Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Bachelorarbeit

Implementierung einer WebDAV-Schnittstelle für die Integration in die „forcont factory FX" Software

Name:	Christof Pieloth
Matrikel-Nr.:	41103
Studiengang:	Informatik

Leipzig, 5. Oktober 2009

Betreuer der Firma:	Dipl.-Ingenieur Bernd Matzke
Betreuender Professor:	Prof. Dr.-Ing. Thomas Kudraß

Abstract

Web-based Distributed Authoring and Versioning (kurz: WebDAV) ist eine Erweiterung zum Hypertext Transfer Protocol (kurz: HTTP), das die Bereitstellung und Übertragung von Daten in einem Netzwerk ermöglicht. HTTP ist das Basisprotokoll des heutigen Internets und wurde entwickelt, um Webseiten zur Darstellung in einen Browser zu laden. Es wird überwiegend für den lesenden Zugriff auf Daten genutzt. Für den schreibenden Zugriff haben sich andere Protokolle durchgesetzt. Um dieser Entwicklung entgegenzuwirken, ohne ein weiteres Protokoll zu entwickeln, und um die hohe Verbreitung von HTTP zu nutzen, wurde beschlossen, das Basisprotokoll des World Wide Webs zu erweitern. Diese Erweiterungen sind in dem Standard Web-based Distributed Authoring and Versioning beschrieben. Mit Hilfe von WebDAV ist eine Datenübertragung in beiden Richtungen zwischen zwei Geräten möglich.

Die Software forcont factory FX ist eine Content-Management-Anwendung zur Verwaltung von geschäftsrelevanten Daten und Dokumenten, die vom Benutzer über einen Browser bedient wird. Zum Editieren eines Dokuments muss jedoch eine lokale Kopie geladen werden, welche nach dem Bearbeiten wieder zurückgeladen werden muss. Mit Hilfe von WebDAV könnte das Dokument ohne eine lokale Kopie editiert werden. Die vorliegende Bachelorarbeit beschreibt die Umsetzung einer WebDAV-Schnittstelle für den Zugriff auf die forcont factory FX Software mit dem Ziel, Aufschluss über den potenziellen Nutzen und mögliche Probleme zu erlangen.

Inhaltsverzeichnis

1 Einleitung

1.1 Relevanz und Anlass der Untersuchung

Den Anlass der Untersuchung gab die forcont business technology gmbh. Das Softwareunternehmen will anhand einer Beispielimplementierung den praktischen Nutzen des WebDAV-Standards für die forcont factory FX erproben. Durch die Umsetzung der Protokollerweiterung von bekannten Softwareherstellern und Anwendungen, wie zum Beispiel Microsoft oder Subversion[1], erkennen immer mehr Benutzer und Entwickler die damit verbundenen Möglichkeiten.

Die forcont business technology gmbh könnte sich mit einer Unterstützung von WebDAV von anderen Produkten in ihren Marktsegment abheben und ihren Kunden neue Funktionen bieten. Gerade in der IT-Branche ist es wichtig, das Potenzial neuer Technologien frühzeitig zu erkennen und als Wettbewerbsvorteil zu nutzen.

1.2 Gegenstand der Untersuchung

1.2.1 forcont factory FX Software

Die Software forcont factory FX ist ein Produkt der forcont business technology gmbh. Sie wird genutzt, um kundenspezifische Anwendungen für das Enterprise-Content-Management (kurz: ECM) zu erstellen. Die Aufgaben einer ECM sind „die Erfassung, Verteilung, Verarbeitung und Speicherung von Daten und Dokumenten im Zusammenhang mit dem jeweiligen Geschäftsprozess“ [FBT09]. Unternehmen haben heutzutage mit einer Vielfalt von Dokumenten[2] zu tun. Mehrere Angestellte können gleichzeitig mit diesen Dokumenten arbeiten und sie mit weiteren Dokumenten und Geschäftsprozessen verknüpfen. Dabei ist es wichtig, den Überblick zu behalten. Um die Informationen effizient nutzen zu können, ist ein schneller Zugriff und eine sinnvolle Ordnung notwendig. Bei der steigenden Anzahl von Dokumenten kann dies nur durch eine professionelle Software sichergestellt werden. ECM-Anwendungen sind auf diese Anforderungen zugeschnitten.

Forcont factory FX bietet „die technologische Basis für standardisierte Produkte, konfigurierbare Lösungen und individuelle Applikationen“ [FBT09]. Häufig benötigte Funktionen wie zum Beispiel das Vertragsmanagement sind fertig implementiert und können mit wenigen Mausklicks eingerichtet werden. Trotz der vorgefertigten Module ist die Anwendung „flexibel genug, um auf spezielle Kundenanforderungen angepasst zu werden“ [FBT09]. Alle Funktionen und Module können über den factory Admin Client eingerichtet und angepasst werden. Nicht nur die Konfiguration der Endanwendung ist sehr flexibel, auch das Backend lässt sich leicht in bestehende IT-Systeme integrieren. So können Daten und Dokumente auf verschiedenen Datenbanken und Dateisystemen verwaltet werden. Dem Endanwender bleiben diese unterschiedlichen Methoden verborgen. Er bedient die Software bequem und einfach über einen Browser. Dies wird durch

[1] Subversion ist eine Open-Source-Software zur Versionsverwaltung.
[2] Zum Beispiel Auftragsbestätigungen, Rechnungen und Bestellungen.

die Verwendung des Framework Adobe Flex sichergestellt, über das Desktop-ähnliche Benutzeroberflächen in einem Browser erstellt werden können.

1.2.2 Web-based Distributed Authoring and Versioning

Web-based Distributed Authoring and Versioning (kurz: WebDAV) bedeutet sinngemäß übersetzt *verteiltes, webbasiertes Verfassen von Inhalten und Versionieren.* WebDAV erweitert das Hypertext Transfer Protocol (kurz: HTTP), welches schon seit 1990 genutzt, aber erst 1996 durch den Request For Comments 1945 (kurz: RFC) genauer festgelegt wurde. 1999 wurde WebDAV im RFC2518 und 2007 im RFC4918 spezifiziert.

Behrens sowie Whitehead und Wiggins geben einen kleinen Einblick in die Entstehungsgeschichte des noch recht jungen Protokolls [Behrens, WW98]. Im Jahre 1990 entwickelte Tim Berners-Lee den ersten Browser, welcher sowohl Informationen lesen als auch schreiben konnte. Die ersten Browser der gängigsten Betriebssysteme besaßen jedoch oft nur die Möglichkeit, lesend auf eine Ressource zuzugreifen. Dies stand aber im Gegensatz zur eigentlichen Grundidee des Internets. So sollte es doch genutzt werden, um Informationen gemeinsam zu nutzen und zu tauschen. Der lesende Zugriff behinderte vor allem Personen, die überwiegend gemeinsam an Daten im Internet arbeiteten. Dadurch entstanden viele unterschiedliche und untereinander inkompatible Lösungen. Als erste kommerzielle Lösungen nennen Whitehead und Wiggins hier NaviSofts NaviPress und FrontPage von Vermeer Technologies im Jahr 1995. Mit diesen Programmen konnte entfernt auf Daten zugegriffen und gearbeitet werden. Jedoch nutzten bzw. erweiterten diese Programme das HTTP auf jeweils eine unterschiedliche Art und Weise. Nach einiger Zeit wurde erkannt, dass eine offene, für alle zugängliche Lösung geschaffen werden musste. Aufgrund der großen Verbreitung und der Entwicklung zum Standardprotokoll des Internets sollte das HTTP als Basis genutzt werden. Da nahezu alle Server und Programme diese Technik erfolgreich einsetzten, konnten die Grundbestandteile des Protokolls nicht geändert werden. Aufgrund dessen wurde 1995 die Entwicklung einer Protokollerweiterung zum bestehenden HTTP beschlossen, welche alle fehlenden Funktionen ergänzen sollte.

Die Idee von WebDAV fordert hauptsächlich drei Erweiterungen:

1. *Web-based Distributed* im Sinne von webbasiert und verteilt,
2. *Authoring* bezeichnet das Verfassen und Bearbeiten von Inhalten, und
3. *Versioning* kann allgemein als Protokollierung von Änderungen unter Vergabe von Versionsnummern und Kommentaren aufgefasst werden.

Durch die Verwendung des HTTP und eines zentralen Speicherorts können die Daten im World Wide Web (kurz: WWW) bereitgestellt und verteilt werden. Die Erweiterung für das *Web-based Distributed* ist somit ohne großen Aufwand umgesetzt, da alle wichtigen Funktionen schon durch das HTTP bereitgestellt werden.

Da im WWW mehrere Autoren an einer Datei arbeiten können, werden für das gemeinsame Verfassen und Bearbeiten vor allem zwei wichtige Funktionen benötigt: Zum einem das Sperren bzw. Entsperren von Ressourcen und zum anderen eine Zugriffskontrolle.

Wenn mehrere Benutzer gleichzeitig auf einer Ressource arbeiten, können die Änderungen anderer Benutzer durch die eigenen Änderungen beim Speichervorgang überschrieben werden. Dieses Problem wird als Lost-Update (deutsch: verlorene Änderung)

bezeichnet und kann durch ein Sperren bzw. Entsperren der Ressource verhindert werden.

Eine Zugriffskontrolle prüft, ob ein Benutzer das Recht besitzt, auf eine Ressource zuzugreifen und verweigert oder erlaubt den Zugriff auf diese. Das Request For Comments 3744 [RFC3744] beschreibt die Methoden der Zugriffskontrolle für den WebDAV-Standard.

Das Versioning oder die Versionskontrolle bildet die letzte Kernkomponente von WebDAV. So können verschiedene Ausgaben bzw. Auflagen von Daten voneinander unterschieden werden. Bei der Benutzung einer Versionskontrolle ist der Arbeitsablauf gewissen Schritten unterlegen. Vor Beginn der Arbeit sollte ein Checkout ausgeführt werden. Hierbei wird eine Arbeitskopie vom Server geladen oder eine vorhandene Datei aktualisiert. Nun kann auf dieser Version gearbeitet werden und alle Speichervorgänge werden in der Regel lokal ausgeführt. Sind alle Änderungen durchgeführt, findet der Checkin oder Commit statt. Hierbei wird die lokale Version auf den Server geladen ohne eine alte Version zu überschreiben. Stattdessen wird entweder eine neue Datei mit einer neuen Version angelegt oder es werden nur die Änderungen gespeichert. Die Hauptvorteile einer Versionskontrolle sind zum einem das Protokollieren von Änderungen und zum anderen die Möglichkeit der Wiederherstellung von alten Versionen. Ähnlich wie die Zugriffskontrolle wurde das Versioning erst später im Request For Comments 3253 [RFC3253] im Jahre 2002 festgelegt.

Im Gegensatz zu vielen anderen Transferprotokollen[3] kann WebDAV aufgrund der Bindung an das HTTP nur auf dem Port 80 betrieben werden. Da über diesen Port in der Regel auch das HTTP für das WWW betrieben wird, gibt es kaum Probleme beim Einsatz von Firewalls.

1.3 Zielsetzung

Über die forcont factory FX können Dokumente editiert werden. Obwohl die Software als Rich Internet Application (kurz: RIA) umgesetzt wurde, bleiben trotz moderner Technologien[4] einige Einschränkungen bestehen. Durch den Browser und das HTTP ist es ohne Zusatzanwendungen nicht möglich, direkt auf Dokumenten zu arbeiten. Stattdessen muss eine lokale Kopie vom Server geladen werden, die nach dem Bearbeiten wieder zurück geladen werden muss. Das Kernziel ist es, diesen Vorgang durch den Einsatz von WebDAV zu optimieren.

Des Weiteren soll eine in den Hauptbestandteilen funktionsfähige serverseitige WebDAV-Schnittstelle implementiert werden, um daraus mögliche Anwendungsfälle für die forcont factory FX Software herzuleiten. Zusätzlich sollen mögliche Probleme erkannt werden, um den Entwicklungsaufwand für eine vollständige Integration besser abschätzen zu können.

Grundsätzlich können drei Teilaufgaben gebildet werden:

1. Dokumente online editieren,
2. Dokumente ablegen und attributieren,
3. Dokumente ohne factory-Client bearbeiten und einstellen.

[3] Zum Beispiel FTP, SMB, SCP.
[4] Adobe Flex, Java Servlets, Java Server Pages, Java Applets.

Die Ergebnisse dieser Aufgaben sind prototypisch in die forcont factory FX einzubinden und zu demonstrieren.

1.3.1 Dokumente online editieren

Ein Microsoft Office-Dokument soll über die Oberfläche der forcont factory FX Software editiert werden. Dazu erzeugt die factory eine spezielle URL, die der Office-Anwendung übergeben wird. Diese URL verweist auf die WebDAV-Schnittstelle, worüber die Anwendung das Dokument lädt. Beim Speichern des Dokuments oder Schließen der Anwendung wird das editierte Dokument über die WebDAV-Schnittstelle in die factory zurück geschrieben.

Die WebDAV-Schnittstelle betrachtet in diesem Szenario nur das Dokument ohne eine Einordnung in eine Verzeichnisstruktur und unabhängig von seiner Attributierung.

1.3.2 Dokumente ablegen und attributieren

Microsoft-Betriebssysteme, wie zum Beispiel Windows XP oder Windows Vista, können ein WebDAV-Verzeichnis als so genannten Web Folder in das Verzeichnissystem einbinden. Dadurch könnten beliebige Anwendungen Dateien in den Web Folder und somit in die factory ablegen. Die WebDAV-Technologie unterstützt eine Attributierung von Dateien. Beim Schreiben eines Dokuments in einen Web Folder könnte dieses mit WebDAV-Mitteln attributiert und anschließend an die factory übergeben werden. Diese kann mit Hilfe der Attribute das Dokument an dem dafür vorgesehenen Ort ablegen. Die zu vergebenden Attribute sind aus der factory zu ermitteln.

1.3.3 Dokumente ohne factory-Client bearbeiten und einstellen

Die factory ordnet Daten und Dokumente in eine Baumstruktur ein. Diese Strukturen werden in kundenspezifischen Anpassungen (englisch: Customizing) der factory abgebildet.

Über WebDAV können Verzeichnisstrukturen in einem Web Folder abgebildet werden. Ziel ist es, einen factory-Baum im Web Folder abzubilden. Es soll möglich sein, Dokumente zu bearbeiten, zu löschen sowie neue Dokumente abzulegen. Beim Ablegen sind Dokumente, wie zuvor beschrieben, zu attributieren.

1.4 Aufbau der Arbeit

Zu Beginn dieser Arbeit werden die Grundlagen und Funktionsweisen des HTTP und WebDAV betrachtet. Die Funktionsweise des HTTP wird hierbei nur kurz erläutert. Detaillierter werden die Methoden der WebDAV-Erweiterung beschrieben.

Nachdem die Grundlagen dieser Protokolle ausreichend erläutert sind, wird mit Hilfe dieser Informationen die Implementierung beschrieben. Hierbei wird das Problem in Hinblick auf die Anforderungen und Möglichkeiten der factory und von WebDAV betrachtet. Nach erfolgreicher Umsetzung wird die Lösung beispielhaft demonstriert, Probleme und Einschränkungen werden erläutert und mögliche Lösungen gesucht.

Zum Ende dieser Arbeit werden die gewonnenen Erkenntnisse zusammengefasst und im Kontext bewertet. Abschließend wird versucht, einen kleinen Ausblick zu geben und mögliche Entwicklungen abzuleiten.

1.5 Quellen

Mit der wachsenden Bedeutung und Nutzung des Internets entwickelten sich die Request For Comments (deutsch: Bitte um Kommentare[5]) zu den bedeutesten Standards des heutigen Internets.[6] In diesen überwiegend technischen Dokumenten werden Standards und Spezifikationen festgelegt.

Da das Thema der vorliegenden Arbeit die Implementierung eines Protokolls ist, bilden folgende RFCs die wichtigsten Quellen:

- RFC1945 - Hypertext Transfer Protocol – HTTP/1.0 [RFC1945]
- RFC2518 - HTTP Extensions for Distributed Authoring – WEBDAV [RFC2518]
- RFC2616 - Hypertext Transfer Protocol – HTTP/1.1 [RFC2616]
- RFC4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV) [RFC4918]

In diesen Dokumenten sind die Spezifikationen vom HTTP und WebDAV festgelegt, wobei die letzten zwei RFCs ihre Vorgänger korrigieren und erweitern.

[5] Ein RFC wird zur Diskussion gestellt und weiterhin ausgearbeitet, bevor es durch seine allgemeine Akzeptanz und Gebrauch als Standard angesehen wird.

[6] URL (RFC 1738), SMTP (RFC 2821), IPv6 (RFC 2460) uvm. wurden in RFCs festgelegt.

2 Grundlagen

2.1 Hypertext Transfer Protocol

In diesem Abschnitt wird das HTTP in seinen Grundzügen erläutert. Die folgenden Erläuterungen basieren auf den RFCs [RFC1945] und [RFC2616].

Das HTTP wird der Anwendungsschicht zugeordnet und ist unabhängig vom eigentlichen Übertragungsprotokoll. In der Praxis wird aber fast ausschließlich das Transmission Control Protocol (kurz: TCP) verwendet, das zum Internet Protocol (kurz: IP) gehört. Ein Anfrage/Antwort-Ablauf (englisch: Request/Response) bildet die Grundlage der Übertragung. So baut ein Client die Verbindung zu einem Server über das IP auf. Anschließend sendet eine Anwendung, wie zum Beispiel ein Browser, eine Anfrage an den Server. Die Anwendung, im HTTP „user agent“ genannt, löst in der Regel den Request aus. Der Server wertet diese Anfrage aus und sendet eine Antwort zurück. Aufbau, Struktur und Ablauf dieser Anfragen und Antworten sind im HTTP genau festgelegt.

In den Spezifikationen wird die Backus-Naur-Normalform (kurz: BNF)[1] zur Beschreibung des Aufbaus einer Nachricht verwendet. Aufgrund der klaren, kurzen und genauen Schreibweise wird die BNF auch an dieser Stelle zur Beschreibung verwendet.

2.1.1 Request

Ein Request besteht allgemein aus einer Methode, einem Uniform Resource Identifier (kurz: URI), der HTTP-Version, den Request- und Client-Informationen und einem optionalen Inhaltsblock. Diese allgemeine Form kann in einer BNF genauer und detaillierter festgelegt werden.

```
Request = Request-Line
        *( General-Header
        |  Request-Header
        |  Entity-Header )
        CRLF
        [ Entity-Body ]
```

In dieser Regel wird das Nichtterminalsymbol `Request`, welches eine Anfragenachricht darstellt, definiert. Dieses setzt sich zusammen aus:

- genau einer `Request-Line`
- 0 bis unbegrenzt `General-`, `Request-` oder `Entity-Header` in beliebiger Kombination
- genau einem `CRLF` (Zeilenumbruch)
- 0 oder 1 `Entity-Body`

`Request-Line`, `General-`, `Request-`, `Entity-Header`, `CRLF` und `Entity-Body` sind Nichtterminalsymbole und werden wiederum genauer definiert.

[1] Die BNF ist eine kompakte formale Notationsform.

Request-Line

Die `Request-Line` wird definiert durch:

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Method       = "OPTIONS"
             | "GET"
             | "HEAD"
             | "POST"
             | "PUT"
             | "DELETE"
             | "TRACE"
             | "CONNECT"
             | Extension-Method
```

Somit setzt sich die erste Zeile eines Requests zusammen aus

- genau einer Methode: `OPTIONS`, `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `TRACE` oder `CONNECT` (`Extension-Method` dient als Platzhalter für mögliche Erweiterungen[2]), gefolgt von einem Leerzeichen,
- genau einem `Request-URI`, welcher in dieser Arbeit nicht genauer definiert wird, gefolgt von einem Leerzeichen,
- der `HTTP-Version`, welche ebenfalls in dieser Arbeit nicht genauer definiert wird,
- und einem abschließenden Zeilenumbruch.

Eine `Request-Line` könnte zum Beispiel so aussehen, wobei der Zeilenumbruch nicht sichtbar ist:

```
GET http://www.example.com HTTP/1.0
```

2.1.2 Response

Eine Response besteht aus einer Statuszeile, welche die HTTP-Version sowie einen Statuscode, Serverinformationen, Eigenschaften der Ressource und einen möglichen Inhaltsblock enthält.

```
Response = Status-Line
         *( General-Header
         |  Response-Header
         |  Entity-Header )
         CRLF
         [ Entity-Body ]
```

Der Aufbau von Request und Response ist sehr ähnlich: Eine Response enthält anstelle der `Request-Line` lediglich eine `Status-Line`.

[2] Wie zum Beispiel WebDAV.

Status-Line

Die `Status-Line` besteht aus

- der `HTTP-Version`, gefolgt von einem Leerzeichen,
- dem `Status-Code`, mit anschließendem Leerzeichen,
- der `Reason-Phrase` bzw. einem Hinweis,
- und dem abschließendem Zeilenumbruch.

```
Status-Line = HTTP-Version SP Status-Code SP ⤸
   Reason-Phrase CRLF
```

Status-Code und Reason-Phrase

Der `Status-Code` besteht aus einer dreistelligen ganzzahligen Zahl. Der `Reason-Phrase` soll den Code kurz beschreiben und dient in der Regel nur als lesbare Hilfestellung für den Benutzer. Der `Status-Code` selbst wird automatisch ausgewertet. Er wird anhand der ersten Ziffer in fünf Gruppen eingeteilt:

1xx Informativ - Anfrage wurde empfangen und wird weiter bearbeitet.

2xx Erfolg - Der Befehl wurde erfolgreich empfangen, verstanden und akzeptiert.

3xx Nachsendung - Weitere Maßnahmen müssen unternommen werden, um die Anfrage zu beenden.

4xx Client Fehler - Die Anfrage war fehlerhaft und kann nicht ausgeführt werden.

5xx Server Fehler - Der Server kann eine scheinbare korrekte Anfrage nicht erfolgreich bearbeiten.

Für jede Gruppe sind vollständige Codes mit einem Hinweis definiert. Erweiternde Codes können verwendet werden, jedoch müssen Anwendungen nicht jeden Code verstehen. Es reicht, wenn die Anwendung die Gruppe des empfangenen Codes interpretieren kann.

2.1.3 Header

Ein Header ist vereinfacht ein Schlüssel-Wert-Paar getrennt durch einen Doppelpunkt. Die speziellen Header setzen meist fest definierte Schlüssel voraus und beziehen sich auf die Nachricht, den Client bzw. Server oder den übermittelten Inhalt. Es wird zwischen den folgenden Headern unterschieden:

`General-Header` beziehen sich nur auf die übertragene Nachricht.

`Request-Header` enthalten Informationen über die Anfrage.

`Response-Header` enthalten Informationen über die Antwort.

`Entity-Header` beschreiben den Inhaltsblock.

Ein Header setzt sich zusammen aus

- genau einem `field-name`, der Schlüssel, dieser besteht aus einfachen Text mit einer begrenzten Anzahl an Sonderzeichen,
- gefolgt von einem Doppelpunkt,
- einem optionalen `field-value`, dem in der Regel aus Text, Leerzeichen und Tabulatoren bestehenden Wert,
- und einem den Header abschließenden Zeilenumbruch.

```
Header = field-name ":" [ field-value ] CRLF

field-name = token
field-value = *( field-content | LWS )
```

Beispiel:

```
Host: www.example.com
Accept: text/html
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859
```

2.1.4 Entity-Body

Der `Entity-Body` bildet den Inhaltsblock. Mit ihm werden die eigentlichen Daten übertragen. Das Format und die Kodierung sind im `Entity-Header` festgelegt. Im `Content-Type` wird das Format der Daten angeben. Das Format einer Portable Network Graphics-Datei (kurz: PNG) sieht zum Beispiel so aus:

```
Content-Type: image/png
```

Für eine eindeutige Kodierung sollte zusätzlich noch ein `Content-Encoding` angegeben werden. Mit Hilfe dieser Angaben können Daten sogar komprimiert übertragen werden, wie zum Beispiel mit:

```
Content-Encoding: gzip
```

2.1.5 Methoden

OPTIONS

Mit Hilfe dieser Methode können die möglichen Funktionen ermittelt werden, die der Server für diese Ressource bearbeiten kann. Diese sind Methoden, erweiternde Header und zusätzliche Anforderungen in Bezug auf die Ressource.

GET

Die GET-Methode liefert jede Art von Information zurück, die mit dem angeforderten URI identifiziert wird. Entweder werden Daten dynamisch berechnet[3] oder der Inhalt einer Ressource[4] wird direkt übermittelt. Diese Daten werden im Entity-Body einer HTTP-Nachricht übertragen.

Zusätzlich können Bedingungen angegeben werden, die für die Übermittlung der angeforderten Informationen zu erfüllen sind. Diese Bedingungen sind:

- If-Modified-Since: Wenn die Ressource seit einem angegebenen Datum geändert wurde.
- If-Unmodified-Since: Wenn die Ressource seit einem angegebenen Datum nicht mehr geändert wurde.
- If-Match: Wenn der angegebene Wert mit dem Entity-Tag übereinstimmt.
- If-None-Match: Wenn der angegebene Wert nicht mit dem Entity-Tag übereinstimmt.
- If-Range: Die Bedingung wird zusammen mit dem Range-Header gesendet, um entweder nur bestimmte Teile einer Ressource oder die gesamte Ressource anzufordern.

HEAD

Die HEAD-Methode ist nahezu identisch mit der GET-Methode. Im Gegensatz zu GET wird bei dieser Methode kein Entity-Body übertragen, sondern nur Header-Felder. Die HEAD-Methode ist daher geeignet, um Metainformationen über eine Ressource zu ermitteln oder Hypertextlinks auf Verfügbarkeit, letzte Änderungen o.ä. zu prüfen.

PUT

Mit Hilfe dieser Methode können Daten auf den Server hochgeladen werden. Als Ziel wird der Request-URI verwendet. Nach erfolgreicher Übertragung sollten die Daten auf diesem URI verfügbar sein.

POST

Die POST-Methode ähnelt PUT, ist aber bei weitem funktioneller. So können Parameter als Schlüssel-Wert-Paare übermittelt und serverseitig ausgewertet werden.

Wie bei PUT können mit dieser Methode ebenfalls Daten auf einen Server geladen werden. Im Gegensatz zu PUT muss aber der Request-URI nicht das Ziel der Übertragung sein, da der Server mit Hilfe der Parameter die Daten an einem anderen Ort ablegen kann. Somit müssen die Daten auch anschließend nicht über diesen URI verfügbar sein. Es können auch nur Parameter übertragen werden, welche serverseitig ausgewertet werden. Auf diese Weise können Daten gespeichert oder Inhalte dynamisch erstellt werden.

Die POST-Methode kann bei folgenden Anwendungsfällen verwendet werden:

- Absenden von Formulardaten, zum Beispiel bei einem Login,

[3] Zum Beispiel eine Suchanfrage bei einer Suchmaschine.
[4] Zum Beispiel der Inhalt einer HTML-Datei zur Darstellung im Browser.

- Eintrag in Foren und Datenbanken,
- Anhängen von Metainformationen, wie Kommentare oder Anmerkungen, an bestehende Ressourcen,
- Hochladen einer Datei, die nicht unter einer angegebenen URI gespeichert werden muss.

DELETE

Diese Methode löscht auf dem Server die im Request-URI angegebene Ressource.

TRACE

Bei der TRACE-Methode sendet der Empfänger die Nachricht so zurück, wie er sie empfangen hat. Eine Anfrage kann über viele Proxys und Gateways zum Server gelangen und dadurch leicht verändert werden. Mit Hilfe von TRACE kann eine Verbindung getestet und Fehler gesucht werden. Die Antwort auf die Anfrage sollte den Status-Code 200, den `Content-Type: message/http` und im Entity-Body die ursprüngliche Nachricht enthalten.

CONNECT

CONNECT wird von Proxys verwendet, die auch als SSL-Tunnel fungieren können.

2.2 Web-based Distributed Authoring and Versioning

Dieser Abschnitt erklärt die Erweiterungen zum HTTP, die als WebDAV bezeichnet werden. Die RFCs [RFC2518] und [RFC4918] sind die Grundlage der folgenden Erklärungen. Das RFC [RFC3253] über Versionierung wird nicht behandelt, da es von einer WebDAV-Implementierung nicht umgesetzt werden muss und auch in dieser Implementierung nicht benötigt wird.

Das HTTP dient WebDAV als Basis und somit wird bei der Nutzung von WebDAV auf die zuvor beschriebenen Header und Methoden zurück gegriffen. An vielen Stellen erweitert WebDAV sein Basisprotokoll: neue Methoden und Header werden definiert oder bestehende Methoden durch Bedingungen ergänzt. Die nötige Funktionserweiterung und die damit verbundene Flexibilität wird jedoch erst durch die neue Nutzung des Entity-Bodys erreicht. In diesen werden nicht mehr nur der Inhalt übertragen, sondern auch zusätzliche Daten für die Anfrage und Antwort.

Ähnlich wie bei der POST-Methode können Informationen für die Anfrage mit gesendet werden. Da jedoch in diesem Fall die Statuszeile nicht mehr als einzige Informationsquelle ausreicht, werden bei der Antwort auch zusätzliche Informationen im Entity-Body übermittelt. Dadurch wird dieser nicht mehr nur für die Übertragung der angeforderten Daten genutzt, sondern auch für Parameter, Argumente und Ergebniswerte. Diese Daten werden in Form der Extensible Markup Language (kurz: XML) übermittelt. So müssen nicht alle möglichen Parameter und Argumente in einem Standard definiert werden, sondern können ohne grundlegende Änderungen verwendet werden.

Diese Flexibilität kommt zum Beispiel den Properties (deutsch: Eigenschaften) zu Gunsten. Aufgrund der Verwendung von XML können diese Properties recht einfach oder beliebig komplex aufgebaut sein.

Eine weitere Neuerung sind die Collections. Mit diesen können Ressourcen zusammengefasst und Hierarchien erstellt werden. Sie sind mit Ordnern in einem Dateisystem vergleichbar.

Aufgrund der neuen Funktion des Locking kann sichergestellt werden, dass nur eine Person zur selben Zeit an einem Dokument arbeitet. Dies verhindert das Lost-Update-Problem.

2.2.1 Properties

Metainformationen über Daten werden im WebDAV-Standard als Properties bezeichnet. Metainformationen enthalten Daten und Informationen über andere Daten. Diese Informationen beschreiben den Inhalt der Daten genauer, wodurch ein effizienterer Umgang mit diesen ermöglicht wird. Metainformationen können entweder in den Daten selbst[5] oder extern gespeichert sein.

Properties bestehen aus Schlüssel-Wert-Paaren. Des Weiteren wird zwischen „live“ und „dead“ Properties unterschieden. Syntax und Semantik werden bei Live-Properties vom Server vorgeschrieben. Außerdem wird der Wert eines Live-Property vom Server geschützt und verwaltet. Der Client kann den Wert ändern, dieser wird jedoch beim Speichern vom Server überprüft. Ein Dead-Property wird vom Client erstellt und vom Server lediglich gespeichert.

Solche Schlüssel-Wert-Paare gibt es auch schon im HTTP-Header (vgl. 2.1.3 auf Seite 12), jedoch sind diese sehr limitiert und unflexibel. Zudem benötigt WebDAV weitaus mehr und komplexere Properties als für HTTP-Header vorgesehen. Außerdem müssen mit diesen Properties verschiedene Aktionen wie das Ändern, Löschen oder Anfragen möglich sein.

Properties werden durch eine wohlgeformte XML dargestellt und beschrieben. Ein Property-Schlüssel oder -Name muss immer eindeutig und einem Schema zugeordnet sein. Des Weiteren wird der XML-Namensraum für die Benennung von Property-Namen genutzt, um Konflikte durch Doppeldeutigkeiten vorzubeugen.

Beispiel:

```
<D:prop xml:lang="en" xmlns:D="DAV:">
  <x:author xmlns:x="http://www.example.com/ns">
    <x:name>Christof Pieloth</x:name>
    <x:departement>IMN</x:department>
    <x:email>a@b.c</x:email>
  </x:author>
  <x:title ⤸
     xmlns:x="http://www.example.com/ns">Implementierung ⤸
     ...</x:title>
</D:prop>
```

Die Angabe der Sprache `xml:lang="en"` ist optional. Die Definition des Namensraum `xmlns:D="DAV:"` kann durch ein Vaterelement vererbt und muss dann nicht mit angegeben werden. Innerhalb des Elements `<D:prop></D:prop>` können die Properties angegeben werden. In diesem Beispiel sind der Autor und der Titel des Dokuments angeben. Die Struktur und die Eindeutigkeit der beiden Properties sind durch die Angabe eines Namensraums überprüfbar und sichergestellt.

[5] Zum Beispiel der ID3-Tag bei MP3-Dateien.

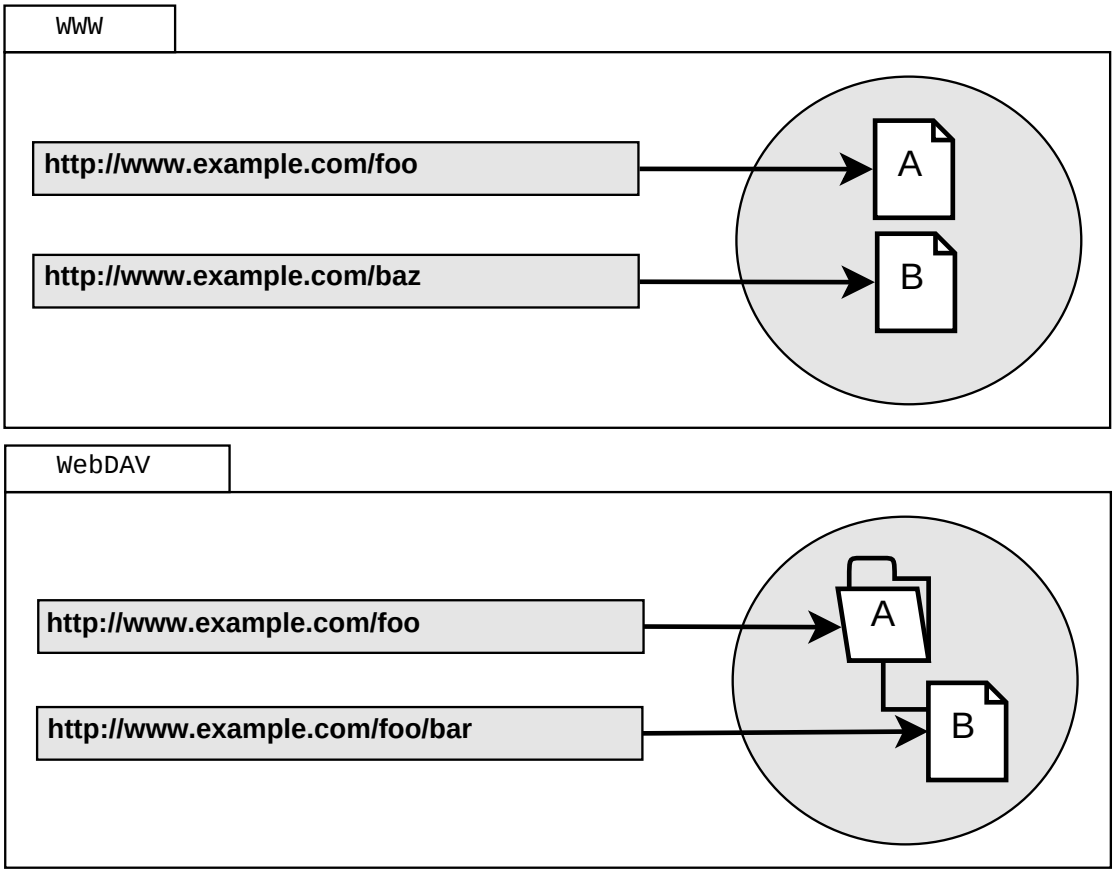

Abbildung 2.1: Abbildung des URL-Namensraums auf WebDAV-Ressourcen.

2.2.2 Collections

Collections sind mit Ordnern in einem Dateisystem vergleichbar. Mit ihrer Hilfe können Dokumente zusammengefasst und Hierarchien aufgebaut werden.

Collections werden auf dem HTTP-URL-Namensraum abgebildet. Dieser baut eine Hierarchie mit dem Trenner „/“ auf. Solch eine Hierarchie ist konsistent, wenn für jede URL in dieser Hierarchie eine Collection existiert, die diese URL als eine interne URL enthält. Von dieser Regel ausgeschlossen ist die Wurzel bzw. die höchste Ebene.

Als Beispiel sind zwei WebDAV-Ressourcen A und B durch die URLs „U“ und „V“ abgebildet. Es wird festgelegt, dass „V“ durch „U/Segment“ gebildet werden kann. Folglich muss die Ressource A, die durch „U“ abgebildet ist, eine Collection sein. Zusätzlich enthält A eine Abbildung von „Segment“ zu B. Wenn die URL `http://www.example.com/foo/bar` auf B zeigt und WebDAV-konform ist und A durch `http://www.example.com/foo/` abgebildet und WebDAV-konform ist, dann muss A eine Collection sein und genau eine Abbildung von bar zu B besitzen. In Abbildung 2.1 ist dieses Beispiel dargestellt.

Collections unterscheiden sich von anderen Ressourcen, da sie wie Container wirken können. Dadurch muss unterschieden werden, ob eine Methode nur auf die Collection oder auf die Collection mit allen enthaltenen Ressourcen ausgeführt werden soll. Falls dies nicht aus der Methode herleitbar ist, kann es durch eine Angabe der Tiefe (englisch: depth) vorgeschrieben werden. Eine Tiefe von 0 führt die Methode nur auf der Collection aus, die Tiefe 1 auf der Collection und allen direkt enthaltenen Ressourcen. Bei der Angabe infinite (deutsch: unbegrenzt) wird die Methode auf der Collection und auf allen enthaltenen Ressourcen rekursiv, d.h. bis zur innersten Ressource, ausgeführt.

2.2.3 Locking

Locking bezeichnet das Sperren von Ressourcen. Dadurch kann der Zugriff auf diese serialisiert werden und für den Client ist sichergestellt, dass während seiner Arbeit an der Ressource diese nicht durch andere geändert werden kann. Dadurch kann das Lost-Update-Problem verhindert werden.

Dies kann jedoch nur gewährleistet werden, wenn alle Clients eine feste Befehlskette einhalten:

1. LOCK (Anfordern einer Sperre)
2. GET (Holen des Inhalts)
3. PUT (Speichern des Inhalts)
4. UNLOCK (Aufheben der Sperre)

Die Idee des Lockings ist nutzlos, wenn ein Client diese Befehlskette nicht einhält. Das folgende Beispiel verdeutlicht einen solchen Problemfall:

Client A holt ein Dokument (GET) ohne dies zu sperren (LOCK).

Client B sperrt das selbe Dokument (LOCK) und fordert es anschließend an (GET).

Client B speichert nach Fertigstellung seiner Änderungen das Dokument (PUT) und entsperrt es (UNLOCK).

Client A speichert das Dokument anschließend (PUT) und überschreibt somit die Änderungen von Client B.

Obwohl Client B die erforderliche Befehlskette eingehalten hat, sind seine Änderungen verloren. WebDAV kann solche Situationen nicht verhindern, sondern nur Methoden zur Vermeidung solcher Probleme anbieten.

Eine Ressource kann direkt und indirekt gesperrt werden. Wird eine Ressource über ihre URL gesperrt, d.h. die LOCK-Anfrage bezieht sich auf diese eine URL, wird dies als direktes Sperren bezeichnet. Als indirektes Sperren wird der Vorgang bezeichnet, wenn eine LOCK-Anfrage mit einer Tiefe größer 0 auf einer Collection, bzw. der URL für diese, ausgeführt wird. Dabei ist die Collection direkt gesperrt und alle enthaltenen Ressourcen sind indirekt gesperrt. Jede Sperre wird durch einen Universally Unique Identifier (kurz: UUID) identifiziert, der als Lock Token bezeichnet wird. Ein UNLOCK löscht die Sperre mit dem dazu gehörigen Lock Token.

Des Weiteren wird zwischen „Exclusive“ (deutsch: exklusiv oder alleinigen) und „Shared“ (deutsch: mehrfach genutzten) Locks unterschieden. Bei einem Exclusive Lock hat nur ein Client Zugriff auf die Ressource. Somit müssen keine Absprachen oder besondere Maßnahmen ergriffen werden, um Konflikte beim Speichern zu verhindern. Oftmals sind Exclusive Locks jedoch zu strikt oder behindern den Arbeitsablauf, da die Freigabe vergessen wurde. Durch einen Shared Lock kann signalisiert werden, dass gerade jemand an dieser Ressource arbeitet ohne diese komplett zu sperren. Der Shared Lock lässt den Zugriff mehrerer Clients auf eine Ressource zu. Hierbei müssen sich die verschiedenen Parteien gegenseitig vertrauen, dass keine andere Partei ihre eigenen Änderung überschreibt.

Die Gültigkeitsdauer eines Lock kann unterschiedlich lang sein und wird als Timeout bezeichnet. Dieser kann zeitlich begrenzt, zum Beispiel 180 Sekunden, oder unendlich

lang sein. Der Timeout kann erneuert werden, indem eine neue LOCK-Anfrage auf die selbe Ressource mit dem selben Lock Token ausgeführt wird. Hierbei wird die verstrichene Zeit der Sperre wieder zurückgesetzt. Ist der Timeout einer Sperre abgelaufen, wird diese gelöscht.

2.2.4 Methoden

PROPFIND

PROPFIND ist die Abkürzung für „Property Find“ (deutsch: finde Eigenschaft). Mit Hilfe dieser Methode können die Properties für eine Ressource ermittelt werden, die durch den Request-URI angegeben ist. Alle WebDAV-konformen Ressourcen, also alle Ordner und Dateien, die abgebildet werden sollen, müssen diese Methode unterstützen.

Da bei dieser Methode eine Tiefe angegeben werden muss, muss der Server mindestens eine Tiefe von 0 und 1 unterstützen. In der Praxis kann eine unbegrenzte Tiefe die Geschwindigkeit und Sicherheit beeinflussen, weshalb die Unterstützung einer unbegrenzten Tiefe abgeschaltet werden darf.

Es gibt drei Arten einer PROPFIND-Anfrage:

1. Einzelabfrage: Es werden bestimmte Properties und deren Werte angefordert.
2. Alle Properties: Es werden alle bekannten Properties und deren Werte angefordert.
3. Namen der Properties: Es werden nur die Namen aller möglichen Properties angefordert.

In einer Einzelabfrage kann eine Liste von Properties angegeben werden, für diese die Werte angefordert werden sollen.

Beispiel:

```
Request:
PROPFIND /file HTTP/1.1
Host: www.example.com
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
  <D:prop xmlns:x="http://www.example.com/ns/">
    <x:author/>
    <x:title/>
  </D:prop>
</D:propfind>

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response xmlns:x="http://www.example.com/ns/">
    <D:href>http://www.example.com/file</D:href>
    <D:propstat>
```

```
      <D:prop>
        <x:author>
          <x:name>Christof Pieloth</x:name>
          <x:departement>IMN</x:department>
          <x:email>a@b.c</x:email>
        </x:author>
        <x:title>Implementierung ...</x:title>
      <D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

In dieser Anfrage werden die Properties `<x:author/>` und `<x:title/>` abgefragt. Die Antwort sendet erfolgreich den Inhalt beider Properties zurück. Bei einer Anfrage für alle Properties wird lediglich das XML-Element `<D:prop></D:prop>` durch `<D:allprop/>` und bei der Anfrage für die Property-Namen durch `<D:propname/>` ersetzt. Die Antworten haben den selben Aufbau, allerdings werden jedoch bei `<D:propname/>` nur leere Elemente als Properties zurückgeliefert.

PROPPATCH

PROPPATCH steht für „Property Patch“ (deutsch: Eigenschaft ausbessern). Mit dieser Methode können Properties einer Ressource, die durch den Request-URI definiert ist, gesetzt und gelöscht werden. Jede WebDAV-konforme Ressource muss diese Funktion unterstützen. PROPPATCH ist eine atomare Methode, d.h. es müssen entweder alle Befehle oder gar keiner ausgeführt werden. Falls irgendein Fehler bei der Ausführung auftritt, müssen alle zuvor ausgeführten Befehle rückgängig gemacht werden. Jede Anfrage muss das XML-Element `<D:propertyupdate xmlns:D=DAV></D:propertyupdate>` enthalten.

Beispiel:

```
Request:
PROPFIND /file HTTP/1.1
Host: www.example.com
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:propertyupdate xmlns:D="DAV:" ➲
   xmlns:x="http://www.example.com/ns/">
  <D:set>
    <D:prop >
      <x:email>c@b.a</x:email>
    </D:prop>
  </D:set>
</D:propertyupdate>

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:" ↩
   xmlns:x="http://www.example.com/ns/">
  <D:response>
    <D:href>http://www.example.com/file</D:href>
    <D:propstat>
      <D:prop>
        <x:email/>
      <D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Das Property `<x:email/>` der Datei `file` wird erfolgreich geändert. Für das Löschen wird das XML-Element `<D:remove></D:remove>` genutzt und die Properties werden als leeres XML-Element angegeben.

MKCOL

MKCOL ist die Abkürzung für „Make Collection“ (deutsch: erstelle Kollektion). Mit dieser Methode können Collections erstellt werden, wenn einige Bedingungen erfüllt sind. So darf die durch den Request-URI angegebene Collection nicht bereits existieren. Des Weiteren müssen die jeweiligen höheren Collections bereits existieren. MKCOL schlägt zum Beispiel fehl, wenn `a/b/c/d` erstellt werden soll, aber die Collection `a/b/c` nicht existiert.

Beispiel:

```
Request:
MKCOL /documents/ HTTP/1.1
Host: www.example.com

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:" ↩
   xmlns:x="http://www.example.com/ns/">
  <D:response>
    <D:href>http://www.example.com/file</D:href>
    <D:propstat>
      <D:prop>
        <x:email/>
      <D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Die Collection `documents` wird erfolgreich erstellt.

GET, HEAD

GET und HEAD haben nahezu die gleiche Funktionalität wie im HTTP. Wird eine dieser Methoden auf einer Collection ausgeführt, kann diese entweder auf einer `index.html` ausgeführt oder der Inhalt der Collection in einer menschenlesbaren Form zurückgegeben werden.[6]

POST

Die POST-Methode wird durch WebDAV nicht genauer definiert, da sie oftmals nur durch den Server genauer bestimmt wird. Außerdem hängt die Funktionalität oft von der entsprechenden Ressource ab,[7] wodurch das Verhalten der Methode bei Collections nicht sinnvoll definiert werden kann.

DELETE

Die Funktionalität der DELETE-Methode entspricht nahezu der im HTTP, wird jedoch durch die zwei wichtigsten Bedingungen erweitert:

1. Alle Locks für diese Ressource müssen gelöscht werden.
2. Jede Abbildung vom Request-URI zu jeder Ressource muss gelöscht werden.

Punkt 2 bezieht sich hauptsächlich auf Collections, da eine DELETE-Anfrage auf Collection so ausgeführt werden muss, als wäre eine unbegrenzte Tiefe angegeben. D.h. es müssen auch alle Ressourcen in einer Collection rekursiv gelöscht werden. Nach Ausführung der Methode muss ein konsistenter URL-Namensraum vorhanden sein.

Beispiel:

```
Request:
DELETE /records/ HTTP/1.1
Host: www.example.com

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:" ↩
   xmlns:x="http://www.example.com/ns/">
  <D:response>
    <D:href>http://www.example.com/records</D:href>
    <D:status>HTTP/1.1 404 Not Found</D:status>
  </D:response>
</D:multistatus>
```

Diese Anfrage soll die Collection `records` löschen, da sie nicht existiert, schlägt dies fehl.

[6] Zum Beispiel eine generierte HTML-Datei, die den Inhalt mit Hyperlinks auflistet und somit ein weiteres Durchsuchen in einem Browser ermöglicht.

[7] Zum Beispiel verschiedene Skripte, die ausgeführt werden.

COPY

Die COPY-Methode kopiert die Ressource, die im Request-URI angegeben ist. Das Ziel wird im Header durch `Destination:<Ziel>` angegeben. Hierbei kann eine Tiefe angegeben werden, wenn keine angegeben ist, wird eine unbegrenzte Tiefe verwendet. Dann werden bei einer Collection alle enthaltenen Ressourcen rekursiv kopiert. Das Erstellen von Ressourcen verhält sich wie bei MKCOL oder PUT, falls diese noch nicht existieren. Mit der Angabe des Header `Overwrite:F` kann das Überschreiben vorhandener Ressourcen verhindert werden. Nach erfolgreichen Kopieren müssen alle Live-Properties vorhanden sein und ein konsistenter URL-Namensraum vorherrschen.

Beispiel:

```
Request:
COPY /documents/ HTTP/1.1
Host: www.example.com
Destination: http://www.example.com/temp/
Overwrite: F
Depth: infinity

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:d="DAV:">
  <D:response>
    <D:href>http://www.example.com/temp/article.doc ↩
       </D:href>
    <D:status>HTTP/1.1 412 Precondition Failed</D:status>
  </D:response>
</D:multistatus>
```

Dieses Beispiel kopiert die Collection `documents` nach `temp`. Die Datei `article.doc` konnte nicht kopiert werden, da sie am Zielort bereits existiert und das Überschreiben ausgeschlossen wurde.

MOVE

MOVE verhält sich wie eine COPY-Anfrage, jedoch werden nach dem Kopieren die Quellressourcen gelöscht.

LOCK

Diese Methode sperrt eine durch den Request-URI angegebene Ressource (vgl. 2.2.3 auf Seite 18). Eine existierende Ressource wird gesperrt, wenn sie nicht bereits mit einem konkurrierendem Lock belegt ist. Die Antwort einer erfolgreich erstellten Sperre muss einen Body mit dem Property `DAV:lockdiscovery` und dem Lock Token im Header enthalten.

Es kann zusätzlich eine Tiefe angegeben werden. Ein Lock-Anfrage mit einer unbegrenzten Tiefe muss atomar ausgeführt werden und darf nur einen Lock Token zurückgeben.

Wird eine Anfrage auf eine nicht existierende Ressource gestellt, so muss diese erstellt und erfolgreich gesperrt werden. Die Bedingungen für ein erfolgreiches Erstellen sind identisch mit denen der PUT- und MKCOL-Methode.

Eine Sperre kann erneuert werden (auch Refresh genannt), wenn ein Lock Request an eine Ressource gesendet wird und dieser im If Header den Lock Token für diese Sperre enthält. Bei dieser Anfrage muss kein Body enthalten sein. Ein erfolgreicher Refresh enthält im Body das Property `DAV:lockdiscovery`, aber nicht den Lock Token im Header.

Beispiel:

```
Request:
LOCK /documents/ HTTP/1.1
Host: www.example.com
Timeout: Second-180
Content-Type: application/xml; charset="utf-8"
Authorization: Digest username="christof", realm="a@b.c
   ", nonce="...", uri="/documents/", response="...",
   opaque="..."

<?xml version="1.0" encoding="utf-8" ?>
<D:lockinfo xmlns:D='DAV:'>
  <D:lockscope><D:exclusive/></D:lockscope>
  <D:locktype><D:write/></D:locktype>
  <D:owner>
    <D:href>http://www.example.com/~christof/contact.
       html</D:href>
  </D:owner>
</D:lockinfo>

Response:
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>http://www.example.com/documents/article.doc
       </D:href>
    <D:status>HTTP/1.1 403 Forbidden</D:status>
  </D:response>
</D:multistatus>
```

Die Collection `documents` soll gesperrt werden. Der Vorgang ist nicht erfolgreich, da `article.doc` nicht gesperrt werden konnte.

UNLOCK

Diese Methode entsperrt eine Ressource wieder. Ist die durch den Request-URI angegebene Ressource eine Collection, so müssen auch alle Ressourcen rekursiv entsperrt werden. Nur wenn alle Ressourcen erfolgreich entsperrt werden können, ist diese Anfrage erfolgreich.

Beispiel:

```
Request:
UNLOCK /documents/ HTTP/1.1
Host: www.example.com
Lock-Token: ⤸
   <urn:uuid:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7>
Authorization: Digest username="christof" ⤸
   realm="a@b.c", nonce="...", uri="/documents/", ⤸
   response="...", opaque="..."

Response:
HTTP/1.1 204 No Content
```

Die Collection `documents` wird erfolgreich entsperrt.

2.3 Datenorganisation der forcont factory FX

Alle Daten in der forcont factory FX sind in „Datasources“ gespeichert. Diese können mit Tabellen von relationalen Datenbanken verglichen werden und bieten einen ähnlichen Funktionsumfang. Dateien oder Dokumente können mit Datensätzen verlinkt und Spalten die Integritätsbedingungen PRIMARY KEY, UNIQUE und NOT NULL zugewiesen werden. Um Spalten mit gleichen Namen von verschiedenen Datasources zu unterscheiden, kann jeder Spalte ein „Alias“, eine Art Alternativname, zugewiesen werden.

Die Präsentation dieser Daten im Frontend wird durch „Views“ konfiguriert. In diesen können die Daten durch Selektionen aufbereitet, d.h. gefiltert oder verknüpft, werden. Die Daten können zum Beispiel durch Baumstrukturen und Listenansichten dargestellt werden. Intern werden die Daten in einer Baumstruktur verwaltet. Diese besteht aus Knoten („Nodes“) und Blättern („Leafs“).

In der factory hat jedes dieser Elemente eine eindeutige ID. Dadurch können mehrere Knoten im Frontend durch den selben Text bzw. Namen angezeigt und trotzdem der Zugriff auf einzelne Knoten unterschieden werden. Des Weiteren muss ein Knoten nicht real in der Datasource existieren. Dies ist zum Beispiel dann der Fall, wenn die Daten nach verschiedenen Kriterien sortiert oder gefiltert werden. Diese Kriterien werden im Frontend durch neue Knoten dargestellt und existieren gewissermaßen nur virtuell, da sie infolge einer Mengenoperation entstehen und ihnen selbst kein Datensatz eindeutig zugeordnet werden kann.

Diese Eigenschaften stehen im Widerspruch zu der Funktionsweise von WebDAV. Der URI einer Ressource wird für den Zugriff und für die Anzeige genutzt. Dadurch muss ein URI immer eindeutig auf eine Ressource abgebildet sein. Dies ist vergleichbar mit einem Dateisystem, da in diesem ein Ordner- oder Dateiname nur einmal im gleichen Pfad genutzt werden darf. In der factory hingegen wird die ID eines Knotens für den Zugriff und der Name für die Darstellung genutzt, wodurch Knoten mit dem selben Namen mehrmals in einer Ebene vorhanden sein dürfen. Des Weiteren können die virtuellen Knoten einige Bedingungen in verschiedenen WebDAV-Methoden verletzen.

Dieses Problem soll anhand der DELETE-Methode verdeutlicht werden. Der WebDAV-Standard legt fest, dass nach dem Löschen einer Ressource B, die durch `http://www.example.com/foo/bar` abgebildet und Teil der Collection A (`http://www.example.`

`com/foo`) ist, die Collection A weiterhin existieren muss. Diese Bedingung wird bei Mengenoperation aber nicht eingehalten. So soll nun in einem Beispiel eine Collection auf einer Kundendatenbank abgebildet werden, die nach dem Wohnort gruppiert wird:

- Kundendatenbank\
  - Berlin\
    - Fleischerei Franz
    - Molkerei Müller
  - Leipzig\
    - Flugschule Pieloth

Nach dem Löschen der Ressource „Flugschule Pieloth“ sollte die Collection „Leipzig“ noch verfügbar sein. Da aber der einzige Eintrag für diese Gruppe gelöscht wurde, ist diese Collection nicht mehr verfügbar.

Das Einhalten aller Bedingungen im WebDAV-Standard ist durch die bereitgestellten Mengenoperationen der factory folglich nicht möglich.

3 Implementierung

Kapitel 3 beschreibt die Umsetzung der in der Zielsetzung beschriebenen Aufgaben. Während der Recherche und Arbeit an der Teilaufgabe „Dokumente online editieren“ zeigte sich, dass das ein Attributieren in der gewünschten Weise nicht möglich ist. Infolgedessen wurden zwei neue Teilaufgaben unter Rücksichtnahme der ursprünglichen Ziele definiert: „Zugriff auf Dokumente über einen Web Folder“ und „Tool zur Demonstration einer Attributierung“.

3.1 Technologie und Entwicklungsumgebung

Die forcont factory FX ist eine Webanwendung und lässt sich somit in Client und Server unterteilen. Auf dem Server muss die WebDAV-Schnittstelle implementiert werden, da dieser die Geschäftslogik ausführt und die Daten bereitstellt.

Bei der Entwicklung der forcont factory FX entschied sich das Softwareunternehmen für Java als Programmiersprache. Mit ihr liegt eine vollwertige und mächtige Sprache vor, die mit Hilfe verschiedener Technologien auch auf Webservern genutzt werden kann. Um die Ausführung von Java-Code auf einem Server zu ermöglichen, werden auf diesem eine Java Runtime Environment (kurz: JRE), teilweise auch das Java Development Kit (kurz: JDK) sowie ein spezieller Webserver benötigt. Die factory verwendet den Open-Source Webserver „Apache Tomcat“ der Apache Software Foundation. Damit Kompatibilitätsprobleme zwischen verschiedenen Java- und Tomcat-Versionen gering gehalten und die verwendeten JavaServer Pages (kurz: JSP) ausgeführt werden können, wird die factory mit einem JDK und einem Tomcat ausgeliefert. Zur Entwicklung werden folgende Versionen verwendet:

- forcont factory FX 1.0 Build 0302
- Apache Tomcat 5.5
- Java Development Kit 5.0

Die Implementierung für die factory sollte leicht einzubinden sein und keine weitere Software voraussetzen. Des Weiteren sollte die bestehende Programmierschnittstelle genutzt werden, um die Wartbarkeit des Codes zu gewährleisten. Im Bereich der Webentwicklung dominieren überwiegend Skriptsprachen bzw. interpretierende Programmiersprachen.[1] Oft fehlende Typunterscheidung und implizit deklarierte Variablen erhöhen jedoch das Fehlerrisiko zur Laufzeit bei Skriptsprachen. Außerdem müsste beim Einsatz einer solchen Sprache zusätzliche Software installiert werden und die Anbindung an die vorhandene Programmierschnittstelle ist schwierig. Für größere Projekte bieten sich deshalb kompilierbare Sprachen an. Diese sind oft performanter und reduzieren Laufzeitfehler durch den Kompiliervorgang.

Da die factory in Java geschrieben ist, können vorhandene Schnittstellen und Klassen bei Verwendung dieser Programmiersprache einfach genutzt werden. Außerdem kann

[1] Zum Beispiel PHP, Perl oder Python.

der mitgelieferte Apache Tomcat und die JRE für die Ausführung des Codes verwendet werden. Nicht zuletzt da Tomcat bereits ein WebDAV-Servlet beinhaltet, fiel die Wahl auf Java. Als Grundlage für die Umsetzung sollte der frei verfügbare Code des Tomcat-Servlets dienen, weil eine komplett eigenständige Implementierung des Standards den Zeitraum und den Rahmen dieser Arbeit überschreiten würde. Stattdessen soll ein verfügbarer Code erweitert und für die Anforderung der factory modifiziert werden. Jedoch konnte das Tomcat-Servlet nicht verwendet werden, da es zu viele Paketabhängigkeiten besitzt, die sich bei der Entwicklung und einer zukünftigen Wartung negativ auswirken. Bei der Suche nach vorhandenen und freien Implementierungen fiel das „webdav servlet"-Projekt auf SourceForge (`http://sourceforge.net/projects/webdav-servlet/`) auf, da es sich am einfachsten einbinden ließ und durch die Verwendung von Schnittstellen leicht erweitert werden kann. Dadurch ist eine Basisfunktionalität gewährleistet und die Entwicklung kann auf die speziellen Anforderungen und Modifikationen für die factory konzentriert werden. Die gewählte Implementierung realisiert den Zugriff auf spezielle Datenstrukturen durch die Bereitstellung von Schnittstellen. Für den Zugriff auf eine Datenstruktur werden diese Schnittstellen speziell für diese umgesetzt. Da die verwendete Implementierung jedoch für den Zugriff auf ein Dateisystem optimiert ist, müssen auch außerhalb der Schnittstellen einige Anpassungen vorgenommen werden.

Als Entwicklungssystem und Clientsoftware kommen neben der zuvor aufgelisteten Konfiguration folgende zum Einsatz:

- Windows XP Service Pack 3
- Microsoft Office 2000 (9.0.6926 SP3)
- Adobe Reader 9 Version 9.1.2
- Eclipse Galileo

Microsoft Office 2000 und Adobe Reader 9 werden als weit verbreitete Bürosoftware zum Testen auf dem Client genutzt. Als IDE für Java wird das freie Eclipse Galileo verwendet, da dieses viele Refactoring-Schritte unterstützt und eine Vielzahl an Erweiterungen existieren.

3.2 Grundaufbau des WebDAV-Servlets

Die Implementierung kann in fünf Funktionsgruppen aufgeteilt werden:

1. Das `WebDavServlet` wird vom Apache Tomcat geladen und initialisiert beim Starten alle Funktionsgruppen mit deren Klassen und Objekten.
2. Die Klasse `WebDavServletBean` lädt alle Implementierungen der speziellen WebDAV-Methoden und leitet die Anfragen an die dafür vorgesehene Klasse weiter.
3. Für jede WebDAV-Methode gibt es eine Klasse, die von der abstrakten Klasse `AbstractMethod` abgeleitet ist.
4. Eine spezielle Implementierung eines WebDAV-Servers wird „Store" genannt. Jeder Store muss das Interface `IWebDavStore` implementieren. Dadurch können die Methoden auch mit unterschiedlichen Implementierungen ausgeführt werden.

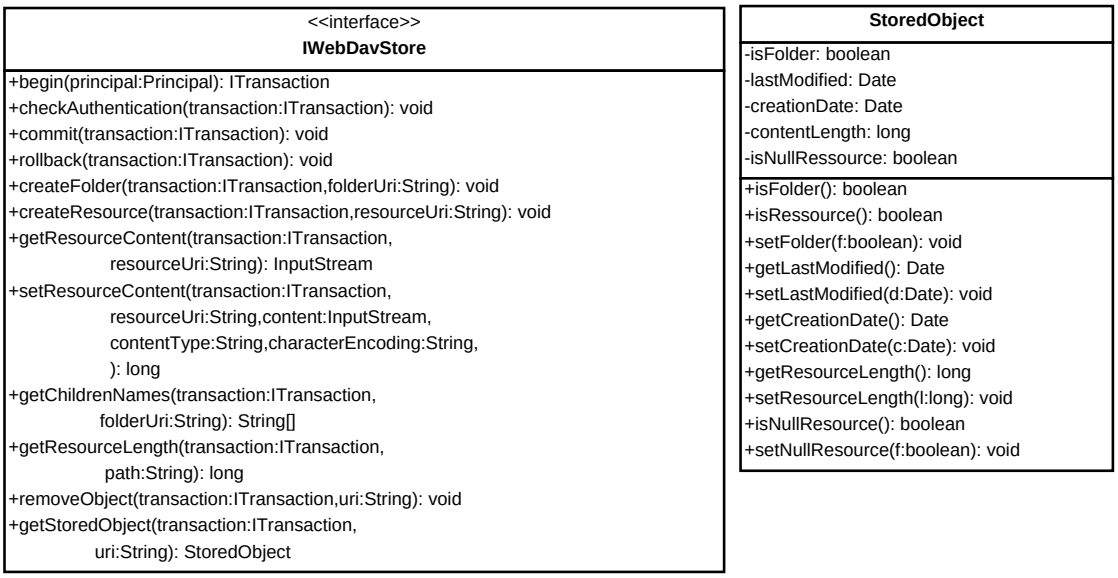

Abbildung 3.1: Klassendiagramm von `IWebDavStore` und `StoredObject`

5. Ressourcen werden durch den Datentyp `StoredObject` dargestellt. Dieses Objekt stellt Funktionen bereit, die vom „Store“ und den WebDAV-Methoden genutzt werden.

In Abbildung 3.1 ist das Interface `IWebDavStore` und die Klasse `StoredObject` als UML-Klassendiagramm dargestellt. Für die factory werden mit diesem Interface spezielle Stores entwickelt, die eine oder mehrere Teilaufgaben umsetzen. Für den Zugriff auf ein lokales Dateisystem existiert bereits der Store `LocalFileSystemStore` als Beispiel. Um diesen zu benutzen, muss nur der Ordner als Parameter übergeben werden, der freigegeben werden soll.

3.3 Dokumente online editieren

Der Vorgang für das Bearbeiten eines Dokuments über den factory-Client soll vereinfacht werden. Dies kann über eine rudimentäre WebDAV-Schnittstelle realisiert werden, die auch vom factory-Client genutzt werden kann.

3.3.1 Analyse

Das Editieren eines Dokuments in der factory ist einem definierten Vorgang unterlegen:

1. Zu Beginn wird durch das Auschecken eine lokale Kopie vom Server geladen.
2. Auf dieser Kopie kann der Benutzer nun mit der entsprechenden Software[2] arbeiten.
3. Durch das Einchecken wird die Datei auf den Server geladen und die Änderungen sind nun öffentlich zugänglich.

Alle drei Schritte werden von der factory unterstützt. Wählt ein Benutzer die Option „Dokument bearbeiten“, wird dieses automatisch heruntergeladen, geöffnet und für andere Benutzer gesperrt. Das Bearbeiten des Dokuments findet nun nicht mehr in der factory

[2]Ein Word-Dokument könnte nun zum Beispiel komfortabel in Microsoft Office bearbeitet werden.

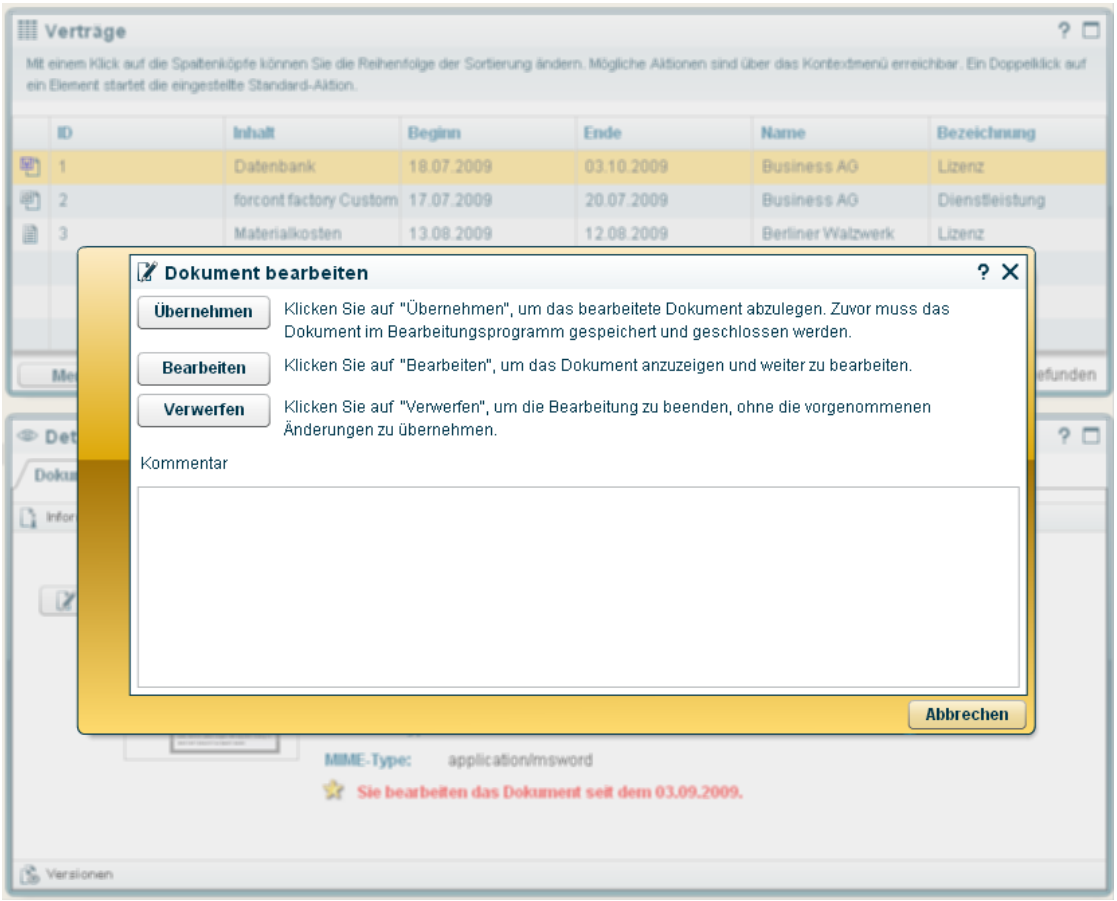

Abbildung 3.2: Ein bearbeitetes Dokument muss zurück in die factory geladen werden.

bzw. im Browser statt, sondern in einem anderen Programm. Nach dem Bearbeiten muss das Dokument jedoch wieder auf den Server geladen und entsperrt werden (siehe Abbildung 3.2). Hierbei wird der Benutzer zwar wieder von der factory unterstützt, aber dieser letzte Schritt wird oft vergessen. Dadurch können Änderungen verloren gehen und durch die Sperre kann das Weiterarbeiten behindert werden.

WebDAV ermöglicht das direkte Arbeiten auf Dokumenten. Um das zuvor beschriebene Probleme zu lösen, müssen folgende Funktionen implementiert werden:

- Öffnen eines Dokuments,
- Speichern eines Dokuments,
- Sperren eines Dokuments,
- Entsperren eines Dokuments.

Für all diese Funktionen gibt es Methoden im WebDAV-Standard. Mit GET und PUT kann ein Dokument geöffnet bzw. gespeichert werden. Zum Sperren und Entsperren stehen LOCK bzw. UNLOCK zur Verfügung. Da die Dokumente nicht in einem Dateisystem gespeichert sind, sondern über die factory und deren Schnittstellen bereitgestellt werden, muss ein neuer Store erstellt werden.

Der Zugriff auf Ressourcen über WebDAV wird über URLs hergestellt. Deswegen muss eine Möglichkeit gefunden werden, wie anhand einer URL ein Dokument in der factory identifiziert werden kann. Daraus entsteht ein Zusammenspiel der einzelnen Komponenten. Auf dem Client wird eine Anwendung gestartet, die mit Hilfe einer übergebenen

URL die zu öffnende Datei lädt. Aufgrund ihrer speziellen URL wird die vom Client gesendete Anfrage vom Webserver als WebDAV-Anfrage erkannt und entsprechend verarbeitet. Dabei wird das angeforderte Dokument aus der factory ermittelt und an den Client gesendet.

3.3.2 Umsetzung

Neuer Store

Für das Online-Editieren wird der Store `FactoryDirectStore` erstellt. Das direkte Öffnen und Speichern benötigt nicht alle vorgeschriebenen Methoden der Schnittstelle `IWebDavStore`. Es genügt, wenn diese Methoden implementiert sind:

- `getResourceContent()` um ein Dokument zu öffnen,
- `setResourceContent()` um ein Dokument zu speichern,
- `getResourceLength()` wird für den Header benötigt und
- `getStoredObject()` wird von den existierenden Klassen genutzt.

Diese Methoden benötigen Daten und Informationen aus der factory, die mit Hilfe einer Adapter-Klasse angefordert werden.

Anbindung an die factory

Die Verbindung zwischen WebDAV und der factory wird durch eine Klasse sichergestellt, die als Adapter fungiert. Die Klasse `WebDav2Factory` stellt alle Methoden bereit, um Daten und Informationen direkt aus der factory zu holen oder diese direkt in die factory zu schreiben. `WebDav2Factory` bündelt die speziellen Programmierschnittstellen zum Zugriff auf die factory in einer Klasse, wodurch diese Methoden auch von anderen Klassen benutzt werden können. Somit ist `WebDav2Factory` der zentrale Zugriffspunkt und zugleich die unterste Abstraktionsebene, die implementiert werden muss.

Für den Store `FactoryDirectStore` werden folgende Methoden benötigt:

- `getDocumentData()` liefert eine Dokument als Byte-Array und wird in der Methode `getResourceContent()` vom Store benutzt.
- `updateDocument()` schreibt ein Dokument zurück in die factory. Diese Methode wird in `setResourceContent()` aufgerufen.
- `lockDocument()` sperrt ein Dokument und wird vor dem Öffnen und nach jedem Speichern eines Dokuments ausgeführt.
- `unLockDocument()` entsperrt ein Dokument.
- `getDocument()` liefert ein Objekt vom Typ `Document` der factory. Dieses Objekt stellt Informationen und Eigenschaften für eine Ressource bereit. Diese Methode wird vom Objekt `StoredObject` genutzt, das im folgenden Abschnitt erklärt wird.

Erweiterung von StoredObject

Jedes Dokument in der factory muss durch den Datentyp `StoredObject` dargestellt werden. Mit Hilfe dieses Objektes können alle Live-Properties einer Ressource abgefragt werden. Ein Dokument in der factory besitzt jedoch mehr Eigenschaften als der Datentyp `StoredObject`. Die Klasse `FactoryDirectObject` erweitert `StoredObject` um weitere Methoden und Eigenschaften. Die Methoden `getResourceLength()`, `isFolder()`, `isRessource()` und `isNullResource()` werden überschrieben, da diese Eigenschaften aus der factory gelesen oder auf eine andere Art ermittelt werden müssen.

Außerdem müssen folgende Eigenschaften von der Klasse `FactoryDirectObject` bereitgestellt werden, damit mit Hilfe von `WebDav2Factory` die gewünschten Daten aus der factory gelesen bzw. in die factory geschrieben werden können:

- Project
- Version
- View
- Datasource
- Primary Keys
- Document Version

Eindeutige URL

Der Zugriff auf eine Ressource kann nur über eine URL erfolgen. Da ein Dokument in der factory mit einem Datensatz verlinkt ist, müssen alle Parameter für den Zugriff auf diesen in der URL übertragen und aus dieser ermittelt werden können. Einer URL können mit dem „Query String“, der durch das Zeichen „?“ von der eigentlichen Adresse getrennt ist, Parameter angehängt werden, die vom Server weiterverarbeitet werden können. Diese Form ist jedoch nicht für den Zugriff auf Ressourcen gedacht und würde tiefgreifende Änderungen in der gewählten Implementierung erfordern, weshalb die Parameter nicht im Query String übertragen werden. Stattdessen wird ein eindeutiges Format für die URL definiert. Dieses enthält die Eigenschaften als Schlüssel-Wert-Paare, die durch Sonderzeichen getrennt sind. Der folgende formale Ausdruck beschreibt das gewählte Format einer URL:

```
URL = "http://" <SERVER> "/ff/" <SERVLET-PATTERN> "/"
      "project=" <PROJECT> "&"
      "version=" <VERSION> "&"
      "view=" <VIEW> "&"
      "datasource=" <DATASOURCE> "&"
      "docversion=" <DOCVERSION> "&"
      "pkfields=" <PKFIELD> *("," <PKFIELD>)
      "/" <FILE>

PKFIELD = <KEY> "+" <VALUE>
```

Alle Nichtterminalsymbole werden sinngemäß durch Werte ersetzt. Anstelle von `<SERVER>` könnte zum Beispiel eine IP-Adresse oder ein Rechnername stehen. Jedes Servlet kann

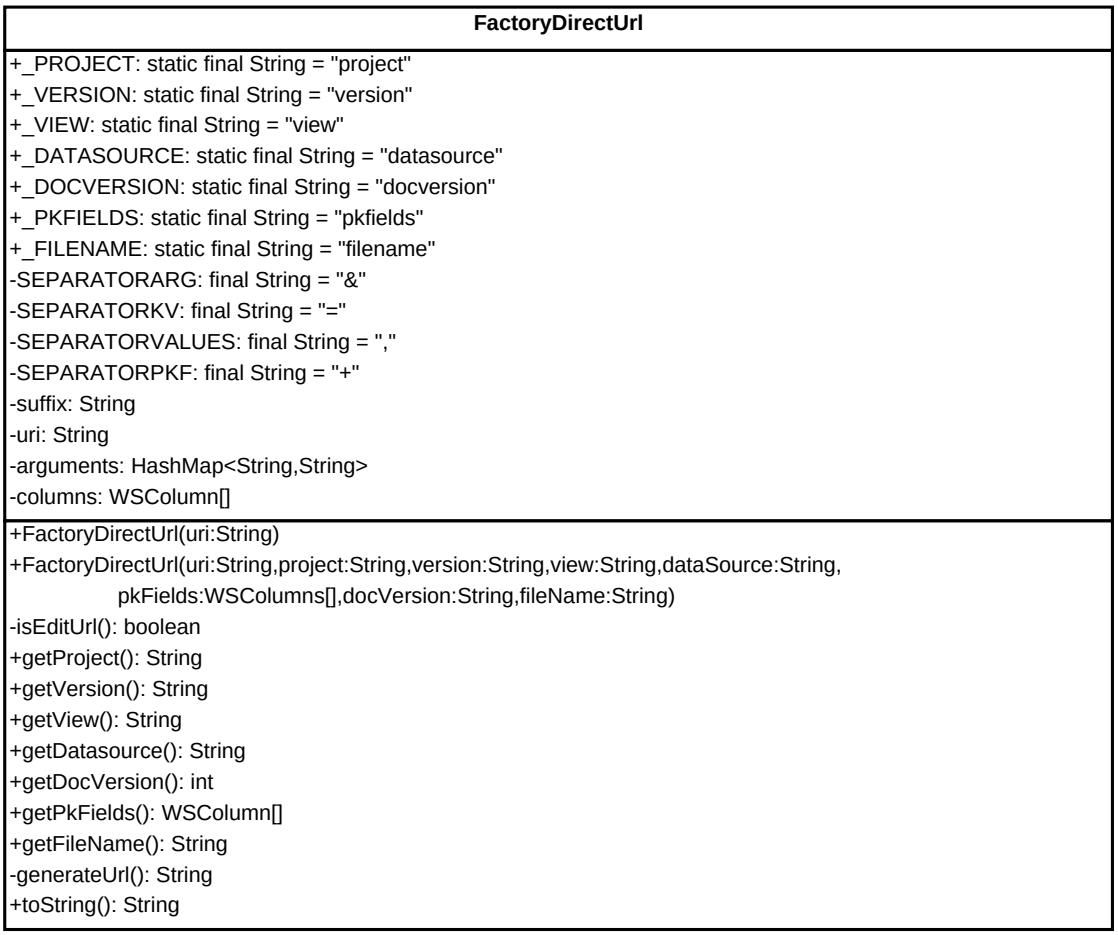

Abbildung 3.3: Klassendiagramm von `FactoryDirectUrl`

über eine definierte Adresse erreicht werden, die im Apache Tomcat angegeben wird. Diese Adresse muss für `<SERVLET-PATTERN>` angegeben werden. Das Nichtterminalsymbol `<FILE>` am Ende einer URL ist nicht zwingend nötig, da ein Dokument auch ohne diese Angabe gefunden wird. Jedoch ist der letzte Teil einer URL in der Regel der Name der angeforderten Ressource. Wird `<FILE>` nicht angegeben, sind in der Titelleiste der Anwendung anstatt des Dateinamens die Schlüssel-Wert-Paare sichtbar. Eine Beispiel URL ist:

```
http://localhost/ff/edit/project=TUTO&version=1.0&view=↩
    Vertraege&datasource=Vertrag&docversion=1&pkfields=ID↩
    +1/test.doc
```

Damit an allen Stellen in der Implementierung bequem mit der URL gearbeitet werden kann, wird die Klasse `FactoryDirectUrl` genutzt. Diese Klasse bietet Methoden, um einerseits eine konforme URL unter Angabe der Parameter erstellen und andererseits aus einer vorhandenen URL die gewünschten Parameter einfach abfragen zu können. Dadurch kann das Format einer URL auch leicht an nur einer Stelle geändert werden. Der Aufbau von `FactoryDirectUrl` ist in Abbildung 3.3 dargestellt.

Einbindung in die Oberfläche

Die zuvor beschriebenen Implementierungen sind für die Bearbeitung von Anfragen an den Server nötig. Um die vom Server bereitgestellten Funktionen auch im factory-Client

nutzen zu können, muss dieser ebenfalls erweitert werden. Im factory Admin Client können nahezu an jedes Element im Frontend Bedienelemente, wie zum Beispiel DropDown-Listen, gebunden werden. Um diesen Bedienelementen Funktionen zu geben, die auf Mausklick ausgeführt werden, gibt es in der factory die so genannten „Actions“. Diese Actions werden an die factory gesendet, welche dann verschiedene Operationen ausführen kann. Die wichtigsten Actions sind in der factory schon implementiert und müssen nur dem Bedienelement zugeordnet werden. Als Beispiel werden folgende Actions genannt:

add document um ein Dokument hinzuzufügen,

search um Einträge zu suchen oder

open url um eine URL zu öffnen.

Des Weiteren gibt es die Action „project extension“. Hiermit können individuelle Erweiterungen in das Frontend eingebunden werden, ohne den bestehenden factory-Code verändern zu müssen. Da die factory für das Öffnen und Starten einer URL erweitert werden muss, wurde diese Action verwendet. Um das Editieren online zu ermöglichen, muss die URL mit der für den Dateityp definierten Anwendung gestartet werden. Hierfür muss die Funktionalität aus zwei Gründen auf den Client verlagert werden: Zuerst muss die richtige Anwendung auf dem Client ermittelt werden, um diese anschließend aus dem Browser heraus zu starten. Der umfangreichste Teil ist das Ermitteln der Anwendung anhand der Dateiendung oder des MIME-Type. Dies ist jedoch nötig, da jeder Benutzer für die Bearbeitung von verschiedenen Dateitypen eigene Anwendungen favorisiert. Zusätzlich können diese Programme auf jeden Computer an einem anderen Speicherort installiert sein. Ist die Anwendung und der Speicherort bekannt, kann diese über die Kommandozeile mit der URL als Parameter gestartet werden. Von nun an verwaltet die Anwendung alle Lese- und Schreibzugriffe unter Nutzung der WebDAV-Methoden alleine.

Das Ermitteln und Starten der Anwendung wird in einem Java Applet implementiert. Applets ermöglichen das Ausführen von Java-Code im Browser auf dem Client. Applets laufen in einer so genannten „Sandbox“ (deutsch: Sandkasten). In Bezug auf Applets bezeichnet Sandbox eine abgeschottete Laufzeitumgebung für das Programm. Das Abgrenzen der Applets vom Browser und Betriebssystem ist für die Sicherheit im Internet unerlässlich, da sonst bösartiger Code großen Schaden anrichten könnte. Dieses Sandkastenprinzip verhindert aber den zum Ermitteln und Starten der Anwendung notwendigen Zugriff auf das Betriebssystem. Durch ein Zertifikat kann ein Applet jedoch mehr Rechte erhalten, wenn der Benutzer diesem zustimmt. Um dies zu ermöglichen, muss das Applet signiert werden, erst dann kann der Benutzer feststellen, wer der Verfasser ist, und ihm vertrauen, dass er kein böswilliges Ziel verfolgt.

Bei der Entwicklung eines Applets sind einige Umstände zu beachten: Da das Applet auf dem Client ausgeführt wird, müssen alle zusätzlichen Pakete mitgeliefert oder nachgeladen werden. Dies kann aber die Größe des Applets erheblich steigern und somit die Datenübertragung verlängern. Deswegen werden so viele Befehle wie möglich auf dem Server ausgeführt. Dem Applet werden drei Parameter für die Ausführung übergeben:

1. die fertige URL,
2. der MIME-Type und
3. die auszuführende Aktion.

Die URL und der MIME-Type ließen sich auch im Applet ermitteln, jedoch werden dafür weitere Klassen benötigt, die wiederum Abhängigkeiten besitzen. Für das Ermitteln der Anwendung wird der Dateityp, der MIME-Type und das Betriebssystem benötigt. Der Dateityp wird aus der URL und dem darin enthaltenen Dateinamen ermittelt. Jedes Betriebssystem benötigt eine eigene Implementierung für das Suchen und Starten der Anwendung. Um das Suchen allgemein und abstrakt zu halten, wurde die Schnittstelle `IOperatingSystemInfo` mit den Methoden `getApplication()` und `startApplication()` definiert. Für Windows wurde die Klasse `WindowsInfo` geschrieben. In dieser liest `getApplication()` in zwei Schritten das Programm für eine Dateiendung mit Hilfe des Konsolenprogramms `reg` aus der Windows-Registry aus und gibt im dritten Schritt das ermittelte Kommando zurück. Anschließend kann das Applet dieses Kommando mit der URL durch das Aufrufen von `startApplication()` auf den Client ausführen. Die übergebene Aktion teilt dem Applet mit, was es ausführen soll (Öffnen oder Download), da das Applet von der factory im Hintergrund und nicht vom Benutzer gesteuert wird.

Da Applet aber nicht direkt über eine Action in die factory eingebunden werden kann, wird hierfür eine „project extension“ durch die Klasse `WebDavAction` eingebunden. Mit Hilfe der neuen Action wird einerseits das Applet mit den nötigen Parametern geladen, andererseits ermöglicht es den bis dahin problematischen Sperrvorgang (weiteres in 3.3.3). `WebDavAction` können die Parameter „action“ und „pattern“ übergeben werden. Durch „action“ kann unterschieden werden, ob das Dokument geöffnet oder nur entsperrt werden soll. Das frei wählbare URL-Pattern, unter dem das Servlet über den Tomcat erreichbar ist, muss durch den Parameter „pattern“ angegeben werden. Außerdem werden die JSPs `showWebDavAction` und `showWebDavApplet` benötigt. `showWebDavAction` generiert XML-Code, der zum Browser gesendet wird. Dieser weist den Browser bzw. den Flex-Client an, ein Dokument im Hintergrund zu laden ohne dieses sichtbar darzustellen. In diesem Fall handelt es sich um die JSP `showWebDavApplet`, die HTML-Code enthält, um das Applet auf den Client zu starten.

Zusammenfassend sind für die Einbindung drei Komponenten nötig:

1. Die „project extension“ `WebDavAction` bindet eine Action in die factory ein und enthält aus dem Applet ausgelagerten Code.

2. Die JSPs `showWebDavAction` und `showWebDavApplet` laden und starten das Applet in der Oberfläche.

3. Das Applet `WebDavApplet` ermittelt und startet die Anwendung auf dem Client.

In Abbildung 3.4 auf der nächsten Seite ist die Einbindung im factory-Client dargestellt. Im Kontextmenü können die Actions „Dokument bearbeiten (WebDAV)“ und „Dokument freigeben (WebDAV)“ ausgewählt werden. Zusätzlich ist für Demonstrationszwecke das Applet in der oberen rechten Ecke eingeblendet. Für den Benutzer ist dieses Applet nicht sichtbar.

3.3.3 Probleme und Anmerkungen

Eine wichtige Funktion in der factory und in WebDAV ist das Sperren und Entsperren eines Dokuments. Da WebDAV hierfür die Methoden LOCK und UNLOCK bereitstellt, sollten ursprünglich diese genutzt werden, um das Dokument in der factory zu sperren oder wieder freizugeben. Im Unterabschnitt 2.2.3 auf Seite 18 wird die Befehlskette

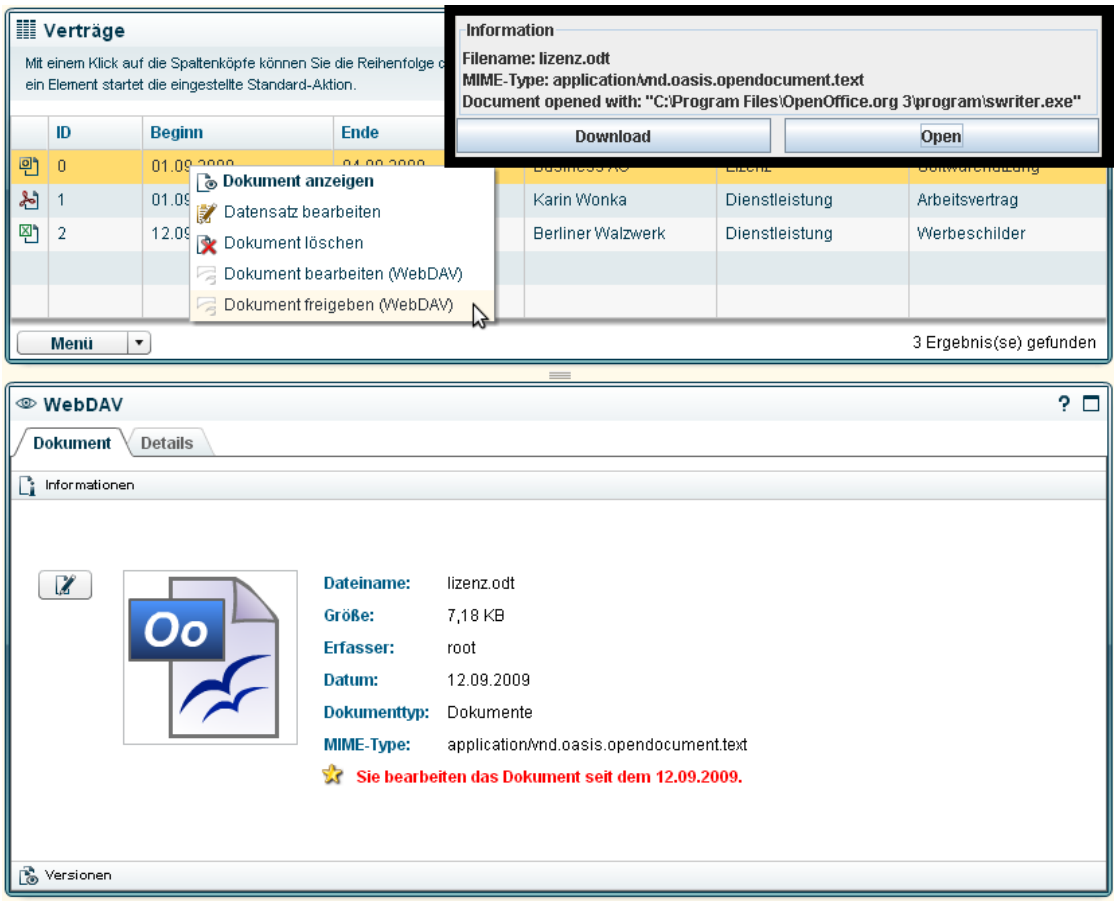

Abbildung 3.4: Dokument bearbeiten oder freigeben mit eingeblendeten Applet.

beschrieben, welche für ein erfolgreiches Zusammenarbeiten mehrerer Benutzer nötig ist. Bei der Nutzung der WebDAV-Methoden mussten jedoch einige Probleme festgestellt werden.

So versucht zum Beispiel Microsoft Office, diese Befehlskette einzuhalten, und sendet vor dem Öffnen eines Dokuments eine LOCK-Anfrage. Diese wird vom Server erfolgreich ausgeführt. Somit wird das Dokument gesperrt und alle nötigen Informationen werden als Antwort zurückgesendet. Microsoft Office scheint diese erfolgreiche Antwort jedoch nicht korrekt zu erkennen und öffnet das Dokument nur mit Lesezugriff („Read-Only“). Da die erfolgreiche LOCK-Anfrage nicht erkannt wurde, wird beim Schließen der Anwendung auch keine UNLOCK-Anfrage gesendet. Somit bleibt das Dokument unnötigerweise gesperrt. Auch nach umfangreicher Analyse der Request und Response, sowie der Installation der Patches KB907306[3] und KB943337[4] konnte dieses Problem nicht gelöst werden. Aufgrund dieser Erkenntnisse wird der Sperrvorgang nicht über die WebDAV-Methoden durchgeführt, sondern über die erstellte Action `WebDavAction` geregelt. Einerseits erfordert diese Lösung wieder das manuelle Freigeben der Dokumente, das eigentlich vereinfacht werden sollte. Andererseits gibt es keine Garantie, dass die empfohlene Befehlskette des WebDAV-Standards von jeder Anwendung eingehalten wird. Durch die Verlagerung des Sperrvorgangs von WebDAV bzw. der Anwendung in die factory kann immer ein korrektes Sperren und Entsperren sichergestellt werden.

Eine fundamentale Komponente für die Funktionalität ist das Applet. Das im Rahmen

[3] `http://support.microsoft.com/kb/907306`
[4] `http://support.microsoft.com/kb/943337`

dieser Arbeit erstellte Applet unterstützt nur die Windows Betriebssysteme von Microsoft. In vielen Unternehmen und Instituten werden jedoch auch zunehmend Linux und Mac OS eingesetzt. Um diese und andere Betriebssysteme zu unterstützen, müssen weitere Klassen entwickelt werden, die für den Dateityp oder den MIME-Type die definierte Anwendung auf den Client suchen und starten. Das Interface `IOperatingSystemInfo` definiert einheitliche Schnittstellen für diesen Zweck. Bei steigender Anzahl der unterstützten Betriebssysteme könnte das Entwurfsmuster „Fabrik“ verwendet werden, um die Erzeugung der richtigen Klasse für jedes Betriebssystem zu vereinfachen.

Beim Testen des Applets gab es bei einigen Anwendungen Startprobleme. So kann zum Beispiel der Adobe Reader keine URL öffnen, wenn dieser über die Kommandozeile gestartet und die URL als Parameter übergeben wird. Wenn die URL jedoch im Öffnen-Dialog eingefügt und geöffnet wird, funktioniert dies ohne Probleme. Falls dieses Problem bei mehreren weit verbreiteten Anwendungen auftritt, muss ein neuer Startmechanismus gesucht werden. Da die factory aber aus dem Browser heraus und möglichst ohne zusätzliche Software auf dem Client bedient werden soll, sind die Möglichkeiten stark begrenzt. Dies könnte die Chancen für den Einzug des Online-Editierens in die forcont factory FX stark mindern, da eine Verbesserung zur bisherigen Lösung nicht in Aussicht steht.

Der Zugriff auf ein Dokument ist in dieser Implementierung für jeden Benutzer möglich. Da die factory aber eine interne Benutzerverwaltung ermöglicht, ist eine mögliche Zugriffskontrolle in Betracht zu ziehen. Für die Umsetzung gibt es grundsätzlich zwei Möglichkeiten: Die einfachste Lösung wäre, die Actions über die interne Zugriffskontrolle der factory zu beschränken. Dadurch können nur autorisierte Benutzer die Action aufrufen und ein Dokument bearbeiten. Diese Zugriffskontrolle greift aber nur, wenn der Benutzer die factory über den Browser, also das eigene Frontend, bedient. Ein Benutzer erhält immer noch Zugriff, wenn er die eindeutige URL auf Dokumente herstellen kann. Diese Hintertür kann durch eine Zwischenschicht geschlossen werden. In der Zwischenschicht wird für jede eindeutige URL eine neue URL gebildet. Dabei werden die Schlüssel-Wert-Paare in der URL zum Beispiel durch einen Universally Unique Identifier ersetzt. Außerdem muss diese Zwischenschicht alle generierten URLs mit den dazu ursprünglichen URLs verwalten, um bei einer Anfrage auch auf das richtige Dokument zugreifen zu können. Bei einer Anfrage muss das WebDAV-Servlet nun zu der neu generierten URL die eindeutige URL ermitteln. Diese Lösung ermöglicht, zumindest theoretisch, immer noch den unautorisierten Zugriff auf ein Dokument, da auch die generierte URL heraus gefunden werden könnte.[5] Jedoch ist eine generierte URL nur für die Dauer der Bearbeitung gültig und die Anzahl der möglichen UUIDs lässt die Chance eines Treffers gegen Null sinken.

Der zweite Lösungsansatz ist die Umsetzung der im [RFC2617] beschriebenen Autorisierung für das HTTP. Mit diesem Ansatz könnte ein „Single Sign-on“ realisiert werden. Dabei führt der Benutzer nur eine Anmeldung am selben Arbeitsplatz aus und hat dann Zugriff auf mehrere Dienste, ohne sich erneut anmelden zu müssen. Bei solch einer Lösung müssten jedoch weitere Aspekte berücksichtigt werden, da die Accounts der Clients und der factory abgeglichen werden müssten.

Ein weiteres Problem tritt im Frontend auf, da die aus den Datensätzen aufgebauten Baumstrukturen im Flex-Client des Frontends zwischengespeichert werden. Bei der Ausführung einer Action oder einer neuen Anfrage an den Server wird angenommen, dass

[5] Zum Beispiel durch eine Brute-Force-Attacke.

Abbildung 3.5: Die Daten im Frontend und auf dem Server unterscheiden sich.

sich die Daten auf dem Server nicht von denen im Zwischenspeicher unterscheiden. Da alle Actions bisher nur über das Frontend ausgeführt wurden, konnten die Daten zwischen Client und Server immer leicht synchronisiert werden. Das Online-Editieren wird aber nur über das Frontend gestartet und arbeitet anschließend autonom über das neue Servlet auf dem Server. Dies ermöglicht das Ändern von Datensätzen und Dokumenten ohne eine Action im Frontend zu nutzen. Dadurch können sich die Daten vom Flex-Client mit denen auf dem Server unterscheiden. Bei einer Anfrage an den Server oder bei Ausführung einer Action wird ein Unterschied zwischen den Daten erkannt und es kommt zu einer Fehlermeldung wie in Abbildung 3.5. Um diese Meldung zu verhindern, muss der Client nach einer Änderung informiert werden, um die Daten zu aktualisieren. Generell ist die Kommunikationsaufnahme zwischen einem Browser und einem Webserver unidirektional, weil immer der Client eine Anfrage an einen Server senden muss, um Daten zu erhalten. Eine vom Server initiierte Kommunikationsaufnahme ist in der Regel nicht möglich. Auch in der forcont factory FX reagiert der Webserver nur auf Anfragen vom Client (Oktober 2009). Somit kann eine Aktualisierung der Daten nur vom Client gestartet werden. Dazu muss der Client ein sogenanntes Polling ausführen. Dabei führt der Client aktive zyklische Anfragen an den Server aus. Der Server prüft bei jeder Anfrage, ob sich die Daten seit dem letzten Zugriff verändert haben und sendet gegebenenfalls die neuen Daten zurück. Hierbei muss die factory zusammen mit der WebDAV-Implementierung eine Datenstruktur verwalten, die die Schreibzugriffe über WebDAV und die Pollinganfragen verarbeitet. Aufgrund der Client-Server-Architektur und den damit verbundenen kontinuierlichen Anfragen wird die Netzwerklast bei dieser Lösung erhöht.

3.4 Zugriff auf Dokumente über einen Web Folder

Windows bietet mit dem Web Folder die Möglichkeit, auf die Ressourcen eines WebDAV-Servers ohne Zusatzsoftware zuzugreifen. Die Ressourcen werden dann in einer Verzeichnisstruktur abgebildet. Dadurch kann mit einem Web Folder auf entfernten Ressourcen wie mit einem lokalen Dateisystem gearbeitet werden. Eine native Einbindung eines factory-Baums durch einen Web Folder würde neue Wege für die Arbeit mit Dokumen-

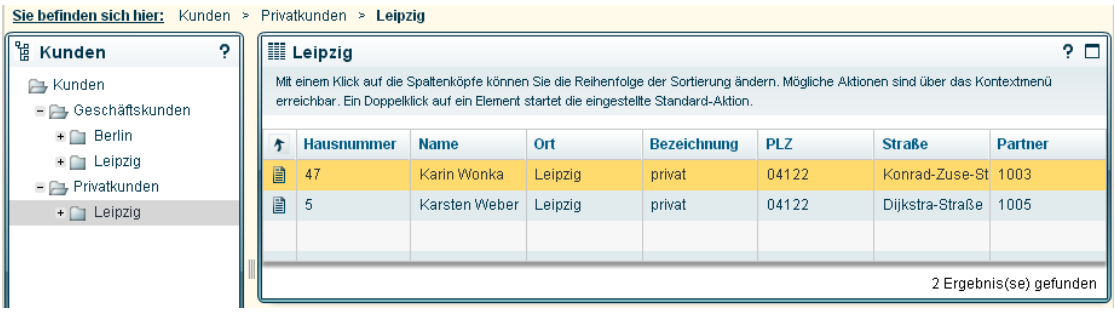

Abbildung 3.6: Diese View ordnet Kunden nach Geschäftsbeziehung und Ort.

ten schaffen. So könnte der Zugriff auf Dokumente über den Windows Explorer oder direkt in einer Anwendung ermöglicht werden.

3.4.1 Analyse

Die Daten in der factory werden in Datasources gespeichert und durch Views für die Präsentation aufbereitet. Um die Kundenanforderungen einfach und effizient umsetzen zu können, werden im factory Admin Client kundenspezifische Anpassungen (englisch: Customizing) der factory vorgenommen. Die in den Datasources gespeicherten Daten werden dabei in unterschiedlichen Baumstrukturen organisiert. Ein entstehender Baum wird allgemein als factory-Baum bezeichnet. Die Regeln und Vorschriften für die Generierung eines factory-Baums sind in einer View gespeichert. Mehrere Views können die selben Datasources nutzen, um die Daten auf unterschiedliche Weise filtern, organisieren und darstellen zu können.

Am Beispiel einer Kundendatenbank sollen eine Datasource, eine View und der daraus gebildete factory-Baum verdeutlicht werden. Die Kundendaten Hausnummer, Name, Ort, Bezeichnung, PLZ, Straße und Partner werden in der Datasource „Kunden“ gespeichert. Wie in einer relationalen Datenbank werden diese Daten in einer Tabelle gespeichert. Um diese Daten aufzubereiten, wird die View „Kunden“ erstellt. In diesem Beispiel sortiert und gruppiert diese die Kundendaten nach Geschäftsbeziehung (Geschäfts- oder Privatkunden) und Wohnort. Der daraus entstehende factory-Baum ist in der Abbildung 3.6 dargestellt.

Solch ein factory-Baum ähnelt stark einem Dateisystem mit Ordnern und Dateien. Für das Beispiel der Kundendatenbank könnte die daraus gebildete Ordnerstruktur die Knoten, also die Geschäftsbeziehung und den Wohnort, als Ordner und die Kunden als Dateien abbilden. Zum Beispiel:

- Kunden\
- Kunden\Geschäftskunden\
- Kunden\Geschäftskunden\Berlin\
- Kunden\Geschäftskunden\Leipzig\
- Kunden\Privatkunden\
- Kunden\Privatkunden\Leipzig\

WebDAV bietet die Möglichkeit, solche Verzeichnisstrukturen abzubilden und die darauf möglichen Operationen auszuführen. Allgemeine Datei- und Ordneroperationen sind:

- Öffnen,
- Erstellen,
- Löschen,
- Kopieren,
- Verschieben.

Durch den Web Folder von Windows können die Benutzer auf eine bekannte Verzeichnisstruktur und somit auch ohne den factory-Client auf Dokumente zugreifen. Dies ermöglicht den Zugriff auf die factory in vielen Anwendungen ohne auf eine zusätzliche Anwendungen zurückgreifen zu müssen.

Die Grundfunktionalität der Web Folder soll das Bearbeiten, Löschen und Ablegen von Dokumenten ermöglichen. Da ein Web Folder das Setzen und Abfragen von Dead-Properties nicht unterstützt, ist das Attributieren eines Dokuments beim Ablegen ohne zusätzliche Software nicht möglich und wird somit getrennt in Abschnitt 3.5 auf Seite 47 betrachtet.

Eine View kann jedoch im Web Folder nicht so dargestellt und erzeugt werden wie in der factory. In einem Dateisystem und somit auch im Web Folder besitzt jede Datei und jeder Ordner in der selben Hierarchieebene einen eindeutigen Namen über den auch der Zugriff auf die Ressource stattfindet. Die Knoten der selben Hierarchieebene in einem factory-Baum können jedoch identische Namen besitzen, da der Zugriff über eine interne ID ausgeführt wird. Als Ordner- und Dateinamen könnte zwar diese ID genutzt werden, jedoch ist diese Lösung nicht sehr anwenderfreundlich, da kaum ein Benutzer sein gesuchtes Dokument zwischen den IDs finden wird. Des Weiteren wird ein Datensatz als Blatt des Baumes dargestellt und das dazugehörige Dokument ist diesem Datensatz angehängt. Diese Eigenschaften der factory müssen auf einen geeignete Art und Weise auf einen Web Folder abgebildet werden.

3.4.2 Umsetzung

Voraussetzung der factory

Der Zugriff auf Ressourcen im Web Folder soll nicht über die IDs der Knoten, sondern über die entstehenden Ordner- und Dateinamen stattfinden. Grundsätzlich können die Namen der Ressourcen mit Hilfe der Attribute generiert werden. Dies hat jedoch zur Folge, dass sehr lange Namen entstehen und diese ebenfalls nicht sehr benutzerfreundlich sind. Stattdessen wird festgelegt, dass jede Datasource, die in einem Web Folder genutzt wird, eine Spalte WEBDAVNAME besitzen muss. Um immer einen eindeutigen Namen für einen Datensatz sicherzustellen, sollten die Integritätsbedingungen UNIQUE und NOT NULL gesetzt sein.

Beim Erstellen von Ordnern und Dateien werden im WebDAV-Standard nur die Namen übertragen, die vom Server für die Spalte WEBDAVNAME eines Datensatzes genutzt werden. Da keine Informationen für weitere Spalten übertragen werden, müssen alle anderen Spalten NULL-Werte erlauben. Aufgrund von UNIQUE und NOT NULL wird eigentlich kein Primärschlüssel benötigt bzw. könnte WEBDAVNAME als Primärschlüssel dienen. Falls dennoch ein Primärschlüssel genutzt wird, muss dieser von der factory automatisch erstellt werden, um das Anlegen neuer Datensätze zu ermöglichen.

Neuer Store

Im Gegensatz zum `FactoryDirectStore`, der nur den direkten Zugriff auf ein Dokument gewährleistet, muss der neue Store eine Baumstruktur verwalten und den Zugriff auf dessen Elemente gewährleisten. Für diese Aufgabe wird der neue Store `FactoryTreeStore` erstellt, der alle Methoden von `IWebDavStore` implementiert. Zusätzlich wird die Schnittstelle `IWebDavStore` um die Methode `renameResource()` erweitert, um Elemente umzubenennen. Beim Initialisieren des Stores werden die benötigten Parameter

- Project,
- Version,
- Instance und
- View

aus der `web.xml` von Tomcat ausgelesen. Aufgrund der eindeutigen Abbildung der URIs auf die Ressourcen können diese Parameter nicht im URI übertragen werden. Somit wird für jede View der factory eine eigene Instanz des Servlets gestartet und durch das URL-Pattern in der `web.xml` kann zwischen den verschiedenen Views unterschiedenen werden. Durch den folgenden Ausschnitt der `web.xml` wird das Servlet für die Kundendatenbank geladen:

```
<servlet>
  <servlet-name>Kunden</servlet-name>
  <servlet-class>ff.webDav.WebDavServlet</servlet-class>
  <init-param>
    <param-name>ResourceHandlerImplementation</param-name>
    <param-value>ff.webDav.FactoryTreeStore</param-value>
  </init-param>
  <init-param>
    <param-name>rootpath</param-name>
    <param-value>D:\\forcont\\factory\\ff\\WebDAV ↩
       </param-value>
  </init-param>
  <init-param>
    <param-name>project</param-name>
    <param-value>TUTO</param-value>
  </init-param>
  <init-param>
    <param-name>version</param-name>
    <param-value>1.00</param-value>
  </init-param>
  <init-param>
    <param-name>instance</param-name>
    <param-value>DOCU</param-value>
  </init-param>
  <init-param>
    <param-name>view</param-name>
    <param-value>Kundenakte</param-value>
  </init-param>
</servlet>
<servlet-mapping>
```

```
  <servlet-name>Kunden</servlet-name>
  <url-pattern>/kunden/*</url-pattern>
</servlet-mapping>
```

Diese Daten werden vom Store gespeichert und für den Zugriff auf die factory bereitgestellt. Die Identifizierung der Datensätze erfolgt über die Ordner- und Dateinamen, die durch den URI ermittelt werden.

Abbildung eines factory-Baums auf eine Verzeichnisstruktur

Eine Baumstruktur wird eigentlich ausgehend vom Wurzelknoten aufgebaut. Da ein factory-Baum jedoch aus Datensätzen einer Tabelle generiert wird, ist dieser Vorgang einfacher zu erklären, wenn bei den Blättern angefangen wird. Die Grundlage und somit die Blätter eines Baumes werden immer durch eine Datasource und deren Datensätze oder Dokumente gebildet. Diese Blätter können durch Kriterien gruppiert und gefiltert werden. Durch diese Kriterien entstehen neue Knoten, die wiederum gruppiert und gefiltert werden können. Somit verringert sich die Anzahl der Elemente in Richtung Wurzel und es entsteht ein Baum. Eine Baumstruktur enthält zwei Elemente. Speziell für die Darstellung eines factory-Baums sind dies:

1. Blätter, die Datensätze und Dokumente darstellen und
2. Knoten, die Selektionen auf ihre Kindelemente darstellen.

Wird ein Dateisystem auf eine Baumstruktur abgebildet, werden Dateien als Blätter und Ordner als Knoten dargestellt. Da ein Datensatz in der Regel mehrere Spalten besitzt, lässt sich dieser nicht sinnvoll in ein Dateisystem integrieren. Jedoch sollen nicht die Datensätze, sondern die angehängten Dokumente bearbeitet werden. Deswegen werden nur die Dokumente als Dateien dargestellt und alle anderen Knoten als Ordner. Da ein Dokument nichts weiter als eine Datei ist, wird der Dateiname zur Darstellung im Web Folder genutzt. Für jeden Datensatz wird der WEBDAVNAME als Ordnername genutzt.

Da diese Struktur von der internen Baumstruktur der factory abweicht und auch neue Funktionen auf den Knoten ausgeführt werden müssen, wird für den Web Folder eine eigene Implementierung verwendet. Eine Baumstruktur ist eine Teil-Ganzes-Hierarchie. Hierbei kann der Baum in einzelne Unterbäume zerlegt werden, die für sich allein betrachtet wieder einen vollständigen Baum repräsentieren. Um solch eine Teil-Ganzes-Hierarchie mit Objekten abzubilden, bietet sich das Strukturmuster Kompositum an. Dieses hat zusätzlich den Vorteil, dass die Unterschiede zwischen verschiedenen Elementen (Selektionen, Datensätzen, Dokumente) verborgen werden. In Abbildung 3.7 auf der nächsten Seite ist dieser Aufbau in einem vereinfachten Klassendiagramm dargestellt. Die Schnittstelle `IFactoryTree` definiert alle Methoden, die auf der Baum- bzw. Verzeichnisstruktur ausgeführt werden sollen und muss von jedem Element implementiert werden.

Die Elemente in dieser Struktur sind `FactoryNode` und `FactoryLeaf`. Alle Datensätze und Selektionen werden durch `FactoryNode` abgebildet und als Ordner im Web Folder dargestellt. Ein `FactoryNode` kann weitere Objekte des Typs `FactoryNode` oder `FactoryLeaf` enthalten. Ein Dokument und somit eine Datei wird durch `FactoryLeaf` abgebildet und ist das Blatt des Baumes. Um diese Struktur aus einem factory-Baum bzw. einer View aufzubauen wird auf die interne Baumstruktur der factory zurückgegriffen und diese erweitert. Hierbei werden die Klassen, in Anlehnung an das Strukturmuster Dekorierer,

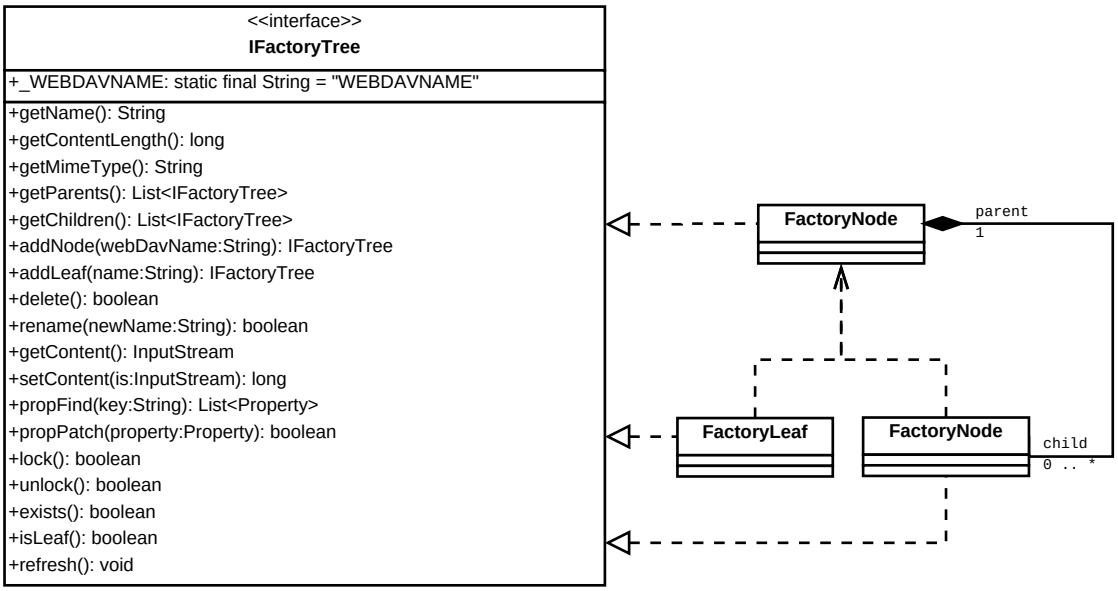

Abbildung 3.7: Vereinfachtes Klassendiagramm der Baumstruktur

um zusätzliche Funktionalitäten erweitert. Die Klassen `FactoryNode` und `FactoryLeaf` dekorieren die Klasse `Node` der factory und greifen somit intern auf die Methoden der zu dekorierenden Klasse zu. Dadurch ist es möglich, die vorhandenen Methoden auf die neuen Anforderungen von WebDAV zu zuschneiden und neue Methoden unter Verwendung vorhandener zu implementieren.

Realisierung der Ordner- und Dateioperationen für die factory

Die Klasse `WebDav2Factory` wird auch in dieser Lösung verwendet, um Daten aus der factory zu holen oder in diese zu schreiben. Aufgrund des höheren Funktionsumfangs wird `WebDav2Factory` um eine Vielzahl an Methoden erweitert.

Öffnen Beim Öffnen einer Datei wird der Inhalt durch eine GET-Anfrage angefordert. `FactoryLeaf` implementiert für diesen Zweck die Methode `getContent()`. Diese liest mit Hilfe von `WebDav2Factory` das Dokument mit `getDocumentData()` aus der factory aus. Beim Öffnen eines Ordners wird der Ordnerinhalt durch eine PROPFIND-Anfrage angefordert. Die Methode `getChildren()` von `FactoryNode` gibt alle Kindelemente und somit die enthaltenen Ordner und Dateien zurück.

Speichern Das Speichern von Dateien wird mit einer PUT-Anfrage durchgeführt. Hierbei wird die Methode `setContent()` von `FactoryLeaf` aufgerufen. Das Dokument wird dann durch `updateDocument()` der Klasse `WebDav2Factory` in die factory geschrieben.

Erstellen Beim Erstellen muss zwischen Ordnern und Dateien unterschieden werden. Ein Ordner wird durch eine MKCOL-Anfrage erstellt. Da ein Ordner durch `FactoryNode` abgebildet wird, muss dem Vaterknoten ein neuer `FactoryNode` angefügt werden. Hierfür existiert die Methode `addNode()`. Ein neuer Ordner kann nur erzeugt werden, wenn dieser Ebene eine Datasource zugrunde liegt. Da Ordner, die durch Selektionen entstehen, nicht real als Datensatz existieren, sondern nur aufgrund der Datensätze selbst entstehen, können diese nicht erstellt werden. Liegt der Ebene eine Datasource zugrunde, wird ein

neuer Datensatz erzeugt. Dazu stellt `WebDav2Factory` die Methode `createRecord()` zur Verfügung.

Eine Datei wird durch eine PUT-Anfrage erstellt, die ebenfalls nur auf einem Ordner, also einem `FactoryNode`, ausgeführt werden kann. Da eine Datei durch ein `FactoryLeaf` abgebildet ist, existiert hierfür die Methode `addLeaf()`. Weil ein Dokument immer einem Datensatz angefügt ist, muss auch hier eine Datasource der Ebene zugrundeliegen. Mit `addDocument()` in `WebDav2Factory` wird ein Dokument in der factory erstellt.

Da einer Datei bzw. einem Blatt keine Elemente angefügt werden können, geben die Methoden `addNode()` und `addLeaf()` in `FactoryLeaf` nur eine Fehlermeldung zurück, um auf ein Fehlverhalten in höheren Abstraktionsebenen hinzuweisen.

Löschen Zum Löschen einer Ressource definiert `IFactoryTree` die Methode `delete()`. Bei `FactoryNode` wird der zugrundeliegende Datensatz durch `deleteRecord()` in `WebDav2Factory` gelöscht. Auch hier muss die Ebene auf eine Datasource abgebildet sein. In `FactoryLeaf` wird die Methode `deleteDocument()` von `WebDav2Factory` aufgerufen.

Kopieren Das Kopieren läuft rekursiv in zwei Schritten ab. Zuerst wird die Ressource erstellt und anschließend ihr Inhalt kopiert. Das rekursive Abarbeiten der Schritte wird durch eine höhere Abstraktionsebene des Servlets gesteuert und muss somit nicht berücksichtigt werden. Somit muss noch das Erstellen und Kopieren des Inhalts ausgeführt werden. Das Erstellen verläuft wie zuvor beschrieben. Der Inhalt eines Ordners, also eines `FactoryNode`, sind die Kindelemente. Diese werden in einer höheren Abstraktionsebene rekursiv über `getChildren()` ausgelesen und erstellt. Beim Kopieren einer Datei, also eines `FactoryLeaf`, wird der Inhalt der Quelldatei durch `getContent()` ausgelesen und mit `setContent()` in die Zieldatei geschrieben. `setContent()` greift hierbei auf `updateDocument()` von `WebDav2Factory` zu, um den Inhalt eines vorhandenen Dokuments in die factory zu schreiben.

Verschieben Beim Verschieben werden die Ressourcen kopiert und anschließend gelöscht. Daher wird beim Verschieben auf die Kopier- und Löschmethoden zurückgegriffen. Weil die zu verschiebende Ressource unter einem anderen Namen gespeichert werden kann, wird diese Methode auch zum Umbenennen verwendet.

Da beim Umbenennen durch das Verschieben immer eine neue Ressource angelegt wird und die Quellressource erst nach erfolgreichem Kopieren gelöscht wird, gibt es einige Nachteile, die insbesondere bei Ressourcen mit hoher Speichergröße problematisch sind. So kann einerseits ein einfaches Umbenennen durch den Kopiervorgang sehr lange dauern und andererseits erhöht sich der Speicherbedarf während des Kopiervorgangs, da bis zum abschließenden Löschvorgang ein Duplikat existiert.

In der factory treten noch zwei andere Probleme auf. Den Ordnern oder Dateien liegen Datensätze einer Datasource und deren Dokumente zugrunde. Beim Umbenennen würde zuerst ein neuer Datensatz unter Angabe des WEBDAVNAME angelegt werden. Da die Spalten eines Datensatzes nicht im Web Folder abgebildet sind, sondern nur der Name, ist der neu angelegte Datensatz, bis auf die Spalte WEBDAVNAME, leer. Abschließend wird der Quelldatensatz gelöscht und die Spalteninhalte sind verloren. Außerdem kann einigen Datensätzen in der factory nur ein Dokument angehängt werden. Dadurch würde das Umbenennen durch ein Verschieben nicht funktionieren, da bei diesen Vorgang ein zweites Dokument an einen Datensatz angehängt werden müsste, um nach erfolgreichem Kopiervorgang das ursprüngliche Dokument zu löschen.

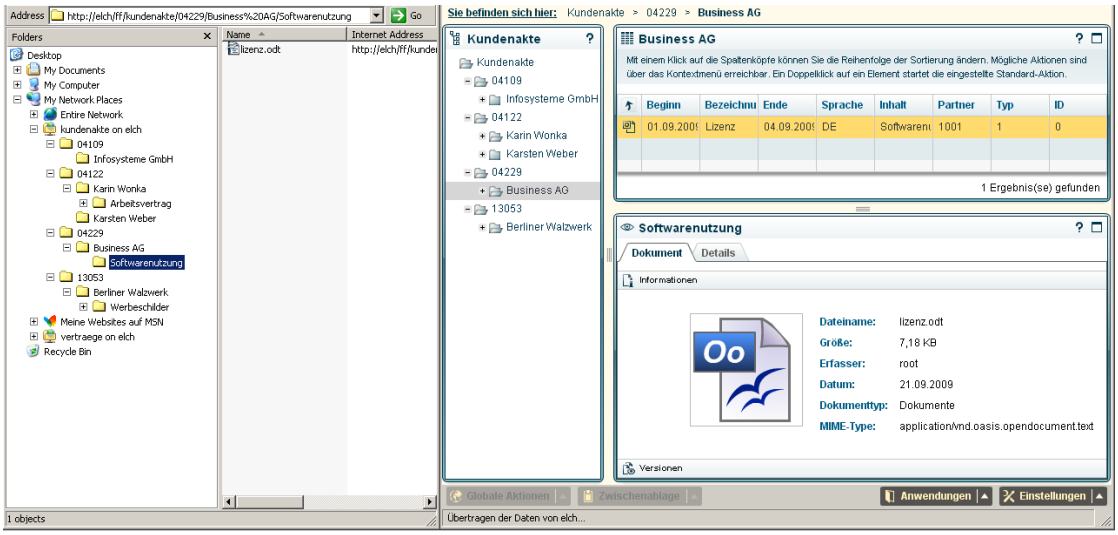

Abbildung 3.8: Gegenüberstellung der View „Kundenakte“ im Web Folder (links) und im Browser (rechts).

Aufgrund dieser Probleme wurde die Klasse `DoMove` des Servlets um die Methode `isRename()` erweitert. Diese Methode prüft, ob das Verschieben innerhalb einer Ebene stattfindet und es sich somit um ein Umbenennen handelt. Wenn die Ressource umbenannt werden soll, wird die Methode `renameResource()` des Stores aufgerufen, die die Implementierung für die Datenstruktur ausführt. Ansonsten wird das Verschieben durch ein Kopieren mit anschließendem Löschen durchgeführt.

Beim Umbenennen eines Ordners bzw. eines `FactoryNode` wird die Spalte WEBDAVNAME aktualisiert. Das Umbenennen einer Datei kann leider nur durch das Erstellen einer neuen Datei ausgeführt werden, da die factory keine anderen Möglichkeiten bietet. Die Methode `rename()` von `FactoryLeaf` arbeitet nach dem Prinzip des Verschiebens. Zuerst wird das Dokument in den Arbeitsspeicher gelesen und anschließend aus der factory gelöscht. Nun wird in der factory ein neues Dokument mit neuem Dateinamen erstellt, in das der Inhalt des Arbeitsspeichers geschrieben wird.

3.4.3 Probleme und Anmerkungen

Die Abbildung eines factory-Baums auf einen Web Folder ist in Abbildung 3.8 illustriert. Im linken Fenster ist die View „Kundenakte“ als Web Folder eingebunden und das rechte Fenster zeigt diese View im factory-Client. Der Datensatz der Datei `lizenz.odt` ist durch den WEBDAVNAME `Softwarenutzung` als Ordner dargestellt. Das an den Datensatz angehängte Dokument ist als Datei eingebunden.

Obwohl die Abbildung eines factory-Baums im Web Folder erstellt werden konnte, gibt es bei der praktischen Anwendung noch einige Schwierigkeiten.

Einige Dateiattribute, wie zum Beispiel das Änderungsdatum, wurde im Windows Explorer nicht korrekt angezeigt. Erst nach der Installation des Patches KB907306[6] konnte dieser Fehler behoben werden.

Ebenfalls schwierig gestaltet sich das Öffnen von Dateien aus dem Web Folder heraus. Das Öffnen aller Microsoft Office-fremden Dateitypen wurde entweder durch eine Fehlermeldung von Windows abgebrochen oder erst gar nicht ausgeführt. Der Patch

[6] `http://support.microsoft.com/kb/907306`

Anwendung	Kommandozeile	Öffnen-Dialog
Microsoft Office 2000	Öffnen, Speichern	Öffnen, Speichern
Microsoft Notepad	nicht möglich	Öffnen einer lokalen Kopie
Microsoft Paint	nicht möglich	Öffnen einer lokalen Kopie
OpenOffice 3.1	Öffnen, Speichern	Öffnen einer schreibgeschützten lokalen Kopie
Adobe Reader 9.1.0	nicht möglich	Öffnen
Dia 0.97	nicht möglich	nicht möglich
GIMP 2.6.7	Öffnen, Fehler beim Speichern	Öffnen, Fehler beim Speichern
LyX 1.6.4	nicht möglich	Öffnen einer schreibgeschützten lokalen Kopie

Tabelle 3.1: Diese Tabelle zeigt, ob die Anwendung ein Dokument mit WebDAV in der gewählten Form Öffnen und Speichern kann.

KB943337[7] konnte dieses Problem nur teilweise beheben. Nach der Installation dieses Patches wird die URL der Datei mit einem Browser geöffnet. Somit erscheint in aller Regel der Download-Dialog des Browsers oder die Datei wird durch ein Plug-in in diesem angezeigt. Als Alternative kann jedoch in jeder Anwendung über den Öffnen-Dialog auf den Web Folder zugegriffen werden. Doch auch hier gibt es leider Einschränkungen. Die Tabelle 3.1 zeigt, dass bei fast allen Testanwendungen eine temporäre Kopie heruntergeladen wird, wodurch ein direktes Arbeiten auf dem Dokument nicht möglich ist. Da dieses Phänomen auch bei Anwendungen und Dateitypen beobachtet wurde, bei denen das Online-Editieren erfolgreich funktioniert, liegt die Vermutung nahe, dass lediglich einzelne Anwendungen Schwierigkeiten mit dem Umgang von Web Foldern haben.

Zum Beispiel gestaltet sich das Online-Editieren mit OpenOffice problemlos. Im Gegensatz dazu wird ein Dokument beim Öffnen über den Web Folder nur durch eine temporäre Kopie geöffnet. Beide Vorgänge unterscheiden sich hauptsächlich in zwei Aspekten: einerseits die Implementierung mit der dazu gehörigen URL und andererseits der Vorgang des Öffnens. Beim Online-Editieren wird die Anwendung direkt über die Kommandozeile gestartet. Um den Startvorgang des Online-Editierens für die neue Implementierung zu testen, wird OpenOffice zum Beispiel mit folgendem Kommando gestartet:

```
"C:\Program Files\OpenOffice.org 3\program\swriter.exe" ↵
   http://elch/ff/kundenakte/04229/Business ↵
   AG/Softwarenutzung/lizenz.odt
```

Bei dieser Methode wird das Dokument korrekt geöffnet und lässt sich direkt bearbeiten, so dass eine grundsätzlich fehlerhafte Implementierung ausgeschlossen wird. Anscheinend ist das Problem von der Art des Öffnens abhängig. Um Unterschiede zwischen den zwei Methoden zu finden, wurden die Request-Header ausgewertet. Beim Starten von OpenOffice über die Kommandozeile wird kein `User-Agent` Header übertragen. Wird jedoch der Öffnen-Dialog und somit der Web Folder genutzt, wird der Header

```
User-Agent: Microsoft Data Access Internet Publishing Provider DAV
```

übertragen. OpenOffice muss folglich zwei verschiedene WebDAV-Clients nutzen. Offensichtlich wird beim Starten über die Kommandozeile ein eigener Client genutzt. Hingegen

[7] `http://support.microsoft.com/kb/943337`

wird beim Öffnen-Dialog die Implementierung von Microsoft, also der Web Folder, genutzt. Dies lässt vermuten, dass Drittanwendungen den Web Folder (oder umgekehrt) nicht vollständig unterstützen.

Als Alternative gibt es Programme wie NetDrive von WiseTodd PTE. LTD.[8], die eine WebDAV-Adresse als Festplatte in das Betriebssystem einhängen (englisch: to mount) können. Dadurch arbeiten die Anwendungen wie auf einem lokalen Datenträger. Alle Lese- und Schreibzugriffe werden von der Zusatzanwendung als WebDAV-Methoden ausgeführt. Im Gegensatz zu Web Foldern ist den Anwendungen nicht bekannt, dass auf eine entfernte Ressource zugegriffen wird. Da einige Anwendungen während der Arbeit mit Dokumenten temporäre Sicherungsdateien erstellen, muss die factory für diese Alternative weitere Voraussetzungen erfüllen.

Aufgrund der Funktionsweise einer View bleiben grundsätzliche Probleme bei der Abbildung eines factory-Baums auf einen Web Folder bestehen. Ein Web Folder soll die Arbeit wie mit einem herkömmlichen Dateisystem ermöglichen. Das heißt, dass im Wesentlichen alle Funktionen, wie das Umbenennen oder Kopieren, auf allen Ordnern und Dateien ausgeführt werden können. Dies ist jedoch nur möglich, wenn alle Ordner und Dateien real auf einen Datenträger existieren, was in einem Dateisystem auch der Fall ist. Durch mögliche Selektionen in einer View entstehen jedoch neue Knoten bzw. Ordner, die keinem Datenträger direkt zugewiesen sind. Ein Web Folder bietet aber grundsätzlich alle Dateioperationen für jeden Ordner und jede Datei an. Jedoch können diese an einigen Knoten bzw. Ordnern nicht ausgeführt werden und es kommt zu einer Fehlermeldung. Außerdem kann nach dem Anlegen eines neuen Ordners bzw. Datensatzes in einer anderen View ein neuer Knoten entstehen. Der daraus abgebildete Ordner besitzt in der Regel aber keinen korrekten Namen, da die Selektion auf einer leeren Spalte basiert. Aufgrund fehlenden Hintergrundwissens sind diese Fehlermeldungen und Phänomene für einen Endanwender nicht nachvollziehbar. Infolgedessen sollte eine View unter Berücksichtigung der Eigenschaften eines Web Folders erstellt werden, wenn die Abbildung in diesem erfolgen soll.

3.5 Tool zur Demonstration einer Attributierung

Metainformationen bzw. Properties von Ressourcen werden in WebDAV über die Methoden PROPFIND und PROPPATCH genutzt. Um ein Dokument in der factory an einen richtigen Datensatz anzuhängen, müssen für dieses Dokument zusätzliche Informationen übertragen werden. Diese könnten zwar theoretisch über die Methoden für Properties realisiert werden, jedoch unterstützt der Web Folder von Microsoft leider keine Dead-Properties. Um die Möglichkeiten mit Properties dennoch untersuchen zu können, muss eine Hilfsanwendung verwendet werden.

3.5.1 Analyse

In der factory ist ein Dokument an einen Datensatz gebunden. Um das Ablegen von Dokumenten auch ohne den factory-Client zu ermöglichen, gibt es zur Zeit (Oktober 2009) zwei Tools zur Desktopintegration. Das „forcont factory addin“ erweitert zum Beispiel die Microsoft Office Anwendungen mit der „forcont factory FX Ablage“. Damit kann ein Dokument aus der Anwendung heraus in der factory abgelegt werden. Beim Ablegen erscheint eine Eingabemaske die für jede Spalte der Datasource ein Eingabefeld enthält

[8] `http://www.netdrive.net`

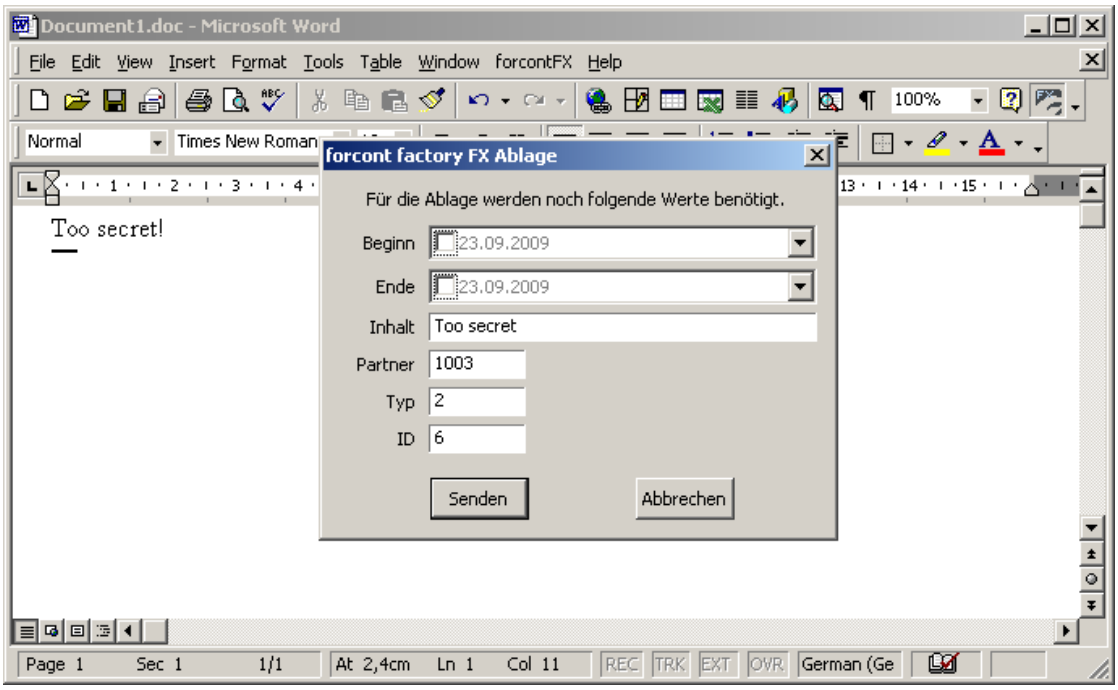

Abbildung 3.9: forcont factory FX Ablage in Microsoft Word

(vgl. Abbildung 3.9). Somit kann ein neuer vollständiger Datensatz erstellt werden, an den das Dokument angehängt wird. Jedoch soll nach neuen Möglichkeiten für diese Lösung gesucht werden, da diese mit zwei Nachteilen verbunden ist: Erstens ist das forcont factory addin eine Zusatzanwendung, die für jeden Benutzer separat installiert werden muss. Zweitens sind solche Tools immer an eine Anwendung oder Hersteller gebunden, wodurch der Entwicklungs- und Wartungsaufwand steigt.

Aufgrund der vorhandenen Methoden wäre es theoretisch möglich das Ablegen von Dokumenten mit WebDAV ähnlich wie zuvor beschrieben umzusetzen. Dies hätte den Vorteil, dass eine Implementierung von allen Anwendungen ohne Zusatzsoftware genutzt werden könnte. Mit PROPFIND können die Spalten einer Datasource abgefragt und somit die Eingabefelder erstellt werden. Beim Senden werden die Eingaben mit PROPPATCH an die Datasource gereicht.

Da die Microsoft Web Folder nur Live-Properties wie Erstellungsdatum oder Dateigröße unterstützen, ist die Entwicklung einer neuen Ablage ohne Zusatzsoftware nicht möglich. Zumindest müsste für die Erstellung einer Eingabemaske und das Absenden der Daten ein eigenes Tool verwendet werden. Da der Nutzen eines weiteren Tools im Bezug auf die bestehende Lösung zu klein ist, wird dieser Weg der Ablage vorläufig nicht weiter untersucht. Stattdessen wird ein Tool entwickelt, das das Auslesen und Ändern von Properties ermöglicht. Dadurch kann der bisher fehlende Teil der Properties getestet und demonstriert werden.

3.5.2 Umsetzung

Die genutzte Implementierung von WebDAV unterstützt keine eigenen Properties. Folglich müssen das Servlet und die Stores mit diesen Funktionen erweitert werden. Dieser Teil muss auf dem Server implementiert werden. Da ein Web Folder keine Funktionen für Properties unterstützt, muss für den Client ein Zusatztool entwickelt werden. Dieses Tool erweitert den Client um die fehlenden Property-Methoden eines Web Folders.

Server

Das Servlet delegiert die Property-Methoden an die Klassen `DoPropfind` und `DoProppatch`. Diese Klassen verarbeiten die Anfragen und unterstützen nur DAV-Properties, die im Standard definiert sind. Da sich individuelle Properties von DAV-Properties durch ihren Namensraum unterscheiden, müssen `DoPropfind` und `DoProppatch` so erweitert werden, dass nur Properties aus anderen Namensräumen speziell verarbeitet werden. Die Art und Weise wie die Zusatzinformationen einer Ressource ermittelt und gespeichert werden, kann sich zwischen verschiedenen Stores unterscheiden. Basiert ein Store auf einem NTFS-Dateisystem, könnten Metainformationen im Alternate Data Stream gespeichert werden. In der Implementierung für die factory sind diese Informationen in einer Datenbank gespeichert. Um jeden Store diese Freiheit der Implementierung zu überlassen, wurde die Schnittstelle `IWebDavStore` mit den folgenden Methoden erweitert:

- `getProperties()` liefert eine Liste der angeforderten Properties zurück,
- `updateProperty()` aktualisiert oder löscht ein Property.

Die Klasse `DoPropfind` bearbeitet eine PROPFIND-Anfrage und muss dadurch drei verschiedene Anfragen unterstützen. Wenn durch die Angabe von `<D:allprop/>` oder `<D:propname/>` alle Properties oder nur deren Namen angefordert werden, werden nun nach dem Generieren der DAV-Properties immer zusätzlich die Dead-Properties durch den Aufruf von `getProperties()` abgefragt. Im Gegensatz dazu sollen bei der Anfrage einzelner Properties nur solche an den Store weitergereicht werden, die keine DAV-Properties sind, also einen anderen Namensraum besitzen.

Eine PROPPATCH-Anfrage wird von der Klasse `DoProppatch` ausgeführt. Wenn das zu aktualisierende Property kein DAV-Property ist, wird die Methode `updateProperty()` des Stores ausgeführt.

Für die Implementierung der Properties wird der zuvor entwickelte Store `FactoryTreeStore` erweitert. Dieser muss nun die neuen Methoden von `IWebDavStore` implementieren. Jede Spalte in einer Datasource wird im `FactoryTreeStore` als Property angesehen. Somit liest `getProperties()` alle oder nur bestimmte Spalten der Datasource aus, die der Ressource zugrunde liegen. Hierfür definiert `IFactoryTree` die Methode `propFind()` und aus der Klasse `WebDav2Factory` kann mit `getRecord()` ein Datensatz und dessen Spalten aus der factory gelesen werden. `updateProperty()` muss die übergebenen Informationen in die factory schreiben. Dazu existiert die Methode `propPatch()` in `IFactoryTree`. Diese Methode aktualisiert einen Datensatz in der factory durch `updateRecord()` in `WebDav2Factory`. Für ein erfolgreiches Aktualisieren der Properties muss der Ressource eine Datasource mit den angeforderten Spalten (Propertynamen) zugrunde liegen.

Client

Für den Client muss eine einfache Anwendung entwickelt werden, die PROPFIND- und PROPPATCH-Anfragen versenden und die Antworten darstellen kann. Da dieses Tool nur zum Testen und Präsentieren der Funktionalität des Servers dienen soll, wird die Benutzerschnittstelle sehr einfach gehalten. In Abbildung 3.10 auf der nächsten Seite ist das entwickelte Property-Tool dargestellt. Die Oberfläche unterteilt sich hauptsächlich in zwei Bereiche: Für die Eingabe und Bedienung sind im oberen Teil Textfelder, Listboxen und Buttons vorhanden. Der untere Teil wird nur zur Anzeige der Anfragen und Antworten genutzt.

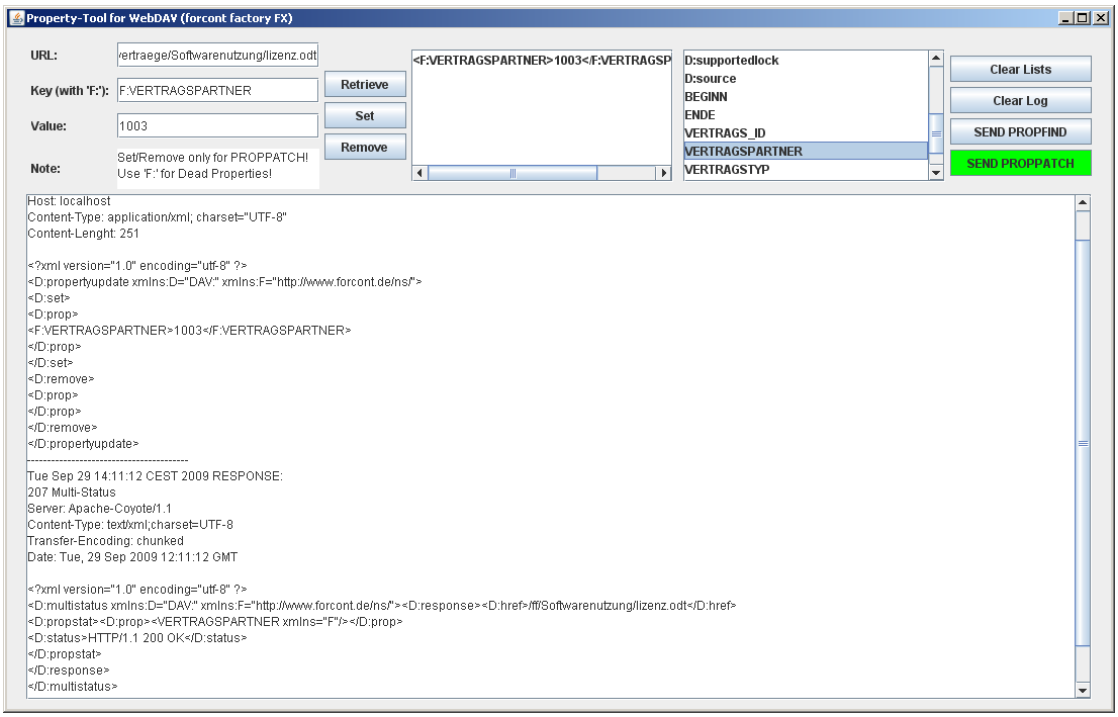

Abbildung 3.10: Property-Tool zur Demonstration einer Attributierung.

Mit den drei Buttons „Retrieve“, „Set“ und „Remove“ kann ein Property für eine Anfrage zu einer Property-Liste hinzugefügt werden. Wird ein Property mit Retrieve (deutsch: Abfragen) auf die Liste gesetzt, wird bei PROPFIND der Wert des Property abgefragt. Wenn die Property-Liste beim Versenden von PROPFIND leer ist, werden alle Properties angefordert. Mit Set für Setzen und Remove für Löschen werden Properties für ein PROPPATCH auf die Liste gesetzt.

Beim Betätigen der Buttons SEND PROPFIND und SEND PROPPATCH wird die Anfrage generiert, in der unteren Textbox ausgegeben und anschließend versendet. Da die empfangene Antwort nicht weiterverarbeitet werden muss, wird der enthaltene XML-Code nicht ausgewertet, sondern nur in der unteren Textbox ausgegeben. Nur wenn ein PROPFIND versendet wurde, werden die empfangenen Propertynamen gefiltert und zur Auswahl in einer Liste aufgeführt.

3.5.3 Probleme und Anmerkungen

Das Property-Tool zeigt auf eine einfache Art und Weise, dass das Auslesen und Bearbeiten von Attributen eines Dokuments möglich ist. Ein Ablegen und gleichzeitiges Attributieren eines Dokuments ist jedoch nicht so einfach zu realisieren, da immer nur eine Anfrage und somit entweder das Versenden der Datei oder das Versenden der Attribute ausgeführt werden kann. Die Einbindung der factory in Windows durch einen Web Folder ermöglicht das Ablegen von Dokumenten aus der Anwendung heraus, falls diese den Web Folder unterstützt. Um das Ablegen mit gleichzeitigem Attributieren zu erreichen, müsste ein neues Tool vier WebDAV-Anfragen nacheinander ausführen:

1. Zuerst muss MKCOL immer einen leeren Datensatz in der Datasources erzeugen,
2. dann kann das Dokument mit PUT abgelegt werden,
3. nun werden die Attribute durch PROPFIND vom Server ermittelt,

4. abschließend werden alle Spalten durch PROPPATCH aktualisiert.

Die PUT-Anfrage kann auch zuletzt ausgeführt werden, da die Attributierung auch am Datensatz ausgeführt werden kann. Mit diesen Schritten könnte der bisherige Ablegevorgang mit WebDAV-Methoden realisiert werden.

4 Schlusswort

4.1 Zusammenfassung

Die Abschnitte 2.1 und 2.2 beschreiben die Grundlagen dieser Arbeit. Die Hauptbestandteile der Protokolle HTTP und WebDAV werden mit einigen Beispielen kurz erklärt. Äußerlich unterscheiden sich die Nachrichten des HTTP und WebDAV kaum. Erst durch eine Analyse des Inhalts können Unterschiede bei den Methoden, im Header, im Entity-Body und bei den Status-Codes erkannt werden. Die Nutzung des Entity-Body für die Übertragung von Parametern in XML-Code kann sicherlich als größte Neuerung bezeichnet werden. Nur dadurch ist es möglich geworden, das HTTP umfangreich zu erweitern, ohne dieses selbst zu verändern. Obwohl als Basis für die vorliegende Arbeit eine bereits funktionsfähige Implementierung genutzt wird, war die Aneignung dieser Grundlagen vor allem für die Fehlersuche und Erweiterung der Grundfunktionalitäten sehr nützlich.

WebDAV wird den Zugriff auf Ressourcen, also Daten, verwendet. Diese können auf unterschiedliche Weise gespeichert und organisiert sein. Im Abschnitt 2.3 wird das Prinzip der Datenorganisation in der factory beschrieben. Hierbei wird festgestellt, dass Unterschiede bei der Datenorganisation zwischen der factory und WebDAV vorhanden sind. WebDAV fordert, dass jede Ressource durch eine URL abgebildet ist. Für den Zugriff auf eine Datei kann somit nur die Hierarchie und der Dateiname herangezogen werden. Durch die Verlinkung der Dokumente mit einem Datensatz in der factory werden für die eindeutige Identifikation einer Ressource aber mehrere Parameter benötigt. Auch die interne Baumstruktur der factory, die der Datenorganisation von WebDAV ähnelt, kann ohne Änderungen nicht genutzt werden, da zum Zugriff auf und für die Darstellung von Ressourcen verschiedene Parameter genutzt werden.

Mit Hilfe der erarbeiteten Kenntnisse wird in Kapitel 3 die Umsetzung der einzelnen Teilaufgaben beschrieben. In Abschnitt 3.1 wird die technische Umgebung aufgeführt und die gewählte Technologie begründet. Aufgrund der Verwendung des Apache Tomcat und der darin enthaltenen WebDAV-Komponente sollte dieser frei verfügbare Code als Basis dienen, um ihn auf die Anforderungen der factory zu zuschneiden. Probleme bei der Verwendung und Einbindung dieser Komponente führten jedoch zur Wahl einer anderen Open Source Implementierung. Abschnitt 3.2 verzichtet auf eine ausführliche Darstellung technischer Details und führt in knapper Form die Hauptkomponenten dieser Vorgehensweise auf. Die wichtigste dieser Komponenten ist der so genannte Store, weil dieser die zentrale Schnittstelle zwischen der speziellen Implementierung und den bereitgestellten Grundfunktionen des Servlets darstellt.

Das Kernziel dieser Untersuchungen ist die Suche nach neuen Möglichkeiten zur Arbeit mit den in der factory verwalteten Dokumenten.

Das Bearbeiten von Dokumenten in der forcont factory FX 1.0 ist durch das Ein- und Auschecken für viele Benutzer zu kompliziert. Deswegen wurde im Abschnitt 3.3 dieses Problem analysiert und nach Lösungen gesucht. WebDAV ermöglicht ein direktes Editieren von entfernten Ressourcen. Somit ist eine lokale Kopie des Dokuments

nicht mehr nötig, wodurch das manuelle Ein- und Auschecken des Anwenders von einem Sperrmechanismus ersetzt werden kann. Das Online-Editieren fordert zwei Implementierungen. Zu Beginn wurde der Zugriff des WebDAV-Servlets auf die factory durch den Store `FactoryDirectStore` ermöglicht. Da die Identifizierung eines Dokument und der Zugriff auf dieses nur über die URL möglich ist, müssen alle benötigten Parameter in dieser übertragen werden. Um einen einheitlichen Aufbau der URL zu gewährleisten, wird zur Erstellung und zum Filtern dieser die Klasse `FactoryDirectUrl` genutzt. Nun muss das Bearbeiten aus dem Frontend heraus gestartet werden können. Da beim Starten einer URL im Browser nicht unterschieden werden kann, ob es sich um den Aufruf einer Website oder eines Dokuments handelt, muss die WebDAV-URL gesondert gestartet werden. Dafür wurde ein Applet im factory-Client eingebunden, das die für dieses Dokument assoziierte Anwendung mit der URL startet. Das gewünschte Ziel einer neuen, vollständigen und funktionsfähigen Methode zur Bearbeitung von Dokumenten konnte nur mit Einschränkungen erreicht werden. Einerseits ist zwar kein explizites Auschecken und die damit verbundene lokale Kopie mehr nötig, andererseits muss das Dokument jedoch manuell wieder entsperrt werden, wenn nicht auf den Ablauf einer Zeitspanne gewartet werden soll. Von einigen kleineren und lösbaren Schwierigkeiten abgesehen gab es vor allem beim Starten einiger Anwendungen Probleme. Diese sind jedoch nicht in der Implementierung, sondern in den Anwendungen selbst begründet.

Die zwei weiteren in der Einleitung aufgeführten Teilaufgaben haben sich aufgrund neuer Erkenntnisse leicht verändert. Da Web Folder keine Möglichkeiten für das Attributieren von Dokumenten bieten, muss dafür eine Zusatztool verwendet werden. Weil für das Ablegen von Dokumenten und das Generieren einer Verzeichnisstruktur ein Web Folder genutzt werden kann, können diese Aufgaben zusammen umgesetzt werden.

Abschnitt 3.4 beschreibt das Vorgehen zur Lösung dieser zwei Aufgaben. Ein Web Folder organisiert die Daten wie ein Dateisystem. Der Zugriff findet über den Datei- bzw. Ordnernamen statt. Die Daten der factory werden intern über eine Baumstruktur verwaltet, die stark einem Dateisystem ähnelt. Jedoch wird im Gegensatz zum Dateisystem über die Knoten-IDs und nicht über die Knotennamen auf diese zugegriffen. Deswegen muss die vorhandene Baumstruktur erweitert werden, damit die Eigenschaften und Operationen eines Dateisystems abgebildet werden können. Durch die Schnittstelle `IFactoryTree` können Baumstrukturen aufgebaut werden, die die interne Struktur der factory erweitern und somit die Anwendung für ein Dateisystem ermöglichen. Zur Sicherstellung eines eindeutigen Ressourcennamens muss jede Datasource eine Spalte für einen WebDAV-Namen enthalten. Damit das Servlet auf diese neue Datenstruktur zugreifen kann, muss der neue Store `FactoryTreeStore` genutzt werden. Mit dieser Lösung können Views der factory als Web Folder abgebildet werden. Dadurch können theoretisch alle Dateioperation auf der factory ausgeführt werden. Dokumente können im Windows Explorer ohne factory-Client geöffnet, erstellt, kopiert, verschoben, umbenannt und gelöscht werden. Durch die Datenstruktur der factory gibt es bei der praktischen Anwendung leichte Einschränkungen. So können nicht alle Operationen erfolgreich auf jedem Ordner oder jeder Datei ausgeführt werden. Des Weiteren scheinen einige Programme nicht korrekt mit einem Web Folder arbeiten zu können.

Mit dem `FactoryTreeStore` wurden alle Methoden von `IWebDavStore` implementiert. Um auch die fehlende Attributierung zu ermöglichen, muss der Store und das Servlet erweitert werden. Damit die Erweiterungen auch getestet werden können, muss ein Zusatztool entwickelt werden. In Abschnitt 3.5 werden die Erweiterungen für die Attributierung beschrieben. Als Properties für ein Dokument werden die Spalten der zugrun-

deliegenden Datasource festgelegt. Mit dem Tool können die Methoden PROPFIND und PROPPATCH ausgeführt und demonstriert werden. So können Properties durch PROPFIND gelesen und durch PROPPATCH aktualisiert werden. Die Ablösung der Zusatztools zur Ablage von Dokumenten ohne factory-Client ist jedoch nicht möglich. Zwar kann ein Dokument und auch ein neuer Datensatz über einen Web Folder angelegt werden, aber durch die fehlende Unterstützung von Properties ist eine Attributierung nicht möglich und die Spalten bleiben somit leer.

4.2 Ausblick

Die in der Zielsetzung beschriebenen Aufgaben konnten mit Einschränkungen gelöst werden. Für das Editieren von Dokumenten wurde eine vielversprechende Lösung gefunden. WebDAV erfüllt theoretisch alle sich aus dieser Aufgabe ergebenen Anforderungen. Damit die erarbeitete Implementierung auch für den Endanwender eingesetzt werden kann, muss jedoch noch etwas Entwicklungsarbeit geleistet werden. Als Erstes sollte die Kompatibilität der kundenseitig am häufigsten genutzten Anwendungen weiter getestet werden, um die Startproblematik besser abschätzen zu können. Falls die Kompatibilität ausreicht oder durch ein neues Startverfahren verbessert werden kann, muss das Synchronisationsproblem gelöst werden. Erst dann sollte das Online-Editieren beim Endanwender Einzug halten. Da anzunehmen ist, dass mit zunehmender Verbreitung von WebDAV immer mehr Anwendungen den Standard korrekt unterstützen werden, sollte auch in Hinblick auf den Nutzen beim Web Folder das von WebDAV zur Verfügung gestellte Locking korrigiert werden. Auf diese Weise kann das clientseitige Entsperren beseitigt und somit die Benutzerfreundlichkeit erhöht werden.

Die Web Folder bieten für die Benutzer eine völlig neue, aber durch die Anlehnung an ein Dateisystem dennoch vertraute Sicht auf die factory. Für die Probleme mit virtuellen Ordnern bzw. Knoten sollte nach weiteren Lösungen gesucht werden. Mit aufwendigen Algorithmen kann versucht werden, einige Operationen auf diesen Ordnern auszuführen. Vorher sollte forcont jedoch ein Konzept erarbeiten, ob und wofür Web Folder bei Endkunden genutzt werden sollen. Der Schwerpunkt der Web Folder ist sicherlich die Verbesserung des Editierens von Dokumenten und weniger die Arbeit mit den Datensätzen. Spätestens bei der Nutzung von Web Foldern muss für das Sperren der Dokumente das Locking-Problem von WebDAV gelöst werden. Für eine Zugriffskontrolle muss zusätzlich die HTTP-Authentifizierung implementiert werden. Das Abschätzen des Aufwands für diese Umsetzung ist schwierig, da die Authentifizierung in dieser Arbeit nicht behandelt wurde.

Das Attributieren von Dokumenten ist ohne Hilfsmittel leider nicht möglich. Aufgrund der vorhandenen Desktopintegrationen sollte diese Thema nicht über WebDAV realisiert werden.

Trotzt einiger Einschränkungen bietet Web-based Distributed Authoring and Versioning das Potenzial für neue und vor allem nützliche Funktionen. Diese Arbeit zeigt, dass die Grundfunktionalität auch auf der forcont factory FX umgesetzt werden kann. Die erlangten Kenntnisse müssen von forcont untersucht werden, damit der Nutzen von WebDAV für ihr Produkt und der noch nötige Entwicklungsaufwand abgeschätzt werden kann. Bei der Bereitstellung einer serverseitigen WebDAV-Schnittstelle ohne eigene Clientsoftware, wird ein Teil der Funktionalität aus der eigenen Hand gegeben. Dadurch entsteht eine Abhängigkeit von Drittherstellern, die die eigene Produktqualität subjektiv senken könnte, da der Anwender eine von der Umsetzung unabhängige, funktionsfähige

Softwarelösung voraussetzt.

Die ständige Weiterentwicklung von Programmiersprachen für Rich Internet Applications sowie das von der Industrie als hoffnungsvoll betitelte Cloud Computing werden in Zukunft immer mehr Dienste wie zum Beispiel Google Docs[1] ermöglichen. RIAs sind Webanwendungen mit einer intuitiven Benutzeroberfläche und Funktionalität, die mit der von Desktopanwendungen vergleichbar sein soll. Als Cloud Computing wird eine IT-Infrastruktur bezeichnet, in der verschiedene Aspekte der Technik (Speicherplatz, Rechenressourcen) und Software (serviceorientiert, Virtualisierung) durch einen Pool vernetzter Computer umgesetzt werden. Dabei spielt die Vernetzung und Verfügbarkeit der Systeme sowie die Flexibilität der vorhandenen Ressourcen eine entscheidende Rolle. In diesem Szenario kann WebDAV sein volles Potential entfalten. Durch die mögliche Kombination eigens entwickelter Clients und Server können alle Funktionen einem bestimmten Anwendungszweck angepasst werden. Dadurch ist eine hohe Kompatibilität und Benutzerfreundlichkeit gegeben. Grundsätzlich könnten RIAs zwar auch Webservices nutzen, jedoch ist der Vorteil von WebDAV die mögliche Unterstützung von Drittsoftware ohne zusätzliche Entwicklungskosten.

[1] `http://docs.google.com`

Abbildungsverzeichnis

Abkürzungsverzeichnis

BNF Backus-Naur-Normalform

CRLF Carriage Return Line Feed

ECM Enterprise-Content-Management

FTP File Transfer Protocol

HTTP Hypertext Transfer Protocol

ID3 Identify an MP3

IDE Integrated Development Environment

IP Internet Protcol

IPv6 Internet Protocol Version 6

JDK Java Development Kit

JRE Java Runtime Environment

JSP JavaServer Pages

MP3 MPEG-1 Audio Layer 3

PNG Portable Network Graphics

RFC Request For Comments

RIA Rich Internet Application

SCP Secure Copy

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SSL Secure Sockets Layer

TCP Transmission Control Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

UUID Universally Unique Identifier

WebDAV Web-based Distributed Authoring and Versioning

WWW Word Wide Web

XML Extensible Markup Language

Literaturverzeichnis

[Behrens] Behrens, Lars: *WebDAV*. http://www.weka-it.ch/praxisreport_view.cfm?nr_praxisreport=577, Abruf: 03.08.2009

[FBT09] forcont business technology gmbh: *forcont - more time for content Technologiekompetenzen im Enterprise Content Management*. 2009

[FN09] forcon business technology gmbh: *forcontNEWS*. März 2009

[Harnisch03] Harnisch, Carsten: *Dateien mit System - Nutzung von WebDAV unter .NET*. Version: Dezember 2003. http://www.jax.de/zonen/portale/psecom,id,101,online,486,.html, Abruf: 03.08.2009

[RFC1521] Borenstein, N. ; Freed, N.: *RFC1521 - MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, September 1993. http://www.rfc-editor.org/rfc/pdfrfc/rfc1521.txt.pdf

[RFC1945] Berners-Lee, T. ; Fielding, R. ; Frystyk, H.: *RFC1945 - Hypertext Transfer Protocol – HTTP/1.0*, Mai 1996. http://www.rfc-editor.org/rfc/pdfrfc/rfc1945.txt.pdf

[RFC2045] Freed, N. ; Borenstein, N.: *RFC2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996. http://www.rfc-editor.org/rfc/pdfrfc/rfc2045.txt.pdf

[RFC2049] Freed, N. ; Borenstein, N.: *RFC2049 - Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*, November 1996. http://www.rfc-editor.org/rfc/pdfrfc/rfc2049.txt.pdf

[RFC2387] Levinson, E.: *RFC2387 - The MIME Multipart/Related Content-type*, August 1998. http://www.rfc-editor.org/rfc/pdfrfc/rfc2387.txt.pdf

[RFC2518] Goland, Y. ; Whitehead, E. ; Faizi, A. ; Carter, S.R. ; Jensen, D. ; The Internet Society (Hrsg.): *RFC2518 - HTTP Extensions for Distributed Authoring – WEBDAV*. The Internet Society, Februar 1999. http://www.rfc-editor.org/rfc/pdfrfc/rfc2518.txt.pdf

[RFC2616] Fielding, R. ; Gettys, J. ; Mogul, J. C. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T. ; The Internet Society (Hrsg.): *RFC2616 - Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society, Juni 1999. http://www.rfc-editor.org/rfc/pdfrfc/rfc2616.txt.pdf

[RFC2617] Franks, J. ; Hallam-Baker, P. ; Hostetler, J. ; Lawrence, S. ; Leach, P. ; Luotonen, A. ; Stewart, L. ; The Internet Society (Hrsg.): *RFC2617 - HTTP Authentication: Basic and Digest Access Authentication*. The Internet Society, Juni 1999. http://www.rfc-editor.org/rfc/pdfrfc/rfc2617.txt.pdf

[RFC3253] Clemm, G. ; Amsden, J. ; Ellison, T. ; Kaler, C. ; Whitehead, J. ; The Internet Society (Hrsg.): *RFC3253 - Versioning Extensions to WebDAV*. The Internet Society, März 2002. http://www.rfc-editor.org/rfc/pdfrfc/rfc3253.txt.pdf

[RFC3744] Clemm, G. ; Reschke, J. F. ; Sedlar, E. ; Whitehead, J. ; The Internet Society (Hrsg.): *RFC3744 - Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol*. The Internet Society, März 2004. http://www.rfc-editor.org/rfc/pdfrfc/rfc3744.txt.pdf

[RFC4918] Dusseault, L.M. ; The IETF Trust (Hrsg.): *RFC4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. The IETF Trust, Juni 2007. http://www.rfc-editor.org/rfc/pdfrfc/rfc4918.txt.pdf

[WG] Whitehead, E. J. ; Goland, Yaron Y.: *The WebDAV Property Design*. http://www.goland.org/Tech/spe-whitehead.pdf

[WW98] Whitehead, E. J. ; Wiggins, Meredith: WEBDAV: IETF Standard for Collaborative Authoring on the Web. In: *IEEE Internet Computing* (1998). http://ftp.ics.uci.edu/pub/ietf/webdav/intro/webdav_intro.pdf

Inhalt der CD-ROM

`Abbildungen\`	Abbildungen aus der Bachelorarbeit
`API-Dokumentation\`	Dokumentation der Implementierung als HTML
`Binärdateien\`	Externe Bibliotheken, kompilierte Implementierung, ausführbares Property-Tool
`Literatur\`	Verwendete Literatur als PDF
`Quelltext\`	Quelltext der Implementierung
`Bachelorarbeit.pdf`	Bachelorarbeit als PDF mit Hyperref-Unterstützung
`Hinweise für WebDAV.pdf`	Erklärungen zum Einbinden von WebDAV in die factory und zur Arbeit mit dem Quelltext

Eidesstattliche Versicherung

Ich versichere, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen oder anderen Quellen entnommen sind, sind als solche eindeutig kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht veröffentlicht und noch keiner Prüfungsbehörde vorgelegt worden.

Ort, Datum	Unterschrift