antwort überschaubar zu halten; in der Praxis wird eine typische Antwort zumeist einen deutlich größeren Umfang aufweisen.

```
>>Anfrage

REPORT /dfs/foo.java HTTP/1.1
Host: Boston.ira.uka.de:4711
Target-Selector: versioned-resource
Content-Length: 185

<?xml version="1.0" encoding="utf-8" ?>
<D:property-report xmlns:D="DAV:">
  <D:revision-set>
    <D:revision-id/>
    <D:creationdate/>
  </D:revision-set>
</D:property-report>

>>Antwort

HTTP/1.0 207 Multi-Status
Connection: close
Content-Type: text/xml
Content-Length: 972
Date: Tue, 15 Aug 2000 15:10:14 GMT
Server: JWebDAV-Server/0.3 mod_dav/0.1 mod_http/0.3 mod_deltav/0.1

<?xml version="1.0"?>
<A:multistatus xmlns:A="DAV:">
  <A:response>
    <A:href>http://Boston.ira.uka.de:4711/dfs/foo.java</A:href>
    <A:propstat>
      <A:prop>
        <A:revision-set>
          <A:response>
            <A:href>http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.1</A:href>
            <A:propstat>
              <A:prop>
                <A:revision-id>1.1</A:revision-id>
                <A:creationdate>2000-08-07T12:54:35.0-00:00</A:creationdate>
              </A:prop>
              <A:status>200 OK</A:status>
            </A:propstat>
          </A:response>
          <A:response>
            <A:href>http://Boston.ira.uka.de:4711/dfs/ev2qn7;1.2</A:href>
            <A:propstat>
              <A:prop>
                <A:revision-id>1.2</A:revision-id>
```

```
            <A:creationdate>1970-01-01T00:00:00.0-00:00</A:creationdate>
          </A:prop>
          <A:status>200 OK</A:status>
        </A:propstat>
      </A:response>
    </A:revision-set>
  </A:prop>
  <A:status>200 OK</A:status>
</A:propstat>
</A:response>
</A:multistatus>
```

Abbildung 3.6: Typische Anfrage und zugehörige Antwort bei Anwendung der Methode `REPORT`

3.2 Modellierung der Java-Schnittstelle von RCE über Delta-V

Die mit dieser Arbeit verbundene Implementierung verwendet RCE als Versionskontrollsystem; somit stellt sich die Frage nach der Verträglichkeit des Versionsmodells von Delta-V zu dem von RCE. Diese Frage stellt sich einerseits bei der Implementierung eines Delta-V-Dienstgebers auf der Basis von RCE. Andererseits ist sie auch bei der Implementierung eines Dienstnehmers wie VRCE von Interesse, der auf der Basis eines Delta-V-Dienstgebers die Funktionalität von RCE emulieren soll. Die Frage nach der Verträglichkeit stellt sich vor allem bezüglich dreier Bereiche der Versionsmodelle von RCE versus Delta-V. Dies sind zum einen die Schablonen von RCE, die bijektiv auf die Arbeits-Resourcen von Delta-V abgebildet werden sollen. Zum zweiten ist es die Art und Weise, in der RCE und Delta-V Verästelungen handhaben. Schließlich möchte man die Attribute von RCE-Archiven und -Revisionen durch Eigenschaften von Resourcen unter Delta-V und umgekehrt darstellen.

3.2.1 Schablonen und Arbeits-Resourcen

Eines der Ziele von Revisionierung ist, das Problem verlorengegangener Aktualisierungen bei konkurrierenden Zugriffen auf einen gemeinsamen Datenbestand zu vermeiden. Daher verwenden Revisionskontrollsysteme üblicherweise Mechanismen, die ein unbeabsichtigtes Überschreiben von Daten ausschließen sollen. Das Konzept des kontrollierten Aus- und Einbuchens dient hierbei als Rahmenwerk, auf welchem sich durch Zurückweisen einer Anforderung zur Einbuchung konkurrierende Aktualisierungen ausschließen lassen.

Als Kriterium zum Zurückweisen einer Einbuchungsanforderung kommen verschiedene Möglichkeiten in Betracht. Die wohl einfachste Methode besteht darin, einen Sperrmechanismus auf Revisionen einzuführen: wird eine Revision ausgebucht, um an ihr Änderungen vorzunehmen und sie später wieder einzubuchen, so wird die bestehende Revision mit einer Sperre markiert, um anzuzeigen, daß sich die betreffende Revision in Bearbeitung findet. Wird versucht, eine mit einer solchen Sperre versehene Revision erneut auszubuchen, so wird die Anforderungen zur Ausbuchung zurückgewiesen. RCS verwendet einen solchen Sperrmechanismus.

Häufig sollen bereits beim Ausbuchen Informationen über die zukünftige Revision vorläufig (z.B. Revisions-Kommentar) oder endgültig (z.B. Revisions-Identifikation) festgelegt werden. Es liegt daher nahe, eine Schablone (*template*) für die zukünftige Revision anzulegen und die Informationen dort zu speichern. Wird die neue Revision durch Einbuchung festgeschrieben, stehen die Daten aus der Schablone bereits zur Verfügung. RCE verfolgt diesen schablonenbasierten Ansatz.

WebDAV unterstützt, wie bereits beschrieben, die Speicherung beliebiger resourcengebundener Eigenschaften. Als Erweiterung von WebDAV ist es für Delta-V naheliegend, diese Funktionalität zu nutzen. Wird eine Resource auf einem Delta-V-Dienstgeber ausgebucht, so erscheint sie dienstgeberseitig als Arbeits-Resource (*working resource*). Die Informationen, die beim schablonenbasierten Ansatz in der Schablone gespeichert werden, stehen in der Regel in direktem Zusammenhang mit der Arbeits-Resource; ein Beispiel hierfür ist die Angabe eines Kommentars zu den in der Arbeits-Resource vorzunehmenden oder bereits vorgenommenen Änderungen. Daher liegt es nahe, solche Informationen nicht in einer Schablone, sondern als Eigenschaften der Arbeits-Resource zu speichern. Wenn es überdies möglich ist, zu einer gegebenen Revision alle ausgebuchten Arbeits-Resourcen aufzuspüren, dann kann auf Schablonen ganz verzichtet werden. Delta-V folgt diesem Ansatz durch die Verwendung mit Eigenschaften versehener Arbeits-Resourcen.

Die im Rahmen dieser Arbeit verfolgte Implementierung verwendet RCE als Revisionskontrollsystem; daher bleibt nun zu prüfen, ob der Ansatz von Delta-V auf der Basis von Arbeits-Resourcen mit dem schablonenbasierten Ansatz von RCE verträglich ist, d.h. ob der Ansatz von Delta-V auf den von RCE abgebildet werden kann.

Da Delta-V im wesentlichen die Eigenschaften, die RCE in den Schablonen speichert, als Eigenschaften von Arbeits-Resourcen realisiert, liegt es zunächst nahe, in einer konkreten Implementierung dem Ansatz von Delta-V zu folgen: erst mit dem Einbuchen werden die mit den Arbeits-Resourcen verbundenen Eigenschaften in die außerhalb des Dienstgebers nicht sichtbare Schablone übernommen, welche sodann zu einer neuen Revision mutiert.

Diese Abbildung allein erlaubt es noch nicht, an bestehenden Revisionen Metadaten wie beispielsweise einen Revisionskommentar jederzeit abändern zu

können. Auch hierfür verwendet Delta-V das Modell der Eigenschaften einer Resource, in diesem Falle einer Revisions-Resource. Eine konkrete Implementierung muß daher entsprechende Anfragen an den Dienstgeber auf Änderungsanforderungen auf Revisionen in RCE-Archiven abbilden. Dann jedoch kann man mit nur geringem zusätzlichen Aufwand auch Änderungen von Eigenschaften an Arbeits-Resourcen direkt auf Schablonen abbilden.

Für eine solche direkte Abbildung auf Schablonen muß allerdings sichergestellt werden, daß zu jeder Arbeits-Resource eine Schablone existiert und aufgefunden werden kann. Delta-V bietet die Möglichkeit, eine Arbeits-Resource einzubuchen, ohne die Resource dabei zu entfernen. Dies erlaubt eine Folge von Einbuchungen ohne dazwischenliegende Ausbuchung. Aus der Forderung, daß zu jeder Arbeits-Resource eine Schablone existieren muß, folgt in diesem Falle, daß der Dienstgeber nach einer Einbuchung *implizit* eine Ausbuchung vornehmen muß. Von der impliziten Ausbuchung zu unterscheiden ist das *explizite* Ausbuchen einer Revision, das vom Dienstnehmer ausdrücklich als Anforderung an den Dienstgeber übermittelt wird, um eine *weitere* Arbeits-Resource zu erzeugen.

Mit der beschriebenen Verfahrensweise wird sichergestellt, daß zu jeder Arbeits-Resource immer eine Schablone im zugehörigen RCE-Archiv existiert. Wird an den Dienstgeber eine Anforderung zum Löschen einer Arbeits-Resource mittels `DELETE` gerichtet, so muß zur Wahrung der Konsistenz der Archive implizit auch die zugehörige Schablone gelöscht werden. Das Löschen einer Arbeits-Resource kann auch grundsätzlich durch den Dienstgeber verweigert werden, da die Methode `UNCHECKOUT` eigentlich für diesen Zweck bestimmt ist. Die Implementierung der HTTP-Methode `DELETE` muß in jedem Falle entsprechend angepaßt beziehungsweise erweitert werden.

Beim Verschieben einer Arbeits-Resource mit der in WebDAV eingeführten HTTP-Methode `MOVE` müssen die in den RCE-Archiven gespeicherten Pfade angepaßt werden. Die Semantik der Anwendung der ebenfalls in WebDAV eingeführten HTTP-Methode `COPY` auf Arbeits-Resourcen wird in Delta-V nicht festgelegt. Denkbar sind die Abweisung des Befehls als Fehler, die Erstellung einer Kopie durch implizites Ausbuchen einer neuen Arbeits-Resource oder das Kopieren der Arbeits-Resource, wobei die Kopie ihre Eigenschaft als Arbeits-Resource verliert.

Weiterhin zu beachten ist, daß eine Schablone keine Revision im Sinne von Delta-V darstellt. Delta-V bietet beispielsweise die Möglichkeit, alle Revisionen einer revisionierten Resource aufzulisten, oder alle Vorgänger oder Nachfolger einer Revision zu erfragen. Schablonen müssen daher bei derartigen Anfragen vom Dienstgeber ausgenommen werden, und der Dienstnehmer muß, wenn er ein schablonenbasiertes Modell verwendet, zusätzlich zu den Revisionen explizit auch Arbeits-Resourcen ermitteln.

3.2.2 Verästelungen (Branching)

Das Revisionsmodell von Delta-V führt, ähnlich wie RCE, die binären Relationen *Vorgänger* und *Nachfolger* ein. Die durch eine Sequenz von Ausbuchungen und Einbuchungen entstehende Struktur von Revisionen im Sinne dieser Relationen kann im allgemeinen Fall ebenfalls Verästelungen in der Art enthalten, wie sie bei RCE auftreten. Dennoch gibt es Unterschiede zwischen den beiden Modellen, die berücksichtigt werden müssen.

Mit dem Konzept des Hauptastes mißt RCE den auf diesem Ast liegenden Revisionen eine besondere Bedeutung zu, da sich entlang dieses Astes üblicherweise die Hauptentwicklungslinie erstreckt. Dies äußert sich semantisch in der besonderen Behandlung beispielsweise zur automatischen Bestimmung einer Revision beim Ausbuchen ohne explizite Angabe einer Revision durch den Benutzer. Ferner unterscheidet RCE explizit zwischen Fortführung eines Astes und Abzweigung auf einen Nebenast. Die automatische Namensvergabe bei der Erzeugung neuer Revisionen spiegelt dies in der Gestalt einer hierarchischen Namensgebung entsprechend der Topologie des Graphen wider. Äste werden durch Erhöhen des Nummernsuffix fortgeführt, beispielsweise mit dem Bezeichner „mybranch.7“ als Nachfolgerevision zu „mybranch.6“. Im Falle einer Verästelung bildet der Name der Vorgängerrevision standardmäßig den Namenspräfix des neuen Astes und der auf ihr liegenden Revisionen, beispielsweise mit „mybranch.6.2.1“ als erste Revision des zweiten Nebenastes, der von der Revision „mybranch.6“ abgezweigt wurde. Diese Unterscheidung zwischen Fortführung eines Astes und Abzweigung und damit Bildung eines Nebenastes berührt auch die topologische Verwaltung des Graphen. RCE verwendet eine Nachfolge*funktion* zur eindeutigen Kennzeichnung der Revision, den Ast fortführt. Auf Fortführungen in der Gestalt von Nebenästen erlaubt RCE Zugriff über einem *Iterator*, der alle Nebenäste im Sinne einer totalen Ordnung aufzählt. Abbildung 3.7 zeigt einen typischen Revisionsgraphen unter RCE.

Delta-V hingegen unterscheidet nicht zwischen Nebenästen und Hauptästen; es erlaubt im allgemeinen die Existenz mehrerer Arbeits-Resourcen zu einer gegebenen Revision, von denen durch Einbuchung eine beliebige zu einer Nachfolgerrevision der gegebenen Revision führen kann. Die Nachfolge*relation* von Delta-V liefert in diesem Sinne zu einer gegebenen Revision eine ungeordnete *Menge* von Nachfolgerevisionen. Dies drückt sich im Protokoll formal darin aus, daß in Delta-V keine Vorgaben zu den vom Dienstgeber zu erzeugenden Revisionsnamen macht und beim Ausbuchen eine Zielrevision (*target revision*) angegeben werden oder voreingestellt worden sein muß. Auf Revisionen wird bei Delta-V ohnehin überwiegend nicht über deren Revisionsnamen, sondern über die weiter oben beschriebenen, vom Dienstgeber erzeugten dauerhaften Resourcenbezeichner zugegriffen. Abbildung 3.8 zeigt einen typischen Revisionsgraphen einschließlich der Arbeits-Resourcen unter Delta-V; zur Bezeichnung der Revisionen und Arbeits-Resourcen im Beispiel wird jeweils eine Ganzzahlvariable verwendet, die bei jeder

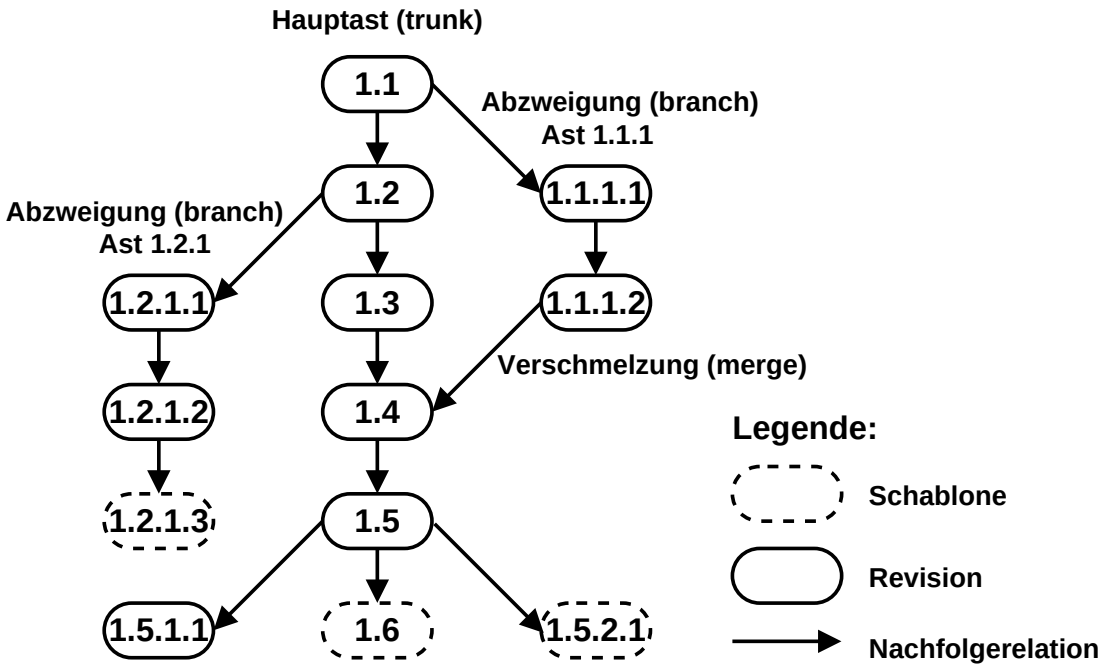

Abbildung 3.7: Typischer Revisionsgraph unter RCE

Erzeugung einer Revision beziehungsweise Arbeits-Resource inkrementiert wird.

Da in RCE die besondere Bedeutung des Hauptastes bei entsprechender Anwendung der Schnittstelle ausgeschaltet werden kann, läßt sich auch in dieser Hinsicht ein Delta-V-Dienstgeber auf der Basis von RCE implementieren. Die umgekehrte Aufgabe, auf der Basis von Delta-V einen Dienstnehmer zu entwickeln, der das Konzept der Haupt- und Nebenäste kennt, ist mit zusätzlichem Aufwand ebenfalls zu bewerkstelligen, beispielsweise indem der Dienstnehmer unter Einsatz von `PROPPATCH` in eigener Verantwortung diejenigen Revisionen markiert, die zum Hauptast gehören. Um auch Nebenäste von Nebenästen zu berücksichtigen, könnte der Dienstnehmer bei jeder Erzeugung einer neuen Revision einen Namen nach dem Revisionsschema von RCE generieren und ihn mittels `PROPPATCH` als zusätzliche Eigenschaft der Revision speichern.

Für eine eindeutige graphische Darstellung des Revisionsgraphen, wie sie von Anwendungen wie VRCE vorgesehen wird, ist die totale Ordnung aller Äste an jedem Verzweigungspunkt des Graphen notwendig. RCE unterstützt, wie beschrieben von sich aus die totale Ordnung. Bei Delta-V ist wegen der ungeordneten Mengen gleichberechtigter Äste eine totale Ordnung derselben nur unter Hinzunahme einer totalen Ordnungsrelation erreichbar; diese ist vom Dienstnehmer in eigener Verantwortung durchzuführen, da eine solche Ordnung außerhalb des Spezifikationsrahmens von Delta-V liegt. Die Eigenschaft `DAV:creationdate`, die den Erstellungszeitpunkt der jeweiligen Resource unter Delta-V repräsentiert, kann zu diesem Zwecke dienen, sofern man eine chronologische Ordnung der Äste

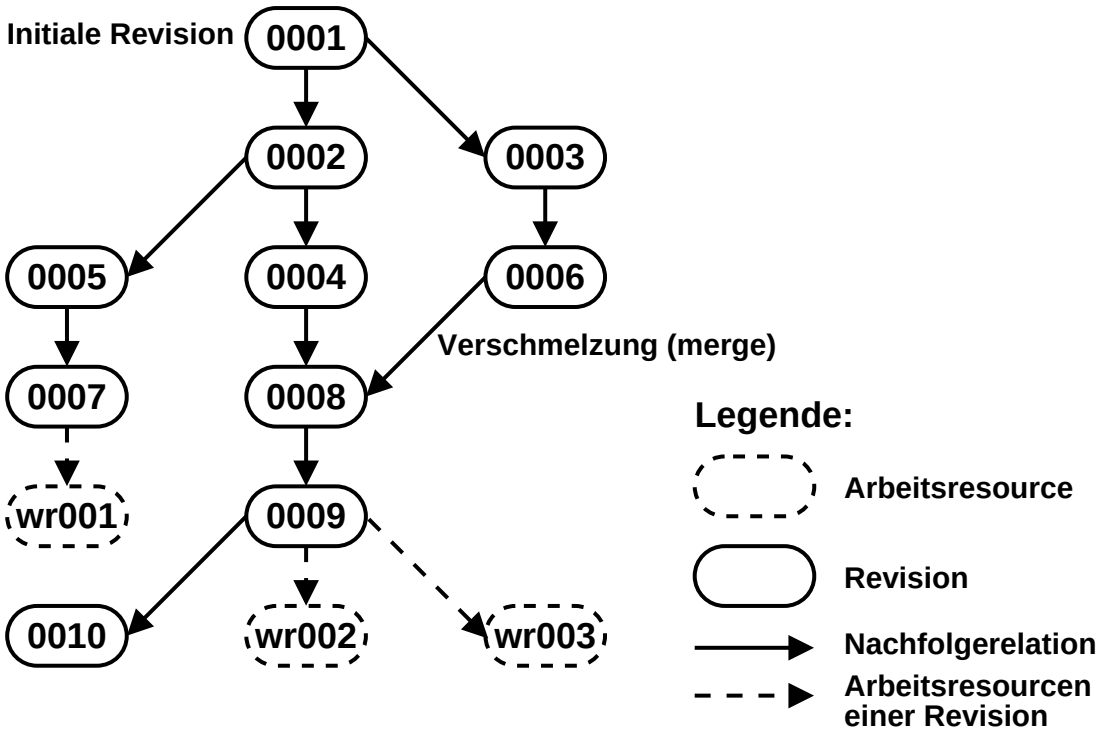

Abbildung 3.8: Typischer Revisionsgraph unter Delta-V

unterstellt. Sie ist hierzu für eine gegebene Revision auf allen ihren Nachfolgerevisionen sowie auf allen ihren Arbeits-Resourcen auszuwerten. Dabei entspricht jede Nachfolgerevision im Sinne von Delta-V der ersten gefüllten Revision eines Haupt- oder Nebenastes im Sinne von RCE; jede Arbeits-Resource im Sinne von Delta-V entspricht einer Schablone im Sinne von RCE, die sich am Ende des Haupt- oder eines Nebenastes befindet. Neben Auswertung der Eigenschaft `DAV:creationdate` kann der Dienstnehmer alternativ mit dem oben beschriebenen Verfahren der Markierung von Ästen deren Ordnung festlegen. Die Markierung an sich mittels Setzen benutzerdefinierter Eigenschaften auf Resourcen fällt dabei unter die Protokollspezifikation von Delta-V; ihre Semantik unterliegt jedoch der Eigenverantwortlichkeit des Dienstnehmers.

3.2.3 Attribute und Eigenschaften

Den Attributen von Archiven, Revisionen und Schablonen bei RCE entsprechen die Eigenschaften von versionierten Resourcen, Revisionen und Arbeits-Resourcen unter Delta-V. Einige RCE-Attribute entsprechen in ihrer Semantik Eigenschaften von Delta-V und lassen sich daher problemlos übertragen. Anderen RCE-Attributen entsprechen Eigenschaften von Delta-V mit leicht differierender Semantik, die sich im Rahmen der mit dieser Arbeit verbundenen Implementierung jedoch als ebenfalls übertragbar erwiesen. Zu einigen der RCE-Attribute

gibt es keine vergleichbaren Eigenschaften in Delta-V. Diese müssen gegebenfalls als benutzerdefinierte Eigenschaften der Resourcen modelliert werden. Umgekehrt kann die Speicherung von Eigenschaften bei Implementierung eines Delta-V-Dienstgebers auf Basis von RCE entweder unter Nutzung des attributierten Dateisystems oder der Attributeverwaltung von RCE erfolgen.

Zu den problemlos übertragbaren Attributen zählen `ARCH/REV_AUTHOR` und `ARCH/REV_DESCRIPTION`, denen die Eigenschaften `DAV:author` und `DAV:comment` unter Delta-V entsprechen. Den drei RCE-Attributen `REV_DATE_OUT`, `REV_DATE_IN` `REV_TIMESTAMP` entsprechen die drei Eigenschaften `DAV:creationdate`, `DAV:checkin-date` und `DAV:getlastmodified`. Bei ihnen müssen lediglich die verschiedenen Datumsformate von HTTP (Datumsformate nach RFC 850 und RFC 1123[33]), WebDAV und Delta-V (Datumsformat nach ISO 8601[26]) sowie RCE (zwei RCE-eigene Datumsformate) berücksichtigt und Datumsangaben gegebenfalls konvertiert werden.

Die Übertragung von RCE-Attributen auf Eigenschaften in Delta-V mit differierender Semantik soll an den Beispielen `DAV:predecessor-set`, `DAV:successor-set`, `DAV:working-resource-id-set` und `DAV:revision-set` exemplarisch erläutert werden. Das Versionsmodell von Delta-V kennt keine Äste im Sinne von RCE, sondern definiert den Revisionsgraphen zu einer versionierten Resource allein über eine Vorgänger/Nachfolger-Relation, bei der alle Nachfolger einer Revision gleichberechtigt sind. Bei RCE hingegen gibt es zu jeder Revision höchstens einen direkten Nachfolger. Die über diese Nachfolgerfunktion definierte Folge von Revisionen entspricht einem Ast im Sinne von RCE. Weitere Nachfolger führen zu Verzweigungen. Hieraus ergibt sich, daß der Menge der Nachfolger einer Revision in Delta-V die Vereinigung aus direktem Nachfolger und den initialen Revisionen der von dieser Revision abgezweigten Äste in RCE entspricht.

Nach dem Versionsmodell von Delta-V kann es zu einer Revision mehrere Vorgänger geben, während bei RCE jeder Revision ein eindeutiger Vorgänger zugeordnet ist. Ausnahme ist bei beiden Modellen selbstverständlich die initiale Revision, zu der es keinen Vorgänger gibt. Der Unterschied zwischen den beiden Modellen liegt darin begründet, daß Delta-V im Falle einer Verschmelzung alle beteiligten Revisionen als gleichwertige Vorgänger auffaßt, während RCE hierfür ein zusätzliches Attribut (`REV_MERGE_FROM`) zum Auffinden von Verschmelzungsrevisionen von Nebenästen verwaltet. Damit ist die Bestimmung der Vorgängermenge für Delta-V auf der Basis einer RCE-Implementierung durch Vereinigung von Vorgängerrevision und Verschmelzungsrevisionen abbildbar. Als Beispiel für den umgekehrten Fall möge ein Dienstnehmer das Modell von RCE auf der Basis eines Delta-V-Dienstgebers implementieren. Der Dienstnehmer erhält bei Anfrage nach der Vorgängerrevision im allgemeinen eine Menge von Revisionen, innerhalb deren er zwischen direkter Vorgängerrevision und Verschmelzungsrevisionen unterscheiden können muß. Dies läuft auf das bereits beschriebene Problem der Unterscheidung zwischen Haupt- und Nebenast hinaus und läßt sich daher, wie bereits erläutert, durch zusätzliche Markierungen oder Auswertung von Zeitstem-

peln unter Berücksichtigung der Topologie des Revisionsgraphen lösen.

In einigen Fällen ist die Beziehung zwischen Attributen und Eigenschaften ein wenig komplizierter. Dies gilt beispielsweise für die Eigenschaft `DAV:working-resource-id-set`, die zu einer gegebenen Revision alle ausgebuchten Arbeits-Resourcen in Form von Resourcenbezeichnern enthält, über die auf die auf dem Dienstgeber gelegenen Arbeits-Resourcen zugegriffen werden kann. RCE hingegen verwendet, wie geschildert, Schablonen, die keine Revisionen im Sinne von Delta-V sind. Die Bestimmung der Menge der ausgebuchten Arbeits-Resourcen einer Revision im Sinne von Delta-V entspricht unter RCE demnach der Aufgabe, zu einer gegebenen Revision über die Nachfolgerattribute `REV_NEXT` und `REV_BRANCHES` zunächst alle Schablonen zu finden, deren Vorgängerrevision gleich der gegebenen Revision ist. Anschließend müssen die zu den Schablonen gehörigen Arbeits-Resourcen ermittelt werden. Hier kann das RCE-Attribut `REV_WORK_PATH` weiterhelfen, welches zu einer Schablone den Pfad der zugehörigen Arbeitsdatei angibt. Aus diesem wiederum kann der Resourcenbezeichner bestimmt werden, unter dem die Arbeitsdatei auf dem Dienstgeber als Arbeits-Resource erscheint. Die Vereinigung dieser Resourcenbezeichner ergibt eine Menge, die dem Inhalt der zu bestimmenden Eigenschaft `DAV:working-resource-id-set` entspricht. Einfacher und effizienter ist es jedoch, diese Eigenschaft nicht bei jeder der häufigen Anfrage neu zu berechnen, sondern mit jeder Revision zu speichern und bei Änderungen wie Einbuchungen und Ausbuchungen fortzuschreiben.

Ähnliches gilt für die Eigenschaft `DAV:revision-set` die zu jeder Revision einer versionierten Resource einen Resourcenbezeichner enthält, über den auf die Revision zugegriffen werden kann. Auch diese Eigenschaft ließe sich durch Aufruf von Funktionen wie `Archive.getFirstRevision()` und Verwendung von RCE-Attributen wie `REV_NEXT` und `REV_BRANCHES` berechnen. Auch hier kann aus Effizienzgründen anstelle der ständigen Neuberechnung die explizite Speicherung und Fortschreibung der Eigenschaft erfolgen, beispielsweise wenn eine häufige Benutzung dieser Eigenschaft etwa zum Durchlaufen eines Revisionsgraphen vorauszusehen ist.

Tabelle 3.9 listet auszugsweise die wichtigsten Beziehungen zwischen RCE-Attributen und Delta-V-spezifischen Eigenschaften von Resourcen auf. Ein Eintrag „N.N." in der Spalte „Delta-V-Eigenschaft" zu einem gegebenen Attribut bedeutet, daß es kein direktes Äquivalent in Delta-V hierfür gibt. Wie bei DAV-Resourcen üblich, sollte der Dienstgeber aber auch beliebige benutzerdefinierte Eigenschaften als „tote Eigenschaften" (*dead properties*) speichern und zurückliefern können. Allerdings kann eine solche tote Speicherung keinen Einfluß auf die Semantik des Dienstgebers im Umgang mit den betroffenen Resourcen ausüben. Zumindest für die RCE-Attribute `ARCH_USERS` und `ARCH_LOCK_LEVEL`, die für die Zugriffskontrolle zuständig sind, lassen sich in der noch in Entwicklung befindlichen ACL-Erweiterung von DAV[44] für Zugriffskontrollisten sowie den Sperrmechanismen von DAV Möglichkeiten erkennen, die Zugriffskontrollmechanismen

von RCE bei Implementierung eines Dienstnehmers auf die Funktionalität eines reinen Delta-V-Dienstgebers abzubilden. Umgekehrt dürften die verhältnismäßig einfachen Mechanismen von RCE nicht ausreichen, um komplexe Zugriffskontrollmechanismen eines vollausgebauten Delta-V-Dienstgebers auf diejenigen von RCE abzubilden. Dies stellt jedoch keine grundsätzliche Einschränkung dar, da die fehlende Funktionalität durch eine zusätzliche Zugriffskontrollschicht in der Gesamtarchitektur des Dienstgebers nachrüstbar ist.

Resourcentyp	Delta-V-Eigenschaft	RCE-Attribut
Versionierte Resource	DAV:author	ARCH_AUTHOR
	DAV:comment	ARCH_DESCRIPTION
	DAV:revision-set	REV_NEXT, REV_BRANCHES
	N.N.	ARCH_WORK_PATH
	N.N.	ARCH_RCA_PATH
	DAV:creationdate	ARCH_DATE
	N.N.	ARCH_COMMENT_LEADER
	N.N.	ARCH_DEFAULT_BRANCH
	N.N.	ARCH_USERS
	N.N.	ARCH_KEYS
	N.N.	ARCH_MODES
	N.N.	ARCH_LOCK_LEVEL
Revision	DAV:author	REV_AUTHOR
	DAV:comment	REV_DESCRIPTION
	DAV:predecessor-set	REV_PREV, REV_MERGE_FROM
	DAV:successor-set	REV_NEXT, REV_BRANCHES
	DAV:creationdate	REV_DATE_OUT
	DAV:checkin-date	REV_DATE_IN
	DAV:getlastmodified	REV_TIMESTAMP
	DAV:revision-id	REV_NAME
	DAV:revision	N.N.
	DAV:working-resource-id-set	REV_NEXT, REV_BRANCHES, REV_WORK_PATH
	DAV:label-set	REV_ALIASES
	N.N.	REV_STATE

Abbildung 3.9: Beziehung zwischen Delta-V-spezifischen Eigenschaften von Resourcen und RCE-Attributen

Bei Erörterung der Schablonen (vgl. Abschnitt 3.2.1) wurde die Möglichkeit diskutiert, Attribute von Schablonen auf Eigenschaften von Arbeits-Resourcen abzubilden. Da bei der Implementierung des Delta-V-Dienstgebers auf der Ba-

sis von RCE und eines attributierten Dateisystems die versionierten Resourcen und Arbeits-Resourcen als RCE-Archive und Arbeitsdateien des attributierten Dateisystems erscheinen, liegt zur Realisierung der Eigenschaften dieser Resourcen allgemein die Verwendung der Dateiattribute nahe. Es muß hierbei allerdings sichergestellt werden, daß etwa beim Einbuchen einer Arbeits-Resource deren Eigenschaften auf die zugehörige Schablone übergehen.

Revisionen hingegen werden als Bestandteil von RCE-Archiven verwaltet. Ein dauerhafter Resourcenbezeichner zu einer Revision läßt sich direkt auf das dazugehörige Archiv abbilden, ohne daß hinter einem solchen Resourcenbezeichner eine tatsächlich existierende Datei stehen muß. In diesem Falle ist es naheliegend, die Eigenschaften von Revisionen als benutzerdefinierte Attribute mit den RCE-Funktionen `Revision.setUserInfo()` und `Revision.getUserInfo()` zu verwalten. Diese beiden Funktionen erlauben die Speicherung und Abfrage benutzerdefinierter Daten unter benutzerdefinierten Schlüsseln.

Leider erlaubt RCE in der aktuellen Version bei Verwendung der beiden Funktionen nur die Speicherung ausgewählter Zeichen; Sonderzeichen und Steuerzeichen können zur Korrumpierung des Archivs führen. Daher ist die Verwendung einer geeigneten Kodierung der Daten vor ihrer Speicherung vorzusehen, um die Funktionen dennoch verwenden zu können. Eine genauere Analyse des Vorrates an sicheren Zeichen, die eine Korrumpierung ausschließen, ergab, daß eine Kodierung gemäß Base64 (vgl. RFC 2045[38], Abschnitt 6.8) geeignet ist.

Ein weiteres Problem der beiden Funktionen ist, daß mit ihnen die Speicherung und Abfrage von Daten nur unter Angabe eines Schlüssels möglich ist; die Möglichkeit einer Anfrage zur Aufzählung aller gespeicherten Daten ist nicht gegeben. Genau eine solche Funktionalität wird jedoch zur Realisierung des Eigenschaftsmodells von Resourcen unter DAV benötigt. Es bleibt die Möglichkeit, *alle* Eigenschaften einer Revision durch geeignete Kodierung zu bündeln und diese Base64-kodiert unter *einem einzigen* Schlüssel zu speichern. Das Konzept von RCE, über Schlüssel einen schnellen und unkomplizierten Zugriff auf benutzerdefinierte Daten zu ermöglichen, wird auf diese Weise allerdings unterlaufen.

Eine Alternative zur Speicherung der Eigenschaften von Revisionen besteht darin, unter dem dauerhaften Resourcenbezeichner jeder Revision eine attributierte leere Datei nur zu dem Zwecke abzulegen, die Eigenschaften als Attribute dieser Datei zu speichern.

3.3 Architektur des Dienstgebers

Die Implementierung eines Delta-V-Dienstgebers auf der Basis von RCE mußte von Grund auf neu erfolgen, da sich alle untersuchten bestehenden WebDAV/HTTP-Dienstgeber als ungeeignet zur Wiederverwendung in dieser Arbeit erwiesen. Die neu geschaffene Implementierung umfaßt daher einen HTTP-Dienstgeber und zahlreiche in WebDAV, Delta-V und anderen Protokollen defi-

nierten Erweiterungen zu HTTP. Nicht zuletzt deswegen macht die Implementierung des Dienstgebers schätzungsweise 80% des gesamten praktischen Teils dieser Arbeit aus.

3.3.1 HTTP

Im Sinne einer modernen Architektur bemüht sich die Implementierung des HTTP-spezifischen Teils des Dienstgebers um Modularität. Ziel ist es, Erweiterungen wie WebDAV als getrennt übersetzbare, optionale Module dem Dienstgeber hinzufügen zu können. Da das HTTP-Protokoll selbst viele optionale Bestandteile enthält, die idealerweise austauschbar sein sollten, wurden Teile des HTTP-spezifischen Teiles selbst in ein eigenes Modul ausgegliedert. Als Architektur des Dienstgebers ergibt sich somit ein kleiner Kern mit Basisfunktionen und Modulschnittstelle, ferner ein HTTP-Modul, welches den Kern zu einem funktionsfähigen HTTP-Dienstgeber erweitert.

3.3.1.1 Dienstgeberkern

Die Basisfunktionen des Kerns enthalten im wesentlichen Funktionen zur Verwaltung und Abwicklung der Kommunikation mit dem Dienstnehmer. Neben der Erstellung von Logbuch-Dateien fallen hierunter das Empfangen von Anfragen vom Dienstnehmer in jeweils einem eigenen Kontrollfaden (*thread*), deren Vorzerteilung und Weitergabe über die Modulschnittstelle, der Empfang von Anfrageergebnissen von der Modulschnittstelle und deren Versand als Antwort an den Dienstnehmer.

3.3.1.2 Modulschnittstelle

Die Modulschnittstelle des Kerns modelliert ein Modul im wesentlichen als Sammlung von Klassen zur Behandlung von HTTP-Methoden, HTTP-Kopfzeilen und HTTP-Statuskodes. Die Klassen zur Behandlung von Kopfzeilen und Statuskodes werden in je einer Tabelle gesammelt; der Zerteiler des Kerns delegiert die Zerteilung der Kopfzeilen an die in der Tabelle vorhandenen Klassen, soweit verfügbar. Die Klassen zur Behandlung von HTTP-Methoden werden in Zuständigkeitsketten gespeichert; die Reihenfolge entspricht dabei der Reihenfolge, in der die Module dem Kern hinzugefügt werden. Auf diese Weise kann ein Modul die Semantik einer HTTP-Methode bei Bedarf erweitern oder redefinieren; im Falle der Nichtzuständigkeit kann die Bearbeitung der Methode entlang der Kette weiterdelegiert werden. Aus Effizienzgründen wird in einer Tabelle für jede HTTP-Methode eine eigene Zuständigkeitskette verwaltet.

Abbildung 3.10 zeigt beispielhaft einen Ausschnitt aus der Zuständigkeitskette für die HTTP-Methode `GET`. Im einfachsten Falle verbirgt sich hinter einer Resource eine reguläre Datei; wird `GET` auf eine solche Resource angewendet,

so wird der Inhalt der Datei dem Dienstnehmer übermittelt. Viele Dienstgeber unterstützen die Anwendung von `GET` auf Verzeichnisdateien; sie erzeugen in diesem Falle eine HTML-Datei mit einer Darstellung des Verzeichnisses. Befindet sich in dem Verzeichnis eine Datei mit dem Namen `index.html`, so liefern gängige Dienstgeber diese automatisch als Index-Datei anstelle der Verzeichnisdarstellung zurück. Unter Delta-V lassen sich über dauerhafte Resourcenbezeichner mit `GET` Revisionen anfordern, die gegebenenfalls aus den jeweiligen versionierten Resourcen extrahiert werden müssen und daher einer gesonderten Behandlung bedürfen. Für versionierte Resourcen sieht Delta-V selbst eine besondere Behandlung der Methode `GET` vor, weshalb sich speziell auch für solche Resourcen ein eigenes Element in der Zuständigkeitskette befindet. Die gezielte Auswahl der in der Zuständigkeitskette vorkommenden Elemente und die Festlegung ihrer Reihenfolge ermöglicht eine flexible Konfiguration des Dienstgebers als Ganzem.

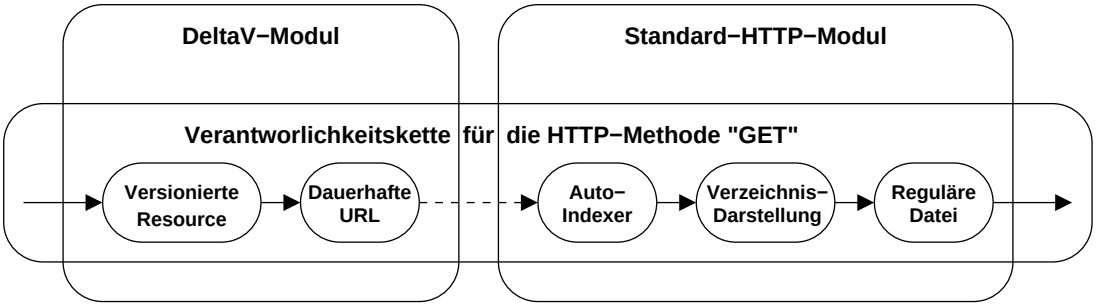

Abbildung 3.10: Zuständigkeitskette am Beispiel der HTTP-Methode `GET`

3.3.1.3 HTTP-Modul

Das HTTP-Modul implementiert die syntaktische und semantische Behandlung aller im HTTP-Protokoll definierten Statuskodes sowie zahlreicher Kopfzeilen und fügt dem Kern Standardimplementierungen zu allen im Protokoll definierten Methoden hinzu.

Die Implementierung der Kopfzeilen zur Authentifizierung unter HTTP unterstützt die Schemen *Basic Authentication* und *Digest Authentication*[40]. Digest Authentication ist ein häufig eingesetztes Authentifizierungsverfahren, das deutlich sicherer als das ohne jegliche Verschlüsselung arbeitende Verfahren Basic Authentication ist. Es basiert auf einem Einweg-Hasch-Verfahren gemäß MD5[34] und wird unter anderem für die WebDAV-Implementierung benötigt.

Das HTTP-Modul bildet den Namensraum des Dienstgebers auf eine Verzeichnishierarchie des lokalen Dateisystems des Dienstgebers ab und setzt Methoden wie `GET`, `PUT` und `DELETE` auf entsprechende Dateioperationen um; dies umfaßt u.a. auch eine Implementierung von `GET` zur Darstellung des Inhaltes von Verzeichnissen.

Gemeinsam mit dem Dienstgeberkern bildet das HTTP-Modul einen schlanken, erweiterbaren HTTP-Dienstgeber.

3.3.1.4 Lexikalische und grammatikalische Analyse

Die von HTTP spezifizierte Syntax stützt sich auf eine Vielzahl von Protokollstandards des Internet, die sich über viele Jahre entwickelt haben. Zur Behandlung der dementsprechend komplexen Syntax ist der Einsatz eines Zerteilergenerators zur automatischen Erzeugung eines geeigneten Zerteilers naheliegend. Eine genauere Analyse bereits der von HTTP verwendeten Syntax sowie der Fähigkeiten gängiger Zerteiler zeigt, daß der Einsatz eines Zerteilergenerators zu diesem Zwecke in der Regel nur mit aufwendigen Transformationen der Syntax möglich ist. Zur Behandlung syntaktischer oder semantischer Kontextabhängigkeiten ist zumeist ein massiver Einsatz handgeschriebenen Kodes notwendig.

Übliche Zerteiler erwarten als Dateneingabe zur grammatikalischen Analyse eine Folge von Symbolen (*tokens*). Die Zerteilung eines zeichenbasierten Eingabestroms in eine Folge von Symbolen erfolgt im Rahmen der lexikalischen Analyse üblicherweise durch einen Symbolerkenner (*tokenizer*), der zumeist als endlicher, deterministischer Automat ausgeführt und durch Angabe regulärer Ausdrücke oder, spezieller, mittels Trennzeichen (*delimiters*) spezifiziert wird. Es erfolgt somit eine strikte Trennung der beispielsweise mittels Backus-Naur-Form (BNF) notierten kontextfreien Zerteiler- oder Symbolgrammatik von einer darunterliegenden regulären lexikalischen oder Zeichengrammatik.

Die hierarchische Gesamt-Grammatik etwa des Kopfes einer HTTP-Anfrage baut auf den einzelnen Zerteilergrammatiken der zugrundeliegenden Protokolle mit ihren jeweils spezifischen lexikalischen Grammatiken auf. Daher definiert die jeweils gerade gültige Zerteilergrammatik, die sich aus der Zerteilung selbst ergibt, die jeweils gerade zuständige lexikalische Grammatik. Es besteht somit eine Abhängigkeit der anzuwendenden lexikalischen Analyse vom inneren Zustand des Zerteilers. Ferner ist der Zerteiler selbst als Hierarchie von Zerteilern zu betrachten. Daher kommt nur die Anwendung eines Zerteilergenerators in Frage, der die Definition hierarchischer Zerteilergrammatiken und mehrerer lexikalischer Grammatiken unterstützt.

Das größte Problem jedoch ist, daß die Trennung zwischen lexikalischer Grammatik und Zerteilergrammatik in den Protokollen des Internet nicht durchgängig durchgehalten wird. Als Folge davon finden sich in einigen Protokollen Grammatikregeln, die sich formal nur durch aufwendige Transformationen in die übrige Grammatik integrieren lassen und daher stattdessen meist natürlichsprachlich umschrieben werden. Die Regel zur Behandlung von „implied linear white space (LWS)“ aus Abschnitt 2.1 des HTTP-Protokolls[39] ist ein Beispiel für diese Problematik. Sie beschreibt verbal, in welchen Situationen Leer- und spezielle Kontrollzeichen der lexikalischen Grammatik zwischen zwei Symbolen der Zerteilergrammatik bei der lexikalischen Analyse ignoriert werden sollen. Die Einbezie-

hung dieser an für sich einfachen Regel in die BNF-ähnliche Darstellung der übrigen, vorwiegend zerteilerorientierten Grammatik von HTTP würde diese deutlich komplizierter werden lassen. Die dieser Arbeit zugrundeliegende Dienstgeberimplementierung führt daher die Symbolerkennung und Zerteilung von Hand, d.h. ohne die Verwendung von Werkzeugen wie Zerteilergeneratoren, aus.

3.3.2 XML

Da in WebDAV Nachrichten unter Verwendung von XML[27] versendet werden, ist die Implementierung von Komponenten zur syntaktischen und semantischen Analyse, aber auch zur Generierung von Datenstrukturen in XML erforderlich. Die Suche nach einer bereits bestehenden Implementierung eines XML-Zerteilers in Java zu Beginn dieser Arbeit ergab eine in noch frühem Stadium befindliches XML-Werkzeug vom Java-Hersteller, an der auch das Web-Konsortium beteiligt war (*Java Project X, Technology Release 2*[29]). Wichtig im Hinblick auf die Verwendung von XML unter WebDAV und Delta-V erschien über die Zusicherung der Wohlgeformtheit auch die Unterstützung von Validation und Namensräumen.

Der Zerteiler des Java Project X unterstützt validierendes XML auf der Basis einer anzugebenden DTD *oder* die Verwaltung von Namensräumen, jedoch nicht beides zugleich. Eine eingehendere Analyse des Quelltextes führte zur Erkenntnis, daß der Zerteiler die Möglichkeit zur Verwendung einer abstrakten Fabrik zwecks Erzeugung anwenderdefinierbarer XML-Elemente vorsieht. Durch Implementierung einer konkreten Fabrik und einer jeweils eigenen Klasse für jedes XML-Element konnte auf diese Weise zusätzlicher validierender Kode dem Zerteiler bei gleichzeitiger Nutzung der Unterstützung von Namensräumen hinzugefügt werden, ohne den Quelltext des Zerteilers verändern zu müssen.

3.3.3 WebDAV

Da seit langem zahlreiche Implementierungen von HTTP-Dienstgebern und -Dienstnehmern, seit einigen Monaten auch einige wenige WebDAV-Implementierungen existieren, lag es zunächst nahe, Delta-V auf der Basis bestehender WebDAV- beziehungsweise HTTP-Implementierungen umzusetzen. Die folgenden Überlegungen zur Anforderungsanalyse zeigen jedoch, daß die Verwendung bestehender Dienstgeber und Dienstnehmer aus den verschiedensten Gründen nicht in Frage kam und somit eine von Grund auf weitgehend neue Implementierung erfolgen mußte. Die Implementierung erfolgte als Erweiterungsmodul des weiter oben beschriebenen HTTP-Dienstgebers auf der Basis eines attributierten Dateisystems.

3.3.3.1 Dienstgeberanbindung

Eine Anbindung von Java-Anwendungen an gängige Dienstgeber erfolgt üblicherweise entweder über die Schnittstelle *Common Gateway Interface* (*CGI*) oder über das mittlerweile in vielen Dienstgebern verfügbare, speziell auf Java-Anwendungen zugeschnittene *Servlet-API*. Eine Realisierung von WebDAV als Anbindung über CGI ist theoretisch zwar möglich, praktisch jedoch aus verschiedenen Gründen unrealistisch. Einer der Gründe ist das ständige, zeitaufwendige Laden des CGI-Programmes vor jeder Ausführung; weitere Gründe wurden bereits bei der Implementierung von WWCM[21] erkannt und beschrieben. Das Servlet-API ermöglicht es, in eine HTML-Seite Programmkode einzubetten. Wird eine solche Seite von einem Dienstnehmer mittels der HTTP-Methode `GET` angefordert, so führt der Dienstgeber vor der Auslieferung der Seite den eingebetteten Kode aus und nimmt dadurch Einfluß auf den Inhalt der Seite. Dies gestattet die Dynamisierung des Inhaltes von Webseiten (*dynamic content embedding*). WebDAV und Delta-V hingegen stellen eine Erweiterung des HTTP-Protokolls einschließlich der Definition neuer HTTP-Methoden, Kopfzeilen und Statuskodes dar und lassen sich daher nicht durch Servlets realisieren.

Für den Betrieb einer in Java realisierten Delta-V-Implementierung ist es zweckmäßig, eine virtuelle Java-Maschine ständig am Laufen zu halten. Zum einen wird das Problem des wiederholten Ladens und Startens der virtuellen Maschine vermieden. Ferner wird es mit dem Konzept eines Speicher-Cache möglich, Daten aus Effizienzgründen über mehrere HTTP-Anfragen hinweg im Speicher zu halten, ohne sie zeitaufwendig in eine Datei auslagern und später wieder einlesen zu müssen. Eine ständig laufende virtuelle Java-Maschine läßt sich auch dazu verwenden, in regelmäßigen Zeitabständen auszuführende Tätigkeiten durchzuführen, beispielsweise das Erstellen einer Datei als Sicherungskopie des Speicher-Cache.

Einige Dienstgeber, wie zum Beispiel Apache[24], stellen eine Schnittstelle zur Einbindung von Modulen zur Verfügung. Solche Module erlauben meist eine Erweiterung des Dienstgebers um neue HTTP-Methoden und neue HTTP-Kopfzeilen. Auch die Aufrechterhaltung von Daten oder Programmen im Hauptspeicher über einzelne Anfragen hinweg ist mit solchen Dienstgebern üblicherweise möglich. Daher bietet es sich an, WebDAV und Delta-V als Module eines solchen Dienstgebers zu implementieren. Dies gilt umso mehr, da sich für Apache derzeit ein Modul (`mod_dav`) in Entwicklung befindet, das die Funktionalität von WebDAV zu implementieren versucht.

Mit Hinblick auf Delta-V als Erweiterung von WebDAV mußte eine WebDAV-Implementierung verwendet werden, die Erweiterungen beispielsweise in Form neuer Eigenschaften erlaubt. Da `mod_dav` zu Beginn dieser Diplomarbeit noch recht experimentell und instabil war und keine Schnittstellen erkennbar waren, die eine Erweiterung im Sinne von Delta-V ohne weitreichende Eingriffe in die Kode-Basis des Moduls ermöglicht hätten, wurde entschieden, ein eigenes WebDAV-

Modul zu entwickeln. Ein weiteres Problem stellt der Übergang zwischen den Programmiersprachen C und Java dar. Da Apache in der Programmiersprache C geschrieben ist, muß auch jedes Modul in C geschrieben werden. Zwar lassen sich von C aus eine virtuelle Java-Maschine starten und Methodenaufrufe ausführen. Allerdings impliziert die Architektur eines modularen Dienstgebers in Verbindung mit den Kommunikationspfaden der einzelnen Module untereinander eine Reihe von Rückrufen (*callbacks*), die einen häufigen Wechsel zwischen C und Java bedingt hätten. Aus Effizienz- wie auch Komplexitätsgründen wurde daher dieser Ansatz aufgegeben.

Benötigt wurde ein modularer, erweiterbarer, in Java implementierter HTTP-Dienstgeber. Dieser sollte zudem frei verfügbar sein, vorzugsweise mit Quelltext, um nötigenfalls in die Kode-Basis eingreifen zu können, falls die Modulschnittstelle sich als unzureichend herausstellen würde. Eine Recherche ergab im wesentlichen nur zwei Dienstgeber, die in Frage kamen: Jigsaw, ein in Java entwickelter umfangreicher und komplexer Dienstgeber, der als Nachfolger des CERN-httpd[23] propagiert wird, und der in Abschnitt 3.3.1 beschriebene, bereits im Rahmen von WWCM[21] entwickelte HTTP-Dienstgeber auf der Basis eines Paketes, das auch die für die Implementierung eines Dienstnehmers notwendigen grundlegenden Funktionen bereits enthielt. Die Entscheidung fiel auf den letzteren HTTP-Dienstgeber wegen der genauen Kenntnis des überschaubaren Quellkodes und der bereits im Paket enthaltenen dienstnehmerseitigen Funktionen. Die Dienstgeberanbindung von WebDAV erfolgt somit über die Modulschnittstelle des HTTP-Dienstgebers.

3.3.3.2 Attributiertes Dateisystem

Die Implementierung eines WebDAV-Dienstgebers erfordert die Möglichkeit zur Speicherung von Resourcen einschließlich beliebiger Eigenschaften, die bei der Übermittlung zwischen Dienstgeber und Dienstnehmer durch XML repräsentiert werden. Ein naheliegender Ansatz zur Implementierung eines solchen Dienstgebers besteht in der Verwendung des dem Dienstgeber zugrundeliegenden Dateisystems zur Speicherung der Resourcen. Die Eigenschaften von Resourcen stellen sich in diesem Falle als zusätzliche, frei definierbare Dateiattribute dar. Somit läuft die Implementierung eines WebDAV-Dienstgebers auf die Realisierung eines attributierten Dateisystems hinaus. Da dem Dienstgeber im allgemeinen die Semantik der als XML dargestellten Eigenschaften nicht bekannt ist, bleibt im wesentlichen die Speicherung der Eigenschaften in einer Form, die eine zeichengetreue Wiederherstellung erlaubt.

Die vorliegende Implementierung realisiert ein solches attributiertes Dateisystem. Zu jeder Resource werden zwei Dateien in zwei identisch strukturierten Dateibäumen verwaltet. Die eine Datei im einen Baum enthält den eigentlichen Inhalt der Resource; die andere im anderen Baum enthält die Attribute der Resource, die als Text-Datei in einer zeichenbasierten Repräsentation eines

XML-Dokumentes gespeichert werden, welches alle Attribute als XML-Elemente enthält. Die beiden identisch strukturierten Dateibäume werden so auf den Namensraum des Dienstgebers abgebildet, daß aus Sicht des Dienstnehmers Resourcen und Attribute als Teile ein und desselben Namensraumes erscheinen. Um Attribute auch auf Verzeichnisdateien zuzulassen, verwendet das attributierte Dateisystem eine bijektive Umbenennung bei der Abbildung von Resourcennamen auf Dateinamen in dem Baum, in dem die Attribute gespeichert werden, denn die Namen der zu attributierenden Verzeichnisdateien selbst sind bereits durch die Strukturierung des Baumes in Verzeichnisse vergeben. Eine integrierte automatische Umbenennung von logischen Pfadnamen in physikalische und umgekehrt durch Substitution eines Wurzelpfades erlaubt überdies das Montieren des attributierten Dateisystems in einem beliebigen logischen Verzeichnis.

XML-Attribute werden in WebDAV als Bestandteil von XML-Dokumenten übermittelt, die in der Regel Deklarationen von Namensräumen enthalten, die bei der Speicherung der Eigenschaften berücksichtigt werden müssen; Präfixe werden daher vor ihrer Speicherung aufgelöst. Umgekehrt werden beim Zusammensetzen eines XML-Dokumentes aus einzelnen Eigenschaften Deklarationen von Namensräumen an geeigneter Stelle eingefügt und die Präfixe entsprechend adaptiert.

Das Zerteilen von XML-Dokumenten erfordert verhältnismäßig viel Zeit, insbesondere, wenn dabei Deklarationen von Namensräumen und deren ordnungsgemäße Anwendung überprüft werden sollen und möglicherweise eine Validation durchgeführt wird. Der umgekehrte Vorgang des Serialisierens etwa zum Schreiben in eine Datei ist vielfach kürzer. Beobachtungen zum Laufzeitverhalten des attributierten Dateisystems zufolge scheinen die Schreib- und Leseraten der zumindest auf dem zur Implementierung verwendeten System eingesetzten Massenspeicher bei diesen Vorgängen vernachlässigbar zu sein. Es liegt daher nahe, Eigenschaften in einem Cache zwischenzuspeichern, um Zerteilungen möglichst zu vermeiden.

Die vorliegende Implementierung des attributierten Dateisystems enthält zu diesem Zwecke einen Durchschreibe-Cache (*write-through cache*). Die Cache-Kohärenz, ja sogar Konsistenz, wird gewährleistet, indem jeder Schreibzugriff auf den Cache zum unmittelbaren Schreiben auf den Hintergrundspeicher führt; dies erscheint aufgrund der geringen beobachteten Zeit für die Serialisierung und das Schreiben auf den Massenspeicher akzeptabel; auf der anderen Seite spart jeder Treffer beim Lesen die Zeit des Einlesens, vor allem aber die des Zerteilens. Als Verdrängungsstrategie kommt das Verfahren gemäß der zweiten Chance (*Second Chance Algorithm*) zum Einsatz, welches eine gute Heuristik für die Strategie des Ausräumens der am längsten nicht benutzten Speicherzelle (*Least Recently Used, LRU*) darstellt. Die derzeitige Dimensionierungen mit 1024 Cache-Zeilen und 16-facher Assoziativität bei linearer Sondierung hat sich – vorbehaltlich genauerer Untersuchungen insbesondere auch unter extremen Belastungen – als geeignet erwiesen. Als Haschfunktion dient die Methode `hashCode()` des Objektes, das

als Schlüssel dient; im Falle einer attributierten Datei ist dies der Pfad der Datei. Eine Erhöhung der Anzahl an Cache-Zeilen bringt eine Erhöhung der Trefferquote (*cache hit rate*) mit sich, belegt aber auch entsprechend mehr Speicherplatz. Höhere Assoziativität bedingt ebenfalls eine höhere Trefferquote, allerdings auch eine langsamere Verwaltung aufgrund der linearen Sondierung.

Das attributierte Dateisystem ist so konzipiert, daß seine Konsistenz gewährleistet bleibt, wenn alle Dateioperationen im Namensraum des attributierten Dateisystems ausschließlich über dessen Schnittstelle ausgeführt werden. Diese Schnittstelle besteht aus einer abstrakten Fabrik zur Erzeugung von Instanzen von Unterklassen der Dateiobjektklassen wie beispielsweise `java.io.File`, `java.io.FileInputStream` oder `java.io.FileOutputStream` sowie einer konkreten Implementierung der Fabrik. Die konkrete Fabrik erzeugt insbesondere als Objekte der Klasse `java.io.File` solche der Unterklasse `durasoft.filesystem.attributed.AttributedFile`, die um Zugriffsmöglichkeiten auf die Attribute der repräsentierten Datei erweitert ist. Ein dem Konzept der Dateisystemfabrik ähnlicher, jedoch nur in Ansätzen umgesetzter Ansatz findet sich im Java-Paket *Swing* in der Klasse `javax.swing.filechooser.FileSystemView`. Diese abstrahiert zur Darstellung von Verzeichnissen in Dateiauswahldialogen durch Verwendung von Fabrikmethoden ebenfalls von konkreten Dateiobjektklassen.

Da das in C entwickelte RCE direkt auf der Dateischnittstelle des Betriebssystems aufsetzt, ist die Konsistenz des attributierten Dateisystems bei Verwendung von RCE im allgemeinen nicht gewährleistet; zusätzliche Maßnahmen zur Kompensation entstehender Inkonsistenzen sind daher erforderlich. Zunächst muß stets sichergestellt sein, daß nicht der logische, sondern der physikalische Pfad einer attributierten Datei an RCE übergeben wird. Erstellt RCE eine Datei im Namensraum des attributierten Dateisystems, so müssen zur Wiederherstellung der Konsistenz die Attribute der von RCE erzeugten Datei nachträglich initialisiert werden. Wenn RCE eine Datei löscht, so müssen auch der Cache-Eintrag und die Datei mit den zugehörigen Attributen gelöscht werden. Ferner erzeugt RCE temporäre Dateien, deren Entfernung unter ungünstigen Umständen ausbleiben kann. Der für die Wiederherstellung der Konsistenz notwendige zusätzliche, fehlerkompensierende, fern von der Fehlerursache plazierte Kode vergrößert unnötigerweise die Komplexität des Gesamtsystems, stört die Kapselung der Verwaltung der Attribute innerhalb des attributierten Dateisystems und vermindert die Verständlichkeit des Gesamtsystems. Nach dem Einbuchen einer Datei beispielsweise muß die Methode `CHECKIN` der Delta-V-Implementierung die gerade eingebuchte Datei nochmals explizit löschen, um auch deren Attributedatei zu löschen und somit die Konsistenz des Systems wiederherzustellen. Zu von RCE erzeugten Dateien generiert das attributierte Dateisystem zur Wiederherstellung der Konsistenz nachträglich die fehlende Attributedatei, sobald es bei einer Ausführung einer Operation auf dieser Datei das Fehlen der Attributedatei entdeckt. Eine Lösung der Probleme bietet auch das Konzept der Fabrik

aus der Klasse `javax.swing.filechooser.FileSystemView` nicht, da ein darauf aufbauendes attributiertes Dateisystem vor denselben Problemen stünde. Eine Anpassung von RCE an das attributierte Dateisystem zwecks Bereinigung der Konsistenzprobleme erscheint unangemessen. Als zukünftige Weiterentwicklung denkbar ist hingegen die Integration von RCE als Teil des attributierten Dateisystem. Das entstehende versionierte attributierte Dateisystem könnte intern Inkonsistenzen selbsttätig vorbeugen.

Anstelle der Abbildung eigenschaftsbehafteter Resourcen auf attributierte Dateien ist ebenso eine Abbildung auf Datensätze einer Datenbank denkbar. Dies brächte vor allem den Vorteil, eine transaktionsorientierte Schnittstelle zur Verwaltung der Eigenschaften auf der Basis des Datenbanksystems verhältnismäßig einfach realisieren zu können. Die derzeitige Implementierung auf der Basis des Dateisystems des zugrundeliegenden Betriebssystems kann die Forderung des WebDAV-Protokolls nach Atomizität bestimmter gebündelter Anfragen hingegen ohne größeren Aufwand nur unvollkommen erfüllen.

3.3.3.3 DAV-Modul

Die Realisierung von WebDAV als Dienstgebermodul definiert neue und redefiniert bestehende Methoden und fügt dem Dienstgeber neue Kopfzeilen und Statuskodes hinzu. Die Methode `PUT` wird redefiniert, um die Eigenschaften einer neu angelegten Resource zu initialisieren; die Methode `OPTIONS` wird um die Rückgabe der Kopfzeile `DAV` zur Angabe der vom Dienstgeber unterstützten DAV-spezifischen Fähigkeiten ergänzt. Neu hinzu kommen die Methoden `PROPFIND`, `PROPPATCH` und `MKCOL`. Die Methoden `MOVE`, `COPY` und `LOCK` werden in der derzeitigen Implementierung noch nicht unterstützt. Die von WebDAV definierten Kopfzeilen werden syntaktisch vollständig und semantisch insoweit unterstützt, wie sie von den implementierten Methoden verwendet werden. Die Statuskodes von WebDAV werden allesamt unterstützt.

Das DAV-Modul definiert ferner eine Schnittstelle zur Verwaltung der Eigenschaften von Resourcen, hinter der sich eine Zuständigkeitskette von Bearbeitern verbirgt. Methoden wie `PROPFIND` und `PROPPATCH` bauen auf dieser Schnittstelle auf. Neben der Möglichkeit, die als XML gespeicherten Eigenschaften einzeln zu setzen und zu erfragen, kann auch eine Aufzählung aller auf einer Resource definierten Eigenschaften angefordert werden. Dies ist für die Implementierung der Methode `PROPFIND` notwendig, die die Auflistung aller Eigenschaften einer Resource unterstützen muß. Module wie Delta-V können durch Hinzufügen zusätzlicher Bearbeiter zur Zuständigkeitskette beliebige neue Eigenschaften definieren, deren Existenz auch auf spezielle Resourcentypen beschränkt sein kann.

3.3.4 Delta-V

Die dienstgeberseitige Implementierung von Delta-V ist als eigenständiges Dienstgebermodul auf der Basis von RCE als Revisionskontrollsystem konzipiert, das den Dienstgeber um die in Delta-V spezifizierten HTTP-Methoden, Nachrichtenkopfzeilen und Statuskodes erweitert. Das Modul erweitert ferner die HTTP-Methode `GET` sowie die Eigenschaftsverwaltung des WebDAV-Moduls, um über die dauerhaften Resourcenbezeichner auf Revisionen zugreifen zu können, so als ob es eigenständige Resourcen unabhängig von der zugehörigen versionierten Resource wären. Zur Realisierung werden alle Zugriffe auf eine Revision über einen dauerhaften Resourcenbezeichner dienstgeberintern auf einen entsprechenden Zugriff auf die Revision der zugehörigen versionierten Resource respektive des dahinterstehenden RCE-Archivs abgebildet.

Ein nicht zu unterschätzendes Problem stellt die Einschränkung dar, daß Archivnamen unter RCE stets die Dateiendung `.rca` aufweisen müssen. Da unter Delta-V versionierte Resourcen nicht dem Namensschema von RCE folgen müssen, muß der Dienstgeber intern eine zusätzliche Namensabbildung durchführen. Weil auf versionierten Resourcen im allgemeinen nahezu alle der in HTTP, WebDAV und Delta-V spezifizierten HTTP-Methoden ausgeführt werden können, erfolgt die Namensabbildung aufwendig durch Erweiterung aller betroffenen Zuständigkeitsketten des HTTP-Dienstgebers. Eine deutlich einfachere Alternative wäre, die Abbildung innerhalb des attributierten Dateisystems selbst vorzunehmen, da alle HTTP-Methoden, die im Namensraum des attributierten Dateisystems operieren, mit logischen Dateinamen auf dessen Schnittstelle arbeiten und daher nicht angepaßt werden müßten. Das für die Zukunft geplante, jedoch noch nicht umgesetzte Konzept der Bindeglieder (vgl. Abschnitt 3.1.3) kann hier Abhilfe schaffen, indem ein Archiv innerhalb des attributierten Dateisystems in einem getrennten Verzeichnis gespeichert wird und über ein geeignetes Bindeglied im Namensraum dem Dienstnehmer gegenüber unter dem gewünschten Namen erscheint.

3.4 Architektur des Dienstnehmers

Unter Verwendung von VRCE beschränkt sich die Realisierung des Dienstnehmers lediglich auf die Implementierung der Repository-Schnittstelle von VRCE zur Anbindung an Delta-V.

3.4.1 Repository-Schnittstelle

Die Implementierung der Repository-Schnittstelle zu VRCE via Delta-V bildet die gesamte von der Schnittstelle zur Verfügung gestellte, RCE-spezifische Funktionalität auf das Versionsmodell eines Delta-V-Dienstgebers ab und realisiert

sie durch Kommunikation mit dem Dienstgeber. Der Verzicht der Repository-Schnittstelle auf Transaktionsprozeduren kommt dem statusfreien Wesen der Kommunikation unter HTTP entgegen. Ferner lassen sich dadurch die Methoden der Schnittstelle als Sammlung von Dienstprozeduren, die nicht untereinander kommunizieren müssen, weitgehend unabhängig voneinander implementieren. Gleichwohl nutzen die meisten der Methoden gemeinsam viele der Bibliotheksfunktionen zu HTTP, WebDAV und Delta-V, die im Zuge der Entwicklung des HTTP-Dienstgebers und seiner Erweiterungsmodule entstanden.

Der nachfolgende Abschnitt betrachtet exemplarisch die Umsetzung einer der Methoden der Repository-Schnittstelle ein wenig genauer. Er zeigt, wie sich das Durchlaufen eines Revisionsgraphen durch Anwendung der Methode `REPORT` mit nur einer einzigen Anfrage an den Dienstgeber realisieren läßt.

3.4.1.1 Durchlaufen des Revisionsgraphen

Eine häufig ausgeführte Funktion von VRCE ist das Durchlaufen des gesamten Revisionsgraphen. Ein Durchlauf findet statt

- wenn eine Revision etwa zum Ausbuchen nach vorgegebenen Kriterien wie Autor oder Datum der Einbuchung gesucht werden soll,
- um eine Darstellung des Revisionsgraphen zu erzeugen sowie
- bei der zusammenfassenden Darstellung von Informationen über ein Archiv in der Art eines Logbuches.

Bei der Architektur des Dienstnehmers verdient die Berücksichtigung einer effizienten Umsetzung solcher Durchläufe besondere Bedeutung. Die Repository-Schnittstelle enthält daher zur Realisierung von Durchläufen in VRCE die Methode `getRevisionAttributes()`, um alle Attribute zu allen Revisionen eines Archivs einschließlich der aller Schablonen in einem einzigen Methodenaufruf zurückzuliefern. Eine sich aufgrund der RCE-Schnittstelle ergebende übliche Implementierung dieser Methode wäre, beginnend bei der ersten Revision des Hauptastes eine Tiefen- oder Breitensuche auf den von dort aus erreichbaren Nachfolgerevisionen durchzuführen. Mit den Methoden `Revision.getNextRevision()`, `Revision.getBranchFirst()` und `Revision.getBranchNext()` ist ein solcher Durchlauf verhältnismäßig effizient durchführbar, da diese Instanzenmethoden unmittelbar auf den Daten der aktuellen Revision arbeiten.

Übertrüge man diese Vorgehensweise sinngemäß auf die Kommunikation mit einem Delta-V-Dienstgeber, so hieße dies, zunächst durch eine Anfrage auf der versionierten Resource mit dem Befehl `PROPFIND` die initiale Revision zu ermitteln. Der in der Antwort enthaltene dauerhafte Resourcenbezeichner böte dann die Möglichkeit, mit einem weiteren `PROPFIND` über diesen Resourcenbezeichner

die Eigenschaften der initialen Revision, darunter auch `DAV:sucessor-set` mit den Resourcenbezeichnern der Nachfolgerrevisionen, zu ermitteln. Auf diese Weise ließen sich sukzessive die Eigenschaften aller benötigten Revisionen Schritt für Schritt ermitteln.

Der Kommunikationsaufwand dieser in zahlreichen Einzelanfragen resultierenden Methode ist enorm. Man mag einwenden, daß unter Ausnutzung eines Pipelineeffektes, d.h. des vorzeitigen Verschickens der nächsten Anfrage an den Dienstgeber noch vor Erhalt einer Antwort der vorhergehenden Frage, die zahlreichen Einzelanfragen kein großes Problem darstellen. Ein möglicher Pipelineeffekt ist zwar vorstellbar (vgl. Abschnitt 8.1.2.2. von HTTP/1.1[39]). Doch müssen sowohl Dienstgeber wie auch Dienstnehmer auf diese spezielle Form der Kommunikation vorbereitet sein, wovon im allgemeinen nicht ausgegangen werden kann. Verglichen mit der Realisierung durch eine einzelne Anfrage bleibt zudem der zusätzliche Verwaltungsaufwand der vielen Einzelnachrichten einschließlich ihrer zu großen Teilen redundanten Nachrichtenköpfe; dies betrifft gleichermaßen den Dienstgeber wie den Dienstnehmer.

Speziell in der Implementierung des Delta-V-Dienstgebers zu dieser Arbeit werden ferner momentan noch keine dauerhaften Verbindungen (*keep-alive connections*) unterstützt. Daher muß mit jeder Anfrage eine neue TCP/IP-Verbindung aufgebaut werden, was zusätzliche Zeit kostet. Sowohl beim Dienstgeber wie auch beim Dienstnehmer werden durch häufiges Öffnen und Schließen von HTTP-Verbindungen zudem Speicherresourcen ständig belegt und wieder freigegeben, was ebenfalls zusätzlichen Aufwand verursacht. Es ergibt sich somit der Wunsch, Eigenschaften von der Gesamtheit aller Revisionen einer versionierten Resource in einer einzigen Antwort auf eine einzige Anfrage hin zu erhalten.

Zur Bündelung von Anfragen nach Eigenschaften bietet WebDAV, wie bereits beschrieben, die Möglichkeit der Anwendung des Befehls `PROPFIND` auf Hierarchien von Resourcen. Das Problem, `PROPFIND` auf allen Revisionen einer versionierten Resource auszuführen, läßt sich auf diese Weise jedoch nicht befriedigend lösen. Denn Delta-V macht keinerlei Vorgaben, wie die dauerhaften Resourcenbezeichner auszusehen haben, über die die Revisionen angesprochen werden; lediglich ihre Eindeutigkeit und Dauerhaftigkeit werden gefordert. Dies erscheint sinnvoll, wenn man bedenkt, daß versionierte Resourcen ihre Resourcenbezeichner ändern können, die Revisionen aber dennoch verläßlich auffindbar bleiben sollen. Im allgemeinen muß daher davon ausgegangen werden, daß die dauerhaften Resourcenbezeichner der Revisionen verstreut auf dem Dienstgeber oder aber auch zusammen mit vielen anderen Resourcen in einer einzigen Sammlung liegen können. Die Fähigkeit von `PROPFIND` zum Sammeln der Eigenschaften vieler Resourcen in einer einzigen Anfrage hilft in solchen Fällen nicht weiter.

Eine elegante Lösung des Problems bietet der in Delta-V eingeführte Befehl `REPORT` (vgl. Abschnitt 3.1.4). Obgleich seine eigentliche Aufgabe darin besteht, einen Bericht über eine versionierte Resource in der Art eines Logbuches anzufertigen, lassen sich mit `REPORT` unter Nutzung der Eigenschaften

`DAV:revision-set` und `DAV:working-resource-id-set` in einer einzigen Anfrage alle Eigenschaften aller Revisionen und Arbeits-Resourcen erfragen, für die andernfalls ein Durchlaufen des Revisionsgraphen in vielen Einzelanfragen durch den Dienstnehmer nötig wäre. Mit `REPORT` wird somit das Durchlaufen des Revisionsgraphen mit nur einer einzigen Anfrage an den Delta-V-Dienstgeber möglich.

Kapitel 4

Bewertung

Zur Bewertung der beiden Ansätze zur verteilten Revisionskontrolle wurden verschiedene Kriterien ausgewählt. Die Schnittstellenarchitektur untersucht, wie die zu erbringenden Aufgaben auf Dienstgeber und Dienstnehmer verteilt werden. Dienstleistungen können in sitzungsorientierter oder statusfreier Form angeboten werden. Weitere Bewertungskriterien sind der Funktionsumfang der Schnittstelle zwischen Dienstgeber und Dienstnehmer, die Portabilität, Interoperabilität, Sicherheit und Effizienz.

Speziell bei VRCE via RMI ist die Serialisierung im Hinblick auf die verwendeten Datenstrukturen zur Übermittlung von Informationen zwischen den beteiligten Dienstpartnern von Interesse. Zu VRCE via Delta-V sollen darüberhinaus Spezifikationsfehler und -schwächen in WebDAV exemplarisch betrachtet und bewertet werden, die sich bei der Implementierung herauskristallisierten.

4.1 Schnittstellenarchitektur

Sowohl VRCE via RMI wie auch VRCE via Delta-V implementieren verteilte Revisionskontrolle nach dem Dienstgeber/Dienstnehmer-Modell. Die Architektur der Schnittstelle zwischen Dienstgeber und Dienstnehmer gibt den Rahmen für die Möglichkeiten der Kommunikation zwischen den beteiligten Partnern vor. Daher verdient die Architektur dieser Schnittstellen besondere Beachtung.

Bei VRCE via RMI ergibt sich die Schnittstellenarchitektur im wesentlichen durch die Entscheidung, welche Klassen als entfernt deklariert und somit über ihre Stellvertreterobjekte adressiert werden. Die Klassen `durasoft.rce.RCE`, `durasoft.rce.Archiv` und `durasoft.rce.Revision` der Java-Schnittstelle von RCE werden aus den in Abschnitt 2.2.2 beschriebenen Gründen über ihre Stellvertreterobjekte adressiert; alle anderen Objekte werden als Wertparameter übergeben. Daraus folgt, daß zwecks Minimierung der Kommunikation zwischen Dienstgeber und Dienstnehmer die Schnittstelle der drei genannten Klassen so beschaffen sein sollte, daß die im Auftrage des Dienstnehmers zu erledigenden Auf-

gaben mit möglichst wenig Methodenaufrufen und möglichst kompakten Parametern realisierbar sind. Dies wiederum läuft auf ein breiter gefächertes Spektrum an Funktionen hinaus, das über eine minimale Kernfunktionalität hinausgeht. Die Java-Schnittstelle von RCE stellt solche zusätzlichen Funktionen zur Verfügung; als Beispiel hierfür können `Archive.deleteRevisions()` und Revisionssuchfunktionen wie `Revision.findLatest()` oder `Revision.branchFindFirst()` genannt werden. Die Java-Schnittstelle erlaubt ferner die Bündelung von Einzelanfragen durch Methoden wie `Revision.getAttributes()` und `Archive.getRevisionAttributes()`, die ebenfalls zur Reduzierung der Anzahl entfernter Methodenaufrufe beitragen.

Im Falle von VRCE via Delta-V ist die Schnittstelle zwischen Dienstgeber und Dienstnehmer durch die in Delta-V und den darunterliegenden Protokollen wie HTTP und WebDAV spezifizierte Kommunikation gegeben. Delta-V verwendet auf dieser Basis verschiedenste Repräsentationsformen für Datenobjekte zur Kommunikation wie HTTP-Kopfzeilen, Nachrichteninhalt mit XML-Anweisungen oder Nachrichteninhalt mit Nutzdaten. Die Vielfalt dieser Repräsentationsformen gepaart mit der Ausdrucksmächtigkeit von Methoden wie `PROPPATCH`, `PROPFIND` und `REPORT` ermöglicht dem Dienstnehmer eine reichhaltige Parametrisierung und damit recht genaue Anforderungsbeschreibung der gewünschten Funktionalität. `PROPFIND` und `REPORT` bieten zudem die Möglichkeit der gebündelten Anfrage.

4.2 Dienstleistungsmodell

Dienstleistungen, die ein Dienstgeber einem Dienstnehmer anbietet, können in loser Form voneinander unabhängiger, statusfreier Dienstprimitive oder in der Form einer Sitzung (*session*) angeboten werden, die Dienstprimitive zu einer Transaktion zusammenfaßt.

RCE verwendet eine sitzungsorientierte Schnittstelle. So können nach einmaligem, exklusiven Öffnen oder Erzeugen eines Archivs vielfache Operationen auf dem Archiv und seinen Revisionen ausgeführt werden. Erst durch explizites Speichern oder mit dem bestätigenden Schließen werden diese Änderungen festgeschrieben; durch Rücksetzen hingegen läßt sich das Archiv jederzeit in den Zustand vor seiner letzten Speicherung versetzen.

Dieses Transaktionsmodell der lokalen Schnittstelle überträgt sich bei Verwendung von VRCE via RMI auf die Dienstgeber/Dienstnehmer-Kommunikation. Auch wenn es hauptsächlich dazu dient, konkurrierende Zugriffe auf Archive auszuschließen, beschleunigt es nebenbei insofern die Kommunikation und Ausführung, als das betroffene Archiv nur einmalig für die gesamte Transaktion geöffnet und wieder geschlossen werden muß.

HTTP als Basisprotokoll zu VRCE via Delta-V ist statusfrei. Während bei sitzungsorientierten Protokollen üblicherweise eine Verbindung aufgebaut, Kommu-

nikationsparameter gegebenfalls einmalig ausgehandelt, Operationen ausgeführt, und die Verbindung schließlich geschlossen wird, erfolgt bei HTTP jede Anfrage unabhängig von den Kommunikationsparametern der vorhergehenden Anfrage. Daher müssen Kommunikationsparameter wie die zur Authentifizierung des Dienstnehmers mit jeder Anfrage neu formuliert werden, was zusätzlichen Verwaltungsaufwand erfordert. Andererseits sind statusfreie Protokolle wie HTTP robuster gegen Störungen, da bei Unterbrechung der Verbindung zwischen Dienstgeber und Dienstnehmer schlimmstenfalls die zuletzt ausgeführte Anfrage verlorengehen oder verfälscht werden kann.

Transaktionen lassen sich bei statusfreien Protokollen nur mit zusätzlichen Maßnahmen realisieren. Daher führt WebDAV das Konzept der Sperren ein, um eine Serialisierung konkurrierend zugreifender Dienstnehmer erzwingen zu können. Zusammen mit HTTP-Methoden, die den Status des Dienstgebers ändern, wie beispielsweise `PUT`, `POST`, `DELETE` oder `MOVE` lassen sich Transaktionen auf der Basis des an sich statusfreien HTTP-Protokolls emulieren. In diesem Sinne ist Delta-V als statusfreies Protokoll zu bewerten, auf dem sich dennoch Transaktionen realisieren lassen, wenn auch mit zusätzlichem Aufwand.

4.3 Funktionsumfang

Bei VRCE via RMI läßt die Transparenz der entfernten Methodenaufrufe den Aufrufer keinen Unterschied in der Art des Zugriffs zwischen lokalem und entferntem Aufruf erkennen. Der Funktionsumfang von VRCE via RMI ist daher identisch mit dem der lokalen Anwendung von VRCE, da beide als Schnittstelle zu RCE im wesentlichen die den Funktionsumfang definierenden drei Klassen `durasoft.rce.RCE`, `durasoft.rce.Archive` und `durasoft.rce.Revision` beziehungsweise deren Stellvertreterobjekte verwenden.

Im Hinblick auf VRCE via Delta-V beschränkt sich, wie bereits geschildert, diese Arbeit auf die Betrachtung der reinen Versionskontrolle von Delta-V. Wie der Abschnitt zur Modellierung der Java-Schnittstelle von RCE über Delta-V gezeigt hat, ist das Versionsmodell von Delta-V dem von RCE in etwa vergleichbar. Der Funktionsumfang von Delta-V entspricht daher in etwa dem von RCE, wenn auch die Handhabung teilweise sehr verschieden ist, wie der Abschnitt über das Durchlaufen eines Revisionsgraphen beispielhaft gezeigt hat.

Delta-V verfügt durch die Infrastruktur von WebDAV und HTTP darüberhinaus über zahlreiche Funktionen zur Verwaltung von Resourcen, die jedoch nicht Bestandteil der Versionskontrolle selbst, sondern als Teil der Kommunikation zwischen Dienstgeber und Dienstnehmer zu sehen sind. Auf der anderen Seite bietet RCE zahlreiche Funktionen, die die Verwendung der Schnittstelle von RCE erleichtern, jedoch keine prinzipielle Erweiterung der Funktionalität darstellen.

Es bleibt festzuhalten, daß der Funktionsumfang von VRCE via RMI ebenso wie der der reinen Revisionskontrolle von Delta-V in etwa dem von RCE ent-

spricht. Die breite Infrastruktur von WebDAV und HTTP bietet bei VRCE via Delta-V darüberhinaus zahlreiche Möglichkeiten einer diversifizierten, den jeweiligen Bedürfnissen anpaßbaren Kommunikation.

4.4 Portabilität

Der in C implementierte Teil von RCE ist für viele verschiedene Plattformen bereits verfügbar, und die Struktur des Kodes erleichtert die Portierung auf weitere Plattformen. Die in Java implementierten Teile sind für alle Plattformen verfügbar, für die es eine lauffähige Version von Java ab der Version 1.2 gibt. VRCE via RMI ist daher auf allen Plattformen verfügbar, die sich als Schnittmenge der beiden erstgenannten Mengen von Plattformen ergibt.

Die Portabilität eines Delta-V-Dienstgebers oder -Dienstnehmers ist unabhängig von Delta-V als Protokoll, da sowohl Delta-V wie auch die Protokolle der darunterliegenden Infrastruktur die Kommunikation zwischen Dienstgeber und Dienstnehmer unabhängig von deren jeweiliger Realisierung spezifizieren. Für die Portabilität zeichnet vielmehr die konkrete Implementierung verantwortlich. Im Falle von VRCE via Delta-V als konkreter Implementierung gelten dieselben Aussagen wie für VRCE via RMI, da auch VRCE via Delta-V auf dem in C implementierten Kern von RCE und der Verwendung von Java für alle übrigen Teile basiert.

4.5 Interoperabilität

VRCE via RMI als proprietäre Lösung eines Ansatzes zur verteilten Revisionskontrolle verwendet als Kommunikationsprotokoll RMI auf der Basis der RCE-inhärenten Datenstrukturen. Eine Interoperabilität mit anderen Revisionskontrollsystemen ist daher nicht gegeben. Plattformspezifische Eigenheiten wie Dateisegmenttrenner behandelt VRCE via RMI in einer Weise, die es zumindest erlaubt, Dienstgeber und Dienstnehmer unter verschiedenen Plattformen laufen zu lassen.

Als plattformübergreifendes Kommunikationsprotokoll des Internet zielt Delta-V auf größtmögliche Interoperabilität zwischen verschiedenen Implementierungen. Das Protokoll beschreibt nur die Erfordernisse der Kommunikation zwischen Dienstgeber und Dienstnehmer, ohne Einschränkungen in bezug auf deren Implementierung zu machen. Somit sollte jede Implementierung eines Delta-V-Dienstnehmers in der Lage sein, mit jeder Implementierung eines Delta-V-Dienstgebers erfolgreich zu interoperieren.

Einschränkungen in der Interoperabilität bei Delta-V können möglicherweise entstehen, wenn verschiedene Dienstgeber oder Dienstnehmer verschiedene Teilmengen der optionalen Funktionalität von Delta-V implementieren. Eine genaue-

re Bewertung dieser Problematik erfordert die in dieser Arbeit nicht unternommene Analyse des Advanced Versioning.

Weitaus bedenklicher ist die Beeinträchtigung der Interoperabilität zwischen verschiedenen Delta-V-Dienstnehmern mit proprietären Erweiterungen, die mit demselben Delta-V-Dienstgeber kommunizieren, etwa wenn einer der Dienstnehmer benutzerdefinierte Eigenschaften verwendet, die ein anderer nicht zu interpretieren vermag. Dies kann schlimmstenfalls zur Verfälschung oder Zerstörung der benutzerdefinierten Eigenschaften eines Dienstnehmers durch einen anderen Dienstnehmer führen.

4.6 Sicherheit

Beide betrachteten Ansätze zur verteilten Revisionskontrolle bieten Möglichkeiten zur Realisierung von Sicherheitmaßnahmen in bezug auf Authentifizierung, Verschlüsselung und Abwehr von Überlastangriffen.

Als proprietärer Lösung stehen VRCE via RMI alle erdenklichen Möglichkeiten zur Authentifizierung und Verschlüsselung der übertragenen Daten offen. RMI schränkt diese Möglichkeiten insofern nicht ein, als die Serialisierung der Daten unter RMI redefinierbar ist. Damit lassen sich im Prinzip beliebige Verschlüsselungsverfahren mit der Serialisierung verbinden. Zudem unterstützt RMI die Verwendung von Verbindungen, die nach dem Standard *Secure Socket Layer* (*SSL*), einem Standard zum Aufbau und Betrieb einer sicheren Transportverbindung über TCP/IP-Verbindungen, arbeiten.

Authentifizierung und Verschlüsselung als Maßnahmen zur Zugriffskontrolle und für eine sichere Transportverbindung für Delta-V sind bereits in HTTP vorgesehen. HTTP sieht als optionalen Authentifizierungsmechanismus Digest Authentication vor, ist zugleich aber offen für die Definition neuer Mechanismen. WebDAV fordert die Unterstützung von Digest Authentication als Authentifizierungsmechanismus, zumindest wenn eine sichere Verbindung nicht gewährleistet werden kann. Ungeachtet dessen bleibt die Unterstützung weiterer, möglicherweise noch sicherer Authentifizierungsmechanismen jeder Implementierung eines Delta-V-Dienstgebers freigestellt. Das allgemeine Schema zur Enkodierung unter HTTP mit der Kopfzeile `Transfer-Encoding` ermöglicht bei Bedarf die Verschlüsselung der zu übertragenden Daten. HTTP ist auch hier offen für die Definition neuer Enkodierungen. Daher können unter Delta-V Daten unter der Anwendung beliebiger Verschlüsselungsverfahren übertragen werden.

Ein beliebte Strategie zum mutwilligen Angriff eines Dienstgebers stellen Überlastangriffe (*denial of service attacks*) dar. Um sich vor Überlastung zu schützen, stellen die meisten Dienstgeber ihre Dienste nur bis zum Erreichen einer Grenzlast zur Verfügung. Ein Dienstnehmer, der nach Erreichen dieser Grenzlast einen Dienst in Anspruch nehmen möchte, wird vom Dienstgeber abgewiesen. Bei einem Überlastangriff versucht ein Angreifer, Dienste eines Dienstgebers miß-

bräuchlich in Anspruch zu nehmen, um durch Erzeugen einer hohen Last die Abweisung von Diensten für andere Dienstnehmer zu erzielen. Ziel der Attacke ist somit die Beeinträchtigung der Verfügbarkeit des Dienstgebers. Solange der Quelltext von VRCE via RMI nicht offen zugänglich ist, werden Angriffe in Form von Überlastangriffen mangels Kenntnis des Systems erschwert. Allerdings ist durch die Verfügbarkeit der Quellen von RMI bereits ein bedeutender Teil der Kommunikation zwischen Dienstgeber und Dienstnehmer dokumentiert, so daß Angriffe auf VRCE via RMI allein unter Ausnutzung möglicher Schwächen von RMI nicht auszuschließen sind. Hier sind gegebenfalls zusätzliche Schutzmaßnahmen zur Erkennung und Abwehr von Angriffen vorzusehen.

Die freie Zugänglichkeit der Protokollspezifikationen zu Delta-V und der darunterliegenden Infrastruktur offenbart mögliche Angriffspunkte der Implementierungen und kann daher unter Ausnutzung protokollspezifischer Eigenheiten beispielsweise zur Ausführung von Überlastangriffen mißbraucht werden. Hier sind gegebenfalls zusätzliche Schutzmaßnahmen vorzusehen, um solche Angriffe zu erkennen und abzuwehren.

4.7 Kommunikationsprotokoll

Sowohl VRCE via RMI wie auch VRCE via Delta-V unterstützen den Zugriff durch verteilte Dienstnehmer auf die Dienste eines entfernten Dienstgebers, der einen Datenbestand zentral verwaltet und speichert. Der wesentliche Unterschied zwischen beiden Ansätzen besteht in der Kommunikation zwischen Dienstgeber und Dienstnehmer, die auf recht verschiedenen Prinzipien aufbaut und daher für die beiden Ansätze schlecht direkt verglichen werden kann. Deshalb soll vielmehr auf besondere Eigenheiten der beiden Protokolle eingegangen werden.

4.7.1 VRCE via RMI

Bei der Serialisierung der Kommunikationsdaten unter VRCE via RMI werden neben den eigentlichen Nutzdaten auch deren Datentypen berücksichtigt, um die typenwiederherstellende Deserialisierung allein auf der Basis der serialisierten Daten durchführen zu können. Bei Objekten enthält die Codierung des Datentyps den vollständig qualifizierten Objektnamen, etwa „`durasoft.rce.client.RemoteFile`“. Das hierfür erforderliche Speicher- beziehungsweise Übertragungsvolumen kann daher leicht das der Nutzdaten übertreffen. Dies trifft insbesondere dann zu, wenn Datenstrukturen verwendet werden, bei denen die Nutzdaten ohne Verwendung homogener Strukturen wie Felder auf viele Objekte verteilt werden.

Alternativ zur Verbesserung der verwendeten Datenstrukturen im Hinblick auf die Serialisierung läßt sich die standardmäßige Serialisierung redefinieren. So wird es beispielsweise auch möglich, die Typinformation von der Übertra-

gung auszunehmen, wenn man wie bei VRCE via RMI davon ausgehen kann, daß Dienstgeber und Dienstnehmer die gleichen Datenstrukturen verwenden und daher die Typinformation der serialisierten Daten kennen.

Die Serialisierung tief verschachtelter Datenstrukturen bei der Wertübergabe kann zu Effizienzproblemen führen. Andererseits ist durch das einmalige Kopieren aller am Objekt beteiligten Daten der nachfolgende Kommunikationsaufwand oftmals deutlich geringer. Die Referenzübergabe führt zu erhöhtem Kommunikationsaufwand, wenn nachfolgend Operationen auf dem Stummel stattfinden, die zum Ursprungsort des übermittelten Parameters weitergeleitet werden müssen, hat aber den Vorteil, daß die Daten, auf denen operiert wird, an ihrem Ursprungsort bleiben.

4.7.2 VRCE via Delta-V

Bei der Durchsicht der für eine Delta-V-Implementierung relevanten Spezifikationen und der anschließenden Umsetzung in eine prototypische Anwendung fanden sich vor allem bei WebDAV zahlreiche Fehler und Schwächen. Hierzu gehören unter anderem Spezifikationsfehler im Sinne von falschen oder fehlenden Spezifikationen, aber auch Schwächen im Entwurf, die bei getreuer Umsetzung beispielsweise die Komplexität der Implementierung unnötig vergrößern. Die Häufigkeit und Schwere dieser Fehler und Schwächen legt eine Analyse des Spezifikationsprozesses nahe.

4.7.2.1 Fehler in der XML-Elemente-Spezifikation

WebDAV spezifiziert ein XML-Element mit dem Namen keepalive: `<!ELEMENT keepalive (#PCDATA | href+) >`. Gemäß der zur Interpretation hier anzuwendenden Syntaxregel der XML-Spezifikation[27] mit der Nummer 51 zur Deklaration eines gemischten Inhaltes (*Mixed Content*) gemäß `[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*' | '(' S? '#PCDATA' S? ')'` handelt es sich hierbei um eine ungültige Spezifikation. Wird als Korrekturmaßnahme diese durch die umfassendere Produktion `<!ELEMENT keepalive (#PCDATA | href)* >` ersetzt, so liegt, wie bei der Implementierung dieser Arbeit geschehen, der Einsatz einer zusätzlichen, geeigneten Gültigkeitsbedingung (*validity constraint, VC*) nahe, um eine Einengung auf die ursprünglich gewünschte Syntax zu erzielen. Dieser Fehler wurde durch automatische Validierung der Dokumententypdeklaration von WebDAV mit einem validierenden Zerteiler gefunden.

Ein weiterer Mangel ist das Fehlen der Deklaration des Wurzelelementes. Entsprechend der Dokumententypdeklaration zu WebDAV (Anhang 1 von RFC 2518[43]) lautet der Name der Dokumententypdeklaration „webdav-1.0“: `<!DOCTYPE webdav-1.0 [ ... ]>`. Damit wird als Wurzelelement jedes XML-Dokumentes unter WebDAV ein Element mit der Bezeichnung `webdav-1.0` festgelegt. Eine Deklaration dieses Elementes sucht man jedoch vergebens. Dieser Man-

gel der aktuellen Spezifikation ließe sich mit der folgenden zusätzlichen Elemente-Definition beseitigen: `<!ELEMENT webdav-1.0 (propfind | multistatus | propertyupdate | propertybehaviour | lockinfo | prop) >`. Hierbei wird unter Berücksichtigung der in Abschnitt 8 des WebDAV-Protokolls diskutierten Szenarien angenommen, daß innerhalb des Wurzelelementes eines gültigen Dokumentes genau eines der angegebenen Elemente als Nachkomme in Frage kommt.

4.7.2.2 Nicht validierbare XML-Syntax

XML unterscheidet, wie in Abschnitt 3.3.2 erläutert, zwischen wohlgeformten und gültigen XML-Dokumenten. Abgesehen von den beiden im vorhergehenden Abschnitt beschriebenen behebbaren Fehlern der Dokumententypdeklaration spezifiziert WebDAV einen legalen Dokumententyp. Um, nach Bereinigung der beiden Fehler, Dokumente gemäß dieses Dokumententyps zerteilen zu können, sieht XML die Verwendung eines validierenden Zerteilers vor.

Die in den Beispielen der WebDAV-Spezifikation angegebenen XML-Dokumente sind zwar wohlgeformt, jedoch aus zwei Gründen nicht gültig im Sinne der fehlerbereinigten DTD. Zum einen enthalten sie nicht, wie von einem gültigen XML-Dokument gefordert, die Angabe des Dokumententyps im Prolog des Dokuments („an XML document is *valid* if it has an associated document type declaration and if the document complies with the constraints expressed in it“). Zum anderen fordert die Gültigkeitsbedingung zu Syntaxregel `[28]` (`[VC: Root Element Type]`), daß der Name in der Dokumententypdeklaration mit dem Elementtyp des Wurzelelementes übereinstimmen muß. Demzufolge müßte daher jedes gültige XML-Dokument unter WebDAV ein Wurzelelement mit dem Namen „web-dav-1.0“ enthalten, was jedoch bei keinem der Beispiele aus der Spezifikation von WebDAV der Fall ist.

Die in diesem und dem vorhergehenden Abschnitt beschriebenen Fehler wurden durch Einsatz eines validierenden Zerteilers gefunden. Auch ein Fehler bei der Verwendung von Namensräumen in einem der als Beispiele angegebenen XML-Dokumente im WebDAV-Protokoll wurde auf diese Weise entdeckt. Einige der genannten Fehler wurden im Rahmen dieser Diplomarbeit erstmalig gefunden, und das, obwohl WebDAV zu diesem Zeitpunkt seit bereits etwa einem Jahr standardisiert war und bereits einige Implementierungen bestanden. Dies unterstreicht die Bedeutung des Einsatzes formaler Methoden zur Verifizierung – in diesem Falle den Einsatz eines validierenden Zerteilers. Daß WebDAV XML in einer Weise verwendet, die den Einsatz eines validierenden Zerteilers nicht vorsieht, ja sogar behindert, ist unverständlich. Mit nur geringfügigen Änderungen des Protokolls wäre ein XML-konformer Einsatz validierender Zerteiler möglich. Zu diesem Schluß ist auch eine vom Verfasser dieser Diplomarbeit angeregte Diskussion in der Arbeitsgruppe zu WebDAV[45] gelangt, wenn auch vorbehaltlich eingehenderer Prüfung. Die meisten Teilnehmer dieser Diskussion wollten jedoch die Vorteile der Validierung nicht anerkennen. Dabei wäre allein die *Möglichkeit*

zur konformen Validierung ein entscheidender Fortschritt, ohne eine Validierung durch WebDAV-Anwendungen zwingend erfordern zu wollen.

4.7.2.3 Datumsformate

Die Eigenschaft `DAV:getlastmodified` enthält gemäß WebDAV den Inhalt der Nachrichtenkopfzeile `Last-Modified`, welche gemäß HTTP eine Datumsangabe in dem unter HTTP üblichen Datumsformat nach RFC[33] enthält. WebDAV führt als neues Datumsformat neben dem aus RFC 1123 das Format gemäß ISO 8601[26] speziell für die Eigenschaft `DAV:creationdate` ein. Eine Begründung für die Einführung oder Erklärung, warum nicht das Format gemäß RFC 1123 verwendet wird, wird nicht gegeben. Als Folge ergibt sich bei der Umsetzung eine uneinheitliche und somit fehlerträchtige Behandlung von Datumsangaben.

4.7.2.4 Weitere Schwächen

Viele weitere Protokollschwächen sind in einer Liste auf einer hierfür eigens eingerichteten Webseite[46] aufgeführt. Das Ausmaß dieser Liste und der Umstand, daß mehr als nunmehr eineinhalb Jahre nach der Festschreibung von WebDAV als Standard immer wieder neue Einträge in die Liste aufgenommen werden, lassen den Schluß zu, daß eine Weiterentwicklung von WebDAV auch in Hinsicht auf die Bereinigung bestehender Schwächen unerläßlich ist.

4.7.2.5 Analyse des Spezifikationsprozesses

WebDAV dient als Ausgangspunkt zahlreicher bereits in Entwicklung befindlicher wie voraussichtlich auch zukünftiger Implementierungen. Damit muß auch die Entwicklung eines solchen Protokolls selbst als Teil eines Softwareentwicklungsprozesses betrachtet werden, der einer Bewertung unterzogen werden kann und sollte. Die auffallende Häufigkeit und Schwere der beschriebenen Schwächen von WebDAV legt daher eine nähere Analyse und Bewertung der Probleme nahe.

Als zukünftigem Standard mit vielen zu erwartenden Implementierungen auf unterschiedlichsten Systemen werden an Protokolle wie WebDAV besondere Anforderungen gestellt, speziell im Sinne bestmöglicher Interoperabilität. Nicht zuletzt zu diesem Zwecke spezifiziert das *Internet Architecture Board* (*IAB*), eine Unterorganisation der *Internet Engineering Task Force* (*IETF*), Verfahrensweisen, die zur Spezifikation von Protokoll-Standards des Internet angewendet werden müssen. Diese Spezifikation des IAB ist selbst eines aus einer Reihe von Protokoll-Standards, welches von Zeit zu Zeit überarbeitet wird (siehe etwa Internet Official Protocol Standards, RFC 2400[42]).

Ein Protokoll-Standard muß hiernach eine Reihe von Reifestufen (*Proposed Standard*, *Draft Standard*, *Standard*) durchwandern, während derer das Protokoll zunehmend verfeinert und getestet wird. An dem Prozeß dieser schrittweisen Entwicklung kann jeder Teilnehmer des Internet mitwirken. Zum Übergang des Pro-

tokolls zu einer neuen Reifestufe bis hin zur Annahme als Standard muß jedoch die *Internet Engineering Steering Group* (*IESG*), eine weitere Unterorganisation der IETF, eine entsprechende Empfehlung aussprechen. Für den Übergang vom Proposed Standard zum Draft Standard gilt es dabei als übliche Praxis, daß neben der Empfehlung der IESG mindestens zwei voneinander unabhängige Implementierungen vorliegen müssen.

WebDAV befindet sich derzeit (August 2000) auf dem Stande eines Proposed Standard; Delta-V hat den Prozeß der Standardisierung gemäß IAB bislang noch nicht begonnen. Die Forderung nach zwei voneinander unabhängigen Implementierungen erscheint in nächster Zeit jedoch für WebDAV und damit auch für Delta-V kaum erfüllbar. Nicht nur bei der Implementierung im Rahmen dieser Arbeit zeigten sich vielfache Schwierigkeiten bei der Umsetzung von WebDAV; auch andere Autoren verweisen auf eine Fülle von Problemen. Der Autor des WebDAV-Moduls für den Dienstgeber Apache beispielsweise fand zahlreiche Unzulänglichkeiten in WebDAV. Diese reichen von sprachlichen Mängeln, die zu mehrdeutigen Interpretationen führen können, über unvollständige Spezifikationen bis hin zu Widersprüchen. Die Problematik ist auf einer eigens eingerichteten Webseite[55] eingehend beschrieben.

Bei Betrachtung der Entwicklung von WebDAV und der darauf aufbauenden Protokolle einschließlich Delta-V zeigt sich ein Wandel im Vergleich zu früheren Protokollen wie HTTP/1.1. Erste experimentelle Implementierungen von HTTP-Dienstgebern und -Dienstnehmern gab es bereits um 1990[22]; hierbei konnten auch Erfahrungen mit dem Gopher-Protokoll[35] einfließen. Dabei war der Quellkode dieser Implementierungen in der Regel frei zugänglich (z.B. CERN-httpd[23]). HTTP/1.0[37] wurde etwa sechs Jahre später unter ausdrücklichen Vorbehalten der IESG als informeller RFC veröffentlicht. Erst Anfang 1997, also sieben Jahre nach den ersten Implementierungen, wurde eine von der IESG vorbehaltlos als Proposed Standard akzeptierte Protokoll-Version unter der Bezeichnung HTTP/1.1[39] freigegeben.

Die Entwicklung von WebDAV hingegen vollzieht sich anders: Obgleich es bereits Anfang 1999 als Proposed Standard veröffentlicht wurde, scheinen – wenn man die Diskussionen in der Arbeitsgruppe zu WebDAV[45] aufmerksam verfolgt – alle wesentlichen derzeit existierenden Implementierungen nur einen Teil des Protokolls umzusetzen und die Spezifikationen nicht selten eigenmächtig abzuändern, sobald Probleme bei der Umsetzung auftreten. WebDAV sieht explizit die Möglichkeit partieller Implementierungen vor, um den vielfältigen zu erwartenden Einsatzmöglichkeiten gerecht zu werden. Solange jedoch keine vollständige Implementierung besteht, ist die Entdeckung zahlreicher weiterer Protokollschwächen nicht auszuschließen. Gelegentlich werden Bestandteile von WebDAV auch offenkundig protokollwidrig umgesetzt. Ein Beispiel hierfür ist die Definition des XML-Elementes `locktoken` durch die Regel `<!ELEMENT locktoken (href+) >`, welches, den Diskussionen der Arbeitsgruppe folgend, von allen wesentlichen bestehenden WebDAV-Implementierungen offenbar vereinfachend als `<!ELEMENT`

`locktoken (href) >` implementiert wird. Als Gründe werden die Komplexität und mangelnde Notwendigkeit der ursprünglichen Regel genannt.

Viele der wesentlichen Implementierungen sind nicht als Quellkode frei verfügbar (z.B. Internet-Explorer). Dies erschwert Bemühungen zur Verbesserung des Protokolls, da ein Vergleich verschiedener Implementierungen dann in der Regel durch Testen erfolgen muß. Zudem scheint es, wenn man den zunehmend aus dem privatwirtschaftlichem Umfeld angehörenden Entwicklern Glauben schenken darf, zuweilen zu Interessenskonflikten zwischen offenen Standards und proprietären Lösungen zu kommen[54]; möglicherweise ist dies im Zusammenhang mit der zunehmenden Zahl von Fällen proprietärer Abweichungen in der Implementierung der an sich offenen Standards zu sehen. Häufige Diskussionen in der Arbeitsgruppe zu WebDAV zu Fragen zur Lesart des Protokolls lassen ferner auf sprachliche Mängel im Protokoll schließen.

4.7.2.6 Fazit

Es entsteht ein Gesamteindruck von WebDAV als zwar sehr nützlichem, vielversprechenden Protokoll, dem es jedoch im jetzigen Zustand noch an Reife fehlt, da es – etwa im Vergleich zum HTTP-Protokoll – offenbar unter großer Eile in kürzester Zeit zum Proposed Standard avancierte. Aufgrund der rasch zunehmenden Anzahl großenteils kommerzieller Entwicklungen, die WebDAV einsetzen, wird es zunehmend schwieriger werden, Änderungen am Protokoll durchzusetzen.

4.8 Effizienz

Um einen Eindruck von der Leistung von VRCE über RMI und VRCE via Delta-V zu erhalten, wurden Messungen zur Bestimmung der Laufzeitkosten durchgeführt.

Es sei nochmals darauf hingewiesen, daß die im Rahmen dieser Arbeit unter engstem Zeitplan angefertigte Implementierung eines Delta-V-Dienstgebers sowie die Erweiterung der bestehenden Anwendung VRCE zur Anbindung an einen Delta-V-Dienstgeber sich in einem sehr frühen Stadium befinden, in dem genaue zeitliche Messungen zur Effizienz nur bedingt tauglich sind; denn unter den gegebenen Bedingungen mußte eine schnelle Umsetzung Vorrang vor einer auf hohe Effizienz zielenden Implementierung haben. Eine genauere Effizienzanalyse deckt daher vor allem Schwächen der vorliegenden Implementierung und weniger die des betrachteten Ansatzes zur verteilten Revisionskontrolle an sich auf. Auch die Implementierung von VRCE via RMI bietet noch Möglichkeiten zur Effizienzverbesserung. Daher erfolgt die Bewertung vorbehaltlich der noch ausstehenden Weiterentwicklung und Verbesserung der zugrundeliegenden Implementierungen nur grob im Sinne einer größenordnungsmäßigen Abschätzung.

Dementsprechend wurden Meßwerkzeuge ausgewählt und der Meßvorgang

festgelegt. Die so erhaltenen Meßergebnisse dienten als Grundlage zur Bewertung der Effizienz. Die Messungen erfolgten auf einem typischen Rechner als Meßumgebung mit typischen Testdaten. Sie wurden für den lokalen Fall von VRCE, für VRCE via RMI und für VRCE via DeltaV unter möglichst denselben Bedingungen durchgeführt, um einen Vergleich zwischen beiden Ansätzen sowie zum lokalen Fall zu ermöglichen.

4.8.1 Meßumgebung

Als Meßumgebung diente ein PC-kompatibler Einzelplatzrechner mit Pentium-133-Prozessor, 48 MByte Hauptspeicher, 100 MByte Auslagerungsspeicher, Festplatte mit 8 ms Zugriffszeit und 512 kByte Cache. Als Betriebssystem wurde Linux 2.2.7 eingesetzt; die Implementierung basiert auf RCE Version 3.1 Linux ELF und Java (build Linux_JDK_1.2_pre-release-v2, green threads, sunwjit).

4.8.2 Meßwerkzeuge

Eine manuelle Messung mittels Stoppuhr als Meßwerkzeug zur Laufzeitbestimmung wurde im Sinne einer groben Messung als ausreichend erachtet. Mit dem Werkzeug `xosview` (vgl. Abbildung 4.1) zur grafischen Anzeige einiger für die Systemlast wichtigen Merkmale unter dem Fenstersystem X11 wurden während der Messungen Beobachtungen durchgeführt. Zur Beobachtung prozeßspezifischer Daten von Dienstgeber und Dienstnehmer diente das Kommandozeilenprogramm `top` (vgl. Abbildung 4.2), welches unter anderem Speicherbedarf und Prozessorausnutzung für jeden Prozeß einzeln anzeigt.

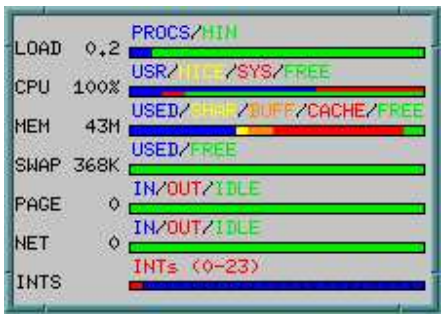

Abbildung 4.1: Das Werkzeug `xosview` zur Anzeige wichtiger Merkmale zur Systemlast unter dem Fenstersystem X11

4.8.3 Testdaten

Als Testdaten dienten die Quelldateien des Dateisystems des Linux-Kernels 2.2.7. Diese sind auf Linux-Installationen typischerweise unter dem Verzeichnispfad `/usr/src/linux/fs` zu finden und wurden in ein Benutzerverzeichnis kopiert,

```
  3:04am  up  5:35,  4 users,  load average: 0.14, 0.14, 0.09
41 processes: 37 sleeping, 4 running, 0 zombie, 0 stopped
CPU states: 35.3% user,  8.9% system,  0.0% nice, 56.1% idle
Mem:   46764K av,  46208K used,    556K free,  17256K shrd,   2392K buff
Swap: 104380K av,   3000K used, 101380K free                 17968K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
 1958 reuter    10   0  4468 4468   832 R       0 38.6  9.5   0:01 java
 1251 reuter     1   0  1140 1140   940 S       0  2.9  2.4   6:05 xosview.bi
 1957 reuter     3   0   768  768   596 R       0  1.5  1.6   0:02 top
 1263 reuter     0   0  1200 1060   612 S       0  0.1  2.2   0:01 tcsh
  165 reuter     0   0   600    0     0 SW      0  0.0  0.0   0:01 tcsh
  190 reuter     0   0   176    4     0 SW      0  0.0  0.0   0:00 startx
  198 reuter     0   0   132    0     0 SW      0  0.0  0.0   0:00 xinit
  202 reuter     0   0  1156 1156   860 S       0  0.0  2.4   0:05 fvwm
 1248 reuter     0   0   736  736   272 S       0  0.0  1.5   0:00 xterm
 1664 reuter     0   0  6048 6048  2696 S       0  0.0 12.9   7:14 emacs
 1943 reuter     0   0  6600 6600   928 S       0  0.0 14.1   0:04 java
```

Abbildung 4.2: Typische Bildschirmausgabe des Werkzeuges `top` zur Anzeige von Prozeßmerkmalen

Anzahl der Dateien (ohne Verzeichnisse)	352
Anzahl der Verzeichnisse	28
Gesamtgröße aller Dateien	3711869 Byte
Durchschnittliche Größe einer Datei	10545 Byte
Durchschnittliche Anzahl von Dateien pro Verzeichnis	13

Abbildung 4.3: Wichtige Merkmale der Testdaten

um sie dort unter Revisionkontrolle zu stellen. Abbildung 4.3 gibt einen Überblick über wichtige Merkmale der Testdaten.

4.8.4 Meßvorgang

Der verwendete Meßvorgang nutzt die Möglichkeit von VRCE, eine Operation auf einem Bündel von Dateien ausführen zu können. VRCE erlaubt hierzu die Auswahl vollständiger Verzeichnisbäume. Die gebündelte Ausführung der Operation erfolgt danach in zwei Phasen. In der ersten Phase werden aus allen ausgewählten Dateien diejenigen herausgefiltert, auf die die Operation angewendet werden kann. In der zweiten Phase wird die Operation auf den gefilterten Dateien ausgeführt. Für die Messungen wurde die Möglichkeit der Bündelung genutzt, um je eine Operation auf allen 352 Testdateien in jeweils einem Meßdurchgang durchzuführen. Der betrachtete Meßvorgang umfaßt die Messung der Laufzeiten der beiden Phasen für die drei Operationen *neues Archiv anlegen*, *letzte Revision ausbuchen* und *Revision einbuchen* auf allen Testdateien.

Operation		Lokal ∑	Lokal ⌀	RMI ∑	RMI ⌀	RMI Ratio	Delta-V ∑	Delta-V ⌀	Delta-V Ratio
anlegen	(1)	0' 07"	0.02"	2' 16"	0.39"	19	1' 50"	0.31"	16
anlegen	(2)	0' 35"	0.10"	4' 41"	0.80"	8	36' 00"	6.14"	62
ausbuchen	(1)	0' 11"	0.03"	1' 59"	0.34"	11	4' 00"	0.68"	22
ausbuchen	(2)	1' 32"	0.26"	4' 01"	0.68"	2.6	90' 00"	15.3"	59
einbuchen	(1)	0' 12"	0.03"	2' 54"	0.49"	15	4' 30"	0.77"	22.5
einbuchen	(2)	1' 32"	0.26"	5' 19"	0.9"	3.5	58' 00"	9.9"	38

Abbildung 4.4: Ergebnisse zur Laufzeitmessungen

4.8.5 Meßergebnisse

Abbildung 4.4 zeigt die Ergebnisse der Zeitmessungen für den lokalen Fall, für VRCE via RMI und VRCE via Delta-V. Die Auftrennung der Operationen in die beiden genannten Phasen ist durch Angabe der Phase in Klammern hinter dem Namen der Operation gekennzeichnet. Die mit dem Summenzeichen gekennzeichneten Spalten bezeichnen die Gesamtdauer für die Ausführung der jeweiligen Operation auf allen 352 Dateien. Die mit dem Symbol „⌀“ überschriebenen Spalten enthalten die durchschnittliche Dauer pro Datei. Die Spalten „Ratio“ schließlich geben die relative Dauer normiert auf den lokalen Fall (= 1) an, um zwischen lokalem Fall, VRCE via RMI und VRCE via Delta-V direkt vergleichen zu können.

Während der Messungen mit VRCE via RMI zeigte `xosview` eine Belegung des Auslagerungsspeichers an, deren Wert sich im Bereich von etwa 19 bis 20 MByte bewegte. Seitenwechsel waren nur in geringem Maße zu beobachten. Den Ausgaben des Werkzeuges `top` zufolge blieben der Speicherbedarf von Dienstgeber und Dienstnehmer nahezu konstant bei 10 MByte respektive 35 MByte.

Bei den Messungen zu VRCE via Delta-V zeigte `xosview` während der Messungen eine Belegung des Auslagerungsspeichers an, deren Wert im Verlauf der Messungen von anfänglich etwa 18 MByte auf schließlich etwa 42 MByte anstieg. Seitenwechsel waren in mäßig geringem Maße zu beobachten; sie dürften die Meßergebnisse leicht, jedoch nicht entscheidend beeinflußt haben. Den Ausgaben des Werkzeuges `top` zufolge stieg der Speicherbedarf des Delta-V-Dienstgebers von anfänglich etwa 10 MByte auf schließlich etwa 17 MByte; der Speicherbedarf des Dienstnehmers blieb nahezu konstant bei etwa 35 MByte.

4.8.6 Bewertung der Meßergebnisse

Daß der Rechner, auf dem die Messung erfolgte, nicht an ein Netzwerk angebunden ist, stellt für die Bewertung der Meßergebnisse keinen Mangel dar. Vielmehr wird dadurch eine Beeinflussung der Messungen durch von fremden Quellen verursachten Netzwerkverkehr ausgeschlossen. Der Hauptspeicher des Rechners ist mit 48 MByte etwas knapp bemessen, wie die beobachtete Belegung des Ausla-

gerungsspeichers zeigt. Eine entscheidende Verfälschung des Meßergebnisses ist jedoch angesichts der wenigen Seitenwechsel auszuschließen.

Der Vergleich von VRCE via RMI mit der lokalen Variante von VRCE zeigt, daß sich die Laufzeit unter Verwendung von RMI im Mittel mehr als verdoppelt, in der ersten Phase sogar auf das bis zu Achtfache ansteigt. Die hohen Kosten für die erste Phase sind verständlich, wenn man bedenkt, daß in dieser Phase zahlreiche, ungebündelte Methodenaufrufe erfolgen, hinter denen sich schnell ausführbare Operationen wie das Testen der Existenz einer Datei verbergen. Somit entstehen via RMI hohe Kommunikationskosten bei geringen Prozeßkosten. Die scheinbar hohen Kommunikationskosten relativieren sich jedoch, wenn man bedenkt, daß die Kosten der ersten Phase insgesamt verhältnismäßig klein gegenüber denen der zweiten Phase sind. Mit steigender Dateigröße wachsen die Prozeß- und Kommunikationskosten in der zweiten Phase. Bei Archiven mit zahlreichen Revisionen und komplexen Deltas und ist zu erwarten, daß das Wachstum der Kommunikationskosten für den Transport der Dateien gering gegenüber dem der Prozeßkosten ist. Dies führt in der Praxis zu einer weiteren Relativierung der scheinbar hohen Kommunikationskosten von VRCE via RMI.

VRCE via RMI weist gegenüber lokalem VRCE für die betrachteten Testdaten eine deutliche Vergrößerung der Laufzeitkosten auf. Ursache sind die zusätzlichen Kommunikationskosten durch den Einsatz von RMI. Für Archive mit vielen Revisionen und komplexen Deltas ist eine Relativierung die Mehrkosten im positiven Sinne zu erwarten.

Der Vergleich mit der lokalen Variante von VRCE zeigt ein auf den ersten Blick ernüchterndes Ergebnis. Zu berücksichtigen ist jedoch der sehr frühe prototypische Status der Implementierung. Diese enthält derzeit noch Anweisungen, die zahlreiche Kontrollausgaben zwecks fortschreitender Fehlerbereinigung erzeugen. Allein durch das Entfernen der Ausgabe des Systemstapels an einer einzigen Programmstelle konnte die Dauer der ersten Phase des Meßvorganges zum Anlegen neuer Archive von ehedem etwa 6 Minuten auf die hier angegebenen 1' 50", also etwa ein Drittel, reduziert werden. Durch die Entfernung aller übrigen Kontrollausgaben ist mit einer weiteren deutlichen Beschleunigung zu rechnen.

Ein weiteres Problem von VRCE via Delta-V stellt die derzeitige Architektur der Eigenschaftsschnittstelle dar, die, wie im Abschnitt 3.3.3.2 bereits erläutert, mangels Mechanismen zur Transaktionsverwaltung das Lesen und Setzen von Eigenschaften nur in voneinander isolierten Operationen erlaubt. Als Folge hiervon kann die Anwendung der Methoden `PROPFIND`, `PROPPATCH` und `REPORT` auf eine Revision oder versionierte Resource unnötigerweise zum mehrfachen Öffnen und Schließen des dahinterstehenden RCE-Archivs führen. Die Anwendung dieser Methoden auf Arbeits-Resourcen hingegen ist im Vergleich dazu unproblematisch, da hier der Cache zur Speicherung der Eigenschaften zwischengeschaltet ist. Beim Ausbuchen wird `REPORT` zum Auffinden einer geeigneten Revision auf die versionierte Resource angewendet, gefolgt von `PROPFIND` auf die gefundene Revision und einem späteren `PROPPATCH` auf die neu geschaffene Arbeits-Resource. Die

anderen beiden betrachteten Operationen sind in dieser Hinsicht weniger problematisch. Der Ausreißer von 90 Minuten zum Ausbuchen der letzten Revision läßt sich daher mit der derzeit unbefriedigend implementierten Eigenschaftsschnittstelle erklären.

Der Anstieg des Speicherbedarfs des Dienstgebers bei VRCE via Delta-V während des Meßvorganges dürfte im wesentlichen auf das allmähliche Auffüllen des Cache-Speichers zum Zwischenspeichern von Eigenschaftsdateien zurückzuführen sein. Ob angesichts des deutlichen Anstiegs des Speicherbedarfs und der nur 48 MByte Hauptspeicher des verwendeten Systems eine geringere Dimensionierung des Cache angebracht ist, bleibt näheren Untersuchungen vorbehalten.

Im übrigen gelten auch hier die genannten Überlegungen zur Relativierung der hohen Kommunikationskosten. Es bleibt festzuhalten, daß VRCE via Delta-V gegenüber lokalem VRCE eine noch deutlichere Vergrößerung der Laufzeitkosten als VRCE via RMI aufweist. Aufgrund des frühen Stadiums der Implementierung ist jedoch noch mit wesentlichen Verbesserungen zu rechnen.

Kapitel 5

Verwandte Arbeiten

Der allgemeine Trend zur zunehmend verteilten Softwareentwicklung macht eine Erweiterung existierender Systeme zur Versionskontrolle und Konfigurationsverwaltung notwendig. Neben der impliziten Verteilung durch Einsatz eines verteilten Dateisystems und WWW-basierten Ansätzen zur verteilten Revisionskontrolle wurden viele existierende Revisionskontrollsysteme explizit um Fähigkeiten zum verteilten Zugriff oder zur verteilten Speicherung revisionierter Daten erweitert. Außer DRCE[6], das RCE erweitert und als Vorgänger von VRCE via RMI angesehen werden kann, zählen hierzu auch bekannte Systeme mit eigenen Ansätzen zur verteilten Revisionskontrolle wie CVS[7], ClearCase ([14], [15]) und Continuus/CM ([12], [13]).

5.1 Implizite Verteilung

Über verteilte Dateisysteme wie das *Network File System* (*NFS*) lassen sich für lokalen Gebrauch konzipierte Revisionskontrollsysteme wie RCS oder RCE in unveränderter Form verteilt verwenden. Dabei läuft das Revisionskontrollsystem lokal auf dem Dienstnehmer ab; lediglich Dateioperationen auf entfernt liegende Dateien wie Archive werden auf Betriebssystemebene über das verteilte Dateisystem weitergereicht und von einem Dateidienstgeber (*file server*) bedient. Dabei entsteht ein verteilter Zugriff, wenn mehrere Dienstnehmer über NFS von verschiedenen Rechnern aus auf dasselbe Archiv zugreifen. Enthält das über NFS verbundene Netzwerk mehrere Dateidienstgeber, so können Archive auch verteilt gespeichert werden. Da verteilte Revisionskontrolle auf der Basis verteilter Dateisysteme auf die Reichweite des Dateisystems beschränkt ist, kommt sie praktisch nur bei lokalen Netzwerken in Betracht.

5.2 WWW-basierte Ansätze

Für eine weltweit verteilte Revisionskontrolle ist der Einsatz des Internet und, spezieller, des World Wide Web, naheliegend; Systeme auf dieser Basis realisieren gemäß der Konzeption des World Wide Web eine Dienstgeber/Dienstnehmer-Beziehung.

In der Entwicklung des World Wide Web spielte der Gedanke, Revisionskontrolle zu integrieren, schon sehr früh eine Rolle. Frühe WWW-basierte Systeme versuchen, diese Integration unter Verwendung gängiger WWW-Dienstgeber (*web server*) und -Dienstnehmer (*web browser*) vorzunehmen. Sie verwenden zumeist spezielle Resourcenbezeichner zur Selektion von Revisionen einer Resource. Der Revisionsbezeichner wird zusammen mit einem Archivbezeichner zu einem Resourcenbezeichner kodiert, etwa in der Form `http://www.foo.bar/user_x/project_y/file_z?read;rev=2.7`. Typische WWW-Dienstgeber lassen sich beispielsweise über eine als *CGI* (*common gateway interface*) bezeichnete Schnittstelle um die benötigte Funktionalität erweitern. Über HTML-Formulare können Parameter vom WWW-Dienstgeber an den WWW-Dienstgeber übergeben und Aktionen wie Einbuchung oder Ausbuchung ausgelöst werden. Die Funktionalität des WWW-Dienstnehmers kann durch Einsatz von *Hilfsapplikationen* (*helper applications*) erweitert werden, die bei Erhalt speziell gekennzeichneter Resourcen durch den Dienstnehmer lokal ausgeführt werden. WWRC[20] ist ein Beispiel für die Anwendung dieser Techniken. BSCW[8] ist ein weiteres System, das die Benutzerführung weitgehend über HTML-Formulare realisiert. Eine Verfeinerung der Technik ergibt sich durch den Einsatz von Java-Applets anstelle von HTML-Formularen. Neben einer verfeinerten Benutzeroberfläche erlaubt der Einsatz von Java auch mehr Gestaltungsspielräume in der Kommunikation zwischen Dienstnehmer und Dienstgeber. WWCM[21] und MKS Web Integrity[11] sind Beispiele für diese Techniken. Das Sicherheitskonzept von Java erfordert allerdings zusätzliche Kommunikationszyklen zwischen Dienstgeber und Dienstnehmer.

Den genannten Systemen gemeinsam ist der Versuch, alleine mit Mitteln des reinen HTTP-Protokolls auszukommen. Dies führt zumeist zu unbefriedigenden Lösungen. Die Systeme unterliegen Beschränkungen in der Benutzbarkeit, sind langsam oder versuchen mit architektonischen Verrenkungen trickreich Unzulänglichkeiten zu umgehen. Der Ansatz von Delta-V besteht daher darin, das HTTP-Protokoll selbst gemäß den Anforderungen eines Revisionskontrollsystems mit verteiltem Zugriff zu erweitern.

5.3 DRCE

Distributed RCE, oder kurz *DRCE*, ist eine frühere Erweiterung von RCE mit dem Ziel, verteilte Revisionskontrolle auf der Basis von RCE zu realisieren. Diese

im Rahmen einer Studienarbeit[6] entstandene Erweiterung kann als Vorläufer von VRCE via RMI angesehen werden, setzt im Unterschied zu VRCE via RMI jedoch direkt auf der C-Schnittstelle von RCE auf. DRCE spezifiziert die Funktionalität von RCE in der in Abschnitt 2.1 bereits erwähnten Schnittstellendefinitionssprache IDL und definiert ein Objektmodell zur Fernbenutzung für RCE nach einem Dienstgeber/Dienstnehmer-Modell ähnlich dem von VRCE via RMI. Die Aufgaben der Registratur von VRCE via RMI zur Herstellung der Verbindung zwischen Dienstgeber und Dienstnehmer übernimmt bei DRCE ein Dienst mit der Bezeichnung *Cell Directory Service*. Wie VRCE via RMI unterstützt DRCE den verteilten Zugriff auf zentral gespeicherte Archive, jedoch keine verteilte Speicherung der Archive.

5.4 CVS

Concurrent Versions System (*CVS*) bezeichnet ein kommandozeilenorientiertes Revisionskontrollsystem, das wie RCS und RCE auf der Basis des Checkin/Checkout-Modells Revisionen von Dateien speichert und verwaltet und den Zugriff auf entfernt gespeicherte Archive ermöglicht. Wie bei RCS werden Operationen wie Einbuchungen und Ausbuchungen im Logbuch der betreffenden Datei protokolliert. Über die Funktionalität einer reinen Revisionskontrolle hinaus ermöglicht CVS die Behandlung ganzer Hierarchien versionierter Dateien in einem Schritt. Zur Unterstützung entfernter Zugriffe sieht CVS ähnlich wie DRCE ein auf entfernten Methodenaufrufen basierendes Dienstgeber/Dienstnehmer-Modell vor. Es realisiert die entfernten Methodenaufrufe, indem der Dienstnehmer eine Verbindung zum Dienstgeber über einen entfernten Kommandointerpreter (*remote shell, rsh* oder *secure shell, ssh*) aufnimmt. Dazu muß der Anwender beim Aufruf eines Kommandos zusätzlich lediglich den Rechnernamen des Dienstgebers sowie einen Benutzernamen angeben, unter dem der Dienstgeber den Dienst ausführen soll. CVS kann durch diese zusätzlichen Angaben Zugriffe auf entfernt liegende Archive von lokalen Zugriffen unterscheiden und ruft im entfernten Fall selbständig einen entfernten Kommandointerpreter auf. Die Authentifizierung und Verschlüsselung der übertragenen Daten erfolgt gegebenenfalls im Rahmen der Verbindung zu dem verwendeten entfernten Kommandointerpreter; die Authentifizierung kann alternativ auch durch einen eigens beim Dienstgeber aufgesetzten Anmelde-Dienst erfolgen.

5.5 ClearCase

ClearCase ist ein komplexes kommerzielles System zur Revisionskontrolle und zum Konfigurationsmanagement, welches wie RCE auf dem Modell des Ein- und Ausbuchens basiert. In der Grundversion ermöglicht es verteilte Revisionskontrol-

le innerhalb lokaler Netzwerke. Die Erweiterung MultiSite ClearCase unterstützt die globale Vernetzung lokaler Netzwerke mit der Möglichkeit der verteilten Speicherung revisionierter Daten.

Die Grundversion stellt innerhalb eines lokalen Netzwerkes ein virtuelles, revisioniertes Dateisystem zur Verfügung. Dieses Dateisystem wird – vergleichbar mit NFS – durch eine Dienstnehmer/Dienstgeber-Architektur innerhalb des Netzwerkes realisiert. Ein- und Ausbuchungen erfolgen nicht explizit durch den Benutzer, sondern implizit durch Schreibe- und Leseoperationen auf dem revisionierten Dateisystem. Durch die Definition einer *Sicht* bestimmt der Benutzer, nach welchen Regeln Revisionen für ihn als Dateien des virtuellen Dateisystems sichtbar sein sollen. Alle übrigen Revisionen bleiben, sofern sie nicht explizit über spezielle Funktionen angesprochen werden, unsichtbar. Das revisionierte Dateisystem macht den Zugriff auf Revisionen transparent; es zeigt sich sowohl dem Benutzer wie auch Anwenderprogrammen gegenüber als ein einfaches, nicht-revisioniertes Dateisystem. Revisionen werden bei ClearCase intern nicht wie bei RCE in Archiven, sondern als Datenbanksätze einer als *versionierte Objekt-Basis* (*versioned object basis, VOB*) bezeichneten Datenbank gespeichert.

ClearCase MultiSite ist eine Erweiterung des Grundsystems, die die Beschränkung auf lokale Netzwerke überwindet. Es erlaubt die Duplizierung von VOBs und deren Speicherung und Verwaltung in räumlich getrennten Netzwerken und unterstützt deren Kommunikation untereinander. Dateioperationen werden dabei stets auf der jeweiligen lokalen Objekt-Basis ausgeführt, so als sei dies die einzig existierende Objekt-Basis. Die hierbei unvermeidlich entstehende Inkonsistenz zwischen den Objekt-Basen wird durch Synchronisations-Aktualisierungen aufgelöst. Bei einer solchen Aktualisierung werden Änderungen einer Objekt-Basis anderen Objekt-Basen mitgeteilt. Der Prozeß der Aktualisierung kann automatisch – etwa in regelmäßigen zeitlichen Abständen – oder explizit durch den Anwender erfolgen. Um Kollisionen bei der Erzeugung neuer Revisionen auf verschiedenen Objekt-Basen zu vermeiden, dürfen Äste nur auf jeweils einer Objekt-Basis fortgeführt werden. Hierzu wird jedem Ast ein eindeutiger Name zugeteilt. Ferner wird sichergestellt, daß zu jedem Zeitpunkt nur einer Objekt-Basis die Erlaubnis gewährt wird, den Ast mit diesem Namen fortzuführen. Soll der Ast von einer anderen Objekt-Basis fortgeführt werden, so muß die Erlaubnis zu dieser anderen Objekt-Basis migrieren.

Es bleibt festzuhalten, daß im Gegensatz zu allen bisher betrachteten rein dienstgeber/dienstnehmer-orientierten Ansätzen bei ClearCase die revisionierten Daten räumlich verteilt verwaltet werden können. Hierzu werden die revisionierten Dateisysteme einzelner lokaler Netzwerke zu einem lose gekoppelten globalen Verbund zusammengeführt. Durch Synchronisations-Aktualisierungen werden entstehende Inkonsistenzen aufgelöst.

5.6 Continuus/DCM

Auch das kommerzielle Konfigurationsmanagementsystem Continuus/CM ist in einer Grundversion für lokale Netzwerke und in einer erweiterten Version für den Netzwerkverbund erhältlich.

Die Grundversion verwendet im lokalen Netzwerk wiederum eine Dienstgeber/Dienstnehmer-Architektur. Im Gegensatz zu ClearCase ist diese Architektur jedoch nicht auf Betriebssystemebene in der Form eines virtuellen Dateisystems realisiert, sondern implementiert Dienstgeber und Dienstnehmer auf der Anwendungsebene. Dementsprechend werden dem Anwender Werkzeuge zur Verfügung gestellt, mit denen er auf die in einer Datenbank verwalteten und gespeicherten Daten zugreifen kann.

Die Erweiterung Continuus/DCM hebt die Beschränkung auf lokale Netzwerke auf und ermöglicht es, Daten global verteilt halten, verwalten und zugreifbar machen zu können. Dazu definiert es drei *Methodologien* der Verteilung, zwischen denen je nach Bedarf gewählt werden kann: *hub and spoke*, *publish and subscribe* und *peer-to-peer*. Bei der mit *hub and spoke* bezeichneten Methodologie wird eine Datenbank als Sammelstelle (*hub*) ausgezeichnet. Die Sammelstelle ist verantwortlich, Änderungen, die in Zuträgerdatenbanken (*spokes*) durchgeführt werden, zu sammeln. Bei der Methodologie *Veröffentlichen und Abonnieren* (*publish and subscribe*) wird eine zentrale Datenbank als *Publizist* ausgezeichnet. Auf dieser Datenbank werden Daten gespeichert, die als *gemeinsame Objekte* global zugreifbar sein sollen. *Abonnenten* bezeichnen Datenbanken, die Informationen vom Publizist beziehen. Der Informationsfluß erfolgt beim Bezug immer in der Richtung vom Publizist zum Abonnenten. Die als *peer-to-peer* bezeichnete Methodologie schließlich verteilt die Daten auf gleichberechtige Datenbanken, ähnlich dem Konzept von ClearCase MultiSite.

Continuus verwendet mit der beschriebenen Konzeption aus Continuus/CM für lokale Netzwerke und Continuus/DCM für globale Verteilung ebenfalls wie ClearCase ein zweistufiges Modell des Konfigurationsmanagements.

5.7 Fazit

VRCE via RMI und VRCE via Delta-V bieten mit ihrem Dienstgeber/Dienstnehmer-Modell verteilten Zugriff auf zentral gespeicherte Archive. Viele andere Systeme wie DRCE und CVS und die meisten der WWW-basierten Systeme fallen ebenfalls in diese Kategorie. Die globale Verteilung revisionierter Daten einschließlich einer Verwaltung zur Wahrung ihrer Kohärenz bleibt augenblicklich großen Systemen wie ClearCase und Continuus vorbehalten.

Kapitel 6

Zusammenfassung und Ausblick

Die Bewertung der beiden in dieser Arbeit betrachteten Ansätze zur verteilten Revisionskontrolle über das Internet hat zwei recht unterschiedliche Konzepte zur Umsetzung der Verteilung auf der Basis entfernter Zugriffe auf einen zentralen Dienstgeber aufgezeigt, die vor dem Hintergrund derselben Benutzeroberfläche VRCE und auf Basis desselben Revisionskontrollsystems RCE betrachtet wurden.

VRCE via RMI als proprietäre Lösung zur verteilten Revisionskontrolle über das Internet ermöglicht durch weitgehende Ungebundenheit von der Einhaltung von Standards entsprechend viele Freiheitsgrade bezüglich der konkreten Umsetzung des verwendeten Kommunikationsprotokolls auf der Basis von RMI. Dies erlaubt zahlreiche Ansatzpunkte zur Verbesserung der Laufzeiteffizienz im Sinne von Latenz und Durchsatz, von denen einige, auch teils noch nicht umgesetzte, in der Bewertung zu diesem System erörtert wurden.

VRCE via Delta-V ist an Delta-V als plattform- und anwendungsunabhängigem Kommunikationsprotokollstandard gebunden, der auf der breiten Infrastruktur rund um HTTP und WebDAV aufbaut. Ansatzpunkte zur Effizienzverbesserung sind nur in dem durch den Protokollstandard gesetzten Rahmen möglich. Mit mächtigen, universellen HTTP-Methoden wie `PROPFIND`, `PROPPATCH` und `REPORT` wird dieser Nachteil jedoch entscheidend gemindert. In der Implementierung der außerhalb des Blickwinkels von Delta-V sich befindenden Bestandteile des Gesamtsystems freilich gelten alle erdenklichen Freiheitsgrade.

Im direkten Leistungsvergleich der beiden Ansätze auf der Basis der durchgeführten Messungen ist VRCE via RMI momentan deutlich überlegen. Zu dem schlechten Abschneiden von VRCE via Delta-V ist jedoch zu berücksichtigen, daß einerseits eine frühe Protokollversion als Basis zur Implementierung herangezogen wurde und die Implementierung selbst in großer Eile erfolgte und daher noch zahlreiche gravierende Mängel enthält, deren Beseitigung nur eine Frage der Zeit darstellt. Als eine der wesentlichen Schwachstellen des Delta-V-Dienstgebers ist die Verwaltung der Eigenschaften zu nennen. Deren Umsetzung auf das Setzen und Lesen von Attributen von RCE-Archiven bei gebündelter Anfrage erfordert momentan noch ein unnötigerweise mehrfaches Öffnen und Schließen von Archi-

ven. Unter diesem und vielen anderen Aspekten ist eine deutliche Verbesserung der Leistung von VRCE via Delta-V möglich.

Die zahlreichen in Entwicklung befindlichen Implementierungen zu WebDAV und, seit wenigen Wochen, eine zunehmende Anzahl in Aussicht gestellter Implementierungen auf der Basis von Delta-V lassen an einer zukünftigen weltweiten Verbreitung von Systemen, die Delta-V unterstützen, kaum noch Zweifel. Die Standardisierung von Revisionskontrolle in der Gestalt von Delta-V und die damit einhergehende Unabhängigkeit von Hardware und Betriebssystem lassen eine hohe Interoperabilität der verschiedensten Lösungen erhoffen, die die Akzeptanz und Verbreitung von Delta-V zunehmend erhöhen dürften. Die besondere Bedeutung von Delta-V liegt somit in dem Versuch der Vereinheitlichung bestehender Modelle der Revisionskontrolle zum Zwecke verteilten Zugriffes.

VRCE via RMI als proprietäre Lösung muß nicht im Gegensatz zu VRCE via Delta-V gesehen werden, sondern kann dieses möglicherweise ergänzen. Es ist schon jetzt prinzipiell möglich, auf mit VRCE via Delta-V erzeugte RCE-Archive über VRCE via RMI und umgekehrt zuzugreifen. Für eine totale bijektive Abbildung zwischen den von VRCE via RMI verwendeten RCE-Attributen und den Eigenschaften von VRCE via Delta-V bedarf es freilich zusätzlicher Anstrengungen. Insbesondere die Möglichkeit der Abbildung der RCE-spezifischen Attribute zur Zugriffskontrolle auf die Mechanismen der Zugriffskontrollerweiterung für WebDAV[44] bleibt zu untersuchen. Grundsätzlich denkbar wird im Falle einer totalen bijektiven Abbildung der konkurrierende Zugriff auf denselben Datenbestand aus RCE-Archiven via RMI als effiziente Lösung einerseits und via Delta-V als universellere Lösung andererseits, möglicherweise auch unter Verwendung anderer Delta-V-Dienstnehmer anstelle von VRCE.

Literaturverzeichnis

[1] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software. Addison Wesley GmbH, 1996. ISBN 3-89319-950-0

[2] Sun Microsystems Inc.: SunOS 5.7 man page: sccs - front end for the Source Code Control System (SCCS). August 1998.

[3] W.F. Tichy: RCS – A System for Version Control. In *Software – Practice and Experience*, vol. 15, no. 7, July 1985, S. 637-654.

[4] J.J. Hunt, W.F. Tichy: The RCE API – Introduction and Reference Manual. DuraSoft GmbH, February 1998.

[5] F. Lamers: Configuration Management Engine – ein programmierbares Werkzeug zur Konfigurationsverwaltung. Diplomarbeit am Institut für Programmstrukturen und Datenorganisation, Fakultät für Informatik, Universität Karlsruhe, Januar 1997.

[6] R. Hille-Döring: Distributed RCE – Ein verteiltes Versionsverwaltungssystem. Studienarbeit am Institut für Programmstrukturen und Datenorganisation, Fakultät für Informatik, Universität Karlsruhe, 1997.

[7] P. Cederqvist et al.: Version Management with CVS. GNU Info manual. Signum Support AB, Free Software Foundation, 1994.

[8] R. Bentley, T. Horstmann, J. Trevor: The World Wide Web as enabling technology for CSCW: The case of BSCW. In: Computer Supported Cooperative Work: The Journal of Collaborative Computing 2-3, S. 111-134. Kluwer Academic Press, Amsterdam, 1997.
http://bscw.gmd.de/Papers/CSCWJ-WWW/CSCW_journal.html

[9] Rational Software Corporation: Rational ClearCase – Featuring Unified Change Management. D-707a. Cupertino, California, January 2000.
http://www.rational.com/products/clearcase/prodinfo/media/ucm_ds_feb8.pdf

[10] Mortice Kern Systems Inc.: Source Integrity – User Guide. 7.5B-0300. Ontario, Canada, 2000.
`http://www.mks.com/support/techinfo/documents/si75a_user.pdf`

[11] Mortice Kern Systems Inc.: Web Integrity. Ontario, Canada, 2000.
`http://www.mks.com/solution/wi/`

[12] Continuus Software Corporation: Learning Guide for developers. Windows Client. LGD-W-044-010. Irvine, California, 1997.

[13] Continuus Software Corporation: Distributed Change Management. DCM-044-010. Irvine, California, 1997.

[14] Atria Software, Inc.: ClearCase Administrator's Manual. Document Number 4000-012-C. Lexington, Massachusetts, November 1996.

[15] Atria Software, Inc.: ClearCase MultiSite Manual. Document Number 4000-062-C. Lexington, Massachusetts, September 1996.

[16] Perforce Software: Perforce 99.2. P4 Command Line User's Guide. Manual 99.2.ug.1. December, 1999.
`ftp://ftp.perforce.com/pub/perforce/r99.2/doc/manuals/cmdref/cmdref.pdf`

[17] Horst Wettstein: Architektur von Rechnerverbundsystemen – Skriptum zur gleichnamigen Vorlesung. Institut für Betriebs- und Dialogsysteme, Universität Karlsruhe, 1995.

[18] Sun Microsystems: Learning the Java Language. In: The Java Tutorial – A practical guide for programmers. Work in progress. Sun Microsystems, 1999.
`http://java.sun.com/docs/books/tutorial/java/index.html`

[19] Ann Wollrath, Jim Waldo: Trail: RMI. In: The Java Tutorial – A practical guide for programmers. Work in progress. Sun Microsystems, 1999.
`http://java.sun.com/docs/books/tutorial/rmi/index.html`

[20] J. Reuter, S. Hänssgen, J.J. Hunt, W.F. Tichy: Distributed Revision Control Via the World Wide Web. In I. Sommerville (Ed.), *Proc. SCM-6, Software Configuration Management: Selected Papers*, LNCS 1167, Springer-Verlag, ICSE'96, SCM-6, Berlin, Germany, March 25-26, 1996, S. 166-174. `ISBN 3-540-61964-X`

[21] J.J. Hunt, F. Lamers, J. Reuter, W.F. Tichy: Distributed Configuration Management via Java and the World Wide Web. In R. Conradi (Ed.), *Proc. SCM-7, Software Configuration Management*, LNCS 1235, Springer-Verlag, ICSE'97, SCM-7, Boston, MA, May 18-19, 1997, S. 161-174. ISBN 3-540-63014-7

[22] Tim Berners-Lee: An executive summury of the World-Wide Web initiative, 1992.
http://www.w3.org/Summary.html

[23] Dan Connolly: Status of the CERN httpd, 1999.
http://www.w3.org/Daemon/

[24] The Apache Group: Apache 1.3 documentation, 1999.
http://www.apache.org

[25] ISO (International Organization for Standardization): ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML). First edition – 1986-10-15. Geneva, 1986.
http://www.iso.ch/cate/d16387.html

[26] ISO (International Organization for Standardization): ISO 8601:1988. Data elements and interchange formats – Information interchange – Representation of dates and times. Geneva, 1988.

[27] T. Bray, J. Paoli, C.M. Sperberg-McQueen: Extensible Markup Language (XML) 1.0. Textuality, Netscape, Microsoft, University of Illinois at Chicago, February 1998.
http://www.w3.org/TR/1998/REC-xml-19980210

[28] T. Bray, D. Hollander, A. Layman: Namespaces in XML. Textuality, Hewlett-Packard, Microsoft, January 1999.
http://www.w3.org/TR/1999/REC-xml-names-19990114

[29] Sun Microsystems: Java Project X. Technology Release 2. Sun Microsystems, May 1999.
http://java.sun.com/xml/

[30] S. Ramaswamy: Version Control Protocol. Internet Draft. Cisco Systems, Inc., February 1999.
http://www.ics.uci.edu/pub/ietf/webdav/versioning/
draft-ramaswamy-version-control-00.txt

[31] E. J. Whitehead, Jr.: Goals for a Configuration Management Network Protocol. In J. Estublier (Ed.), *Proc. SCM-9, System Configuration*

Management, LNCS 1675, Springer-Verlag, 9th Intern. Symp., SCM-9, Toulouse, France, Sept. 1999, S. 186-203. ISBN 3-540-66484-X

[32] J. Postel, J. Reynolds: File Transfer Protocol (FTP). RFC 959. ISI, October 1985.
http://www.ietf.org/rfc/rfc959.txt

[33] R. Braden (Editor): Requirements for Internet Hosts – Application and Support. RFC 1123. Internet Engineering Task Force, October 1989.
http://www.ietf.org/rfc/rfc1123.txt

[34] Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321. MIT, RSA Data Security, Inc. April 1992.
http://www.ietf.org/rfc/rfc1321.txt

[35] Anklesari, McCahill, Lindner, Johnson, Torrey & Alberti: The Internet Gopher Protocol. A distributed document search and retrieval protocol. RFC 1436. University of Minnesota, March 1993.
http://www.ietf.org/rfc/rfc1436.txt

[36] N. Borenstein, N. Freed: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521. Bellcore, Innosoft, September 1993.
http://www.ietf.org/rfc/rfc1521.txt

[37] Berners-Lee, et al: Hypertext Transfer Protocol – HTTP/1.0. RFC 1945. MIT/LCS, May 1996.
http://www.ietf.org/rfc/rfc1945.txt

[38] N. Freed, N. Borenstein: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045. Innosoft, First Virtual, November 1996.
http://www.ietf.org/rfc/rfc2045.txt

[39] R. Fielding, et. al.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2068. MIT/LCS, January 1997.
http://www.ietf.org/rfc/rfc2068.txt

[40] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart: An Extension to HTTP: Digest Access Authentication. RFC 2069. Northwestern University, CERN, Spyglass Inc., Microsoft, Netscape Communications, Open Market, January 1997.
http://www.ietf.org/rfc/rfc2069.txt

[41] J. Slein, F. Vitali, E. Whitehead, D. Durand: Requirements for a Distributed Authoring and Versioning Protocol for the World Wide Web. RFC 2291. Xerox Corporation, University of Bologna, U.C. Irvine, Boston University, February 1998.
`http://www.ietf.org/rfc/rfc2291.txt`

[42] J. Postel, J. Reynolds: Internet Official Protocol Standards. RFC 2400. Internet Architecture Board, September 1998.
`http://www.ietf.org/rfc/rfc2400.txt`

[43] Y. Goland, E.Whitehead, A.Faizi, S.R.Carter, D.Jensen: HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518. Microsoft, U.C.Irvine, Netscape, Novell, February 1999.
`http://www.ietf.org/rfc/rfc2518.txt`

[44] G. Clemm, A. Hopkins, E. Sedlar: Access Control Extensions to WebDAV. Work in progress. Rational Software, Microsoft Corporation, Oracle Corporation, July 2000.
`http://www.webdav.org/acl/protocol/`
`draft-ietf-webdav-acl-02.htm`

[45] E. Whitehead et al.: IETF WEBDAV Working Group Home Page. University of California, Irvine.
`http://www.ics.uci.edu/pub/ietf/webdav/`

[46] E. Whitehead et al.: WebDAV Distributed Authoring Protocol Issues List. University of California, Irvine.
`http://www.ics.uci.edu/pub/ietf/webdav/protocol/issues.html`

[47] IBM Corporation: Websphere Dav for Java: another alphaWorks technology.
`http://www.alphaworks.ibm.com/tech/DAV4J/`

[48] Jim Whitehead: IETF Delta-V Working Group Home Page. University of California, Irvine.
`http://www.webdav.org/deltav/`

[49] J. Amsden, C. Kaler, J. Stracke: Goals for Web Versioning. IBM, Microsoft, Netscape, Juni 1999.
`http://www.ics.uci.edu/pub/ietf/webdav/versioning/`
`draft-ietf-webdav-version-goals-01.txt`

[50] J. Amsden, G. Clemm: Web Versioning Model. IBM, Rational, February, 1999.
`http://www.ics.uci.edu/pub/ietf/webdav/versioning/`
`draft-ietf-webdav-versionmodel-00.html`

[51] G. Clemm, C. Kaler: Versioning Extensions to WebDAV. Rational Software, Microsoft, February 2000.
`http://www.ics.uci.edu/pub/ietf/webdav/versioning/draft-ietf-deltav-versioning-04.5.txt`

[52] J. Slein, E.J. Whitehead Jr., J. Davis, G. Clemm, C. Fay, J. Crawford: WebDAV Redirect Reference Resources. Xerox, University of California, Irvine, CourseNet, Rational, FileNet, IBM, December 1999.
`http://www.ics.uci.edu/pub/ietf/webdav/collection/draft-ietf-webdav-redirectref-protocol-02.txt`

[53] J. Slein, E.J. Whitehead Jr., J. Davis, G. Clemm, C. Fay, J. Crawford: WebDAV Bindings. Xerox, University of California, Irvine, CourseNet, Rational, FileNet, IBM, December 1999.
`http://www.ics.uci.edu/pub/ietf/webdav/collection/draft-ietf-webdav-binding-protocol-02.txt`

[54] Y. Goland: Message 2000 Jan/Mar, 0360.
`http://lists.w3.org/Archives/Public/w3c-dist-auth/2000JanMar/0360.html`

[55] Greg Stein: mod_dav: a DAV module for Apache.
`http://www.webdav.org/mod_dav/`