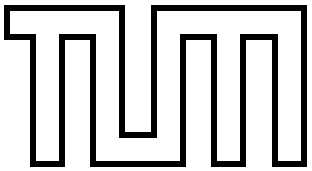


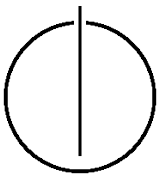
FAKULTÄT FÜR INFORMATIK

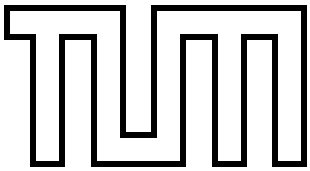
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Konzeption und Implementierung einer dateibasierten Lösung zum Offline-Zugriff auf eine Enterprise 2.0 Plattform

Kilian Wischer





FAKULTÄT FÜR INFORMATIK

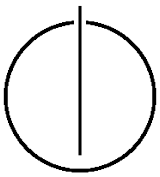
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Wirtschaftsinformatik

Konzeption und Implementierung einer dateibasierten Lösung zum Offline-Zugriff auf eine Enterprise 2.0 Plattform

Design and Implementation of a File-based Solution to Provide Offline Access to an Enterprise 2.0 Platform

Bearbeiter:	Kilian Wischer
Aufgabensteller:	Prof. Dr. Florian Matthes
Betreuer:	Dr. Thomas Büchner
Datum:	16. August 2011



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 16. August 2011 Kilian Wischer

Abstract

In dieser Arbeit wird ein Konzept für einen Offline-Zugriff auf die Enterprise 2.0 Plattform Tricia entwickelt und implementiert. Zunächst werden dazu die drei Ansätze Browser-Plugin basierte Offline-Funktionalität, die Offline-Möglichkeiten in HTML5 und ein dateibasierter Offline-Zugriff hinsichtlich ihrer Eignung für die Implementierung in der Webapplikation Tricia untersucht. Der dateibasierte Offline-Zugriff stellt sich als der beste Ansatz für Tricia heraus. Anschließend wird ein Konzept erarbeitet, wie der dateibasierte Offline-Zugriff in Tricia integriert werden kann. Dieses Konzept wird anschließend implementiert. Als Hauptanforderung aus Benutzersicht an eine Offline-Version von Tricia stellt sich für den Lesezugriff das Betrachten der hybriden Wikiseiten heraus. Für den Offline-Schreibzugriff wird das Anlegen von Notizen zu Wikiseiten als Hauptanforderung identifiziert. Als Schnittstelle zwischen Online- und Offline-Version von Tricia wird das bereits implementierte SMB-Laufwerk gewählt. Die Dateien der einzelnen Wikiseiten werden dem Benutzer auf diesem SMB-Laufwerk als HTML-Dateien in einer neu angelegten Ordnerstruktur angeboten. Die HTML-Dateien werden über das schon vorhandene Template-System von Tricia generiert. Die Ordnerstruktur, in welcher die HTML-Dateien angeboten werden, wird in die bestehende Implementierung des SMB-Laufwerks als neues Plugin integriert. Das Speichern von Offline-Notizen wird über JavaScript implementiert und orientiert sich an dem Konzept der Wikisoftware TiddlyWiki[1]. Für die Synchronisation wird das Konzept der „Two-Step"-Synchronisation entwickelt. Dabei wird eine „Changeset"-Datei, die lokal auf dem Rechner generiert wird, mit dem SMB-Laufwerk von Tricia synchronisiert. Auf dem Server wird diese Datei ausgelesen und in die Datenbank eingespielt.

[1] `http://www.tiddlywiki.com/`, aufgerufen am 11. August 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltexte

1 Einleitung und Motivation

Ziel dieser Arbeit ist es, ein Konzept für einen Offline-Zugriff auf die Enterprise 2.0 Plattform Tricia zu erarbeiten und anschließend zu implementieren.

Tricia ist eine Enterprise 2.0 Plattform, die auf einem Server installiert wird. Tricia ist also eine Webapplikation, die von den Benutzern typischerweise über einen Internetbrowser bedient wird. Die Software dient der Kommunikation und dem Wissensmanagement von unternehmensinternem sowie unternehmensexternem Wissen. Diese Informationen können in Tricia in Form von Wikis und Wikiseiten abgelegt werden. Dabei wurde das Konzept von hybriden Wikis entwickelt. In hybriden Wikis lassen sich unstrukturierte Informationen, z.B. Texte, Bilder und andere Dateien in Wikiseiten ablegen. Zusätzlich lassen sich zu den Wikiseiten „Formulare“ anlegen, die den Wikiseiten eine gewisse Struktur verleihen. Die Idee der hybriden Wikis wird in Kapitel 3.1 vorgestellt.

Tricia ist ein Produkt der infoAsset AG[1], welches in Kooperation mit dem Lehrstuhl Software Engineering for Business Information Systems (sebis)[2] der Technischen Universität München entwickelt wird.

Die Grundidee für einen Offline-Zugriff auf Tricia ist, dass Benutzer sich einzelne Wikis lokal auf dem Rechner speichern können. Der Benutzer soll die Wikiseiten des Wikis einsehen können. Außerdem soll der Benutzer die Möglichkeit haben, lokale Veränderungen auf den Wikiseiten vorzunehmen. Anschließend soll es eine Möglichkeit geben, diese lokalen Änderungen wieder in die Webapplikation einspielen zu können.

1.1 Pro und Contra Offline-Funktionalität

Es stellt sich die Frage, wozu braucht man überhaupt einen Offline-Zugriff auf eine Webapplikation wie Tricia. Grundsätzlich spricht der zusätzliche Aufwand gegen eine Implementierung einer solchen Funktionalität. Eine Offline-Funktionalität steigert die Komplexität der gesamten Anwendung. Gegen die Implementierung spricht weiterhin der sehr gute Ausbau des Breitband-Netzes in Westeuropa und den USA. Außerdem besteht eine gute Netzabdeckung des mobilen Internet in Ballungsräumen, so dass vermeintlich kein Bedarf für einen Offline-Zugriff besteht. Allerdings gibt es auch immer wieder Situationen, in denen man aus vielerlei Gründen keinen Zugriff auf das Internet hat. Sei es auf Geschäftsreisen oder in suburbanen bzw. ländlichen Gegenden. Aber gerade auf Geschäftsreisen kann der Zugriff auf Unternehmens-Wissen essenziell für den geschäftlichen Erfolg sein. Auch wenn der Server, auf dem Tricia installiert ist, ausfällt, kann man trotz Internetzugang nicht auf das System zugreifen. Zusätzlich hat man durch eine Offline-Version von Tricia ein lokales Back-up. Dieses Back-up kann im schlimmsten

[1] `http://www.infoasset.de/`, aufgerufen am 18. Mai 2011

[2] `http://wwwmatthes.in.tum.de/`, aufgerufen am 18. Mai 2011

Fall eines Datenverlustes auf dem Tricia Server dazu verwendet werden, zumindest einen Teil der Daten wieder in die Webapplikation zurückzuspielen.

Da die Argumente für eine Offline-Funktionalität in Tricia überwiegen, versucht diese Arbeit ein Konzept für Tricia zu erarbeiten und umzusetzen.

1.2 Aufbau der Arbeit

Zunächst werden in Kapitel 2 drei unterschiedliche Ansätze für einen Offline-Zugriff auf eine Webapplikation vorgestellt und hinsichtlich ihrer Eignung für Tricia bewertet. Untersucht werden eine Browser-Plugin basierte Offline-Funktionalität, die Offline-Möglichkeiten in HTML 5 und ein dateibasierter Offline-Zugriff. Der dateibasierte Offline-Zugriff wird schließlich für die Implementierung einer Offline-Funktionalität für Tricia gewählt.

Das Kapitel 3 erarbeitet ein Konzept, wie der dateibasierte Offline-Zugriff in Tricia integriert werden kann. Dabei werden zunächst Anforderungen, die Benutzer an eine Offline-Version von Tricia haben, erhoben. Anschließend wird über das Dateiformat und die Ordnerstruktur für die Offline-Dateien diskutiert. Nach dieser Klärung wird ein Entwurf für den Lese- und Schreibzugriff auf diesen Dateien erarbeitet. Anschließend wird ein Konzept für die Synchronisation von lokalen Änderungen mit der Webapplikation entwickelt.

In Kapitel 4 wird die Implementierung des in Kapitel 3 erarbeiteten Konzepts gezeigt. Dabei wird zunächst auf die schon bestehende Architektur von Tricia eingegangen. Anschließend wird gezeigt, wie die entworfene Ordnerstruktur in die bestehende Implementierung integriert werden kann. Daraufhin wird auf die Generierung der HTML-Dateien in dieser Ordnerstruktur eingegangen. Anschließend wird die Implementierung des lokalen Schreibzugriffs auf diese HTML-Dateien gezeigt. Zuletzt wird die Implementierung der Synchronisation von lokalen Änderungen auf den HTML-Dateien mit der Webapplikation erläutert.

Das letzte Kapitel 5 fasst die Ergebnisse dieser Arbeit noch einmal zusammen.

2 Ansätze für einen Offline-Zugriff auf Tricia

Wie schon in Kapitel 1 erwähnt, ist ein Offline-Zugriff auf Tricia ein wünschenswertes Feature. Deswegen beschäftigt sich Kapitel 2 damit, einen passenden Ansatz zu finden, mit welchem am besten eine Offline-Funktionalität in Tricia implementiert werden kann. Dazu wird der Ansatz einer Plugin basierten Offline-Funktionalität, die Offline-Möglichkeiten von HTML5 und ein dateibasierter Ansatz für einen Offline-Zugriff auf Webapplikationen im Folgendem näher erläutert. Darauf aufbauend wird eine Bewertung über die Verwendbarkeit des jeweiligen Ansatzes für eine konkrete Implementierung einer Offline-Funktionalität in Tricia vorgenommen.

2.1 Browser-Plugin basierte Offline-Funktionalität

Um Webapplikationen offlinefähig zu machen, bieten viele Hersteller Plugin basierte Lösungen an. Die drei meist benutzten Lösungen sind Adobe Flash[1], Java Applets[2] und Gears[3]. Adobe Flash und Java Applets kommen für die Offline-Funktionalität in Tricia nicht in Betracht, da diese einen anderen Use-Case haben. Das Front-End von Tricia besteht hauptsächlich aus HTML, JavaScript und CSS-Dateien. Dafür eignet sich von den drei genannten nur das Gears Plugin, welches stellvertretend für Plugin basierte Lösungen vorgestellt wird.

Gears ist ein Open-Source Projekt von dem Unternehmen Google, das 2007 noch unter dem Namen „Google Gears“ veröffentlicht wurde. Das Projekt steht unter der BSD-Lizenz. Gears enthält eine umfangreiche JavaScript API, um Webapplikationen ohne Internetverbindung lokal auf dem Client verfügbar zu machen. Damit der Benutzer auf Webapplikationen zugreifen kann, die Gebrauch von der Gears API machen, muss dieser die Gears Webbrowsererweiterung installieren. Diese Erweiterung steht für die gängigsten Browser auf den meist verbreitetsten Betriebssystemen zur Verfügung [CN10]. Dies ist zugleich der größte Nachteil von Plugin basierten Lösungen. Der Entwickler muss voraussetzen, dass der Endbenutzer das entsprechende Plugin in seinem Webbrowser installiert hat [Mah11].

Die zwei wichtigsten Bestandteile der Gears API für den Offline-Betrieb von Webapplikationen sind der lokale Server und die eingebaute SQLite[4] Datenbank. Entwickler können über eine Manifest-Datei Anwendungsressourcen (z.B. HTML, JavaScript, etc.) lokal speichern lassen, welche dann vom lokalen Webserver für den Benutzer im Offline-Modus bereitgestellt werden können [VVVE09]. Die lokale Datenbank dient zur Speicherung von Änderungen, die der Benutzer im Offline-Modus in der Anwendung erzeugt. Diese Änderungen können dann, sobald ein Internetzugang besteht, mit dem entfernten Webserver

[1] `http://www.adobe.com/de/products/flashplayer/`, aufgerufen am 20.April 2011
[2] `http://www.oracle.com/technetwork/java/index.html`, aufgerufen am 20. April 2011
[3] `http://gears.google.com/`, aufgerufen am 20. April 2011
[4] `http://www.sqlite.org/`, aufgerufen am 18. April 2011

synchronisiert werden.

Am 11. März 2011 verkündete Google das Aus für das Gears Projekt [Boo11]. Google wird keine neue Funktionalität in Gears einbauen und auch nicht die neuesten Internetbrowser-Generation unterstützen. Dies ist der nächste große Nachteil an Plugin basierten Lösungen. Die Entwickler von Webapplikationen sind darauf angewiesen, dass der Plugin Hersteller sein Produkt weiter pflegt [Mah11].

Die meisten Features von Gears wurden inzwischen in den HTML5 Standard aufgenommen, deswegen besteht in Zukunft keine Notwendigkeit mehr für Gears. Momentan wird noch keine Migration von Gears auf HTML5 angeboten. Aufgrund dieser Fakten ist Gears keine Alternative, die für eine Offline-Funktionalität in Tricia in Betracht kommt.

2.2 HTML5

Unter HTML5 versteht man heute nicht mehr nur die reine Spezifikation der Hypertext Markup Language. Vielmehr wurde HTML5 zu einem Sammelbegriff vieler Technologien für moderne Webanwendungen. Darunter befinden sich u.a. Multimedia Schnittstellen, CSS3 und Offline Speicherung. Die Offline Möglichkeiten von HTML5 werden in diesem Kapitel vorgestellt und darauf aufbauend eine Bewertung über den Einsatz von HTML5 für die Implementierung einer Offline-Funktionalität in Tricia vorgenommen.

Die Offline Möglichkeiten von HTML5 lassen sich in die zwei Bereiche Application Cache und Offline Storage einteilen. Der Application Cache dient vorrangig dafür, die Benutzeroberfläche ohne Internetverbindung darstellen zu können. Der Offline Storage unterstützt die Speicherung von Daten, welche der Benutzer im Offline-Betrieb der Applikation produziert. Zunächst geht dieses Kapitel auf den Application Cache ein, anschließend auf das Thema Offline Storage.

2.2.1 Application Cache

Der große Vorteil am Application Cache ist, dass man sehr einfach die Benutzeroberfläche einer Webapplikation im Offline-Betrieb zur Verfügung stellen kann. Dazu deklariert man in einer Manifest-Datei (siehe Quelltext 2.2) alle Webressourcen (z.B. HTML-Dateien, JavaScript-Dateien, CSS-Dateien, Bilder, ...) die offline verfügbar sein sollen. Unter dem Punkt `NETWORK` können explizit Dateien angegeben werden, welche nicht heruntergeladen werden sollen. Als dritte Option können unter `FALLBACK` Ausweich-Dateien angegeben werden, die im Fall, dass im Offline-Modus eine Ressource angefragt wird, welche nicht im Cache liegt, angezeigt werden. Die Manifest-Datei muss in einer HTML-Datei referenziert sein (siehe Quelltext 2.1). Alle HTML-Dateien, welche auf die Manifest-Datei referenzieren, werden automatisch gespeichert [WHA11].

Quelltext 2.1: Referenzierung einer Manifest-Datei

```
<html manifest="example.app cache">
  ...
</html>
```

Quelltext 2.2: Manifest-Datei

```
CACHE MANIFEST

CACHE:
index.html
help.html
style/default.css
images/logo.png
images/backgound.png

NETWORK:
login.php

FALLBACK:
*.html /offline.html
```

Die in der Manifest-Datei aufgelisteten Ressourcen werden bei dem erstmaligen Aufruf der HTML-Datei, die auf die Manifest-Datei referenziert, initial vom Browser in den Cache heruntergeladen. Wenn sich in einer Webressource eine Änderung ergibt, wird die Ressource nicht automatisch neu heruntergeladen. Der Cache wird nur aktualisiert, wenn sich bei einem erneuten Aufruf der HTML-Datei, die referenzierte Manifest-Datei selbst ändert [WHA11]. Außerdem besteht die Möglichkeit, den Cache über JavaScript Befehle zu aktualisieren. Bei beiden Methoden werden aber immer alle in der Manifest-Datei aufgeführten Dateien neu heruntergeladen [WHA11]. Damit kommen wir schon zu einem ersten Problem für Tricia. Um ein gesamtes Wiki lokal auf dem Client verfügbar zu halten, müssten sehr viele HTML-Dateien in den Offline Cache geladen werden. Bei jeder Änderung an einer Wikiseite müssten alle Dateien neu geladen werden, was zu sehr hohem Traffic führen könnte. Ein weiteres Problem ist die Größenrestriktion des Offline Cache auf 5 MB. Diese Grenze kann bei einem großen Wiki mit eingebetteten Bildern schnell erreicht werden. Der Application Cache wird noch nicht von allen modernen Browsern unterstützt, z.B. dem Internet Explorer 9 [Leb10]. Somit wäre die Offline-Nutzung einer Webapplikation auf bestimmte Benutzergruppen beschränkt.

2.2.2 Offline Storage

Der Offline Storage in HTML5 dient dazu, Daten, die auf dem Client im Offline-Betrieb erzeugt wurden, lokal auf dem Client zwischenzuspeichern, um diese später, wenn wieder eine Internetverbindung besteht, mit der Webapplikation zu synchronisieren. Im Bereich des Offline Storage gibt es drei Technologien, die potentiell für Tricia in Frage kommen, um Daten lokal zu speichern. Diese sind der Web Storage, die Web SQL Datenbank und die IndexedDB. Der Zugriff auf diese verschiedenen Speicher erfolgt jeweils durch eine JavaScript Schnittstelle.

Der Web Storage, auch Local Storage und DOM Storage genannt, ist ein einfacher Key-Value Speicher, der lokal auf dem Client angesiedelt ist [Mah11]. Es gibt zwei verschiedene Arten des Web Storage. Um die zwei Arten des Web Storage anzusprechen, bietet der HTML5 Standard die zwei JavaScript Objekte `localStorage` und `sessionStorage` an [Hog10]. Über diese zwei Objekte lassen sich zu selbst definierten Schlüsseln Werte in den Web Storage ablegen und wieder aus dem Speicher herausholen (siehe Quelltext 2.5). Der Unterschied zwischen diesen zwei Methoden liegt darin, dass der Inhalt des Session Storage nach Beenden der Browser-Session gelöscht wird. Der Inhalt des Local Storage bleibt

darüber hinaus gespeichert [LASS10]. Aufgrund dieser Eigenschaften ist der Web Storage vor allem für die Wiedererkennung von Benutzern einer Webapplikation gedacht und ist somit ein Ersatz für die sehr limitierten Cookies aus früheren HTML Versionen [Hog10]. Der Web Storage dient daher in sehr eingeschränkter Weise dazu, Webapplikationen offlinefähig zu machen.

Quelltext 2.3: Web Storage

```
sessionStorage.setItem('key','value');
sessionStorage.getItem('key');
```

Die Web SQL Datenbank in HTML5 ist eine vollständige relationale Datenbank und wird von den meisten Browsern als eine SQLite Datenbank implementiert [Pil10]. Über JavaScript kann man mit ganz normalen SQL-Statements auf diese Datenbank zugreifen (siehe Quelltext 2.5). Da auch hinter Tricia eine relationale Datenbank hängt, welche die Wikis, Wikiseiten, etc. abspeichert, würde sich die Web SQL Datenbank grundsätzlich anbieten, um Änderungen in Tricia im Offline-Modus zu speichern und diese Änderungen später mit der Remote-Datenbank zu synchronisieren. Allerdings ist auch die Web SQL Datenbank, wie der Application Cache, auf eine Größe von 5 MB begrenzt. Diese Restriktion könnte bei vielen Offline-Änderungen mit Einsatz von Mediendateien zu einem Kapazitätsproblem werden. Außerdem wird die Web SQL Datenbank nicht von allen neuen Browsern unterstützt und wird wahrscheinlich auch nicht in Zukunft als Technologie in alle Browser einziehen [LASS10]. Daher wird die Web SQL Datenbank auch nicht mehr im HTML5 Standard aufgeführt [Wor11]. Wegen dieser Negativpunkte enfällt diese Option für die Implementierung einer Offline-Funktionalität in Tricia.

Quelltext 2.4: Web SQL Datenbank

```
var db = null;
db = openDatabase('ToDo', '1.0', 'ToDo-Liste', 2 * 1024 * 1024);

db.transaction(function (tx) {
   tx.executeSql('CREATE TABLE IF NOT EXISTS notes (id INTEGER, note TEXT)');
   tx.executeSql('INSERT INTO notes (id, note) VALUES (0, "notiz")');
});

db.transaction(function (tx) {
   tx.executeSql('SELECT note FROM notes', [],
        function(tx, results) {
                var note = results.rows.item(0).note;
        });
});
```

Die IndexedDB soll einen Kompromiss zwischen Web Storage und Web SQL Datenbank darstellen [Mah11]. Es handelt sich hierbei ebenfalls um einen Key-Value Speicher (`objectStore`). Auf den Values lassen sich Indexe legen, um in den Datensätzen schneller und exakter suchen zu können. Über einen Zeiger (`cursor`) kann man die Datensätze aus dem Speicher auslesen (siehe Quelltext 2.5). IndexedDB wird noch von keinem Browser unterstützt [Leb10]. Deswegen ist die IndexedDB auch nicht geeignet für Tricia.

Quelltext 2.5: IndexedDB (Quelle: In Anlehnung an: [RW11])

```
//Öffnen und Befüllen einer Datenbank
var request = indexedDB.open('Notes', 'Notizbuch');
request.onsuccess = function(event) {
        var db = event.result;
        //Anlegen eines ObjectStore mit dem Key "notes";
        //Index über dem automatisch erstellten Feld "id"
        db.createObjectStore("notes","id",true);
        var objectStore = event.result.objectStore("notes");
        objectStore.add({note: "Notiz1"});
        objectStore.add({note: "Notiz2"});
};

//Einträge aus der Datenbank lesen
var request = indexedDB.open('Notes', 'Notizbuch');
request.onsuccess = function(event) {
        request = event.result.objectStore("notes").openCursor();
        request.onsuccess = function(event) {
                var cursor = event.result;
                var note = curser.value.note;
                console.log(note);
                cursor.continue();
        };
};
```

2.2.3 Fazit

Da die HTML5 Offline-Technologien noch nicht ganz ausgereift sind und noch kein einheitlicher Konsens zwischen den Browserherstellern über die angebotenen Schnittstellen besteht, ist HTML5 zum heutigen Zeitpunkt noch nicht für Tricia geeignet. Auch wenn in Zukunft ein Konsens der Hersteller bestehen sollte, ist eine Offline-Funktionalität für Tricia durch HTML5 schwer vorstellbar, da alle HTML5 Offline-Features über JavaScript angesprochen werden. Der JavaScript Code in Tricia verbindet das Front-End, bestehend aus HTML-Dateien und CSS-Dateien, mit der Back-End Serveranwendung, welche in Java geschrieben ist. Will man eine Offline-Funktionalität über HTML5 in Tricia integrieren, so müsste man Back-End Tätigkeiten, z.B. das Abspeichern von Wikiseiten in den JavaScript Code implementieren. Dadurch müsste der bisher bestehende JavaScript Code von Tricia auf die zwei Fälle Online- und Offline-Betrieb angepasst werden. Daraus folgt eine starke Erhöhung der Komplexität des bisher bestehenden JavaScript Codes und eine Aufweichung der Trennung von Front- und Back-End Code. Das verursacht eine hohe Fehleranfälligkeit und einen hohen Wartungsaufwand des Quelltextes. Aus den genannten Gründen wird HTML5 nicht für die Implementierung einer Offline-Funktionalität in Tricia verwendet.

2.3 Dateibasierter Offline-Zugriff

Die Idee eines dateibasierten Offline-Zugriffs auf eine Webapplikation ist ein neuer Ansatz, der versucht, die Nachteile vorher genannter Technologien für eine Implementierung in Tricia zu umgehen.

Die grundsätzliche Idee eines dateibasierten Offline-Zugriffs besteht darin, dass die für einen Benutzer wichtigen Daten einer Webapplikation lokal auf dem Rechner als Dateien

abgespeichert werden. Auf die lokal abgelegten Dateien kann dann, ohne dass eine Verbindung zum Server besteht, auf dem die Webapplikation läuft, über weitere Programme zugegriffen werden, um die Inhalte betrachten zu können. Desweiteren kann auch eine Möglichkeit geschaffen werden, den Inhalt der Dateien zu ändern. Sobald wieder eine Internetverbindung besteht, muss es einen Rückkanal zur Webapplikation geben, damit diese Veränderungen synchronisiert werden können. Abbildung 2.1 verdeutlicht dieses Konzept. Der Benutzer kann sich bei Schritt ① die wichtigsten Daten der Webapplikation in Dateiform auf seine Festplatte herunterladen. Bei Schritt ② greift der Client über lokal installierte Programme, z.B. Internetbrowser oder lokaler Webserver auf diese Dateien zu, um diese anzusehen oder Veränderungen an diesen vorzunehmen. Bei Schritt ③ werden die lokal veränderten Dateien mit dem Webserver synchronisiert.

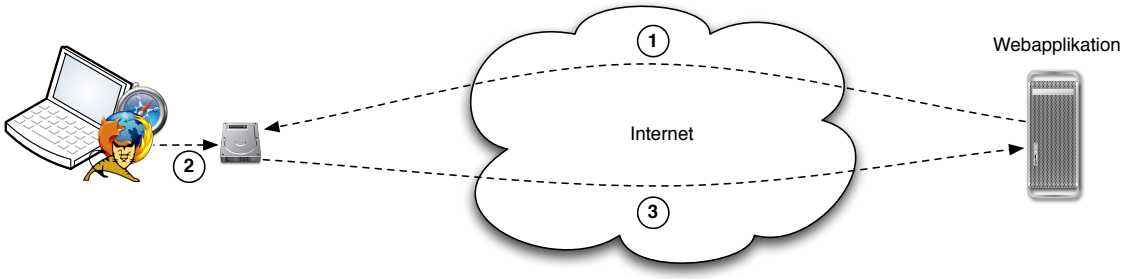


Abbildung 2.1: Übersicht eines dateibasierten Offline-Zugriffs auf eine Webapplikation

Diese Strategie eignet sich vor allem für Webapplikationen, die vorrangig aus mehreren statischen Seiten bestehen und deren Inhalt textueller Natur ist. Darunter fallen also vor allem Wiki- und Content-Management-Systeme und damit auch Tricia als eine Enterprise 2.0 Plattform. Im Fall von Tricia bestehen die wichtigen Daten aus dem Inhalt von Wikiseiten.

Der Nachteil einer dateibasierten Lösung für Tricia ist, dass alle Daten in Tricia in einer relationalen Datenbank abgelegt werden und somit zunächst nicht in Dateiform vorliegen. Diese Umwandlung von relational abgelegten Daten in Dateien muss neu implementiert werden. Der Nachteil der Neu-Implementation erweist sich aber schnell als Vorteil. Im Gegensatz zu den in den vorherigen Kapiteln vorgestellten Ansätzen müssen kaum Änderungen an dem bisher vorliegenden Quelltext vorgenommen werden. Der Offline-Zugriff kann aufbauend auf der bisher bestehenden Plugin-Architektur von Tricia realisiert werden. Eine genaue technische Betrachtung dieser Plugin-Architektur erfolgt in Kapitel 4.1.

Ein weiterer Nachteil ist, dass bei einem dateibasierten Ansatz eine Benutzerschnittstelle geschaffen werden muss, mit welcher ein Benutzer von Tricia die Dateien herunterladen und bei lokalen Änderungen die Dateien auch wieder mit dem Webserver synchronisieren kann. Dafür bietet sich bei Tricia die vorhandene Schnittstelle zu einem virtuellen Dateisystem an. Tricia-User können über eine Samba-Verbindung ein virtuelles Laufwerk der Tricia Installation auf ihrem lokalen Rechner einbinden. In diesem Laufwerk werden alle Dateien, z.B. Dateianhänge einer Wikiseite gelistet, auf die der jeweilige Nutzer Zugriff hat. Diese Schnittstelle lässt sich auch ideal nutzen, um die für den Offline-Zugriff auf Tricia notwendigen Dateien anzubieten, die sich der Benutzer auf seine lokale Festplatte

kopieren kann. Das Laufwerk ist auch als Synchronisations-Kanal geeignet. Über bereits vorhandene Synchronisationstools von Drittherstellern, z.B. DirSync Pro[5] können die lokal veränderten Daten mit den Daten auf dem virtuellen Laufwerk synchronisiert werden.

Aufgrund dieser Vorraussetzungen und der großen Vorteile wird für einen Offline-Zugriff auf Tricia ein dateibasierter Ansatz gewählt. Die Anpassung des dateibasierten Ansatzes für die Umsetzung einer Offline-Funktionalität in Tricia erfolgt in Kapitel 3. Die Implementierung dieses Konzepts in Tricia wird in Kapitel 4 erläutert.

[5] `http://www.dirsyncpro.org/`, aufgerufen am 28. April 2011

3 Konzeption einer dateibasierten Lösung zum Offline-Zugriff auf Tricia

In Kapitel 2 wurde gezeigt, dass ein dateibasierter Ansatz für die Implementierung einer Offline-Funktionalität in Tricia der geeignete Ansatz ist. In diesem Kapitel wird ein Konzept entwickelt, wie der dateibasierte Offline-Zugriff in Tricia integriert werden kann.

Dazu werden zunächst diejenigen Features der Webapplikation Tricia identifiziert, die auch im Offline-Modus verfügbar sein sollen. Anschließend wird ein Grundkonzept entwickelt, wie das in Kapitel 2.3 schon angesprochene virtuelle Samba-Laufwerk als Schnittstelle zwischen dem Offline- und Online-Modus von Tricia fungieren soll. In diesem Zusammenhang wird diskutiert, welches Format die Dateien haben sollen, die sich der Benutzer beim dateibasierten Ansatz herunterladen und wieder mit der Serveranwendung synchronisieren kann. Die Wahl des Dateiformats ist abhängig von den davor identifizierten Features, die in die Offline-Funktionalität von Tricia implementiert werden sollen. Das Dateiformat hat einen entscheidenden Einfluss auf die Komplexität der gesamten Implementierung. Nach der Entscheidung über das Format der Dateien wird ein Konzept entwickelt, in welcher Weise diese Dateien in einer Ordnerstruktur auf dem virtuellen Samba-Laufwerk abgelegt werden. Auf Basis des Dateiformates wird anschließend erläutert, wie der Benutzer diese Dateien lokal verändern kann. Zum Schluss wird gezeigt, wie diese lokal editierten Dateien wieder mit der Tricia-Serveranwendung synchronisiert werden können.

3.1 Offline-Features

Zunächst muss man sich überlegen, welche Funktionalität von Tricia auch in der Offline-Variante verfügbar sein soll. Da der Offline-Modus von Tricia nicht in Konkurrenz zur Webanwendung stehen soll und möglichst nur dann Anwendung findet, wenn keine Internetverbindung besteht, soll nur ein Teil der Funktionalität in die Offline-Variante einziehen. Dazu werden zunächst die Hauptfeatures der Webanwendung von Tricia erläutert und aus Sicht der Anwender priorisiert.

Wie schon in Kapitel 1 erwähnt, ist Tricia eine Enterprise 2.0 Plattform, die sich flexibel um Plugins erweitern lässt. Den Kern von Tricia bilden allerdings die hybriden Wikis. In hybriden Wikis lassen sich unstrukturierte Inhalte, z.B. Texte, Bilder, Dateien, etc. sowie strukturierte Inhalte in Form von Formularen anlegen [inf11a].

Abbildung 3.1 zeigt die typischen Bestandteile einer hybriden Wikiseite in Tricia. Im Rahmen von ① sieht man den Inhalt der Wikiseite (unstrukturierter Teil). Der Inhalt einer Wikiseite lässt sich mit Hilfe des WYSIWYG-Editor TinyMCE[1] bearbeiten. Dabei lassen sich auch Verlinkungen auf andere Wikiseiten anlegen. Zu den einzelnen Wikiseiten

[1] `http://tinymce.moxiecode.com/`, aufgerufen am 20. Mai 2011

können Dateien, sogenannte Attachments (siehe ③), angehängt werden. Auch auf diese Attachments kann man innerhalb des Inhalts der Wikiseite referenzieren.

Im Rahmen von ② sieht man den strukturierten Teil einer hybriden Wikiseite. Zu Wikiseiten des gleichen Typs lassen sich Typen deklarieren. Zu diesen Typen können Attribute definiert und auf den Wikiseiten mit Werten belegt werden. Zu den Attributen können zusätzlich Konsistenzregeln definiert werden, welche die zulässige Wertemenge einschränken [inf11a]. Dieses Feature nennt sich in Tricia *Hybrid Table*.

Bei ④ sieht man die Suchfunktion von Tricia. Mit der Suchfunktion kann der Benutzer über die Eingabe des Titels der Wikiseite schnell auf die entsprechende Wikiseite springen. Außerdem lässt sich eine Volltextsuche durchführen.

Innerhalb des Rahmens von ⑤ lassen sich Tags vergeben, welche den Inhalt der Wikiseite möglichst gut beschreiben. Bei ⑥ kann man eine Historie über die verschiedenen Versionen der hybriden Wikiseite einsehen und ältere Versionen wiederherstellen. Wird die Kommentarfunktion für eine Wikiseite erlaubt, kann man unter ⑦ Kommentare zu einer Wikiseite hinterlassen. Bei ⑧ werden Meta-Informationen über den letzten Bearbeiter und den letzten Bearbeitungszeitpunkt der Wikiseite angezeigt.

Abbildung 3.1: Features von hybriden Wikis in Tricia

Die Wichtigkeit der einzelnen Features bei der Online-Anwendung ist durch die Nummerierung in Abbildung 3.1 dargestellt. Oberste Priorität hat das Anlegen, Editieren und Anzeigen von Wikiseiten. Durch die Verknüpfung der Wikiseiten mit den strukturierten Formulardaten und Dateianhängen lassen sich Informationen noch besser aufbereiten und miteinander verknüpfen. Das Anlegen und die Anzeige von Tags hilft Informationen wiederzufinden. Versionshistorie, Kommentare und Meta-Informationen zu Wikiseiten sind nützlich, aber nicht essenziell für das Ablegen von Informationen in Tricia.

Auf der Basis der vorgestellten Features wird in den nächsten beiden Kapiteln die Funktionalität herausgearbeitet, welche die Offline-Version von Tricia jeweils für den Lese- sowie Schreibzugriff bieten soll.

3.1.1 Lesezugriff

Das wichtigste für einen Offline-Zugriff auf Tricia ist zunächst der Lesezugriff. Wenn keine Internetverbindung besteht, ist man zunächst daran interessiert, die Informationen, die in Tricia hinterlegt sind, trotzdem einsehen zu können. Das Erzeugen und Ablegen von neuen Informationen kann im Notfall auch über Umwege geschehen. Zum Beispiel, indem man die Informationen zunächst in einer lokalen Textdatei ablegt und anschließend, wenn wieder eine Internetverbindung besteht, in die Tricia Plattform überträgt. Von daher hat zunächst die Implementierung eines Offline-Lesezugriffs höhere Priorität als der Offline-Schreibzugriff auf Tricia.

Es stellt sich die Frage, welche der im vorherigen Kapitel vorgestellten Features der hybriden Wikis den Benutzer beim reinen Lesezugriff besonders interessiert. Das primäre Interesse gilt natürlich dem Inhalt der hybriden Wikiseiten. Die Verknüpfung des Inhalts mit den strukturierten Formular-Daten (*Hybrid Table*) und Dateianhängen bilden den Kern der Tricia-Anwendung. Diese drei Features müssen dem Benutzer im Offline-Betrieb für den Lesezugriff zugänglich gemacht werden. Ein wichtiges Feature in Tricia ist außerdem das Anlegen von Verlinkungen auf Dateien und andere Wikiseiten. Beim Offline-Lesezugriff auf Tricia ist es daher auch wünschenswert, wenn diese Verlinkungen weiterhin bestehen und nachverfolgt werden können. Die Benutzung der Suchfunktion hilft dem Benutzer beim Finden von Inhalten und sollte daher auch in der Offline-Version implementiert werden. Das Einsehen von auf den Wikiseiten angelegten Tags und Kommentaren der Nutzer bildet einen zusätzlichen Mehrwert beim Betrachten der Wikiseiten. Diese beiden Features sollten, wenn es keinen zu hohen Aufwand erfordert, ebenfalls implementiert werden. Die Versionshistorie und die Meta-Informationen sind nicht so wichtig, können aber implementiert werden. Die vorgenommene Kategorisierung der Features wird in Tabelle 3.1 zusammengefasst.

3.1.2 Schreibzugriff

Wenn man gerade keine Internetverbindung hat, ist man nicht vorrangig daran interessiert, ganze Wikiseiten neu anzulegen oder bestehende Wikiseiten zu editieren. Vielmehr möchte man eine Möglichkeit haben, Ideen, die einem z.B. beim Betrachten einer Wikiseite einfallen, als Notiz zu der jeweiligen Wikiseite ablegen zu können. Diese Notizen werden dann, wenn wieder eine Verbindung zum Internet besteht, mit der Webanwendung synchronisiert. Diese Notizen können anschließend in der Webanwendung ausformuliert wer-

Priorität	Funktionalität
Hoch	Betrachten des Inhalts der Wikiseiten
	Betrachten der *Hybrid Table*
	Betrachten der Dateianhänge
	Verfolgen von Referenzen
Mittel	Suchfunktion
	Betrachten der Tags
	Betrachten der Kommentare
Niedrig	Betrachten der Versionshistorie
	Betrachten der Meta-Informationen

Tabelle 3.1: Feature-Kategorisierung für die Implementierung des Offline-Lesezugriffs

den und in den Text der bestehenden Wikiseite einfließen. Aus dieser Überlegung heraus hat die Funktion Notizen im Offline-Modus zu Wikiseiten anzulegen die höchste Priorität bei der Implementierung des Offline-Schreibzugriffs.

Das Editieren des textuellen Inhalts ganzer Wikiseiten hat, wie erläutert, nicht die höchste Priorität. Diese Funktionalität ist für die Implementierung des dateibasierten Ansatzes auch nicht ganz unproblematisch, da es bei gleichzeitigen Veränderungen einer Wikiseite im Offline-Betrieb und in der Webanwendung durch einen anderen Benutzer zu Konflikten kommt.

Das Editieren der anderen Features, z.B. Tags, *HybridTable* und Dateianhänge kann vernachlässigt werden, da dies für den Benutzer im Offline-Modus von Tricia keinen Anwendungsfall darstellt.

Die folgende Tabelle 3.2 fasst die Priorisierung dieser Features für die Implementierung des Offline-Schreibzugriffs noch einmal zusammen.

Priorität	Funktionalität
Hoch	Anlegen von Notizen zu Wikiseiten
Mittel	Editieren des Inhalts von Wikiseiten
	Anlegen neuer Wikiseiten
Niedrig	Anlegen/ Editieren von *HybridTables*
	Anlegen/ Editieren von Tags
	Anlegen/ Editieren von Dateianhängen
	Führen einer Offline-Versionshistorie
	Editieren der Meta-Informationen von Wikiseiten

Tabelle 3.2: Feature-Kategorisierung für die Implementierung des Offline-Schreibzugriffs

3.2 Grundkonzept

In Kapitel 2.3 wurde grundsätzlich das Konzept eines dateibasierten Ansatzes für den Offline-Zugriff auf eine Webapplikation erläutert. In diesem Kapitel wird versucht, dieses allgemeine Konzept auf eine Implementierung in Tricia zu übertragen. In Abbildung 3.2 ist die Grundidee eines dateibasierten Offline-Zugriffs auf Tricia zu sehen. Dieses Konzept wird nun näher erläutert.

Bei der Implementierung des in Kapitel 2.3 vorgestellten dateibasierten-Ansatzes muss eine Funktionalität geschaffen werden, mit der ein Benutzer die notwendigen Dateien herunterladen und diese Dateien wieder mit dem Server synchronisieren kann. Der Vorteil in Tricia ist, dass bereits eine Dateisicht auf Tricia besteht. D.h., Benutzer können in Tricia Ordnerhierarchien anlegen und Dateien darin ablegen.

In Kapitel 2.3 wurde auch schon erwähnt, dass in Tricia ein virtuelles SMB-Laufwerk implementiert ist. Dieses Laufwerk lässt sich vom Benutzer als Netzwerklaufwerk in sein Betriebssystem einbinden. Auf dem Laufwerk wird wiederum die Dateisicht von Tricia dargestellt. Das Laufwerk dient also als eine weitere Schnittstelle auf die Ordnerhierarchien, die in Tricia angelegt sind. Eine genaue technische Betrachtung, wie das SMB-Laufwerk implementiert ist und wie dieses mit der Dateisicht von Tricia zusammenspielt, wird in den Kapiteln 4.2 und 4.3 erläutert.

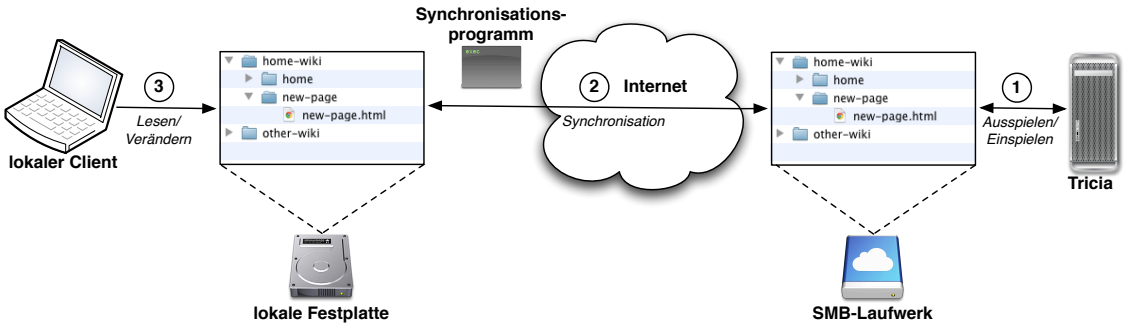


Abbildung 3.2: Grundkonzept eines dateibasierten Offline-Zugriffs auf Tricia

Dieses virtuelle Tricia-Netzwerklaufwerk soll als Schnittstelle zwischen Online- und Offline-Version von Tricia dienen. In Abbildung 3.2 stellt Tricia unter Punkt ① die notwendigen Dateien für die Offline-Version in dem SMB-Laufwerk für den Benutzer zur Verfügung. Die Dateien für den Offline-Modus sollen dabei nur auf dem SMB-Laufwerk und nicht in der Dateiansicht der Webapplikation zu sehen sein. Zunächst muss der Benutzer das SMB-Laufwerk auf seinem Computer einbinden. Der Benutzer kann sich nun diese Dateien mit Hilfe eines Synchronisationsprogramms eines Drittherstellers, z.B. DirSync Pro[2] auf seine lokale Festplatte synchronisieren (siehe Punkt ②). Dabei soll jede hybride Wikiseite einer eigenständigen Datei entsprechen. Auf dem SMB-Laufwerk sollen nur die Dateien der Wikiseiten zu sehen sein, auf denen der Benutzer mindestens Leserechte besitzt. Auf diese lokalen Dateien kann der Benutzer dann ohne Internetverbindung zugreifen, um deren Inhalt zu lesen oder zu verändern (siehe Punkt ③). In welcher Form die Daten der Wikiseiten in diesen Dateien hinterlegt sind und mit welchen

[2] `http://www.dirsyncpro.org/`, aufgerufen am 25. Mai 2011

Programmen man auf die Daten lokal zugreifen kann, wird in Kapitel 3.3 diskutiert. Das Kapitel 3.4 diskutiert, an welchen Stellen diese Dateien in die Ordnerstruktur auf dem SMB-Laufwerk eingespeist werden. Wenn Änderungen an den lokalen Dateien vorgenommen wurden, können, sobald wieder eine Internetverbindung besteht, die lokalen Dateien wieder mit den Dateien auf dem SMB-Laufwerk zurücksynchronisiert werden. Auf dem Server müssen dann die Veränderungen in das Live-System übertragen werden. Auf das Konzept der Synchronisation der Dateien und das Einspielen der lokalen Veränderungen in die Webapplikation wird detailliert in Kapitel 3.7 eingegangen.

3.3 Dateiformat

Wie schon in Kapitel 3.2 erwähnt, soll jede hybride Wikiseite einer eigenständigen Datei entsprechen. In diesem Kapitel wird diskutiert, in welchem Format die Daten der Wikiseiten in die jeweilige Datei geschrieben werden soll. Für das Format der offline gespeicherten Dateien von Tricia kann man sich zwei verschiedene Konzepte vorstellen.

Beim ersten Konzept werden die Daten in einem Format gespeichert, welche von einem Internetbrowser interpretiert werden können, z.B. HTML-Dateien.

Das zweite Konzept besteht aus einem lokal installierten Webserver, der auf die gespeicherten Dateien zugreift und diese als Webseiten ausliefert. Die Dateien enthalten die notwendigen Information im JSON- oder XML-Datenformat. Diese beiden Konzepte werden nun näher vorgestellt.

3.3.1 HTML-Dateien

Das Konzept der Speicherung jeder Wikiseite als HTML-Datei verdeutlicht Abbildung 3.3. Als erstes werden über das in diesem Kapitel vorgestellte virtuelle Laufwerk die Dateien von dem Server heruntergeladen (siehe Schritt ①). Die Dateien haben ein Format, welches von einem Webbrowser gelesen werden kann, z.B. HTML-Dateien. Der Benutzer kann also auf die heruntergeladenen Dateien über seinen Webbrowser zugreifen und sieht dadurch den Inhalt der jeweiligen Wikiseite (siehe Schritt ②). Will der Benutzer Änderungen an den Wikiseiten vornehmen, kann er über einen Text-Editor die Dateien öffnen (siehe Schritt ③). Die Synchronisation der Änderungen erfolgt wieder über das virtuelle Laufwerk (siehe Schritt ④).

Der Vorteil der Speicherung der Wikiseiten in HTML-Dateien ist, dass im Gegensatz zu dem im nächsten Kapitel 3.3.2 vorgestellten Konzept kein weiteres Programm implementiert werden muss, welches auf die Offline-Dateien zugreift und diese für den Benutzer aufbereitet. Die Speicherung der Wikiseiten als HTML-Dateien ist sehr einfach zu implementieren. Der Inhalt der Wikiseiten ist nämlich schon im HTML-Format in der Datenbank gespeichert. Um eine vollständig valide HTML-Datei zu erstellen, muss nur noch das Grundgerüst einer jeden HTML-Datei (Header, etc.) um den eigentlichen Inhalt herum erzeugt werden. Damit das Design der Offline-Wikiseiten dem Design der Online-Version entspricht, müsste man nur zusätzliche CSS-Dateien an den Benutzer ausliefern und in den jeweiligen HTML-Dateien referenzieren. Auch lassen sich Verlinkungen der einzelnen Wikiseiten untereinander sowie Verlinkungen auf Dateianhänge implementieren (siehe Kapitel 3.5.3).

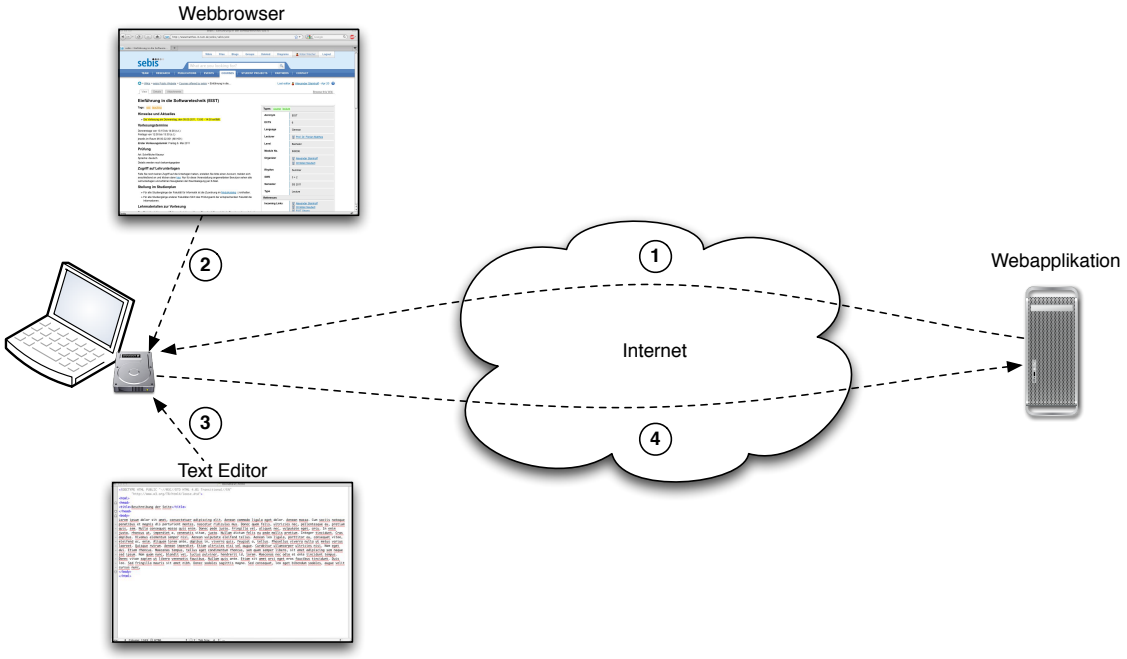


Abbildung 3.3: Dateibasierter Offline-Zugriff auf eine Webapplikation über lokal gespeicherte HTML-Dateien

Wie man sieht, kann man dieses Konzept sehr stark perfektionieren und damit zu einem optisch ansprechenden und benutzerfreundlichen Produkt für den Benutzer kommen.

Der Nachteil an diesem Konzept ist, dass der User nicht, wie er es von der Online-Version von Tricia gewöhnt ist, Wikiseiten verändern kann. Um Änderungen an Wikiseiten vornehmen zu können, müsste der Nutzer mit einem Texteditor den Quelltext der HTML-Dateien bearbeiten. Eine Lösung für dieses Problem wird in Kapitel 3.6 erarbeitet. Eine Suchfunktion, wie sie der Nutzer aus der Webanwendung gewohnt ist, lässt sich durch dieses Konzept auch nicht realisieren. Einen Ausgleich für diesen Nachteil wird in Kapitel 3.5.2 beschrieben.

Daher ist dieses Konzept vor allem dann geeignet, wenn der Hauptanwendungsfall ein Offline-Lesezugriff auf Tricia ist und auf eine komfortable Bearbeitung von Wikiseiten verzichtet werden kann.

3.3.2 Strukturierte Dateien

Der deutliche komplexere Ansatz wird in Abbildung 3.4 gezeigt. Dabei werden die Dateien nicht als HTML-Dateien, welche direkt vom Browser interpretiert werden können, ausgeliefert. Die Informationen der Wikiseiten werden in speziellen strukturierten Dateien (z.B. JSON- oder XML Dateien) gespeichert. Um die Dateien betrachten zu können, muss der Benutzer einen lokalen Webserver installieren, der mit den Informationen dieser Dateien umgehen und diese in einem benutzerfreundlichen Format dem Benutzer zur Verfügung stellen kann (siehe Schritt ②). Der lokale Webserver implementiert eine abgespeckte Variante der Tricia-Serveranwendung. Über den Browser kann dann die lokal

installierte Tricia-Anwendung (in Form des lokalen Webservers) benutzt werden (siehe Schritt ③). Über diese Anwendung können dann auch komfortabel Veränderungen an Wikiseiten vorgenommen werden. Die Synchronisation erfolgt wiederum über das virtuelle Laufwerk.

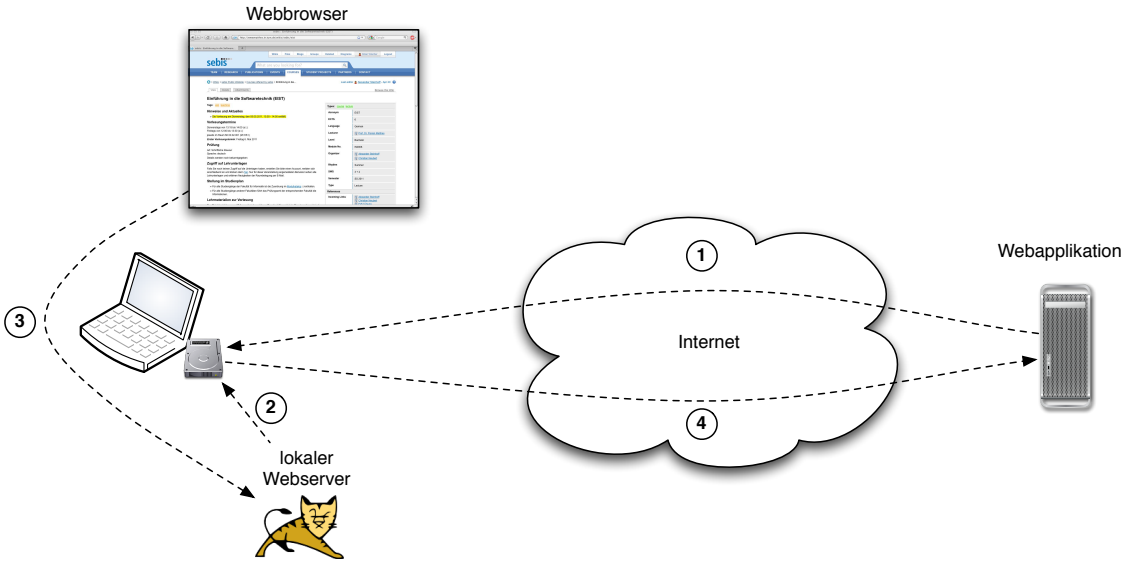


Abbildung 3.4: Dateibasierter Offline-Zugriff auf eine Webapplikation über lokal gespeicherte strukturierte Dateien

Der große Vorteil dieser Variante ist, dass durch die Zuhilfenahme eines lokalen Webservers sich mehr Features der Hauptanwendung integrieren lassen als bei dem Konzept der HTML-Dateien, z.B. eine integrierte Volltextsuche. Dadurch kann der Benutzer auf der Offline-Version von Tricia genau so arbeiten, wie er es von der Online-Version gewohnt ist.

Dieser lokale Webserver muss allerdings neu implementiert werden. Dadurch nimmt die Komplexität und der Aufwand der Implementierung einer Offline-Lösung enorm zu. Es ist schwer eine Grenze zu ziehen, welche Funktionalität der Online-Anwendung auch in die Offline-Version integriert werden soll und welche nicht. Bei Veränderungen der Hauptanwendung muss unter Umständen auch die Implementierung des lokalen Servers angepasst werden.

3.3.3 Fazit

Es stellt sich die Frage, ob die Komplexitäts- und Aufwandssteigerung des zweiten Konzepts, die durch die Implementierung des lokalen Webservers entsteht, in der richtigen Relation zum Nutzen für den Anwender steht. Die Offline-Variante von Tricia soll nicht in Konkurrenz zur Benutzung der Webapplikation stehen. Der primäre Einsatz der Offline-Variante ist nur dann gegeben, wenn keine Internetverbindung besteht. Dabei ist für den Benutzer vor allem der Lesezugriff interessant. Der Lesezugriff lässt sich durch das Konzept der HTML-Dateien, wie oben erläutert, zufriedenstellend lösen. Das lokale Editieren

von Wikiseiten lässt sich mit Hilfe eines JavaScript WYSIWYG-Editor (TinyMCE[3]), der auch bei der Webanwendung verwendet wird, bewerkstelligen. Dieses Konzept der lokalen Veränderungen der Dateien wird in Kapitel 3.6 erläutert.

Aufgrund dieser Abwägung wird für das Dateiformat das Konzept der HTML-Dateien gewählt.

3.4 Ordnerstruktur

Das vorherige Kapitel kam zu dem Ergebnis, dass jede Wikiseite als eine HTML-Datei für den Benutzer zur Verfügung stehen soll. In diesem Kapitel wird diskutiert, an welcher Stelle der Ordnerhierarchie auf dem SMB-Laufwerk diese Dateien angezeigt werden sollen. Da das SMB-Laufwerk die einzige Schnittstelle zwischen Online- und Offline-Modus von Tricia sein soll, sollen die HTML-Dateien nur auf dem SMB-Laufwerk dem Benutzer angezeigt werden und nicht in der Dateiansicht von der Tricia-Webanwendung.

Zunächst wird die Ausgangssituation der Ordnerhierarchie auf dem Laufwerk gezeigt. Anschließend werden drei Konzepte von Ordner-/Dateistrukturen gegeneinander abgewogen. Das erste Konzept sieht das Anlegen der Dateien auf dem SMB-Laufwerk im bereits bestehenden „Attachments“-Ordner vor. Beim zweiten Konzept wird eine unabhängige Ordner-/Dateistruktur an der Wurzel des SMB-Laufwerks eingerichtet. Das dritte Konzept bildet eine Mischform der ersten beiden Konzepte.

Verbindet sich der Benutzer zum SMB-Laufwerk von Tricia, sieht dieser die Ordner und Dateien, die von Benutzern in Tricia angelegt wurden sowie Ordner und Dateien, die automatisch von Tricia generiert werden. Fester Bestandteil des Wurzelverzeichnisses auf dem Laufwerk ist unter anderem der „Attachments“-Ordner. Das Wurzelverzeichnis des Laufwerks sowie der „Attachments“-Ordner ist in Abbildung 3.5 zu sehen. Im „Attachments“-Ordner werden die Dateianhänge der Wikiseiten nach Wiki-Instanzen und Wikiseiten aufgeschlüsselt und dem Benutzer angezeigt. Im Beispiel der Abbildung 3.5 gibt es die zwei Wiki-Instanzen „home-wiki“ und „new-wiki“. In der Instanz von „home-wiki“ befinden sich die zwei Wikiseiten „home“ und „konferenz“. In der Wikiseite „konferenz“ wurde eine Datei „Agenda.pdf“ angehängt. Auf diese Ausgangssituation beziehen sich nun auch die Beispiele der Konzepte, die nun vorgestellt werden.

Das erste Konzept sieht vor, die HTML-Dateien zusätzlich zu den Dateianhängen in die Ordner der jeweiligen Wikiseiten einzuspeisen (siehe Abbildung 3.6). Das bedeutet, dass im „Attachments“-Ordner zu jeder Wiki-Instanz ein Ordner existiert. In dem Ordner der Wiki-Instanz muss zu jeder Wikiseite ein Ordner existieren, auch wenn kein Dateianhang auf der Wikiseite abgelegt wurde. Momentan werden die Ordner der Wikiseiten auf dem SMB-Laufwerk erst angelegt, wenn ein Dateianhang zu der entsprechenden Wikiseite hochgeladen wurde. In diesen Ordnern der einzelnen Wikiseiten sollen nun die HTML-Dateien der jeweiligen Seite zusätzlich gelistet sein. In dem Beispiel in Abbildung 3.6 wird in dem „Attachments“-Ordner von der Wikiseite „konferenz“ neben dem Dateianhang „Agenda.pdf“ zusätzlich der Inhalt der Wikiseite als Datei „konferenz.html“ angezeigt. Der Vorteil dieser Variante ist, dass der Benutzer nur noch den Ordner der Wiki-Instanz aus dem „Attachments“-Order mit seiner lokalen Festplatte synchronisieren

[3] `http://tinymce.moxiecode.com/`, aufgerufen am 20. Mai 2011

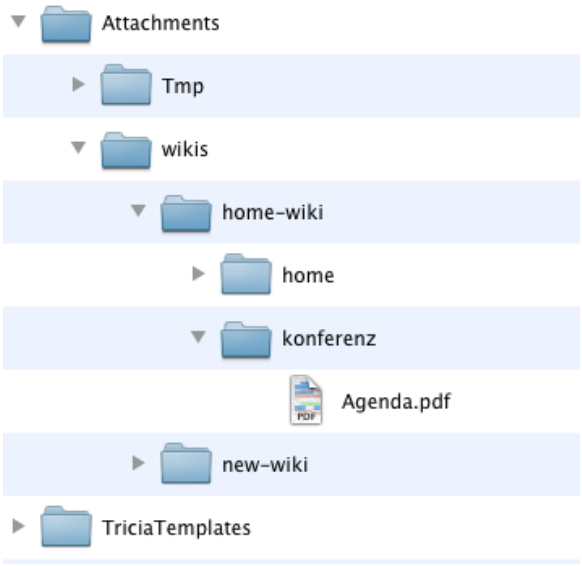


Abbildung 3.5: Initiale Ordnerhierarchie auf dem SMB-Laufwerk von Tricia

muss. Dabei erhält der Benutzer alle HTML-Dateien der einzelnen Seiten sowie die dazugehörigen Dateianhänge. Der Nachteil ist, dass Dateianhänge und die Dateien, die für den Offline-Modus vorgesehen sind, miteinander vermischt werden. Dies würde große Verwirrung bei den Benutzern hervorrufen. Das Konzept widerspricht außerdem dem in Kapitel 2.3 genannten Vorteil des dateibasierten Offline-Zugriffs, dass die Implementierung einer Offline-Funktionalität ohne große Veränderungen der bisherigen Webapplikation auskommt. Es müssten Veränderungen an der Implementierung des Anlegens von Dateianhängen zu Wikiseiten vorgenommen werden. Außerdem muss für die zusätzliche Anzeige der Offline-HTML-Dateien in dem „Attachments“-Ordner auf dem SMB-Laufwerk die Implementierung verändert werden.

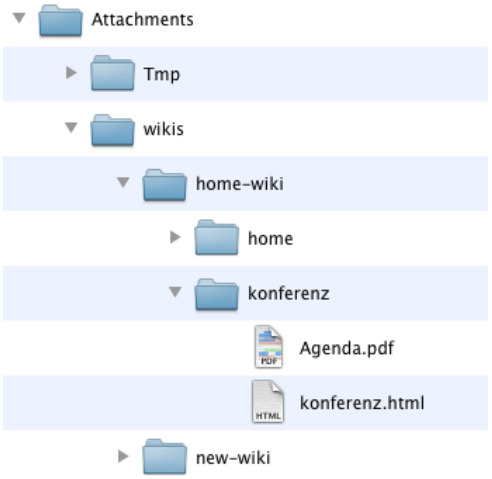


Abbildung 3.6: Konzept 1: Anzeigen der HTML-Dateien im „Attachments“-Ordner

Im zweiten Konzept wird im Wurzelverzeichnis ein weiterer Standard-Ordner mit dem Namen „Offline“ eingerichtet (siehe Abbildung 3.7). Unter diesem Ordner wird ähnlich wie im „Attachments“-Ordner zu jeder Wiki-Instanz ein Ordner angezeigt. In diesen Ordnern werden dann die HTML-Dateien der Wikiseiten aus dieser Instanz angezeigt. Der Offline-Ordner soll nur im SMB-Laufwerk angezeigt werden und nicht in der Dateiansicht der Tricia Webapplikation. Nachteil dieser Idee ist, dass der Benutzer, wenn er in der Offline-Version auch Zugriff auf die Dateianhänge haben will, die zwei Ordner „Attachments“ und „Offline“ synchronisieren muss. Bei der Benutzung eines Synchronisationsprogramms muss der Benutzer dabei aufwendige Einstellungen vornehmen. In der Regel will der Benutzer nur bestimmte Wikis offline verfügbar haben. Dazu müsste sich der Benutzer den Wiki-Instanz-Ordner aus dem „Offline“-Ordner lokal abspeichern. Zusätzlich muss der Benutzer den enstprechenden Wiki-Instanz-Ordner aus dem „Attachments“-Ordner an die richtige Stelle auf seinem lokalen Dateisystem ablegen, damit sichergestellt ist, dass Verlinkungen auf Dateianhänge in den HTML-Dateien der Wikiseiten nicht ins Leere führen (siehe Kapitel 3.5.3).

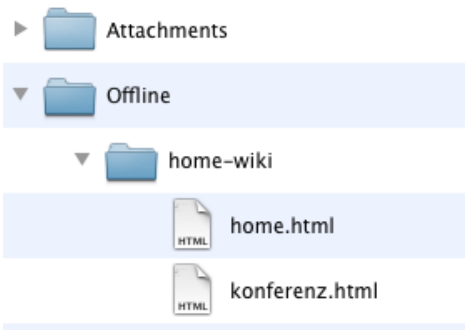


Abbildung 3.7: Konzept 2: Anzeigen der HTML-Dateien in neuer Ordnerstruktur

Einen guten Kompromiss zwischen den beiden Ansätzen bildet das dritte Konzept (siehe Abbildung 3.8). Für die Offline-Dateien wird eine eigenständige Ordnerstruktur auf dem SMB-Laufwerk eingerichtet („Offline“-Ordner). Unter dem „Offline-Ordner“ gibt es zu jeder Wiki-Instanz einen eigenen Ordner. In diesen Wiki-Instanz-Ordnern werden die HTML-Dateien der Wikiseiten in einem „Pages“-Ordner abgelegt. Die zum Wiki gehörenden Dateianhänge werden unterhalb des Wiki-Instanz-Ordners in einem „Attachments“-Ordner gelistet. Dateianhänge werden nun sowohl im „Attachments“-Ordner im Wurzelverzeichnis als auch im enstprechenden „Attachments“-Ordner im „Offline“-Ordner gelistet. Dies sollte aber zu keiner großen Verwirrung für den Benutzer führen, da die zusätzliche Ordnerstruktur explizit für die Offline-Nutzung deklariert ist. Der Vorteil an diesem Konzept ist, dass der Benutzer nur noch den Ordner der Wiki-Instanz, welche er offline verfügbar haben will, auf die lokale Festplatte kopieren und später wieder mit dem SMB-Laufwerk synchronisieren muss. Dabei erhält der Benutzer alle HTML-Dateien der Wikiseiten sowie die Dateianhänge auf diesen Wikiseiten. Diese fest vorgegebene Ordnerhierarchie macht es leichter, Referenzierungen von Offline-Wikiseiten auf die entsprechenden Dateianhänge zu implementieren (siehe Kapitel 3.5.3).

Da das dritte Konzept die Vorteile der beiden ersten Konzepte vereint und so gut wie keine Nachteile hat, wird dieses Konzept für die weitere Implementierung der Offline-

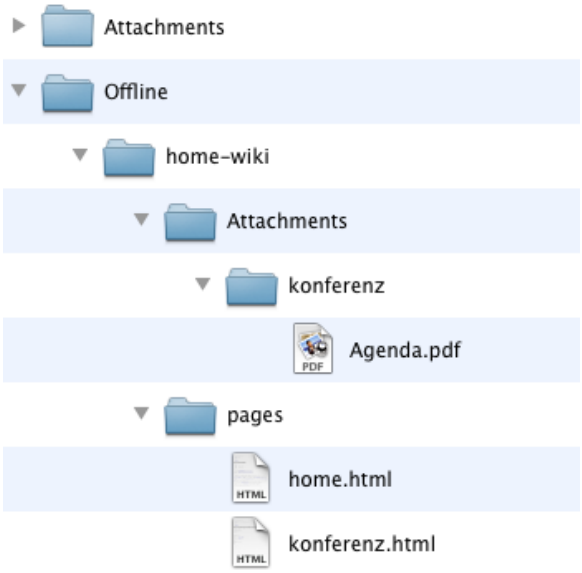


Abbildung 3.8: Konzept 3: Anzeigen der HTML-Dateien in neuer Ordnerstruktur mit zusätzlichen Dateianhängen

Funtionalität von Tricia gewählt. In den weiteren Kapiteln wird die vorgestellte Ordnerstruktur von Konzept 3 noch um weitere Ordner und Dateien erweitert. Das Grundgerüst bleibt bestehen.

3.5 Lokaler Lesezugriff

Kapitel 3.3 kommt zu dem Ergebnis, dass die Dateien für den Offline-Modus von Tricia als HTML-Dateien implementiert werden sollen. Kapitel 3.4 zeigt, wie diese HTML-Dateien in eine Ordnerstruktur auf dem SMB-Laufwerk von Tricia eingebettet werden sollen. In diesem Kapitel wird ein Konzept erarbeitet, wie die in Kapitel 3.1 identifizierten Features für den Offline-Lesezugriff auf Tricia unter Berücksichtigung des Dateiformats und der Ordnerstruktur umgesetzt werden können. Dabei geht das Kapitel zunächst auf den äußerlichen sowie inhaltlichen Aufbau der HTML-Dateien ein. Anschließend wird versucht, die fehlende Suchfunktion durch Index-Dateien zu ersetzen. Der letzte Abschnitt behandelt Referenzierungen in Wikiseiten. Dabei werden auch Probleme, die durch den Synchronisationsprozess entstehen, betrachtet.

3.5.1 Aufbau und Design

Die Offline-Variante von Tricia soll für den Benutzer keine zu große Umstellung in der Benutzung gegenüber der Online-Version darstellen. Wie schon in Kapitel 3.3 erwähnt, soll jede Wikiseite in Tricia einer HTML-Datei in der Offline-Version entsprechen. Deswegen soll der inhaltliche Aufbau und das äußerliche Design der Offline-Wikiseiten dem Design der Wikiseiten aus der Tricia Webanwendung entsprechen. Der typische Aufbau und das Design einer hybriden Wikiseite in Tricia ist in Abbildung 3.9 zu sehen. Im folgenden wird

auf die markierten Punkte der Abbildung 3.9 eingegangen und diese hinsichtlich einer möglichen Integration in die Offline-HTML-Dateien bewertet. Das Ergebnis für den Aufbau und das Design der Offline-HTML-Dateien zeigt Abbildung 3.10.

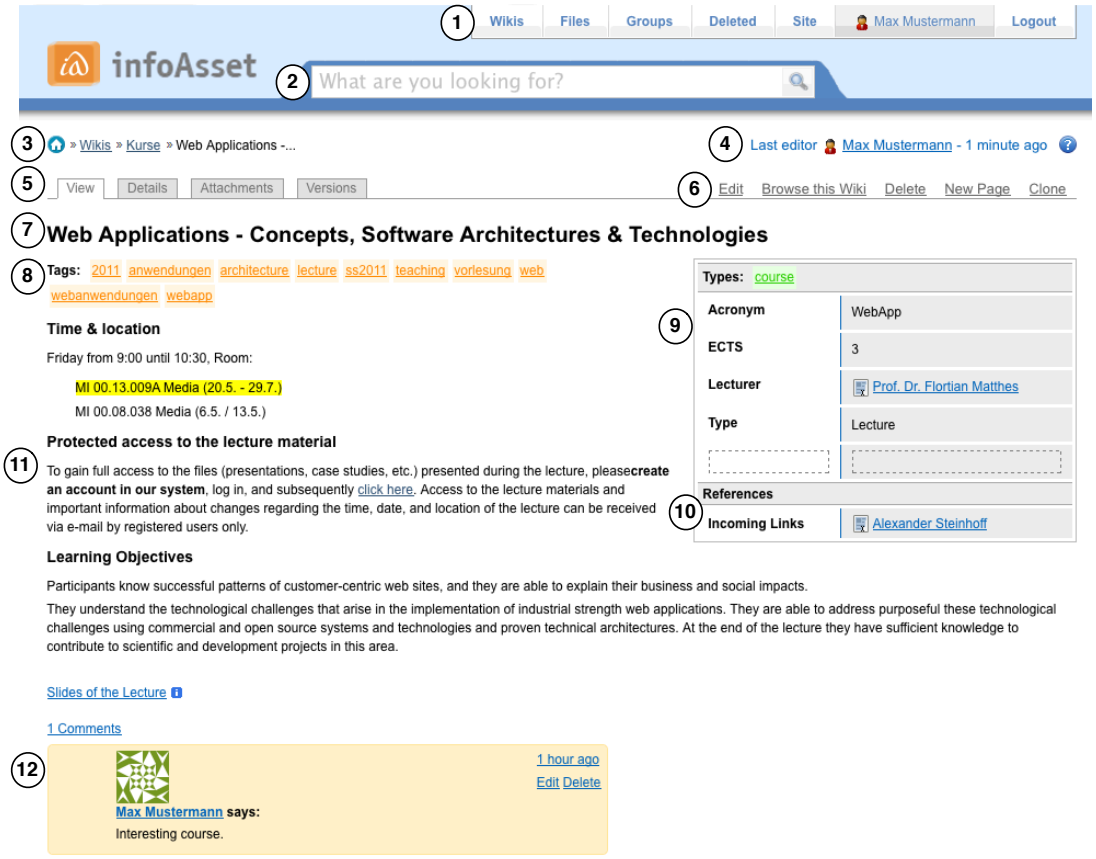


Abbildung 3.9: Aufbau und Design hybrider Wikiseiten in der Online-Version von Tricia

Bei ① ist die Navigationsleiste zu sehen. Hier kann der Benutzer schnell zwischen bestimmten Features von Tricia navigieren. Diese Navigationsleiste ist in den Offline-HTML-Dateien nicht notwendig, da dort die hybriden Wikiseiten im Vordergrund stehen und es keine anderen HTML-Seiten für andere Features geben wird.

Die Suchfunktion von Tricia sieht man bei ②. Diese Suchleiste verschwindet auch in den Offline-Dateien, da, wie in Kapitel 3.3 schon angesprochen, eine Suchfunktion mit dem Konzept der HTML-Dateien nicht zu implementieren ist. Eine Lösung dieses Problems wird in Kapitel 3.5.2 erläutert.

Bei ③ sieht man die Breadcrumb-Navigation. Hier hat der Benutzer einen schnellen Überblick, auf welcher Wikiseite und in welcher Wiki-Instanz er sich befindet. Diese Navigationsleiste soll auch in leicht abgeänderter Form in die HTML-Dateien integriert werden. Die Breadcrumb-Navigation der Offline-Dateien soll als Navigationspunkte eine Referenz auf die „Homepage“ der aktuellen Wiki-Instanz und den Titel der aktuellen Wikiseite enthalten.

Neben der Breadcrumbnavigation befinden sich bei ④ Metainformationen über den Au-

tor und den letzten Bearbeitungszeitpunkt einer Wikiseite. Diese Metainformationen sollen auch übernommen werden. Der letzte Bearbeitungszeitpunkt soll keine relative Zeitangabe, z.B. „18 minutes ago“, sondern von Anfang an einen festen Datumswert haben, z.B. „May 12“.

Bei ⑤ befinden sich Tabs, welche auf weitere Metainformationen von Wikiseiten, z.B. die Versionshistorie verweisen. Von diesen Tabs sollen nur noch der „View“-Tab und eventuell der „Attachments“-Tab in der Offline-Datei übrig bleiben. Wie in Kapitel 3.3 erläutert, lässt sich eine Revisionshistorie durch die Implementierung der Offline-Dateien im HTML-Format nur schwer bzw. gar nicht umsetzen. Deswegen wird dieser Tab genauso wie der „Details“-Tab in den HTML-Dateien nicht berücksichtigt.

Bei ⑥ befinden sich „Knöpfe“, um Veränderungen an der Wikiseite bzw. Wiki-Instanz vorzunehmen. In Tricia werden diese Features *Actions* genannt. Von diesen *Actions* soll in der Offline-Variante nur noch der „Browse this Wiki“-Knopf übrig bleiben. Über den „Browse this Wiki“-Knopf gelangt der Benutzer auf eine Übersichtsseite, auf der alle Wikiseiten der jeweiligen Wiki-Instanz aufgelistet werden. In der Offline-Variante soll der Benutzer über diesen Knopf auf die Index-Datei der Wiki-Instanz gelangen (siehe Kapitel 3.5.2). Zusätzlich soll auf den Offline-HTML-Dateien ein „Add Note“ Knopf integriert werden. Über diesen Knopf können Benutzer Notizen zu Wikiseiten hinzufügen. Ein Konzept für diesen Offline-Schreibzugriff wird in Kapitel 3.6 erarbeitet.

Die Überschrift einer Wikiseite sieht man bei ⑦. Diese soll unverändert in die HTML-Datei übernommen werden.

Unter der Überschrift werden bei ⑧ die *Tags* angezeigt, die Benutzer zu der Wikiseite angelegt haben. *Tags* sollen auch in der Offline-Version angezeigt werden, allerdings sollen die *Tags*, im Gegensatz zur Webanwendung, keine Hyperlinks auf die Suchfunktion haben.

Am rechten Rand befindet sich bei ⑨ der strukturierte Teil der hybriden Wikiseite. Diese Tabelle nennt sich in Tricia *Hybrid Table*. Die *Type Tags* sollen wie die normalen Tags angezeigt werden, aber keine Verlinkungen enthalten. Die Einträge in der *Hybrid Table* werden angezeigt und können auch in der Offline-Version Verlinkungen auf andere Wikiseiten, Dateianhänge, etc. enthalten.

Bei ⑩ werden Wikiseiten gelistet, welche auf die aktuelle Wikiseite referenzieren. Auch diese Information soll in die HTML-Dateien integriert werden.

Der eigentliche Inhalt der Wikiseiten ist bei ⑪ zu sehen. Dieser Inhalt soll eins zu eins in die HTML-Dateien übernommen werden. Der Text der Wikiseiten kann auch Verlinkungen auf andere Wikiseiten, Attachments, etc. enthalten. Das Thema der Referenzierungen wird ausführlich in Kapitel 3.5.3 behandelt.

Auch die Kommentare bei ⑫ sollen in die HTML-Datei integriert werden. Dabei muss allerdings auf das Profilbild des Kommentar-Autors verzichtet werden, da die Generierung dieses Profilbilds eine Internetverbindung erfordert. Die Zeitangabe soll, wie bei den Metainformationen bei ④ einen festen Datumswert besitzen.

Aus dieser Betrachtung der verschiedenen Aufbauelemente entsteht als Ergebnis der Entwurf für den äußerlichen Aufbau der Offline-HTML-Dateien in Abbildung 3.10.

Damit in den HTML-Dateien das Design, d.h. die örtliche Platzierung der einzelnen logischen Aufbauelemente, die Schriftarten und die Farben dem Design der Online-Version entsprechen, müssen zusätzlich zu den HTML-Dateien dem Benutzer noch CSS-Dateien, Icons, etc. zur Verfügung gestellt werden. Als CSS-Dateien können die original CSS-Dateien der Webapplikation von Tricia verwendet werden. Somit ist sichergestellt, dass

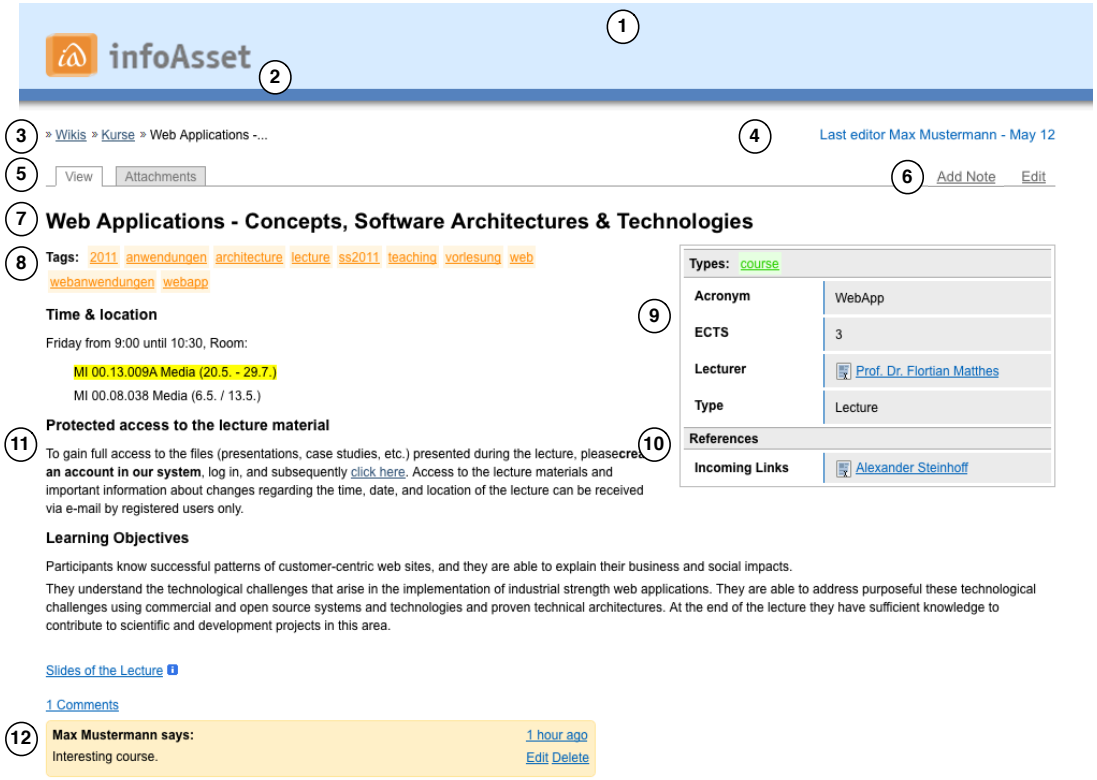


Abbildung 3.10: Entwurf des Aufbaus und Designs einer Offline-Wikiseite

Veränderungen am Design der Webapplikation auch automatisch in die Offline-Version übernommen werden. Diese „Design“-Dateien müssen dem Benutzer zusätzlich zu den HTML-Dateien auf dem SMB-Laufwerk angeboten werden. Dazu soll unterhalb eines Wiki-Instanz-Ordners in der Offline-Ordnerstruktur ein Ordner mit dem Namen „_metaData“ eingespeist werden (siehe Abbildung 3.11). In dem Ordner „_metaData“ werden dann die benötigten CSS-Dateien, Icons, etc. abgelegt. Durch diese Platzierung des „_metaData“-Ordners ist sichergestellt, dass der Benutzer beim Synchronisieren einer Wiki-Instanz sich immer die benötigten „Design“-Dateien auf die lokale Festplatte speichert.

3.5.2 Suche und Index-Dateien

Um die fehlende Suchfunktion (siehe Kapitel 3.3) zu ersetzen, soll zusätzlich zu den HTML-Dateien der Wikiseiten auch pro Wiki-Instanz eine Index-Datei auf dem Samba-Laufwerk angeboten werden. Die Index-Dateien sollen auch im HTML-Format ausgeliefert werden. Eine Index-Datei ist eine Art Inhaltsverzeichnis über die HTML-Dateien der Wikiseiten der entsprechenden Wiki-Instanz. In einer Index-Datei soll eine Liste über alle Wiki-Seiten, auf welche der Benutzer Leserechte besitzt, angezeigt werden. Die einzelnen Listeneinträge entsprechen den jeweiligen Wikiseiten und sollen einen Hyperlink auf die jeweilige HTML-Datei der Wikiseite enthalten. Das Thema der Referenzierungen wird in Kapitel 3.5.3 behandelt. Ein Beispiel einer solchen Index-Datei ist in Abbildung 3.12 zu sehen.

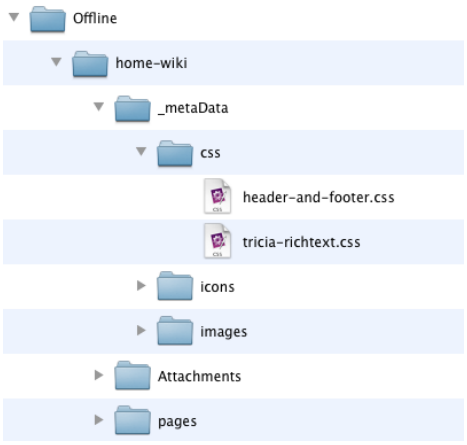


Abbildung 3.11: Meta-Dateien für das Design der HTML-Dateien

Abbildung 3.12: Index-Datei für eine Wiki-Instanz

In der Abbildung 3.12 gibt es eine Wiki-Instanz „Home Wiki“ mit den drei Wikiseiten „Home“, „Konferenz“ und „Workshop“. Die Index-Datei wird in den Ordner der Wiki-Instanz in der Ordnerstruktur abgelegt. In der Index-Datei sieht man die Auflistung der drei Wikiseiten mit der jeweiligen Referenz auf die entsprechende HTML-Datei.

Die Index-Dateien versuchen die Suchfunktion der Webanwendung zumindest ein bischen zu ersetzen. Sucht der Benutzer im Offline-Modus nach einer bestimmten Wikiseite, kann er in der jeweiligen Index-Datei manuell oder über die Volltext-Suchfunktion seines Internetbrowsers die entsprechende Seite suchen. Über die Referenz auf die entsprechende Seite gelangt der Nutzer schnell an das gewünschte Ziel.

Als weitere Suchmöglichkeit steht dem Benutzer auch die Suchfunktion seines Betriebssystems zur Verfügung. Die meisten Betriebssysteme bieten eine Volltextsuche über die Dateien auf dem lokalen Rechner an. Der Benutzer kann also auch über diese Funktion an die HTML-Datei der gesuchten Wikiseite gelangen.

3.5.3 Referenzierung

Im Inhalt einer Wikiseite sowie in der *Hybrid Table* kann man Referenzen auf *Assets* anlegen. In Tricia sind *Assets* Entitäten, die in einer Datenbank gespeichert werden. *Assets* sind z.B. Wikis, Wikiseiten, Personen, Dateianhänge, etc. So kann man z.B. in dem Text einer Wikiseite auf eine andere Wikiseite oder eine Datei referenzieren. Diese Referenzen erscheinen als Hyperlinks im Text der Wikiseite. Drückt der Benutzer auf diesen Hyperlink, gelangt er zur Ansicht der entsprechenden Wikiseite bzw. Datei. Durch das gewählte Dateiformat für die Wikiseiten aus Kapitel 3.3, die Ordnerstruktur aus Kapitel 3.4 und den Index-Dateien aus Kapitel 3.5.2 ergibt sich, dass in der Offline-Version von Tricia nur eine sinnvolle Ansicht auf die *Assets* „WikiPage", „Wiki", „Directory" und „Document" besteht.

Die Ansicht der Wikiseite entspricht der jeweiligen Offline-HTML-Datei, welche der Benutzer über den Browser öffnet. Für den Inhalt einer Wiki-Instanz gibt es die aus Kapitel 3.5.2 angesprochenen Index-Dateien. Die Ordner und Dateianhänge einer Wikiseite kann der Benutzer im „Attachments"-Ordner des lokal gespeicherten Wiki-Instanz-Ordner einsehen.

Verlinkungen, die in der Online-Version von Tricia auf die vier genannten Assets gemacht wurden, sollen auch in der Offline-Version existieren. D.h., wurde in der Webversion von Tricia von einer Wikiseite „x" auf eine andere Wikiseite „y" verlinkt, soll in der HTML-Datei von der Wikiseite „x" ein Hyperlink auf die entsprechende HTML-Datei der Wikiseite „y" verweisen. Eine Referenz, die auf einen Dateianhang oder einen Ordner verweist, soll in der Offline-Version auf die entsprechende Datei bzw. Ordner im „Attachments"-Ordner verweisen. Eine Verlinkung auf ein Wiki zeigt in der Offline-Version auf die entsprechende Index-Datei aus Kapitel 3.5.2.

Verlinkungen in der Webanwendung von Tricia sind absolute Links. Z.B. hat ein Link auf eine Wikiseite folgendes Format: *„/wikis/{wikiInstanzUrlName}/{wikiPageUrlName}"*. Die absoluten Pfadangaben dieser Verlinkungen müssen in der Offline Version in relative Pfade umgewandelt werden, da der Benutzer die Ordner für die Offline-Funktion an einen beliebigen Ort in seinem lokalen Dateisystem synchronisieren kann. In der Offline-Version gibt es daher keine absoluten Pfade. Man kann nur über relative Pfade an das gewünschte Ziel gelangen. Wird z.B. von einer Wikiseite auf eine andere Wikiseite referenziert, muss der Link in der Offline-Version von Tricia aufgrund der gewählten Ordnerhierarchie (siehe Kapitel 3.4) folgendes Format haben: *„../../{wikiInstanzUrlName}/pages/{wikiPageUrlName}.html"*. Vom Ort der aktuellen HTML-Datei der Wikiseite muss man zunächst zwei Hierarchiestufen nach oben, um dann in den entsprechenden Wiki-Instanz-Ordner zu gelangen. Von dort navigiert man in den „Pages"-Ordner, in dem die referenzierte Wikiseite als HTML-Datei vorliegt. Auf die Implementierung der Umwandlung dieser Hyperlinks geht Kapitel 4.4.1 ein.

Ein Problem ist, dass man nicht wissen kann, welche Wiki-Instanzen oder Dateianhänge sich der Benutzer auf seine lokale Festplatte synchronisiert hat. Deshalb kann es dazu kommen, dass Referenzen auf Dateien verweisen, die nicht existieren. Dies lässt sich aber nicht vermeiden. Folgt der Benutzer im Offline-Modus einem „kaputten" Link, wird der Internetbrowser dem Benutzer mitteilen, dass er die Datei nicht finden kann. Über die „Zurück"-Funktion des Internetbrowsers gelangt der Benutzer auf die Ansicht der Wikiseite, von der er auf den „kaputten" Hyperlink gedrückt hat.

3.6 Lokaler Schreibzugriff

In Kapitel 3.1.2 wurde herausgearbeitet, dass es für den Benutzer im Offline-Betrieb von Tricia am wichtigsten ist, Notizen zu Wikiseiten anlegen zu können. Die Konzeption des Offline-Schreibzugriffs beschränkt sich im weiteren auch auf diese Möglichkeit. Das Editieren ganzer Wikiseiten würde kein Problem für die Implementierung des Schreibzugriffs darstellen. Aber das anschließende Synchronisieren ganzer Wikiseiten steigert die Komplexität in einem zu hohen Maße, sodass der Nutzen nicht mehr in Relation zum Aufwand steht. Diese Problematik wird ausführlicher in Kapitel 3.7 angesprochen.

Die Grundidee des Schreibzugriffs ist, dass der Benutzer beim Betrachten der HTML-Datei einer Wikiseite die Möglichkeit hat, Informationen in textueller Form an die Wikiseite anzuhängen. Damit dieser Vorgang für den Benutzer interaktiv gestaltet werden kann, der Benutzer also das Gefühl hat, mit einer richtigen Webseite zu interagieren, muss in den HTML-Dateien zusätzlicher JavaScript Quelltext mit ausgeliefert werden. Durch JavaScript lässt sich ein im Browser ausführbarer Programmcode erstellen. Die nun beschriebenen Interaktionen auf den Offline-HTML-Dateien lassen sich durch JavaScript implementieren. Die technischen Details der Implementierung werden in Kapitel 4.5 ausgeführt.

Im vorherigen Kapitel 3.5.1 wurde gezeigt, dass es einen „Add Note“ Button geben soll. Dieser Button ist zum Anlegen einer Notiz gedacht. Drückt der Benutzer auf diesen Button, soll ein Textfenster im unteren Bereich der Seite erscheinen, in welchem der Benutzer eine Notiz verfassen kann. Das Verfassen einer Offline-Notiz soll für den Benutzer ähnlich funktionieren wie das Anlegen/Editieren eines Wikitextes in der Webapplikation. Deswegen soll als Texteditor der in der Webanwendung eingesetzte Javascript WYSIWYG-Editor TinyMCE[4] verwendet werden. Das Design der beiden Editoren im Offline- und Online-Betrieb von Tricia soll möglichst gleich sein. Dem Benutzer soll es aber beim Verfassen einer Offline-Notiz nur möglich sein, formatierten Text anlegen zu können. Verlinkungen auf andere Wikiseiten oder Dateien würden die Komplexität beim Schreibzugriff sowie der Synchronisation zu stark erhöhen. Daher soll der TinyMCE-Editor nur Bedienelemente enthalten, um den eingegebenen Text zu formatieren (Fettdruck, Schriftfarbe, Aufzählungen, ...). Abbildung 3.13 zeigt einen Entwurf, wie der TinyMCE-Editor auf der HTML-Seite angezeigt werden soll.

Hat der Benutzer seine Notiz fertig, soll er über den „Save“ Knopf die Notiz speichern können.

Um den TinyMCE-Editor in die HTML-Datei zu integrieren, müssen im Meta-Dateien-Ordner (siehe Kapitel 3.5.1) die dafür notwendigen Dateien abgelegt werden.

Das Speichern einer Offline-Notiz orientiert sich an dem Konzept der Wiki-Software TiddlyWiki[5]. TiddlyWiki ist eine Wiki-Software, die nur aus einer einzelnen HTML-Datei mit zusätzlichem JavaScript besteht. Legt man Wikiseiten in TiddlyWiki an, wird über JavaScript die HTML-Datei eingelesen und mit dem alten Inhalt und dem Inhalt der neuen Wikiseite überschrieben. Das Einlesen und Schreiben von lokalen Dateien über Javascript funktioniert nur in den Internetbrowsern Firefox und Internet Explorer. Aufgrund des hohen Marktanteils dieser beiden Browser stellt dies keinen großen Nachteil dar. Wird über JavaScript auf die lokale Festplatte zugegriffen, fragt der Internetbrowser den Benutzer, ob

[4] `http://tinymce.moxiecode.com/`, aufgerufen am 24. Juli 2011
[5] `http://www.tiddlywiki.com/`, aufgerufen am 17. Juli 2011

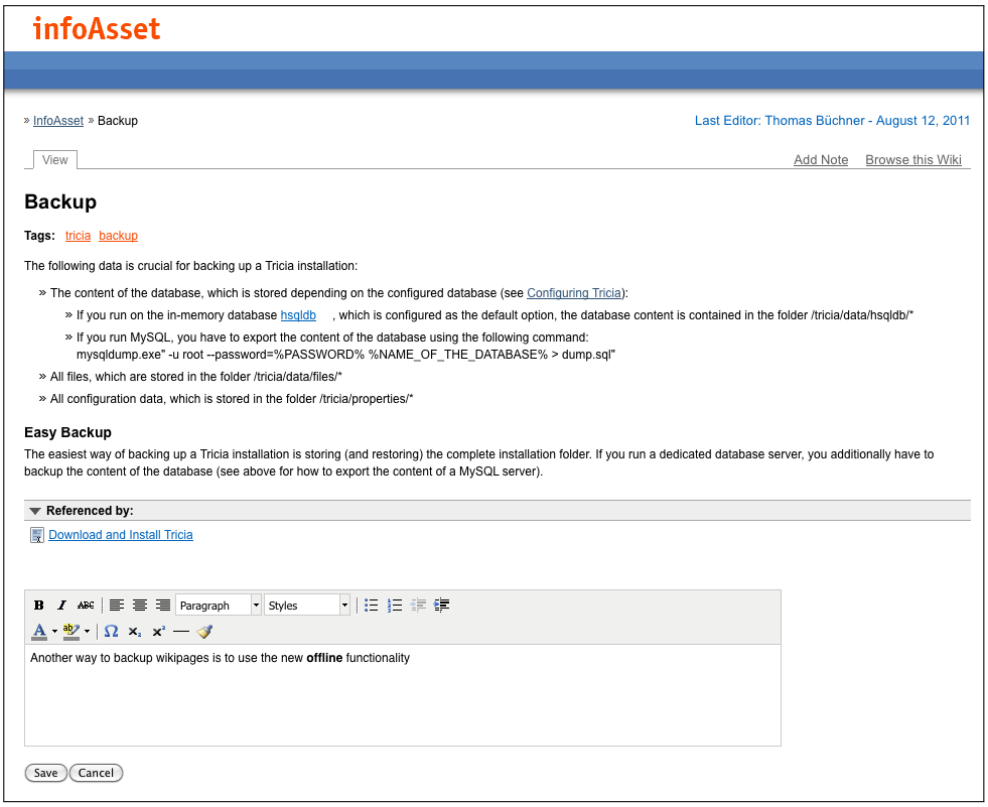


Abbildung 3.13: Schreiben einer Offline-Notiz

er diese Operation zulassen will. Dies muss der Benutzer gestatten, ansonsten funktioniert das Einlesen und Abspeichern der neuen HTML-Seite nicht.

Beim Speichern einer Offline-Notiz soll der Text an das Seitenende der HTML-Datei der Wikiseite angehängt werden. Dazu muss die HTML-Datei zunächst über JavaScript eingelesen und anschließend mit dem neuen HTML-Quelltext neu geschrieben werden. Zusätzlich zur HTML-Datei der Wikiseite soll noch eine JSON-Datei geschrieben werden, welche später mit der Webanwendung von Tricia synchronisiert wird. Auf das Konzept, welches hinter dieser Datei liegt, geht das nächste Kapitel 3.7 ein. Der TinyMCE-Editor soll nach dem Speichern wieder verschwinden. Einen Entwurf für das Aussehen einer Offline-Notiz in den Offline-HTML-Dateien ist in Abbildung 3.14 zu sehen.

Räumlich getrennt werden soll der Offline-Kommentar vom Rest der Wikiseite durch einen horizontalen Strich. Unter diesem Strich steht dann der Inhalt der geschriebenen Offline-Notiz. Will der Benutzer seiner Notiz noch etwas hinzufügen oder sie verändern, kann er wieder auf den „Add Note“ Knopf drücken. Die schon vorhandene Notiz im Inhalt der Wikiseite soll verschwinden und im Textfeld des TinyMCE-Editors zum Editieren erscheinen.

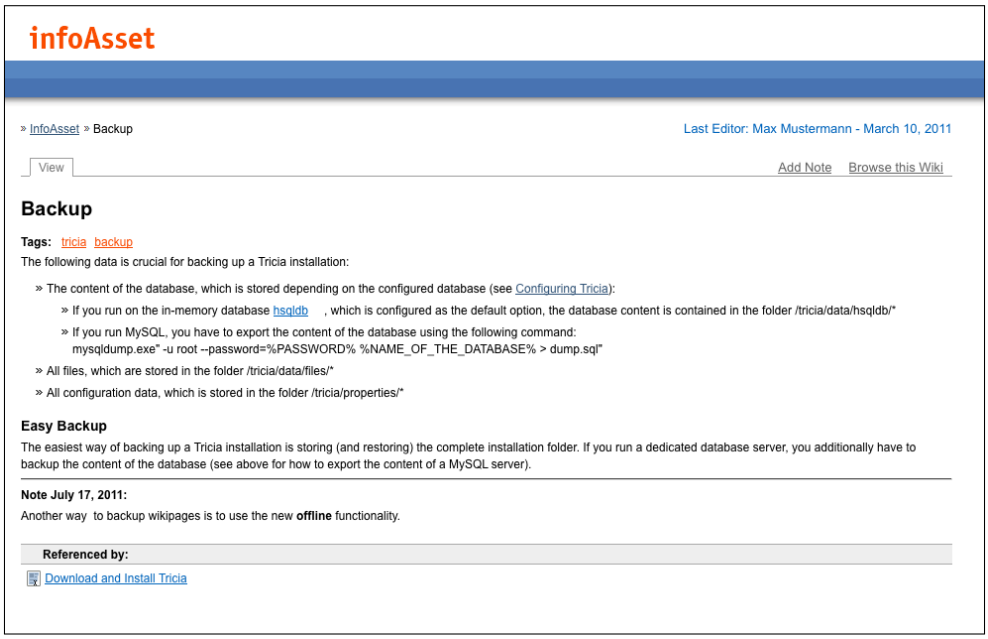


Abbildung 3.14: Ansicht einer Offline-Notiz

3.7 Synchronisation

In diesem Kapitel wird ein Konzept für das Zusammenspiel von der Online- und Offline-Version von Tricia erarbeitet.

Das Grundkonzept des dateibasierten Ansatzes für die Offline-Funktionalität in Tricia aus Kapitel 3.2 hat gezeigt, dass das schon implementierte SMB-Laufwerk als Schnittstelle zwischen Online- und Offline-Betrieb von Tricia dienen soll. Das Kapitel 3.3 kam zu dem Ergebnis, dass auf diesem Laufwerk für jede Wikiseite eine HTML-Datei angeboten werden soll. Darauf folgend wurde in Kapitel 3.4 eine Ordnerstruktur entwickelt, in welche die HTML-Dateien abgelegt werden. Diese Ordnerstruktur wird in diesem Kapitel um einen „Sync“-Ordner erweitert. Der lokale Lesezugriff auf die HTML-Dateien der Wikiseiten wurde in Kapitel 3.5 erläutert. In Kapitel 3.6 wurde gezeigt, wie man lokale Veränderungen auf den HTML-Dateien der Wikiseiten vornehmen kann. Dabei wurde auch schon Rücksicht auf Probleme genommen, die bei der Synchronisation auftreten können.

Will der Benutzer erstmalig die Offline-Funktionalität von Tricia benutzen, muss er sich mit dem SMB-Laufwerk verbinden. Die Ordner und Dateien der einzelnen Wikis befinden sich im „Offline“-Ordner. Pro Wiki-Instanz befindet sich ein Ordner in diesem „Offline“-Ordner (siehe Kapitel 3.4). Nun kann sich der Benutzer die Wikis, welche er offline zur Verfügung haben will, auf die lokale Festplatte kopieren. Dieses Kapitel zeigt nun ein Konzept, wie die Synchronisation zwischen dem lokal kopierten Ordner einer Wiki-Instanz und dem entsprechenden Ordner auf dem SMB-Laufwerk funktionieren kann.

Im ersten Teil wird gezeigt, warum eine beidseitige Synchronisation („Two-Way“-Synchronisation) des lokalen Ordners mit dem Ordner auf dem SMB-Laufwerk nicht funktioniert. Anschließend wird ein Konzept („Two-Step“-Synchronisation) gezeigt, welches die

Problematik, die bei der „Two-Way“-Synchronisation entsteht, löst.

3.7.1 „Two-Way“-Synchronisation

Die erste Idee für die Synchronisation war, dass der Benutzer ein Synchronisationsprogramm verwendet und dieses Programm so einrichtet, dass der lokale Ordner der Wiki-Instanz mit dem Ordner der Wiki-Instanz auf dem SMB-Laufwerk beidseitig synchronisiert wird. D.h., wurde eine Veränderung auf einer Wikiseite in der Webapplikation vorgenommen, wird die entsprechende HTML-Datei beim Synchronisieren auf das lokale Laufwerk kopiert. Wurde eine lokale Veränderung auf einer Wikiseite vorgenommen, wird diese HTML-Datei auf das SMB-Laufwerk synchronisiert und auf dem Server in die Datenbank eingespielt. Dieses Konzept würde es auch zulassen, ganze Wikiseiten im Offline-Modus zu editieren und diese mit der Webapplikation zu synchronisieren. Ein großes Problem entsteht nun aber, wenn eine Wikiseite nach der letzten Synchronisation im Offline-Betrieb und in der Webapplikation verändert wurde. Bei der nächsten Synchronisation würde es dann zu einem Konflikt darüber kommen, welche Version auf welches Laufwerk kopiert werden soll. Es gibt viele Ordner-Synchronisationsprogramme, die solche Konflikte erkennen. Aber es gibt kein bekanntes Programm, welches im Konfliktfall zwei Dateien manuell miteinander zu vereinen erlaubt. Bei den meisten Programmen muss der Benutzer wählen, welche Version der Datei für die Synchronisation hergenommen werden soll. Aufgrund dieser Problematik scheidet eine „Two-Way“-Synchronisation für Tricia aus.

3.7.2 „Two-Step“-Synchronisation

Um dieses Problem zu umgehen, wird für die Synchronisation in Tricia das Konzept der „Two-Step“-Synchronisation entwickelt. Dazu wird in die Ordnerhierarchie aus Kapitel 3.4 ein zusätzlicher „Sync“-Ordner eingeführt. Dieser Ordner befindet sich eine Ebene unter dem Wiki-Instanz-Ordner. D.h., jede Wiki-Instanz hat zusätzlich zum „Pages“-, „MetaData“- und „Attachments“-Ordner noch einen „Sync“-Ordner. In Kapitel 3.6 wurde für den Offline-Schreibzugriff ein Konzept entwickelt, welches vorsieht, dass Benutzer Notizen auf den HTML-Dateien der Wikiseiten ablegen können. Beim Speichern einer Offline-Notiz wird die HTML-Datei der Wikiseite neu geschrieben, damit die Notiz am unteren Rand der Wikiseite zu sehen ist. Für die „Two-Step“-Synchronisation muss zusätzlich eine Datei geschrieben werden, die alle Notizen, die im Offline-Betrieb seit der letzten Synchronisation gemacht wurden, speichert. Diese „Changeset“-Datei wird im vorher vorgestellten „Sync“-Ordner abgelegt. Bei der „Two-Step“-Synchronisation wird die Datei auf den Server kopiert und dort verarbeitet. Auf den Aufbau der Datei und die Verarbeitung auf dem Server geht das nächste Kapitel 3.7.3 ein.

Die „Two-Step“-Synchronisation sieht zwei Synchronisationsschritte vor. Diese sind in der Abbildung 3.15 schemenhaft dargestellt. Die beiden Synchronisationsschritte sind unidirektional. D.h., es wird nur in eine Richtung synchronisiert, z.B. von Ordner A nach Ordner B. Wird die gleiche Datei im Ordner A und B verändert, wird beim Synchronisieren die Version der Datei im Ordner B mit der Version aus Ordner A überschrieben. Wird eine Datei im Ordner A gelöscht, wird diese Datei bei der Synchronisation auch im Ordner B gelöscht. Im Endeffekt wird bei der unidirektionalen Synchronisation ein Ordner A auf einen Ordner B gespiegelt.

Die beiden nun vorgestellten Synchronisationsschritte lassen sich in den meisten Synchronisationsprogrammen einrichten. Durch das Testen vieler Synchronisationsprogramme konnte festgestellt werden, dass diese alle sehr unterschiedliche Algorithmen beim Abgleich von zwei Ordnern verwenden. D.h., die Sychronisationsprogramme führen sehr unterschiedliche Operationen auf dem Dateisystem aus. Optimiert wird die Implementierung für das Synchronisationsprogramm DirSync Pro[6], welches plattformübergreifend verfügbar ist und rsync[7], welches ein Standardtool auf Unix-/Linux-Betriebssystemen ist. Die Funktionsweisen dieser beiden Programme wird in Kapitel 4.6.2 erläutert.

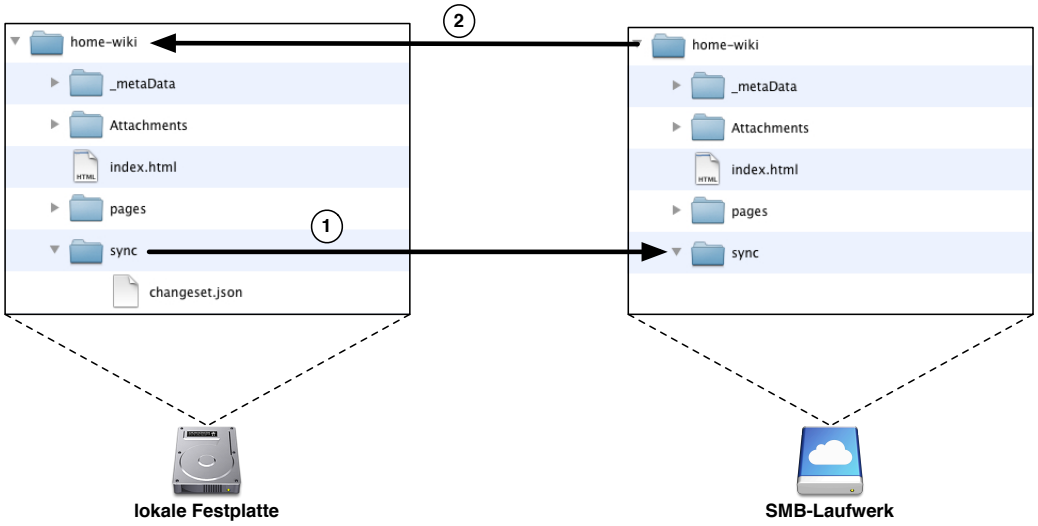


Abbildung 3.15: Konzept der „Two-Step"-Synchronisation

Der erste Synchronisationsschritt bei Tricia ist eine unidirektionale Synchronisation zwischen dem „Sync"-Ordner der jeweiligen Wiki-Instanz auf dem lokalen Laufwerk und dem „Sync"-Ordner der Wiki-Instanz auf dem SMB-Laufwerk (siehe Abbildung 3.15). Sollten also Offline-Notizen angelegt worden sein, würde die Datei „changeset.json" in den „Sync"-Ordner auf dem lokalen Laufwerk geschrieben werden. Auf dem SMB-Laufwerk ist der „Sync"-Ordner immer leer. Bei diesem Synchronisationsschritt wird diese Datei also in den „Sync"-Ordner auf dem SMB-Laufwerk kopiert. Die „Changeset"-Datei wird anschließend auf dem Server verarbeitet, was im nächsten Kapitel 3.7.3 erläutert wird. Bei der Verarbeitung werden alle Offline-Notizen in die Wikiseiten eingearbeitet.

Beim zweiten Schritt wird eine unidirektionale Synchronisation vom gesamten Wiki-Instanz-Ordner auf dem SMB-Laufwerk zum Wiki-Instanz-Ordner auf dem lokalen Laufwerk durchgeführt (siehe Abbildung 3.15). Dadurch erhält der Benutzer die HTML-Dateien der Wikiseiten, die in der Webanwendung durch andere Benutzer oder durch die Verarbeitung der „Changeset"-Datei aktualisiert wurden. Dabei werden alle lokal veränderten HTML-Dateien durch die HTML-Dateien auf dem SMB-Laufwerk überschrieben. Zudem wird die „Changeset"-Datei im „Sync"-Ordner auf dem lokalen Laufwerk automatisch gelöscht.

[6] `http://dirsyncpro.org/`, aufgerufen am 26. Juni 2011
[7] `http://rsync.samba.org/`, aufgerufen am 26. Juni 2011

Nun hat der Benutzer eine aktualisierte Offline-Version des gewählten Wikis und kann neue Offline-Notizen anlegen.

3.7.3 „Changeset"-Datei

Im ersten Synchronisationsschritt wird die Datei „changeset.json" auf das SMB-Laufwerk synchronisiert. In dieser Datei sollen alle Notizen stehen, die seit der letzten Synchronisation angelegt wurden. Als Dateiformat bietet sich das JSON-Format an, da die Notizen mit JavaScript gespeichert werden (siehe Kapitel 3.6). In dieser Datei soll sich ein Array aus JSON-Objekten befinden. Jedes JSON-Objekt entspricht einem Offline-Kommentar. Als Information muss in dem JSON-Objekt die „Id" der Wikiseite sowie der Text der Offline-Notiz gespeichert sein.

Bei der Synchronisation der „Changeset"-Datei mit dem SMB-Laufwerk soll diese JSON-Datei eingelesen werden. Anschließend sollen die Notizen an die jeweiligen Wikiseiten angehängt und in der Datenbank gespeichert werden. Dabei sollen die Offline-Notizen in der Webanwendung genau so aussehen wie auf den HTML-Dateien (siehe Abbildung 3.14).

Nun kann der Benutzer die Wikiseiten besuchen, auf denen er eine Notiz hinterlassen hat. Eine Notiz erscheint, getrennt durch einen horizontalen Strich, als ganz normaler Text am Ende der Wikiseite. Der Benutzer kann über das gewohnte Editieren von Wikiseiten den Text der Notiz in den Inhalt der Wikiseite einpflegen.

Durch das Konzept dieser „Changeset"-Datei im Zusammenspiel mit der „Two-Step"-Synchronisation ließe sich für den Offline-Schreibzugriff auch das Editieren ganzer Wikiseiten realisieren, denn ein Zusammenführen zweier Dateien müsste nicht durch ein Synchronisationsprogramm übernommen werden. Dabei müsste der gesamte Text einer geänderten Wikiseite in der „Changeset"-Datei gespeichert werden. Allerdings hätte man dann nach der Synchronisation das Problem, dass man beim Einlesen der „Changeset"-Datei mögliche Konflikte bei gleichzeitiger Änderung der Wikiseite in der Webapplikation auf dem Server erkennen und auflösen müsste. Da die Konflikterkennung sowie das Zusammenführen der Wikitexte nur mit sehr viel Aufwand zu implementieren wäre, beschränkt man sich beim Schreibzugriff und der Synchronisation auf die wichtigste Anforderung: das Anlegen von Offline-Notizen (siehe auch Kapitel 3.1.2). Durch diese Einschränkung umgeht man die Problematik der Konflikterkennung und -auflösung, denn die Notizen werden einfach immer an das Ende einer Wikiseite gehängt.

4 Realisierung einer dateibasierten Lösung zum Offline-Zugriff auf Tricia

In diesem Kapitel wird gezeigt, wie das in Kapitel 3 erarbeitete Konzept für einen dateibasierten Offline-Zugriff auf Tricia implementiert wird.

Dazu wird zunächst die Plugin-Architektur von Tricia vorgestellt und erläutert, wie die neu implementierte Offline-Funktionalität in diese Architektur integriert wird. Als nächstes wird die vorhandene Implementierung des SMB-Laufwerks vorgestellt. Darauf aufbauend wird gezeigt, wie die in Kapitel 3.4 konzipierte Ordnerstruktur in diese Implementierung des SMB-Laufwerks integriert wird. Anschließend wird erläutert, wie die Dateien in dieser Ordnerstruktur generiert werden. Zum Schluss wird gezeigt, wie auf den HTML-Dateien der Wikiseiten ein Schreibzugriff für die Offline-Notizen realisiert wird. Der letzte Teil stellt die Implementierung der Synchronisation dieser Offline-Notizen mit der Webanwendung von Tricia vor.

4.1 Architektur von Tricia

Die Enterprise 2.0 Plattform Tricia ist modular aufgebaut [inf11c]. Die Plattform besteht aus mehreren Plugins, die aufeinander aufbauen. Die in Kapitel 3.1 identifizierten Features von Tricia, die auch im Offline-Betrieb zur Verfügung stehen sollen, werden in unterschiedlichen Plugins implementiert. Einen Ausschnitt der wichtigsten Plugins von Tricia zeigt Abbildung 4.1.

Die modulare Plugin-Architektur von Tricia erlaubt es, den Funktionsumfang in Tricia zu erhöhen, ohne große Veränderungen an der schon vorhandenen Implementierung vornehmen zu müssen. Dadurch ist es auch möglich, eine Offline-Funktionalität in Tricia zu integrieren, ohne die bestehende Implementierung groß verändern zu müssen. Dies wurde in Kapitel 2.3 auch als ein Hauptvorteil der Implementierung des dateibasierten Offline-Zugriffs gesehen.

Die Offline-Funktionalität soll daher als ein neues eigenes Plugin (*OfflineWiki*-Plugin) implementiert werden. Im Folgendem wird kurz auf jedes der in Abbildung 4.1 dargestellten Plugins eingegangen und deren Abhängigkeiten untereinander betrachtet.

Den Kern von Tricia bildet das *Core*-Plugin. Das *Core*-Plugin implementiert die Kernfunktionalität von Tricia. In diesem Plugin befindet sich auch die *Main*-Methode, die beim Starten von Tricia ausgeführt wird.

Das *File*-Plugin implementiert einen hierarchischen Dateispeicher. Durch das Plugin lassen sich Dateien und Ordner in Tricia anlegen. Das *File*-Plugin ermöglicht auch das Anhängen von Dateien an Wikiseiten.

Darauf aufbauend befindet sich das *Jlan2*-Plugin. Das Plugin implementiert ein virtuelles Samba-Laufwerk. Das *Jlan2*-Plugin zeigt den hierarchischen Dateispeicher aus dem

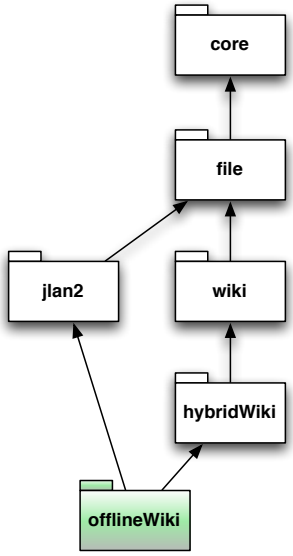


Abbildung 4.1: Plugin-Architektur von Tricia

File-Plugin in einem virtuellen Samba-Laufwerk an. Auf das Zusammenspiel des *File*- und *Jlan2*-Plugins geht das nächste Kapitel 4.2 ein.

Auf dem *Jlan2*-Plugin soll nun das neue *OfflineWiki*-Plugin aufbauen. Wie in Kapitel 3.2 erläutert, soll das Samba-Laufwerk als Austausch-Medium der Dateien für den Offline-Betrieb dienen. Desweiteren dient das Samba-Laufwerk als Rücksynchronisationskanal, um Veränderungen, die im Offline-Betrieb gemacht wurden, wieder in die Live-Instanz von Tricia einzuspielen.

Das *OfflineWiki*-Plugin baut desweiteren auf dem *Wiki*- und *HybridWiki*-Plugin auf. Diese beiden Plugins implementieren die hybriden Wikis in Tricia. Sie enthalten also die meisten Features, die in Kapitel 3.1 für den gewünschten Funktionsumfang der Offline-Variante von Tricia herausgearbeitet wurden.

Die Tabelle 4.1 gibt einen kurzen Überblick, welches Plugin welche Funktionalität in Tricia implementiert.

Plugin	**Funktionalität**
Core	Kernfunktionalität
File	hierarchischer Dateispeicher
Jlan2	Samba-Laufwerk
Wiki	Wiki-Funktionalität
HybridWiki	hybride Wikis
OfflineWiki	Offline-Funktionalität

Tabelle 4.1: Übersicht über die Features einiger Plugins von Tricia

4.2 Jlan2- und File-Plugin

Wie schon erwähnt, implementiert das *File*-Plugin eine hierarchische Dateistruktur. Benutzer können durch das *File*-Plugin Ordner und Dateien in Tricia anlegen. Auf diese hierarchische Ordnerstruktur kann nun entweder über das Webinterface oder über das virtuelle Samba-Laufwerk zugegriffen werden. Das virtuelle Laufwerk wird durch das *Jlan2*-Plugin implementiert. Diese Abhängigkeit der beiden Plugins ist auch in Abbildung 4.1 zu sehen.

Der hierarchische Datei-Speicher wird im *File*-Plugin durch ein Composite-Pattern aus den drei Assets `Path`, `Directory` und `Document` implementiert (siehe Abbildung 4.2). Ein `Path` kann entweder ein `Directory` oder ein `Document` sein. Ein `Directory` kann eine oder mehrere Instanzen der Klasse `Path` enthalten. Ein `Document` entspricht letztendlich einer echten Datei, welche in Tricia angelegt und heruntergeladen werden kann.

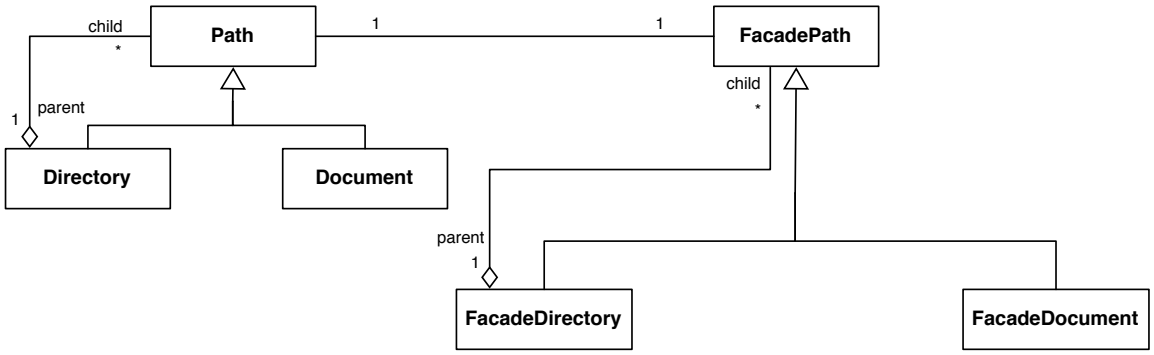


Abbildung 4.2: Aufbau des hierarchischen Datei-Speichers inklusive Fassaden-Klassen

Für den Zugriff auf diese Klassen aus dem *Jlan2*-Plugin implementiert das *File*-Plugin die drei Fassaden-Klassen `FacadePath`, `FacadeDirectory` und `FacadeDocument`. Diese drei Klassen sind das jeweilige Äquivalent zu den vorher vorgestellten Asset-Klassen. Die Klassen der Fassade bilden eine schlanke Schnittstelle auf die komplexe Implementierung der Assets des *File*-Plugins. Der Zusammenhang dieser Klassen ist ebenfalls in Abbildung 4.2 dargestellt.

Das *Jlan2*-Plugin verwendet das Alfresco Jlan Server Framework[1] für die Implementierung eines virtuellen Samba-Laufwerks. Einige der Klassen des *Jlan2*-Plugins für die Implementierung des Laufwerks sind in Abbildung 4.3 zu sehen. Um die Funktionalität des Frameworks zu nutzen, muss die vom Framework angebotene Schnittstelle `DiskInterface` implementiert werden. Das Framework übernimmt die ganzen Netzwerkprotokoll-Aktivitäten, die beim Zugriff eines Benutzers auf das Laufwerk entstehen. Die `DiskInterface` Schnittstelle sieht Methoden vor, die für die Interaktion mit dem virtuellen Dateisystem wichtig sind. Es muss z.B. eine Methode (`startSearch()`) implementiert werden, die beim Öffnen einer Datei oder eines Ordners ausgeführt wird.

Die Klasse `TriciaDiskDriverProxy` bzw. `TriciaDiskDriver` (siehe Abbildung 4.3) implementiert die von der Schnittstelle geforderten Methoden. Die Klasse `TriciaDiskDriver` ist indirekt gegen die Fassaden-Klassen implementiert. Ein Aufruf der Methode `startSearch()` impliziert über den Umweg der Klasse

[1] `http://www.alfresco.com/products/aifs/`, aufgerufen am 27. April 2011

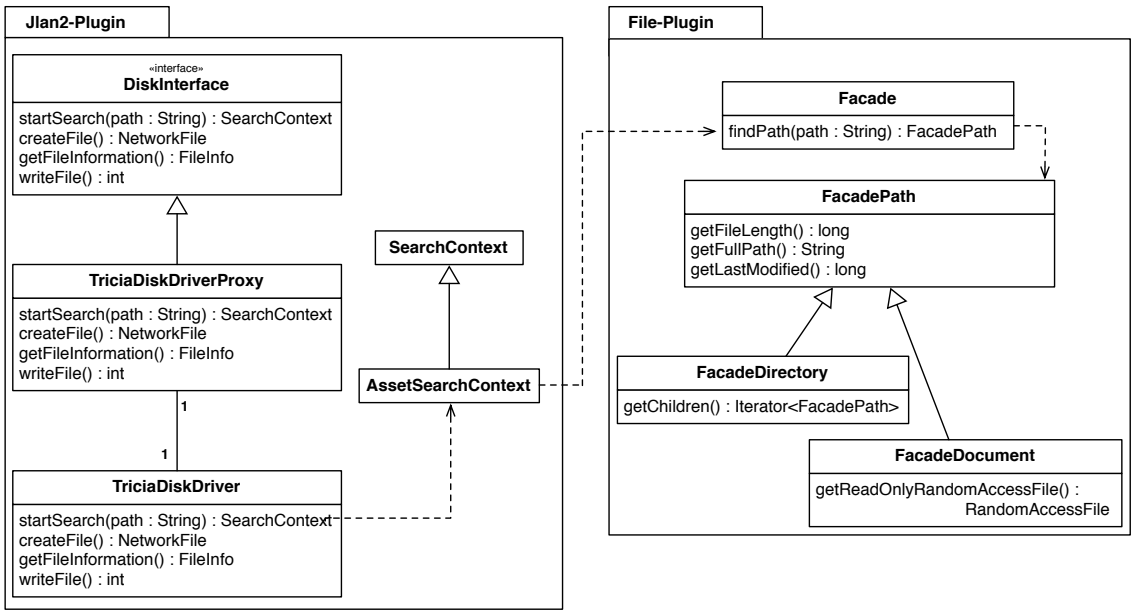


Abbildung 4.3: Klassen des *Jlan2*- und *File*-Plugins

`AssetSearchContext` einen Aufruf der statischen Methoden `findPath()` in der Klasse `Facade`. Die Methode `findPath()` initialisiert anhand des Pfades eine Instanz der Klasse `FacadeDirectory` bzw. `FacadeDocument`. Wird gerade ein Ordner auf dem Samba-Laufwerk geöffnet, wird auf der Instanz der Klasse `FacadeDirectory` die Methode `getChildren()` aufgerufen. Diese Methode liefert als Ergebnis einen Iterator über den aktuellen Ordnerinhalt. Dieser beschriebene Zusammenhang ist im Sequenzdiagramm in Abbildung 4.4 dargestellt. Als Beispiel wird in dem Sequenzdiagramm versucht, den „Attachments“-Order auf dem Samba-Laufwerk zu öffnen.

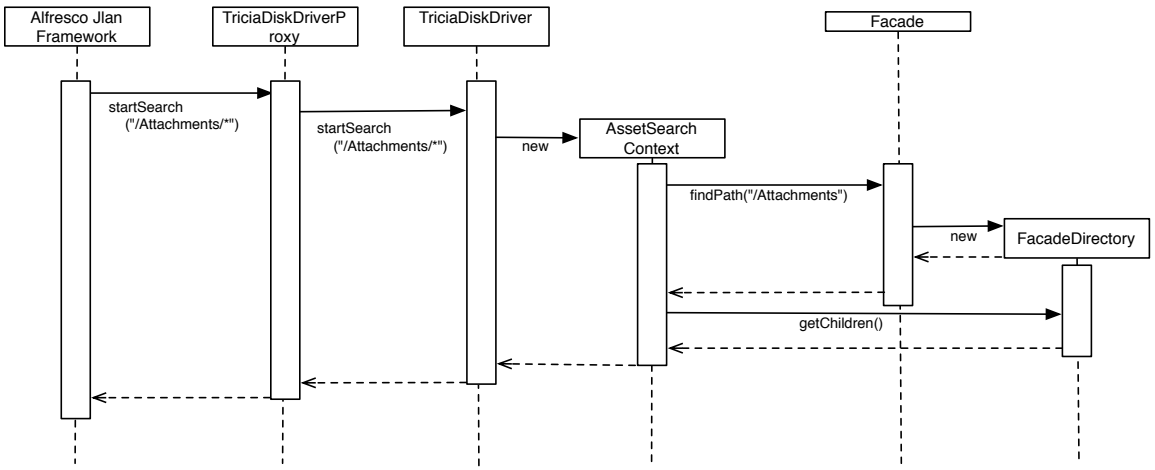


Abbildung 4.4: Sequenzdiagramm über das Zusammenspiel des *Jlan2*- und *File*-Plugins

Die drei Fassaden-Klassen implementieren Methoden, die für die Interaktion eines Be-

nutzers mit den Ordnern und Dateien auf dem Samba-Laufwerk notwendig sind.

Die Klasse `FacadePath` implementiert z.B. eine Methode (`getFileLength()`), die Auskunft über die Dateigröße eines Ordners bzw. einer Datei gibt. Einige Methoden dieser Klasse sind ebenfalls in Abbildung 4.3 zu sehen.

Die Klasse `FacadeDirectory` implementiert u.a. die Methode `getChildren()`. Diese Methode liefert als Rückgabewert einen Iterator über alle im zu öffnenden Ordner befindlichen Ordner und Dateien, also Instanzen der Klasse `FacadeDirectory` und `FacadeDocument`.

Um eine Datei auf dem Laufwerk öffnen zu können, implementiert die Klasse `FacadeDocument` u.a. die Methode `getReadOnlyRandomAccessFile()`. Diese Methode liefert als Rückgabewert eine Instanz der Klasse `RandomAccessFile`, welche vom Alfresco Framework für diese Aktion benötigt wird.

4.3 Samba-Laufwerk

In dem Kapitel 3.4 wurde ein Konzept entwickelt, wie die HTML-Dateien der Wikiseiten in eine Ordnerstruktur auf dem virtuellen Samba-Laufwerk eingebettet werden sollen. In diesem Kapitel wird gezeigt, wie diese Ordner-/Dateistruktur im Zusammenspiel mit dem *Jlan2*- und *File*-Plugin (siehe Kapitel 4.2) implementiert wurde. Die Offline-Funktionalität wird als ein neues Plugin (*OfflineWiki*-Plugin) implementiert.

Zunächst wird gezeigt, wie die Fassaden-Klassen des *File*-Plugins erweitert werden, um eine Ordnerstruktur für die Offline-HTML-Dateien zu implementieren.

Anschließend wird gezeigt, wie diese Ordnerstruktur in die bestehende Implementierung des Samba-Laufwerks integriert wird.

4.3.1 Ordner-/Dateistruktur

In Kapitel 4.2 wurde gezeigt, dass im *File*-Plugin die Fassaden-Klassen `FacadePath`, `FacadeDirectory` und `FacadeDocument` als Schnittstelle für den Zugriff aus dem *Jlan2*-Plugin auf die Asset-Klassen `Path`, `Directory` und `Document` implementiert sind (siehe Abbildung 4.2).

Für die gewünschte Ordner-/Dateistruktur für die Offline-Funktionalität aus Kapitel 3.4 müssen Subklassen von `FacadeDirectory` und `FacadeDocument` implementiert werden. Die Ordner und Dateien für den Offline-Zugriff haben keine Referenz auf Ordner und Dateien, die im Tricia-Dateisystem existieren. Denn die Ordner und Dateien für den Offline-Zugriff sollen gemäß Kapitel 3.2 nur auf dem Samba-Laufwerk dem Benutzer angeboten werden.

Die im *OfflineWiki*-Plugin zu implementierenden Subklassen von `FacadeDirectory` und `FacadeDocument` sind in Abbildung 4.5 zu sehen. Jeder Ordner und jeder Dateityp aus der Ordnerstruktur aus Kapitel 3.4 wird durch eine eigene Klasse repräsentiert. Für den „Offline“-Ordner wird die Klasse `OfflineDirectory` implementiert. Unterhalb des „Offline“-Ordners gibt es für jede Wiki-Instanz einen eigenen Ordner. Dieser wird repräsentiert durch die Klasse `WikiInstanceDirectory`. Unterhalb des Wiki-Instanz Ordners befinden sich der „MetaData“-Ordner, der „Pages“-Ordner und der Ordner für die Dateianhänge. Diese drei Ordner werden repräsentiert durch die Klassen

`MetaDataDirectory`, `PagesDirectory` und `OfflineAttachmentsDirectory`. Für die HTML-Dateien der Wikiseiten im „Pages“-Ordner sowie die Index-Datei wird die Subklasse `VirtualDocument` implementiert. Für die Meta-Dateien gibt es die Klasse `MetaDataDocument`.

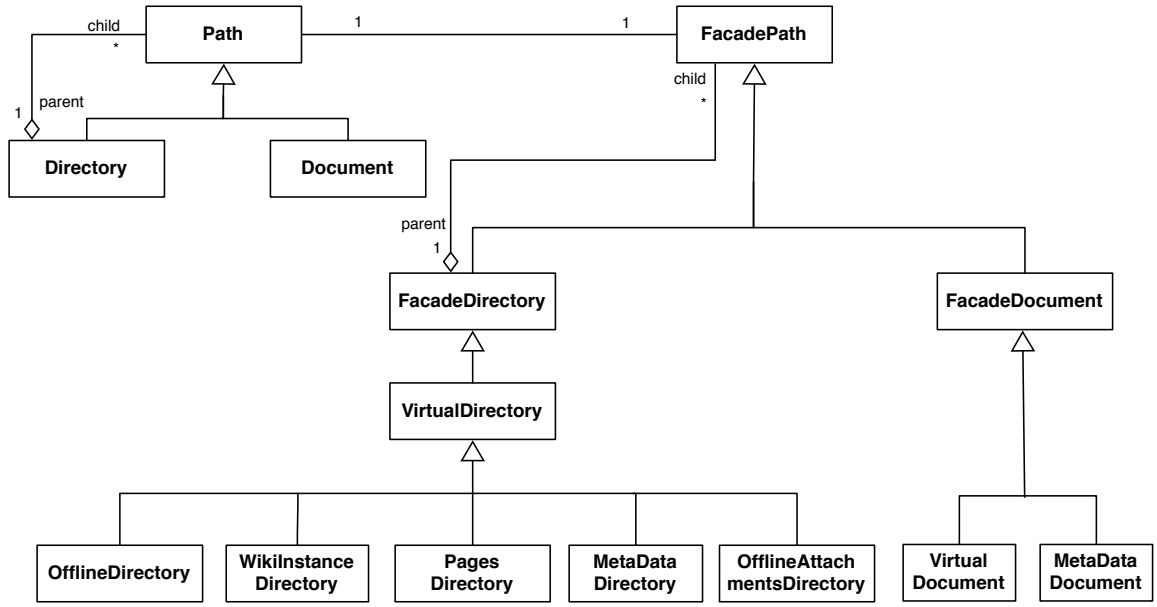


Abbildung 4.5: Erweiterung der Fassaden-Klassen um virtuelle Ordner und Dateien

Die Klasse `VirtualDirectory` überschreibt alle Methoden der beiden Klassen `FacadePath` und `FacadeDirectory`. Diese beiden Klassen greifen nämlich innerhalb ihrer Methoden auf die Referenz zu der Klasse `Path` und ihren Subklassen zu. Diese Referenz ist aber in der Ordnerstruktur für die Offline-Dateien nicht mehr gegeben. Auch die Klassen `VirtualDocument` und `MetaDataDocument` überschreiben die Methoden der Oberklassen.

In Kapitel 4.6 wird für die Implementierung der Synchronisation dieses Klassendiagramm noch um eine weitere Subklasse von `FacadeDirectory` und eine Subklasse von `FacadeDocument` erweitert.

4.3.2 Einbindung der Ordner-/Dateistruktur

Das Sequenzdiagramm in Abbildung 4.4 zeigt die schon vorhandene Implementierung des Samba-Laufwerks. Die statische Methode `findPath()` wird ausgehend aus dem Alfresco Jan Framework für mehrere Operationen aufgerufen. Die wichtigste Operation ist die Methode `startSearch()`, die immer aufgerufen wird, wenn ein Ordner auf dem Laufwerk geöffnet wird.

Die Methode `findPath(String path)` initialisiert anhand des Suchstrings eine der Fassaden-Klassen aus Abbildung 4.2. Um eine Ordnerstruktur für die Offline-Dateien auf dem Samba-Laufwerk anzuzeigen, wurde im vorherigen Kapitel 4.3.1 eine Erweiterung der Fassaden-Klassen vorgenommen. Deswegen muss nun die Methode `findPath()` an

diese neuen Klassen angepasst werden. Dazu wurde eine statische Methode `getFacadePathByPath()` in der Klasse `FacadePath` angelegt. Diese Methode wird im ersten Teil dieses Kapitels gezeigt.

Aufgrund der Plugin-Struktur von Tricia (siehe Kapitel 4.1) muss die Methode `getFacadePathbyPath()` Gebrauch machen von dem Konzept der *Extension* in Tricia. Dazu wird die Klasse `VirtualDirectoriesExtension` implementiert. Das Konzept und die Implementierung dieser *Extension* zeigt der zweite Teil dieses Kapitels.

Damit das Jlan Framework den Inhalt des Ordners, der durch die Methode `getFacadePathByPath()` gefunden wurde, anzeigen kann, muss auch in den Subklassen von `FacadeDirectory` die Methode `getChildren()` implementiert werden. Auf diese Methode geht der letzte Teil dieses Kapitels ein.

Abbildung 4.6 ist eine Erweiterung des Klassendiagramms aus Abbildung 4.3. Für die Einbindung der Ordnerhierarchie für die Offline-Funktionalität müssen die oben angesprochenen Klassen und Methoden implementiert werden. Die zu implementierenden Methoden/Klassen sind blau markiert.

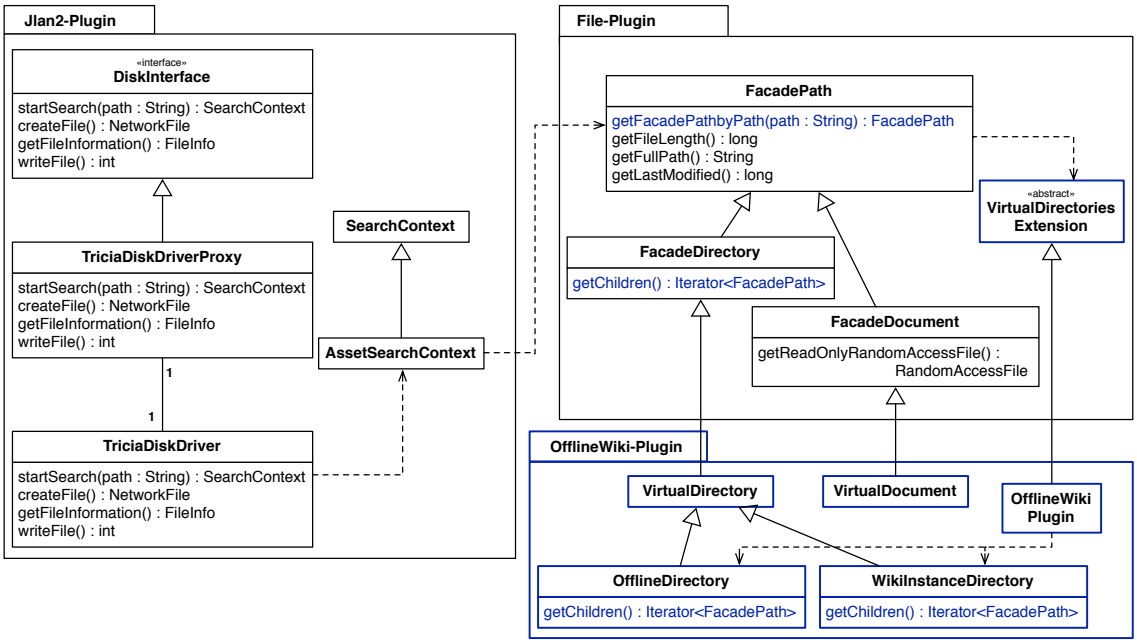


Abbildung 4.6: Klassendiagramm des *Jlan2*-, *File*- und *OfflineWiki*-Plugins

Methode getFacadePathbyPath()

In der vorhandenen Implementierung des Samba-Laufwerks (siehe Kapitel 4.2) ruft die Klasse `AssetSearchContext` u.a. beim Öffnen eines Ordners die statische Methode `findPath()` in der Klasse `Facade` auf. Diese Methode hat anhand des Suchpfades die richtige Instanz einer der Klassen `FacadeDirectory` bzw. `FacadeDocument` des zu öffnenden Ordners bzw. Datei zurückgeliefert. Diese Methode muss nun ausgetauscht werden, damit beim Öffnen eines Ordners bzw. einer Datei aus der Ordnerstruktur für

die Offline-Funktionalität von Tricia auch die richtige Instanz einer der Subklassen von `FacadeDirectory` und `FacadeDocument` (siehe Abbildung 4.2) zurückgeliefert wird. Die neue Methode hat den Namen `getFacadePathbyPath()`.

Als Argument wird der Methode der Suchpfad des zu öffnenden Ordners bzw. Datei übergeben. Die Methode zerlegt den Suchpfad in seine einzelnen Bestandteile. Entspricht der erste Teil des Suchpfades dem Namen des „Offline-Ordners“, wird über weitere if-Abfragen geprüft, welcher Ordner bzw. Datei, also welche Instanz der Subklassen von `FacadeDirectory` bzw. `FacadeDocument` gerade geöffnet werden soll. Entspricht der erste Teil des Pfades nicht dem Offline-Ordner, wird die alte Methode `findPath()` aufgerufen. Wird die Methode mit dem Suchstring *„/Offline/home-wiki/pages/home.html“* aufgerufen, liefert die Methode als Rückgabewert eine Instanz der Klasse `VirtualDocument`. Um die richtige Instanz als Ergebnis zurückzuliefern, müssen dabei aufgrund der Plugin-Struktur von Tricia die Methoden von der oben schon angesprochenen `VirtualDirectoryExtension` Klasse benutzt werden. Einen Ausschnitt der Methode zeigt der folgende Quelltext 4.1.

Quelltext 4.1: Auffinden der richtigen Instanz von FacadePath anhand des Pfades

```
public static FacadePath getFacadePathByPath(String path) {
        FacadePath p = null;
        String[] splittedPath = path.split("/");
        VirtualDirectoriesExtension vde = Main.INSTANCE().getExtension(
            VirtualDirectoriesExtension.class);
        if(vde != null){
                if(path.equals("/"+vde.getOfflineDirectoryName())){
                        p = vde.getOfflineDirectory();
                }
                else if(splittedPath.length > 2 && splittedPath[1].equals(vde.
                    getOfflineDirectoryName())){
                                if(splittedPath.length == 3){
                                        p = getFirstLevelDirectory(splittedPath, vde);
                                }
                                //...
                                else if(splittedPath.length >= 5){
                                        p = getThirdLevelDirectory(path, vde);
                                }
                        }
                        //...
        }
        else {
                p = Facade.INSTANCE().findPath(path);
        }

        return p;
}

private static FacadePath getThirdLevelDirectory(String fullPath,
    VirtualDirectoriesExtension vde){
        String[] splittedPath = fullPath.split("/");
        FacadePath p = null;
        //...
        else if (splittedPath[3].equals(vde.getPagesDirectoryName())){
                String wikiPageDocumentName = splittedPath[4];
                p = vde.getWikiPageDocument(wikiPageDocumentName, fullPath);
        }
        //...
        return p;
}
```

In der Methode `getFacadePathByPath()` wird herausgefunden, dass der erste Teil des Pfades *„/Offline/home/pages/home.html“* dem Ordner-Namen des „Offline“-Ordners entspricht. Auf der Basis der Überprüfung der Pfadlänge wird der Pfad der Methode `getThirdLevelDirectory()` übergeben. Diese Methode stellt fest, dass der dritte Teil des Pfades dem „Pages“-Ordner entspricht. Über den Aufruf der Methode `getWikiPageDocument()` wird eine Instanz der Klasse `VirtualDocument` erzeugt, welche anschließend auch zurückgegeben wird. In dieser Methode wird anhand des Pfades die entsprechende Wikiseite aus der Datenbank geholt. Mit Hilfe dieser Wikiseite wird dann eine Instanz der Klasse `VirtualDocument` erzeugt. Auf eine detaillierte Beschreibung dieser Methode wird verzichtet.

Klasse VirtualDirectoriesExtension

Das Konzept der *Extension* in Tricia erlaubt lose Abhängigkeiten zwischen Plugins herzustellen. In Abbildung 4.6 ist die abstrakte Klasse `VirtualDirectoriesExtension` zu sehen. Diese Klasse wird innerhalb der Klasse `OfflineWikiPlugin` implementiert. Die Klasse `VirtualDirectoriesExtension` erbt von der Klasse `Extension`. Diese *Extension* bildet eine Art Brücke zwischen dem *File*- und *Jlan2*-Plugin und dem neu geschaffenen *OfflineWiki*-Plugin.

In Kapitel 4.3.1 wurde die Erweiterung der Fassaden-Klassen vorgestellt. Diese Subklassen werden im *OfflineWiki*-Plugin implementiert. Um die Ordnerstruktur für die Offline-Funktionalität in das Samba-Laufwerk einzuspeisen, muss im *File*- und *Jlan2*-Plugin auf diese Klassen zugegriffen werden können. Wie man aber in der Plugin-Struktur in Abbildung 4.1 sieht, geht die Abhängigkeit der Plugins vom *OfflineWiki*-Plugin über das *Jlan2*-Plugin zum *File*-Plugin. Die abstrakte Klasse `VirtualDirectoriesExtension` und deren Implementierung in der Klasse `OfflineWikiPlugin` erlaubt den indirekten Zugriff auf die Fassaden-Klassen. So wird auch im obigen Quelltext 4.1 in der Methode `getFacadePathByPath()` über den Aufruf der Methode `getWikiPageDocument()` eine Instanz der Klasse `VirtualDocument` instanziert. Einige der Methoden der Klasse `VirtualDirectoriesExtension` sind in Abbildung 4.7 zu sehen.

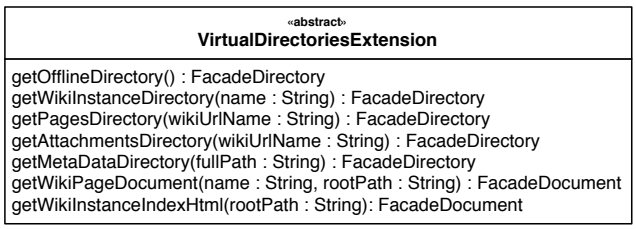


Abbildung 4.7: Klassendiagramm von `VirtualDirectoriesExtension`

Methode getChildren()

Wie schon in Kapitel 4.2 ausgeführt, liefert die Methode `getChildren()` in der Klasse `FacadeDirectory` einen Iterator über den Inhalt des aktuellen Ordners. Diese Methode muss daher in den Subklassen von `VirtualDirectory` implementiert werden. So

liefert diese Methode in der Klasse `OfflineDirectory` einen Iterator über alle Wiki-Instanz-Ordner (`WikiInstanceDirectory`). Die `getChildren()` Methode der Klasse `WikiInstanceDirectory` liefert einen Iterator über dessen Ordnerinhalt. Unterhalb des Wiki-Instanz Ordners befinden sich gemäß der Ordnerstuktur aus Kapitel 3.4 der „Pages"-, „Attachments"- und „MetaData"-Ordner sowie eine Index-Datei. Der Iterator, den diese Methode zurückliefert, enthält also jeweils eine Instanz der Klassen `PagesDirectory`, `OfflineAttachmentsDirectory`, `MetaDataDirectory` und `VirtualDocument`. Diese Methode ist im Beispiel des Sequenzdiagramms in Abbildung 4.8 zu sehen.

Der „Pages"-Ordner soll alle HTML-Dateien der Wikiseiten enthalten. Die Methode `getChildren()`-in der Klasse `PagesDirectory` generiert einen Iterator über alle Wikiseiten, also Instanzen der Klasse `VirtualDirectory`, der jeweiligen Wiki-Instanz. Die Implementierung dieser Methode ist im folgenden Quelltext 4.2 zu sehen.

Quelltext 4.2: Ordnerinhalt des Wiki-Instanz-Ordners

```
@Override
public Iterator<FacadePath> getChildren() {
        Iterator<WikiPage> wikiPageIterator = Wiki.SCHEMA.getEntity(wikiId).pages.
            getAssetsIterator();
        List<FacadePath> facadePathList = new ArrayList<FacadePath>();
        while (wikiPageIterator.hasNext()) {
                WikiPage wikiPage = wikiPageIterator.next();
                VirtualDocument virtualDocument = new VirtualDocument(wikiPage);
                virtualDocument.setFullPath(this.getFullPath()+"/"+wikiPage.urlName.get()+
                    ".html");
                facadePathList.add(virtualDocument);
        }
        return facadePathList.iterator();
}
```

Über das „Query-Framework" von Tricia wird zunächst die Wiki-Instanz über deren „Id" gefunden. Die Wiki-Id wird der Klasse `PagesDirectory` im Konstruktor übergeben. Über die Wiki-Instanz lassen sich alle darin befindlichen Wikiseiten aus der Datenbank herausholen. Somit lässt sich ein Iterator zurückgeben, der für jede Wikiseite eine Instanz der Klasse VirtualDocument enthält. Auf die Generierung der HTML-Dateien der Wikiseiten wird ausführlich in Kapitel 4.4 eingegangen.

Um die Ordnerstruktur für die Offline-HTML-Dateien initial in das virtuelle Dateisystem einzuhängen, muss die `getChildren()` Methode in der Klasse `FacadeDirectory` leicht modifiziert werden. Ein Ausschnitt dieser Methode ist weiter unten zu sehen. Wenn der aktuelle Ordner, der im virtuellen Dateisystem geöffnet wird, das Wurzelverzeichnis ist, so soll dort zusätzlich zu den anderen existierenden Ordnern, z.B. der „Attachments"-Ordner, der neue „Offline"-Ordner (`OfflineDirectory`) angezeigt werden.

Aufgrund der Plugin-Struktur von Tricia (siehe Abbildung 4.1) kann dem Iterator der `getChildren()` Methode in der Klasse `FacadeDirectory` nicht direkt eine Instanz der Klasse `OfflineDirectory` hinzugefügt werden. Deswegen wird sich der schon beschriebenen Klasse `VirtualDirectoriesExtension` bedient. Diese Klasse implementiert die Methode `getOfflineDirectory()` und liefert als Rückgabewert eine Instanz der Klasse `OfflineDirectory`.

Der folgende Quelltext 4.3 zeigt einen Ausschnitt aus dieser Methode.

Quelltext 4.3: Ordnerinhalt des Root-Ordners

```
public Iterator<FacadePath> getChildren() {
        Iterator<FacadePath> result;
        //...
        VirtualDirectoriesExtension vde = Main.INSTANCE().getExtension(
            VirtualDirectoriesExtension.class);
                if (getDirectory().isRoot()) {
                        if (vde != null) {
                                result = new ChainedIterator<FacadePath>(result,
                                                new OneIterator<FacadePath>
                                                        (vde.getOfflineDirectory()));
                        } else {
                        }
                }
       // ...
        return result;
}
```

Beispiel

Nun wird noch einmal versucht, die beschriebenen Klassen und Methoden (siehe auch Abbildung 4.6) in einen Gesamtzusammenhang zu bringen. Es wird versucht, die Verflechtung der einzelnen Klassen und Methoden anhand eines Sequenzdiagramms (siehe Abbildung 4.8) darzustellen. Das Sequenzdiagramm zeigt eine Operation eines Nutzers auf dem Samba-Laufwerk. Das Beispiel geht davon aus, dass der Benutzer auf dem virtuellen Laufwerk einen Wiki-Instanz-Ordner in der Ordnerhierarchie für die Offline-Funktionalität öffnen will.

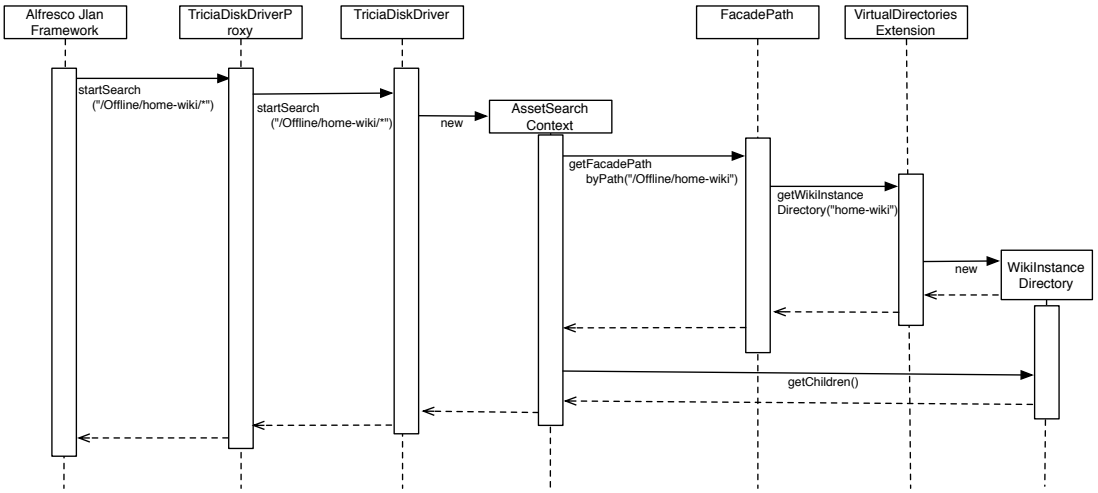


Abbildung 4.8: Sequenzdiagramm über das Zusammenspiel des *Jlan-*, *File-* und *OfflineWiki*-Plugin

Der Benutzer ist mit dem Samba-Laufwerk verbunden und will den Wiki-Instanz-Ordner mit dem Pfad *„/Offline/home-wiki“* öffnen. Das Alfresco Jlan Framework ruft bei dieser Operation die Methode `startsearch("/Offline/home-wiki/*")` in der Klasse `TriciaDiskDriverProxy` bzw. `TriciaDiskDriver` auf. Als Rückgabewert erwartet das Framework eine Instanz der Klasse `SearchContext`. In der Klasse

`TriciaDiskDriver` wird eine Instanz der Klasse `AssetSearchContext` initialisiert. Im Konstruktor der Klasse `AssetSearchContext` wird die statische Methode `getFacadePathbyPath()` mit dem String *„/Offline/home-wiki“* als Argument aufgerufen. In der Methode wird anhand des Pfades dann herausgefunden, dass der Nutzer den Ordner der Wiki-Instanz „home-wiki“ öffnen will. Um die entsprechende Instanz der Klasse `WikiInstanceDirectory` zu erhalten, wird auf der Klasse die Methode `getWikiInstanceDirectory()` in der Klasse `VirtualDirectoriesExtension` aufgerufen. Diese gibt eine Instanz der Klasse `WikiInstanceDirectory` zurück. Um den Ordnerinhalt dieser Wiki-Instanz anzuzeigen, ruft nun die vorher geschaffene Instanz der Klasse `AssetSearchContext` die Methode `getChildren()` auf dieser Instanz auf. Diese Methode liefert den Ordner-Inhalt in Form eines Iterators über Instanzen der Klasse `VirtualDocument`. Wie oben erwähnt, enthält der Iterator jeweils eine Instanz der Klassen `PagesDirectory`, `OfflineAttachmentsDirectory`, `MetaDataDirectory` und `VirtualDocument`.

4.4 Generierung der Dateien

Im letzten Kapitel 4.3 wurde die Implementierung der Offline-Ordnerstruktur sowie deren Einbindung in das Samba-Laufwerk gezeigt. Dieses Kapitel zeigt die Generierung der Dateien, die in dieser Ordnerstruktur angezeigt werden.

Für das Öffnen einer Datei auf dem Samba-Laufwerk wird die Methode `getReadOnlyRandomAccessFile()` in der Klasse `FacadeDocument` implementiert (siehe Quelltext 4.4). Als Ergebnis liefert die Methode eine Instanz der Klasse `RandomAccessFile`. Diese Instanz enthält den Inhalt der entsprechenden Datei und wird vom Alfresco Jlan Framework benutzt, um diesen Inhalt zu lesen und dem Benutzer auszuliefern.

Dieser Zusammenhang wird im Sequenzdiagramm in Abbildung 4.10 dargestellt. Will der Benutzer die Datei *„/Offline/home-wiki/pages/home.html“* auf dem Samba Laufwerk öffnen bzw. auf seine lokale Festplatte kopieren, versucht zunächst das Jlan-Framework die Datei zu öffnen (`open()`). Die Klasse `TriciaDiskDriver` implementiert, wie schon in Kapitel 4.2 erwähnt, die Schnittstelle des Frameworks. In der Klasse `TriciaDiskDriver` wird nun versucht die Instanz einer der Fassaden-Klassen zu finden, welche die Datei repräsentiert (`getFacadePathbyPath()`). In unserem Beispiel wird mit Hilfe der Klasse `VirtualDirectoriesExtension` eine Instanz der Klasse `VirtualDocument` erzeugt. Ist die entsprechende Instanz gefunden, erzeugt die Klasse `TriciaDiskDriver` eine Instanz der Klasse `TriciaNetworkFile` und übergibt dem Konstruktor die vorher gefundene Instanz. Die Klasse `TriciaNetworkFile` implementiert die vom Framework bereitgestellte abstrakte Klasse `NetworkFile`. Diese Klasse repräsentiert, wie der Name schon ausdrückt, eine Netzwerkdatei und implementiert Methoden, um die jeweilige Datei auf dem Samba-Laufwerk zu öffnen, zu beschreiben, etc (siehe Klassendiagramm in Abbildung 4.9).

Das Alfresco Jlan Framework ruft nun die Methode `readFile()` auf und übergibt als Argument die soeben generierte Instanz der Klasse `TriciaNetworkFile` sowie weitere Argumente, die Angaben machen über die zu lesende Stelle in der angeforderten Datei. Die Klasse `TriciaDiskDriver` ruft auf der mitgegebenen Instanz die Methode

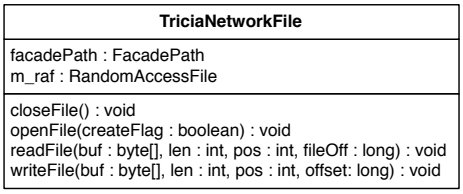


Abbildung 4.9: Klassendiagramm von `TriciaNetworkFile`

`readFile()` auf. Im `TriciaNetworkFile` wird dann, um den Inhalt des `FacadePath` Attributs zu lesen, die Methode `getReadOnlyRandomAccessFile()` auf diesem Attribut (in unserem Fall eine Instanz der Klasse `VirtualDocument`) aufgerufen. Die vom Framework angeforderte Stelle des `RandomAccessFile` wird in ein Byte-Array geschrieben. Dieses Byte-Array wird dann vom Alfresco Framework benutzt, um den Inhalt der Datei an den Benutzer auszuliefern.

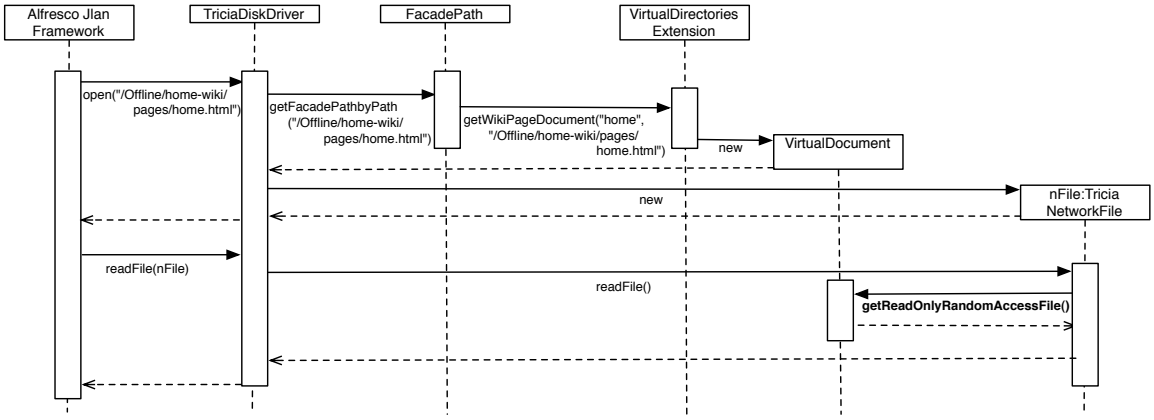


Abbildung 4.10: Sequenzdiagramm über das Öffnen einer Datei auf dem Samba-Laufwerk

Wie man sieht, muss, um Dateien an die Benutzer ausliefern zu können, die Methode `getReadOnlyRandomAccessFile()` in den Subklassen von `FacadeDocument` implementiert werden.

Dieses Kapitel zeigt im weiteren Verlauf zunächst die Implementierung dieser Methode für die HTML-Dateien der Wikiseiten. Anschließend wird die Generierung der Index-Dateien beschrieben. Zum Schluss wird noch auf die Auslieferung der Meta-Dateien, die u.a. für das Design der Wikiseiten notwendig sind, eingegangen.

4.4.1 HTML-Dateien der Wikiseiten

Das Kapitel 3.1.1 hat die Features identifiziert, die für den Lesezugriff bei der Offline-Variante von Tricia implementiert werden sollen. Im Kapitel 3.5.1 wurde auf Basis dieser Features gezeigt, wie die HTML-Dateien der Wikiseiten aufgebaut sein sollen. Die Klasse `VirtualDocument` implementiert eine Offline-HTML-Datei. Das Klassendiagramm in Abbildung 4.11 zeigt einige Attribute und Methoden dieser Klasse.

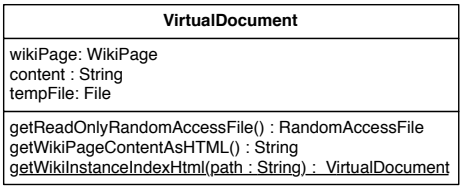


Abbildung 4.11: Klassendiagramm von `VirtualDocument`

Wie in der Einleitung zu diesem Kapitel beschrieben wurde, fordert das Alfresco Jlan Framework beim Öffnen einer Datei über die Methode `getReadOnlyRandomAccessFile()` eine Instanz der Klasse `RandomAccessFile` an. Die Implementierung dieser Methode ist im folgenden Quelltext 4.4 zu sehen.

Quelltext 4.4: Methode für den Lesezugriff auf eine Wikiseite im Samba-Laufwerk

```
@Override
public RandomAccessFile getReadOnlyRandomAccessFile()
                throws FileNotFoundException {
        RandomAccessFile raf = null;
        try {
                tempFile = File.createTempFile("tempFile", ".tmp");
                FileOutputStream fos = new FileOutputStream(tempFile);
                fos.write(getContent().getBytes());
                raf = new RandomAccessFile(tempFile, "r");
        } catch (IOException e) {
                e.printStackTrace();
        }
        return raf;
}
```

Der Konstruktor der Klasse `RandomAccessFile` benötigt als Argumente eine Instanz der Klasse `File` und den Modus als String. Wie der Methodenname schon ausdrückt, wählen wir als Modus den Lesezugriff, übergeben also als Modus den String „r“. Für die benötigte Instanz der Klasse `File` wird eine temporäre Datei angelegt (Attribut `tempFile`). In diese temporäre Datei wird der Inhalt der HTML-Datei, welcher als String in dem Attribut `content` gespeichert ist, geschrieben. Die temporäre Datei wird nach Abschluss des Lesevorganges wieder gelöscht.

Template-System

Die Frage ist nun, wie das String-Attribut `content` erzeugt wird. In diesem String soll der HTML-Quelltext einer Offline-Wikiseite stehen.

In Tricia gibt es eine eigene Template-Sprache [inf11b]. Durch das Tricia-Template-System lassen sich Platzhalter in HTML-Dateien definieren, welche zur Laufzeit ersetzt werden. Um einen Platzhalter in einer HTML-Datei zu definieren, muss in Tricia am Anfang und Ende des Platzhalters das „$“ Zeichen stehen. Die Substitution des Platzhalters wird durch Java-Code implementiert. Dazu implementiert man Funktionen, die als Methodennamen den gleichen Namen wie die HTML-Datei besitzen. In der Funktion muss eine Instanz der Klasse `TemplateSubstituion` zurückgegeben werden. Dieses System wird normalerweise für die Generierung der Wikiseiten in der Webanwendung von Tricia benutzt.

Das Grundgerüst der Offline-HTML-Dateien orientiert sich an dem HTML-Quelltext der Online-Wikiseiten. Die Elemente, die in den Offline-Seiten enthalten sein sollen, wurden in Kapitel 3.1.1 herausgearbeitet. Die Ausgangsdatei für die Wikiseiten ist die Datei „offlineWikiPage.htm“. Folgender Quelltext 4.5 zeigt einen Ausschnitt aus dieser HTML-Datei.

Quelltext 4.5: Haupt-Template für die Offline-Wikiseite

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>$name.value()$</title>
        $cssReference$
    </head>
    <body>
        <!-- ... -->
        $offlineHybridTable()$
        <div id="wikiPageContent" class="tricia-richstring">
            $wikiPageContent$
        </div>
        <!-- ... -->
    </body>
    $javaScript$
</html>
```

Der Platzhalter `$wikiPageContent$` soll zur Laufzeit, z.B. wenn die Offline-HTML-Datei vom Samba-Laufwerk auf die lokale Festplatte kopiert wird, durch den Inhalt der jeweiligen Wikiseite ersetzt werden. Dazu wird eine Funktion `offlineWikiPage()` implementiert, die eine Instanz der Klasse `TemplateSubstitution` als Ergebnis zurückliefert. Einen Ausschnitt aus der Implementierung dieser Methode zeigt folgender Quelltext 4.6.

Quelltext 4.6: Substitutionsmethode für das Haupt-Template einer Offline-Wikiseite

```
public static TemplateSubstitution offlineWikiPage(final WikiPage wikiPage, final
    FunctionParameters params) {
        return new TemplateSubstitution() {
                @Override
                public void putSubstitutions(Template template) {
                        //...
                        template.put("wikiPageContent", new PrintSubstitution(
                                        Escaping.none) {
                                public String print() {
                                        return repairLinks(wikiPage.content.get());
                                }
                        });
                        //...
                }
        };
}
```

Der Methode wird die aktuelle Wikiseite übergeben. Für den Platzhalter `$wikiPageContent$` wird eine `PrintSubstitution` definiert. Diese `PrintSubstitution` ersetzt den Platzhalter mit dem Inhalt der Wikiseite (`wikiPage.content.get()`). Die Methode `repairLinks()` verändert die Hyperlinks im Inhalt der Wikiseiten, damit diese im Offline-Modus funktionieren. Diese Methode wird später genauer vorgestellt.

Dieser Substitutionsmechanismus funktioniert auf gleiche Art und Weise auch für die anderen Elemente der Offline-HTML-Datei, die in Kapitel 3.5.1 herausgearbeitet wurden. Z.B. ist im obigen Quelltext 4.5 noch ein Platzhalter `$offlineHybridTable()$` zu sehen. Dieser wird durch den HTML Quelltext für die *HybridTable* ersetzt.

Bei der Erzeugung einer Instanz der Klasse `Virtualdocument` wird das String Attribut `content` gesetzt. Das Attribut wird mit dem in der oben gezeigten Methode `offlineWikiPage()` (siehe Quelltext 4.6) erzeugten HTML-Quelltext inklusive der vorgenommenen Substitutionen belegt. Beim Lesezugriff auf die entsprechende Datei wird dieser String in eine temporäre Datei geschrieben.

Hyperlinks

Der Inhalt von Wikiseiten kann Hyperlinks auf andere *Assets* von Tricia oder andere Internetadressen enthalten. In Kapitel 3.5.3 wurde erläutert, dass die Hyperlinks im Inhalt der Wikiseiten für die Offline-Version geändert werden müssen. Desweiteren wurde festgestellt, dass bei dem gewählten Konzept für den Offline-Modus von Tricia nur Hyperlinks auf die vier *Assets* `WikiPage`, `Wiki`, `Directory` und `Document` Sinn machen.

Im obigen Quelltext 4.6 sieht man für die Substitution des Inhalts der Wikiseite die Methode `repairLinks()`. Diese Methode nimmt als Argument den Inhalt der Wikiseiten, der in der Datenbank gespeichert ist entgegen und verändert alle darin enthaltenen Hyperlinks, damit diese in der Offline-Version auf das richtige Ziel zeigen. Der folgende Quelltext 4.7 zeigt einen Ausschnitt aus dieser Methode.

Quelltext 4.7: Anpassen der Hyperlinks im Inhalt der Wikiseite für die Offline-Version von Tricia

```
private static String repairLinks(String content){
        List<TransientLink> links = RichStringProperty.getLinks(content);
        List<StringInsert> inserts = new ArrayList<StringInsert>();
        for (TransientLink link : links) {
                String newLink = "";
                String[] splittedLink = link.url.split("/");
                if(splittedLink.length >= 3){
                        // => Link on wiki/wikiPage
                        if(splittedLink[1].equals("wikis")){
                                newLink = "../../" + splittedLink[2];
                                if (splittedLink.length < 4) { //=> Link on wiki
                                        newLink = newLink + "/index.html";
                                } else {  // => Link on wikiPage
                                        newLink = newLink + "/pages/" + splittedLink[3] +
                                            ".html";
                                }
                        }
                        // => Link on Attachment
                        else if (splittedLink[1].equals("file") && splittedLink[2].equals(
                            "Attachments")){
                                String wikiUrlName = splittedLink[4];
                                newLink = "../../"+wikiUrlName+"/Attachments";
                                for(int i = 5; i < splittedLink.length; i++){
                                        newLink = newLink+"/"+splittedLink[i];
                                }
                        }
                        //...
                        inserts.add(link.getInsert(newLink));
                }
        }
```

```
        return StringInsert.insertAll(inserts, content);
}
```

Hyperlinks auf Wikis bzw. Wikiseiten haben in der Webanwendung folgendes Format: „/wikis/{*wikiInstanceUrlName*}“ bzw. „/wikis/{*wikiInstanceUrlName*}/{*wikiPageUrlName*}“. Hyperlinks auf Ordner und Dateien aus dem Attachments-Ordner haben das Format: „/file/Attachments/wikis/{*wikiInstanceUrlName*}/{*Pfad zur Datei/Ordner*}“.

In der oben gezeigten Methode werden zunächst über schon implementierte Methoden alle Hyperlinks, die im Inhalt der Wikiseite enthalten sind, herausgefiltert. Für jeden dieser Links wird nun anhand des ersten Teils des Pfades herausgefunden, ob es sich um einen Link auf ein Attachment oder auf ein(e) Wiki/Wikiseite handelt.

Bei einem Link auf ein Wiki wird der Link umgebaut, so dass er auf die Index-Datei des Wikis (siehe Kapitel 3.5.2) zeigt. Ein Link auf ein Wiki von einer Wikiseite aus hat im Offline-Modus aufgrund der gewählten Ordnerstruktur (siehe Kapitel 3.5.2) folgendes Format: „../../{*wikiInstanceUrlName*}/*index.html*“. Die HTML-Dateien der Wikiseiten liegen im „Pages“-Ordner. Geht man von diesem Ordner zwei Ebenen höher, befindet man sich im „Offline-Ordner“. Ausgehend von diesem Ordner navigiert man in den richtigen „Wiki-Instanz“-Ordner. Dort befindet sich dann die Index-Datei. Auf die Generierung dieser Index-Dateien geht das folgende Kapitel 4.4.2 ein.

Ein Link auf eine Wikiseite zeigt in der Offline-Version auf die HTML-Datei der entsprechenden Wikiseite. Ein Link von einer Wikiseite auf eine andere hat im Offline-Modus daher folgendes Format: „../../{*wikiInstanceUrlName*}/*pages*/{*wikiPageUrlName*}.*html*“. Zunächst navigiert man, ausgehend vom „Pages“-Ordner, zwei Ebenen höher in den „Offline“-Ordner. Von dort führt der Pfad in den „Pages“-Ordner der entsprechenden Wiki-Instanz und zeigt dort auf die entsprechende HTML-Datei der Wikiseite.

Wird auf einen Dateianhang im Inhalt einer Wikiseite verlinkt, muss in der Offline-Version der Link im Offline-Modus folgendes Format haben: „../../{*wikiInstanceUrlName*}/*Attachments*/{*Pfad zur Datei/Ordner*}“. Wie vorher, wird in den Ordner der Wiki-Instanz navigiert. Von dort führt der Pfad auf die entsprechende Datei im „Attachments“-Ordner.

Für Hyperlinks im strukturierten Teil der Wikiseiten, also in der *Hybrid Table*, müssen die Links über das oben gezeigte Template-System richtig in die vorgesehenen Platzhalter eingesetzt werden. Dazu wurde die Funktion `getLinkForAsset()` implementiert (siehe Quelltext 4.8). Diese Funktion wird in den Substitutions-Methoden aufgerufen, um Hyperlinks auf *Assets* in der Offline-Ordnerhierarchie zu setzen.

Quelltext 4.8: Anpassen der Hyperlinks auf *Assets* im strukturierten Teil der Offline-Wikiseite

```
private static String getLinkForAsset(PersistentEntity entity){
        String oldLink = entity.getUrl();
        String[] splittedLink = oldLink.split("/");
        String newLink = "";
        if(entity instanceof WikiPage){
                String wikiUrlName = splittedLink[2];
                String wikiPageUrlName = splittedLink[3];
                newLink = "../../" + wikiUrlName+"/"+PagesDirectory.DIRECTORY_NAME+"/"+
                    wikiPageUrlName+".html";
        }
        else if(entity instanceof Wiki){
                //...
        }
        else if(entity instanceof Path){
```

```
                //...
        }
        return newLink;
}
```

Sie funktioniert ähnlich wie die oben gezeigte Methode `repairLinks()`. Als Argument wird der Methode eine Instanz der Klasse `PersistentEntity` übergeben. Die Methode überprüft, um welche konkrete Instanz es sich handelt. Verlinkungen auf die drei *Assets* werden nach dem oben gezeigten Format aufgebaut.

CSS-Referenz

Damit die Offline-Wikiseiten das gleiche Design haben wie die Online-Version (siehe Kapitel 3.5.1), wird in den HTML-Dateien auf die entsprechenden CSS-Dateien referenziert, die auch für die Webanwendung benutzt werden. Diese CSS-Dateien werden im _metaData-Ordner an den Benutzer ausgeliefert. In dem Quelltext 4.5 sieht man den Platzhalter `$cssReference()$`. Dieser wird durch die Referenzen auf die entsprechenden CSS-Dateien ersetzt. Auf die Generierung dieser Meta-Dateien geht Kapitel 4.4.3 ein.

4.4.2 Index-Dateien

Für die Generierung einer Index-Datei (siehe auch Kapitel 3.5.2) gibt es in der Klasse `VirtualDocument` die statische Methode `getWikiInstanceIndexHtml()` (siehe Abbildung 4.11). Diese Methode erzeugt eine Instanz der Klasse `VirtualDocument`. Dabei wird das `WikiPage`-Attribut nicht gesetzt. Wie im vorherigen Kapitel erläutert, muss das String-Attribut `content` richtig gesetzt werden. Denn dieser String ist letztendlich der Inhalt der Index-Datei. Wie schon bei der Generierung der Offline-Wikiseiten, wird für die Erzeugung dieses Strings das Template-System von Tricia benutzt. Es gibt für die Index-Dateien ein eigenes HTML-Template. Einen Ausschnitt aus dem Grundgerüst für die Index-Datei einer Wiki-Instanz zeigt folgender HTML-Quelltext 4.9.

Quelltext 4.9: HTML-Template für die Generierung einer Index-Datei

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
        <!-- ... -->
        </head>
        <body>
                <!-- ... -->
                <div id="wrapper">
                        <div id="pageContent" class="tricia-page-content">
                                <h1>Index of $wikiInstanceName$</h1>

                                <div class="tricia-richstring">
                                        $instanceWikiPageLinks$
                                </div>
                        </div>
                </div>
        </body>
</html>
```

Der Platzhalter `$instanceWikiPageLinks$` wird zur Laufzeit durch eine Liste von Wikiseiten inklusive Hyperlinks auf deren HTML-Dateien ersetzt. Dazu wird der ent-

sprechenden Substitions-Methode `offlineInstanceIndexPage()` ein Iterator über alle Wikiseiten der Wiki-Instanz übergeben. Dies sieht man im folgenden Quelltext 4.10 der Methode `getWikiInstanceIndexHtml()`.

Quelltext 4.10: Generierung einer Index-Datei

```
public static VirtualDocument getWikiInstanceIndexHtml(String rootPath){
        String wikiInstanceUrlName;
        String[] splitted = rootPath.split("/");
        wikiInstanceUrlName = splitted[splitted.length-1];
        Query q = new QueryEquals(Wiki.SCHEMA.prototype().urlName, wikiInstanceUrlName);
        Wiki wiki = Wiki.SCHEMA.queryEntities(q).iterator().next();
        Iterator<WikiPage> iterator = wiki.pages.getAssetsIterator();
        String content = Template.getString(Functions.offlineInstanceIndexPage(wiki.
            getName(), iterator));
        VirtualDocument indexHtml = new VirtualDocument();
        indexHtml.urlName = "index.html";
        indexHtml.title = "index";
        indexHtml.setFullPath(rootPath+"/"+"index.html");
        indexHtmlcontent = content;
        return indexHtml;
}
```

Anhand des Pfades, der als String der Methode übergeben wird, wird über das „Query“-Framework die entsprechende Wiki-Instanz gefunden. Anschließend wird ein Iterator über die Wikiseiten dieser Wiki-Instanz erzeugt.

Die Funktion `offlineInstanceIndexPage()` aus der Klasse `Functions` ist die Methode, die für die Substitution der Platzhalter in der obigen HTML-Datei (siehe Quelltext 4.9) zuständig ist. Dieser Methode wird der Name der Wiki-Instanz sowie der Iterator der Wikiseiten übergeben. Anschließend wird eine Instanz der Klasse `VirtualDocument` erzeugt und das String-Attribut `content` mit dem Ergebnis der Substitution belegt.

4.4.3 Meta-Dateien

Die Meta-Dateien sind für das Design der HTML-Dateien der Wikiseiten und der Index-Dateien zuständig. Die Meta-Dateien werden im Samba-Laufwerk in den Ordner „_metaData“ eingespeist. Für die Meta-Dateien wurden die beiden Klassen `MetaDataDirectory` und `MetaDataDocument` angelegt (siehe Abbildung 4.5). Erstere Klasse repräsentiert den „_metaData“-Ordner sowie dessen Unterordner. Die Klasse `MetaDataDocument` repräsentiert die Dateien in diesen Ordnern. Das Klassendiagramm in Abbildung 4.12 zeigt einige Attribute und Methoden dieser Klasse.

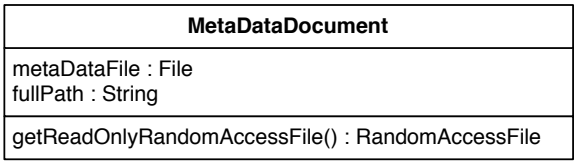


Abbildung 4.12: Klassendiagramm von `MetaDataDocument`

Bei der Generierung der HTML-Dateien für die Wikiseiten und der Index-Dateien wurde in der Methode `getReadOnlyRandomAccessFile()` immer eine temporäre Datei

erstellt (siehe Quelltext 4.4). Diese temporäre Datei wurde dann benutzt, um dem Alfresco Jlan Framework eine Instanz der Klasse `RandomAccessFile` zurückzuliefern.

Für die Meta-Dateien muss keine temporäre Datei erzeugt werden. Dem Konstruktor der Klasse `MetaDataDocument` wird die schon bestehende Datei (z.B. CSS-Datei, Icons, etc.) übergeben. Der Konstruktor belegt dann das Attribut `file` mit dieser Datei. In der Methode `getReadOnlyRandomAccessFile()` wird dann mit dieser Datei eine Instanz der Klasse `RandomAccessFile` erzeugt. Dies ist im folgenden Quelltext 4.11 zu sehen.

Quelltext 4.11: Methode für den Lesezugriff auf eine Meta-Datei im Samba-Laufwerk

```
private RandomAccessFile getReadOnlyRandomAccessFile(){
        RandomAccessFile raf = null;
        try {
                raf = new RandomAccessFile(metaDataFile, "r");
        } catch (IOException e) {
                e.printStackTrace();
        }
        return raf;
}
```

4.5 Lokaler Schreibzugriff

Im Kapitel 3.6 wurde ein Konzept für den Schreibzugriff in der Offline-Variante von Tricia entwickelt. Benutzer sollen auf den HTML-Dateien der Wikiseiten Notizen anlegen können. Dieses Kapitel zeigt die technische Umsetzung dieses Konzepts. Für das Einbinden der gezeigten Quelltexte wird das im vorherigen Kapitel gezeigte Template System verwendet.

Für die Umsetzung muss JavaScript Quelltext in die HTML-Dateien der Wikiseiten integriert werden. Dafür gibt es auf dem Ausgangs-Template der HTML-Dateien (*offlineWikiPage.htm*) einen Platzhalter `$javaScript$` (siehe HTML-Quelltext 4.5), der durch den JavaScript Quelltext in der Datei *javaScript.htm* substituiert wird. Im Folgenden wird auf die für den Schreibzugriff relevanten Stellen dieses JavaScript-Quelltextes eingegangen.

Zunächst erläutert das Kapitel, wie der JavaScript-Editor TinyMCE in die HTML-Dateien der Wikiseiten eingebunden wird. Anschließend wird gezeigt, wie der in diesem Editor verfasste Text als Notiz in der HTML-Datei abgespeichert wird.

4.5.1 Einbettung von TinyMCE

Das Konzept aus Kapitel 3.6 sieht für das Anlegen und Editieren der Notizen den Editor TinyMCE vor. TinyMCE ist komplett in JavaScript geschrieben. Um den Editor auf einer HTML-Seite einzubinden, müssen die JavaScript-Dateien von TinyMCE in der HTML-Datei referenziert werden (siehe Quelltext 4.13). Damit der Editor auch ohne Internetverbindung funktioniert, müssen, wie schon in Kapitel 3.6 erwähnt, die notwendigen JavaScript Dateien an den Benutzer mit ausgeliefert werden. Deshalb werden die Dateien im „MetaData“-Ordner eingespeist. Die Generierung der Meta-Dateien wurde in Kapitel 4.4.3 erläutert. Das Einbinden von Dateien auf dem Samba-Laufwerk zeigt Kapitel 4.3. Um TinyMCE in den HTML-Dateien einsetzen zu können, muss ein Textbereich im HTML Quelltext deklariert werden. Der Platzhalter `$offlineTinyMCE$` im Template der Datei *offlineWikiPage.htm* (siehe Quelltext 4.5) wird substituiert durch den HTML-Quelltext für

den TinyMCE Editor. Der HTML-Quelltext für diese Substitution steht in der entsprechenden HTML-Datei *offlineTinyMCE.htm*. Der folgende Quelltext 4.12 zeigt den Inhalt dieser Datei.

Quelltext 4.12: Template für den Editor TinyMCE

```
<div id="tinymce" class="hidden">
    <form method="post" action="$urlName.value()$.html" onsubmit="return saveNote();">
        <textarea id="tinyMCE" name="offlineTinyMCE" rows="15" cols="80" style="width: 80%
            ">
        </textarea>
        <br/>
        <input type="submit" name="save" value="Save" /><input type="reset" name="reset"
            value="Cancel" />
    </form>
</div>
```

Das `<textarea>` Element wird dann zur Laufzeit, also wenn der Benutzer die HTML-Datei im Browser öffnet, durch den TinyMCE-Editor ersetzt. Im darumliegenden`<form>` Element wird im Attribut `onsubmit` deklariert, dass beim Drücken des „Save“ Knopfes die JavaScript Methode `saveNote()` aufgerufen wird (siehe Quelltext 4.19). Diese Methode wird später erläutert. Im Attribut `action` wird deklariert, welche HTML-Datei nach dem Drücken des „Save“ Buttons geöffnet werden soll. In unserem Fall soll die aktuelle HTML-Seite mit dem neuen Inhalt neu geladen werden. Daher wird hier der Platzhalter `$urlName.value()$` durch den *Url-Namen* der aktuellen Wikiseite, was dem Dateinamen der HTML-Seite entspricht, ersetzt.

Damit das oben deklarierte Eingabefeld durch den TinyMCE-Editor ersetzt wird, muss dieser initialisiert werden (`tinyMCE.init()`). Die Initialisierung wird im oben schon angesprochenen JavaScript Quelltext in der Datei *javaScript.htm* angestoßen (siehe Quelltext 4.13).

Quelltext 4.13: Initialisierung des Editors TinyMCE

```
<script type="text/javascript" src="../_metaData/jquery-1.4.2.js"></script>
<script type="text/javascript" src="../_metaData/tiny_mce/tiny_mce.js"></script>
<script type="text/javascript">
tinyMCE.init({
    mode: "textareas",
    theme: "advanced",
    body_class: "tricia-page-content tricia-richstring",
    content_css: "../_metaData/tricia-richtext.css,../metaData/undohtml.css",
    theme_advanced_toolbar_location: "top",
    theme_advanced_toolbar_align: "left",
    theme_advanced_buttons1: "bold,italic,strikethrough,separator," +
        "justifyleft,justifycenter,justifyright,formatselect,styleselect,separator," +
        "bullist,numlist,outdent,indent",
    theme_advanced_buttons2: "forecolor,backcolor,separator,charmap,sub,sup,replace,hr,
        cleanup",
    theme_advanced_buttons3: ""
});
//...
</script>
```

Über das Attribut `body_class` wird festgelegt, welche HTML-Klasse der geschriebene Text im Textfeld des Editors haben soll. Diese Klasse ist wichtig, um über CSS-Dateien das Design des Textes festzulegen. Die CSS-Dateien werden im Attribut `content_css` referenziert. Die restlichen Attribute definieren die Menüleiste des Editors. Dabei wird zunächst die Position der Leiste festgelegt. Anschließend werden die zur

Verfügung stehenden Knöpfe deklariert. Laut dem Konzept aus Kapitel 3.6 soll es nur Bedienelemente geben, um einen Text zu formatieren. Daher werden in der Menüleiste nur die Knöpfe integriert, die diese Funktionalität erfüllen.

Um den TinyMCE-Editor als Benutzer aufzurufen, soll laut Kapitel 3.6 der „Add Note“ Knopf gedrückt werden. In der *Actions*-Leiste der HTML-Datei soll also, sofern der entsprechende Benutzer Schreibrechte auf dieser Wikiseite hat, der Knopf „Add Note“ zur Verfügung stehen (siehe auch Kapitel 3.5). Das entsprechende Template für die *Actions*-Leiste befindet sich in der Datei *tabsholder.htm*. Einen Ausschnitt aus diesem Grundgerüst zeigt Quelltext 4.14.

Quelltext 4.14: HTML-Template der *Actions*-Leiste

```
<div class="tabs-holder">
    <!-- ... -->
    <ul class="actions">
        $[hasWriteAccess$
        <li>
            <div id="edit"><a>Add Note</a></div>
        </li>
        $hasWriteAccess]$
        <li>
            <a rel="bookmark" href="./index.html">Browse this Wiki</a>
        </li>
    </ul>
</div>
```

Dieses Template enthält nur die konditionale Substitution `$hasWriteAccess$`. Die konditionale Substitution wird in der entsprechenden Methode `tabsholder()` in der Klasse `Functions` ausgewertet. Diese Substitutionsmethode zeigt folgender Quelltext 4.15.

Quelltext 4.15: Substitutionsmethode für das Template in der Datei *tabsholder.htm*

```
public static TemplateSubstitution tabsholder(final WikiPage wikiPage, final
    FunctionParameters params) {
    return new TemplateSubstitution() {
        @Override
        public void putSubstitutions(Template template) {
                template.put("hasWriteAccess",new ConditionalSubstitution() {
                                @Override
                                public boolean test() {
                                        return wikiPage.mayEdit();
                                }
                        });
        }
    };
}
```

Diese Methode prüft, ob der Benutzer, der sich die Dateien vom Samba-Laufwerk herunterlädt, das Recht hat, die Wikiseiten zu bearbeiten. Wird die konditionale Substitution mit `true` ausgewertet, wird in der HTML-Datei der Button „Add Note“ angezeigt. Wird die Substitution mit `false` ausgewertet, wird der entsprechende HTML-Quelltext für den Button nicht in die HTML-Datei geschrieben und somit erscheint der Button auch nicht.

Damit der TinyMCE-Editor beim Drücken des vorgestellten „Add Note“ Knopfes erscheint, muss dies über JavaScript und CSS gesteuert werden. Folgender Quelltext 4.16 zeigt einen Ausschnitt aus der CSS Datei *offlineWiki.css*, welche auch im „MetaData“-Ordner zu finden ist.

Quelltext 4.16: CSS-Datei für das Einblenden des Editors TinyMCE

```
.hidden{
        visibility: hidden;
        display: none;
}
```

Hat also ein HTML-Element als Klassen-Attribut den Wert `hidden`, wird der entsprechende Inhalt dieses Elements im Browser nicht angezeigt. Im obigen HTML-Quelltext 4.12 des TinyMCE-Editors sieht man, dass das `<div>`-Element, welches um das Textfeld des Editors herumliegt, als Klassenattribut den Wert `hidden` besitzt. D.h., wird die HTML-Seite der Wikiseite im Browser geladen, sieht der Benutzer zunächst nicht den TinyMCE-Editor. Dieser soll erst erscheinen, wenn der „Add Note“ Knopf gedrückt wurde. Über Java-Script kann man diesen Knopfdruck „abfangen“.

Dazu wird die JavaScript-Bibliothek jQuery[2] benutzt. Diese Bibliothek stellt Funktionen zur Verfügung, um in einem HTML-DOM zu navigieren und diesen zu verändern. Durch die Bibliothek ist es sichergestellt, dass die Funktionen in verschiedenen Browsern funktionieren. Die JavaScript Dateien der jQuery-Bibliothek müssen, wie die Dateien für TinyMCE, zusätzlich dem Benutzer im MetaData-Ordner zur Verfügung gestellt werden. Der Quelltext 4.17 zeigt, wie der Knopfdruck abgefangen und der TinyMCE-Editor für den Benutzer sichtbar wird.

Quelltext 4.17: Einblenden des Editors TinyMCE

```
<script type="text/javascript" src="../_metaData/jquery-1.4.2.js"></script>
<script type="text/javascript" src="../_metaData/tiny_mce/tiny_mce.js"></script>
<script type="text/javascript">
// ...
jQuery('#edit').click(function(){
    jQuery("#hr").remove();
    jQuery("#offlineNote").addClass("hidden");
    jQuery("#tinymce").removeClass("hidden");
    var offlineNote = jQuery("#offlineNote").html();
    tinyMCE.activeEditor.setContent(offlineNote, {
        format: 'text'
    });
});
// ...
</script>
```

Der „Add-Note“ Knopf hat die „Id“ `edit` (siehe Quelltext 4.14). Über jQuery wird das Klicken auf diesen Knopf abgefangen (`jQuery('#edit').click()`). Anschließend werden Funktionen deklariert, die nach dem Klicken ausgeführt werden sollen. Das Konzept aus Kapitel 3.6 sieht vor, dass die Offline-Notiz durch einen horizontalen Strich vom Rest der Wikiseite getrennt werden soll. Im HTML-Quelltext besitzt dieser Strich die „Id“ `hr`. Nach dem Drücken des „Add-Note“ Knopfes wird dieser Strich, falls vorhanden, entfernt. Offline-Notizen auf den HTML-Seiten haben die „Id“ `offlineNote` (siehe Quelltext 4.21). Sollte schon eine Notiz auf der HTML-Seite bestehen, wird diese durch das Hinzufügen der Klasse `hidden` ausgeblendet. Dem `<div>`-Element des TinyMCE-Editors wird die Klasse `hidden` entfernt. Dadurch wird der TinyMCE-Editor für den Benutzer sichtbar. Durch die Methode `tinyMCE.activeEditor.setContent()` wird der Inhalt dieser Offline-Notiz in das Textfeld des TinyMCE-Editors übernommen. Sollte keine No-

[2] `http://jquery.com/`, aufgerufen am 19. Juli 2011

tiz bestehen, ist das Textfeld des Editors leer. Somit sieht der Benutzer nun den TinyMCE Editor mit dem evtl. schon bestehenden Text einer Offline-Notiz.

4.5.2 Speichern der Offline-Notiz

Für die Speicherung von HTML-Dateien sind die drei Methoden `loadFile()`, `saveFile()` und `getLocalPath()` von besonderer Bedeutung. Das Konzept für den lokalen Schreibzugriff aus Kapitel 3.6 orientiert sich bei der Speicherung an dem Konzept der Wiki-Software TiddlyWiki. TiddlyWiki besteht aus einer einzigen HTML-Datei. Beim Anlegen von Wikiseiten überschreibt der Java-Script Code diese HTML-Datei mit dem neuen Inhalt. Die drei genannten Methoden stammen aus dem Quelltext von TiddlyWiki und wurden mit kleinen Veränderungen in den JavaScript-Quelltext der HTML-Dateien für die Offline-Verison von Tricia übernommen. Das Copyright dieser drei Methoden und deren Hilfsmethoden liegt bei Jeremy Ruston, dem Entwickler von TiddlyWiki. Die Verwendung dieser Methoden wird gestattet. Die Methode `loadFile()` lädt, wie der Name schon sagt, eine lokale Datei in einen String. Die Methode `saveFile()` schreibt einen String in eine Datei. Die Methode `getLocalPath()` ermittelt den Speicherpfad einer Datei, abhängig vom darunterliegenden Betriebssystem. Das Schreiben bzw. Lesen von Dateien ist nur für die beiden Browser Firefox und Internet Explorer implementiert. Stellvertretend wird im Folgendem die Methode `saveFile()` bzw. `mozillaSaveFile()` näher vorgestellt (siehe Quelltext 4.18).

Quelltext 4.18: Speichern einer Datei über JavaScript im Internetbrowser Firefox

```
function saveFile(fileUrl,content){
        var r = mozillaSaveFile(fileUrl,content);
        if(!r)
                r = ieSaveFile(fileUrl,content);
        return r;
}

function mozillaSaveFile(filePath, content){
    if (window.Components) {
        try {
            netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
            var file = Components.classes["@mozilla.org/file/local;1"].createInstance(
                Components.interfaces.nsILocalFile);
            file.initWithPath(filePath);
            if (!file.exists())
                file.create(0, 0664);
            var out = Components.classes["@mozilla.org/network/file-output-stream;1"].
                createInstance(Components.interfaces.nsIFileOutputStream);
            out.init(file, 0x20 | 0x02, 00004, null);
            out.write(content, content.length);
            out.flush();
            out.close();
            return true;
        }
        catch (ex) {
            return false;
        }
    }
    return null;
}
```

Die Methode `saveFile()` versucht über die Methode `mozillaSaveFile()` den Inhalt in eine Datei zu schreiben. Wie der Name schon sagt, ist die Methode dafür gedacht,

falls der Benutzer die HTML-Datei mit dem Browser Firefox betrachtet. Funktioniert dies nicht, versucht die Methode den String mit der Methode `ieSaveFile()`, welche für den Browser Internet Explorer vorgesehen ist, zu schreiben. Im Folgenden wird kurz die Speicher-Methode für den Internetbrowser Firefox erläutert. Zunächst wird über die Methode `enablePrivilige("UniversalXPConnect")` das Recht vom Browser angefordert, lokale Dateien zu lesen. Wird dieses Recht gewährt, wird die Variable `file` mit Hilfe des übergebenen Dateipfades erzeugt. Diese Variable entspricht einer lokalen Datei. Existiert die Datei noch nicht, wird diese neu angelegt. Anschließend wird der String `content` in die neue Datei geschrieben.

Hat der Benutzer den Text seiner Notiz verfasst, kann er über den „Save"-Button die Notiz abspeichern. Dazu wurde die Funktion `saveNote()` im JavaScript-Code implementiert. Im HTML-Quelltext des Editors TinyMCE (siehe Quelltext 4.12) sieht man im `<form>` Element das Attribut `onsubmit=return saveNote();`. Wird also der „Save" Button gedrückt, wird die Funktion `saveNote()` aufgerufen. Auf diese Methode (siehe Quelltext 4.19) wird nun genauer eingegangen.

Quelltext 4.19: Abspeichern einer Offline-Notiz

```
function saveNote(){
    var input = tinyMCE.activeEditor.getContent({
        format: 'text'
    });
    var newContent = createNewHTMLContent(input);
    var localPath = getLocalPath(document.location.toString());
    saveFile(localPath, newContent);
    createChangeSetFile(input);
}
```

Zunächst wird in der Variablen `input` der vom Benutzer geschriebene Text gespeichert. Die Methode `createNewHTMLContent()`, welche später vorgestellt wird, nimmt als Argument die Variable `input` auf und erzeugt den neuen HTML-Quelltext der Wikiseite mit der neu geschriebenen Offline-Notiz. Das Ergebnis dieser Methode wird in der Variablen `newContent` abgelegt. Die Methode `getLocalPath()` generiert einen Dateipfad der aktuellen HTML-Datei. Die Methode `saveFile()` ist für die Speicherung der neuen HTML-Seite zuständig (siehe Quelltext 4.18). Die Methode nimmt als Argumente den Dateipfad sowie den vorher generierten neuen HTML-Quelltext. Die Methode `createChangeSetFileString()` ist für die Synchronisation wichtig und wird deshalb erst im nächsten Kapitel 4.6 behandelt. Wurde die Methode `saveNote()` vollständig ausgeführt, wird die aktuelle HTML-Seite, wie oben erläutert, neu geladen. Der Benutzer sieht nun die von der Methode `createNewHTMLContent()` neu generierte Wikiseite inklusive der Offline-Notiz.

Auf diese Methode wird nun genauer eingegangen. Die Implementierung ist im Quelltext 4.20 zu sehen.

Quelltext 4.20: Generierung des neuen HTML-Quelltextes

```
function createNewHTMLContent(newContent){
        var originalPath = document.location.toString();
        var localPath = getLocalPath(originalPath);
        var oldHtmlFile = loadFile(localPath);
        var indexOflineNote = oldHtmlFile.indexOf('<div id="offlineNote">');
        var numberOfCharacters = oldHtmlFile.indexOf('<div id="endOfOfflineNote">')-
            oldHtmlFile.indexOf('<div id="offlineNote">');
```

```
        var oldOfflineNote = oldHtmlFile.substr(indexOflineNote,numberOfCharacters);
        var newOfflineNote = '<div id="offlineNote"><hr id="hr">' + newContent + '</div>';
        var newHtmlFile = oldHtmlFile.replace(oldOfflineNote, newOfflineNote);
        return newHtmlFile;
}
```

Zunächst wird über die Methode `getLocalPath()` der Dateipfad der aktuellen HTML-Datei in die Variable `localPath` eingelesen. Über die Methode `loadFile` wird der Quelltext der aktuellen HTML-Datei in der Variablen `originalHTML` gespeichert. Aus diesem String wird nun eine evtl. schon vorhandene alte Offline-Notiz extrahiert. Offline-Notizen werden im HTML-Quelltext in ein `<div id="offlineNote">...</div>` Element geschrieben (siehe Quelltext 4.21). Im HTML-Template befindet sich direkt hinter diesem Element das `<div id="endOfOfflineNote">` Element. Dieses `<div>`-Element markiert das Ende der Notiz und dient dazu, den Quelltext für die Offline-Notiz aus dem HTML-Quelltext zu extrahieren.

Quelltext 4.21: HTML-Template für eine Offline-Notiz

```
<div id="offlineNote"> Hier steht der Text der Offline-Notiz</div>
<div id="endOfOfflineNote"></div>
```

Der nun extrahierte HTML-Quelltext der Notiz wird mit dem Quelltext für die neue Offline-Notiz ersetzt. Als Ergebnis liefert die Methode den Quelltext der gesamten HTML-Datei der Wikiseite inklusive des JavaScript Quelltextes als String zurück.

Der Rückgabewert dieser Methode wird in der Hauptmethode `saveNote()` (siehe Quelltext 4.19) zusammen mit dem Dateipfad der Methode `saveFile()` (siehe Quelltext 4.18) übergeben. Diese Methode überschreibt nun die HTML-Datei der Wikiseite inklusive des JavaScript Quelltextes.

4.6 Synchronisation

Das Kapitel 3.7 hat ein Konzept für die Synchronisation zwischen der Webapplikation und der Offline-Variante von Tricia erarbeitet. Dabei wurde das Konzept der „Two-Step"-Synchronisation entwickelt. Bei der „Two-Step"-Synchronisation wird im ersten Schritt der „Sync"-Ordner des lokalen Laufwerks mit dem „Sync"-Ordner auf dem SMB-Laufwerk synchronisiert. Dabei wird, falls eine Offline-Notiz angelegt wurde, die Datei „changeset.json" auf das SMB-Laufwerk kopiert. Auf die Generierung dieser „Changeset"-Datei geht der erste Teil dieses Kapitels ein. Beim zweiten Synchronisationsschritt wird der gesamte Wiki-Instanz-Ordner vom SMB-Laufwerk mit dem Wiki-Instanz-Ordner auf dem lokalen Laufwerk synchronisiert. Das Bereitstellen und Ausspielen der HTML-Dateien, Wikiseiten, Meta-Dateien, Index-Dateien und der Dateianhänge wurde schon in den vorherigen Kapiteln 4.3 und 4.4 vorgestellt. Im zweiten Teil dieses Kapitels wird daher nur die Implementierung des neuen „Sync"-Ordners und der „Changeset"-Datei vorgestellt. Im letzten Teil wird das Einspielen der „Changeset"-Datei in die Datenbank der Webapplikation erläutert.

4.6.1 Generierung der „Changeset"-Datei

Bei der „Two-Step"-Synchronisation ist es notwendig, eine lokale Datei zu schreiben, welche alle Offline-Notizen enthält, die seit der letzten Synchronisation angelegt wurden.

Diese Datei wird, wie auch die Offline-Notizen, über JavaScript generiert. Im JavaScript-Quelltext 4.19 sieht man in der Methode `saveNote()` den Aufruf der Methode `createChangeSetFile()`. Diese Methode ist für die Generierung der „Changeset"-Datei zuständig. Es wird also jedes Mal, wenn eine Offline-Notiz gespeichert werden soll, eine solche Datei generiert. Die evtl. schon vorhandene „Changeset"-Datei wird dabei überschrieben. Als Argument nimmt die Methode den zu speichernden Inhalt einer Offline-Notiz auf. Der folgende Quelltext 4.22 zeigt die Implementierung dieser Methode. Das Ziel ist die Generierung einer Datei, in der ein Array aus JSON-Objekten gespeichert ist. Jedes JSON-Objekt entspricht dabei einer Offline-Notiz auf der HTML-Datei einer Wikiseite.

Quelltext 4.22: Generierung der „Changeset"-Datei

```
function createChangeSetFile(content){
    var wikiPageId = "$wikiPageId$";
    var changeSetFile = loadFile(getChangeSetFilePath());
    var changeSetString = "";
    var change = {
        "wikiPageId": wikiPageId,
        "content": content
    };
    if (!changeSetFile) {
        var newChangesetJsonArray = new Array(1);
        newChangesetJsonArray[0] = change;
        changeSetString = JSON.stringify(newChangesetJsonArray);
    }
    else {
        var oldChangesetJsonArray = JSON.parse(changeSetFile);
        var oldJsonArrayLength = jQuery(oldChangesetJsonArray).length;
        var newChangesetJsonArray = new Array(oldJsonArrayLength + 1);
        for (var i = 0; i < oldJsonArrayLength; i++) {
            newChangesetJsonArray[i] = oldChangesetJsonArray[i];
        }
        newChangesetJsonArray[oldJsonArrayLength] = change;
        changeSetString = JSON.stringify(newChangesetJsonArray);
    }
    saveFile(getChangeSetFilePath(), changeSetString);
}

function getChangeSetFilePath(){
    var originalPath = document.location.toString();
    var splittedPath = originalPath.split("/");
    var changeSetDocumentPath = "file:";
    for (var i = 1; i < splittedPath.length - 1; i++) {
        changeSetDocumentPath = changeSetDocumentPath + "/" + splittedPath[i];
    }
    changeSetDocumentPath = changeSetDocumentPath + "/../sync/changeset.json";
    return getLocalPath(changeSetDocumentPath);
}
```

In der Variablen `wikiPageId` wird die „Id" der Wikiseite gespeichert. Dazu wird über das schon vorgestellte Template System von Tricia bei der Generierung der HTML-Datei der Platzhalter `$wikiPageId$` mit der „Id" der Wikiseite ersetzt. Diese „Id" wird später beim Schreiben der „Changeset"-Datei benötigt. In die Variable `changeSetFile` wird über die Methode `loadFile()`, welche von TiddlyWiki adaptiert wurde (siehe Kapitel 4.5.2), die alte „Changeset"-Datei, falls sie schon existiert, eingelesen.

Über die Methode `getChangeSetFilePath()` wird der Pfad der „Changeset"-Datei herausgefunden. Die Datei soll sich immer im „Sync"-Ordner unterhalb des Wiki-Instanz-

Ordners befinden (siehe Kapitel 3.7). In der Methode `getChangeSetFilePath()` wird der Pfad der aktuellen HTML-Datei bestimmt. Dieser hat folgendes Format: „{*Pfad zum Speicherort des Wikis*}/{*wikiInstanceUrlName*}/*pages*/{*wikiPageUrlName*}.*html*“. Von diesem Pfad werden die letzten beiden Teile weggeschnitten und durch den Pfad zum Speicherort der „Changeset“-Datei ersetzt. Der Pfad der „Changeset“-Datei hat danach folgendes Format: „{*Pfad zum Speicherort des Wikis*}/{*wikiInstanceUrlName*}/*sync*/*changeset.json*“.

Als nächstes wird in der Methode `createChangeSetFile()` die Variable `changeSetString` initialisiert. Diese Variable wird am Schluss in die Datei „changeset.json“ geschrieben. Die Variable `change` wird mit einem JSON-Objekt belegt. Das JSON-Objekt besitzt die beiden Attribute „wikiPageId“ und „content“. Im Attribut „wikiPageId“ wird die vorher in der Variablen `wikiPageId` abgelegte „Id“ der Wikiseite gespeichert. Im Attribut „content“ wird der Inhalt der Offline-Notiz, welche der Funktion `createChangeSetFile()` als Parameter übergeben wird, gespeichert.

Nun wird über eine „If“-Abfrage herausgefunden, ob bereits eine „Changeset“-Datei im „Sync“-Ordner existiert. Existiert diese Datei noch nicht, wird ein neues Array angelegt und das vorher erzeugte JSON-Objekt darin abgelegt. Anschließend wird dieses Array in einen String umgewandelt und in der Variablen `changeSetString` abgelegt. Existiert die „Changeset“-Datei schon, wurde bereits mindestens eine Offline-Notiz abgespeichert. Es wird nun das Array, welches sich in der schon vorhandenen „Changeset“-Datei befindet, eingelesen. Als nächstes wird ein neues Array erzeugt, welches Platz für die JSON-Objekte aus der „Changeset“-Datei hat und das neue JSON-Objekt aus der Variablen `change` aufnehmen kann. Dem neuen Array werden nun alle JSON-Objekte des alten Arrays hinzugefügt. Zusätzlich wird das neue JSON-Objekt hinzugefügt. Anschließend wird dieses Array wieder als String in die Variable `changeSetString` abgelegt. Zum Schluss wird dieser String in die neue Datei „changeset.json“ über die Methode `saveFile()` (siehe Quelltext 4.18) geschrieben.

Wird eine Offline-Notiz nach dem erstmaligen Speichern noch einmal geändert, wird diese Notiz als JSON-Objekt hinten an das Array angehängt. Auf dem Server muss dann beim Auslesen der „Changeset“-Datei darauf geachtet werden, dass immer nur die zuletzt gespeicherte Version einer Offline-Notiz in die Datenbank eingespielt wird (siehe Kapitel 4.6.3). Folgender Ausschnitt 4.23 zeigt ein Beispiel, wie ein Array von JSON-Objekten in die „Changeset“-Datei geschrieben wird.

Quelltext 4.23: Inhalt einer „Changeset“-Datei

```
[
{"wikiPageId":"19y6qlzrlibj9","content":"<p>Termin: 15. Oktober 2011</p>"},
{"wikiPageId":"133y2fn1mx2xs","content":"<p>Die neue Offline-Funktionalität muss ich
    ausprobieren</p>"},
{"wikiPageId":"19y6qlzrlibj9","content":"<p>Termin: 15. Oktober 2011 um 11 Uhr</p>"}
]
```

In diesem Beispiel wurden zwei Notizen angelegt. Bei der ersten Notiz wurde ein Termin auf der Wikiseite mit der „Id“ `19y6qlzrlibj9` angelegt. Die zweite Notiz wurde auf der Wikiseite mit der „Id“ `133y2fn1mx2xs` verfasst. Zum Termin in der ersten Notiz wurde anschließend noch eine Uhrzeit hinzugefügt.

4.6.2 Datei-Synchronisation

Für die„ Two-Step"-Synchronisation muss ein „Sync"-Ordner unterhalb des Wiki-Instanz-Ordners geschaffen werden (siehe Kapitel 3.7). Dazu wird die Klasse `SyncDirectory` implementiert. `SyncDirectory` ist eine weitere Subklasse von `VirtualDirectory`. Das Klassendiagramm aus Abbildung 4.5 wird daher in Abbildung 4.13 um diese Klasse erweitert. Für die „Changeset"-Datei, welche beim ersten Synchronisationsschritt auf das SMB-Laufwerk kopiert wird, wird die Klasse `SyncDocument` implementiert. `SyncDocument` ist eine Subklasse von der Klasse `FacadeDocument`. Auch dies ist in der Abbildung 4.13 zu sehen.

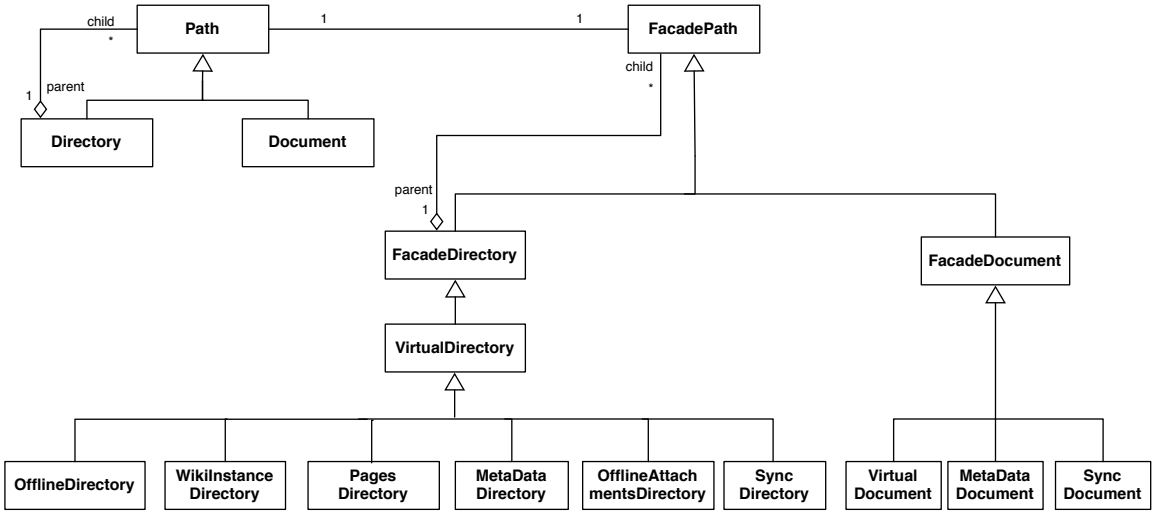


Abbildung 4.13: Erweiterung der virtuellen Ordnerstruktur um die Klassen `SyncDirectory` und `SyncDocument`

Wie man in Kapitel 4.3.2 schon gesehen hat, wird in der Methode `getChildren()` in den Subklassen von `VirtualDirectory` der Inhalt eines Ordners erzeugt. Um den „Sync"-Ordner unterhalb des Wiki-Instanz-Ordners einzuhängen, wird in der `getChildren()`-Methode der Klasse `WikiInstanceDirectory` eine Instanz der Klasse `SyncDirectory` dem Iterator, welcher von der Methode zurückgegeben wird, hinzugefügt. Die `getChildren()` Methode der Klasse `SyncDirectory` liefert immer einen leeren Iterator zurück, denn dieser Ordner muss immer leer sein, damit die „Changeset"-Datei (repräsentiert durch die Klasse `SyncDocument`) beim Synchronisieren auf das SMB-Laufwerk kopiert wird.

In Kapitel 3.7.2 wurde erwähnt, dass die Implementierung der Synchronisation auf die beiden Programme „rsync" und „DirSyncPro" ausgerichtet wird. Um die Implementierung der Synchronisation zu verstehen, wird nun kurz auf die Funktionsweise dieser beiden Synchronisationsprogramme eingegangen. Die Erläuterung erfolgt anhand des konkreten Falls der Spiegelung des lokalen „Sync"-Ordners auf den „Sync"-Ordner einer Wiki-Instanz auf dem SMB-Laufwerk. Dieser Synchronisationsschritt muss vom Benutzer einmalig manuell im Synchronisationsprogramm eingerichtet werden.

Bei „rsync" wird bei der unidirektionalen Synchronisation, also Spiegelung eines Ord-

ners auf einen anderen, zunächst im lokalen „Sync"-Ordner nachgesehen, welche Dateien dort vorhanden sind. Laut dem Konzept aus Kapitel 3.7 soll dort nur die Datei „changeset.json" (siehe vorheriges Kapitel 4.6.1) liegen. Anschließend sucht „rsync" auf dem entsprechenden „Sync"-Ordner des SMB-Laufwerks, ob diese Datei dort vorhanden ist. Wie oben erläutert sollte der „Sync"-Ordner auf dem SMB-Laufwerk immer leer sein. Das Programm „rsync" legt nun eine temporäre Datei im „Sync"-Ordner auf dem SMB-Laufwerk an. In diese Datei wird der Dateiinhalt der lokalen Datei „changeset.json" geschrieben. Anschließend wird die temporäre Datei in „changeset.json" umbenannt. Das Sequenzdiagramm in Abbildung 4.15 zeigt die Operationen, welche „rsync" auf dem SMB-Laufwerk bei der Synchronisation ausführt.

Die Synchronisation bei „DirSync Pro" läuft ähnlich ab. Es wird im Gegensatz zu „rsync" bei der Synchronisation keine temporäre Datei angelegt, welche später umbenannt wird. Stattdessen wird sofort die richtige Datei auf dem SMB-Laufwerk geschrieben.

Das Klassendiagramm in Abbildung 4.14 zeigt die wichtigsten Attribute und Methoden der beiden Klassen `SyncDirectory` und `SyncDocument`.

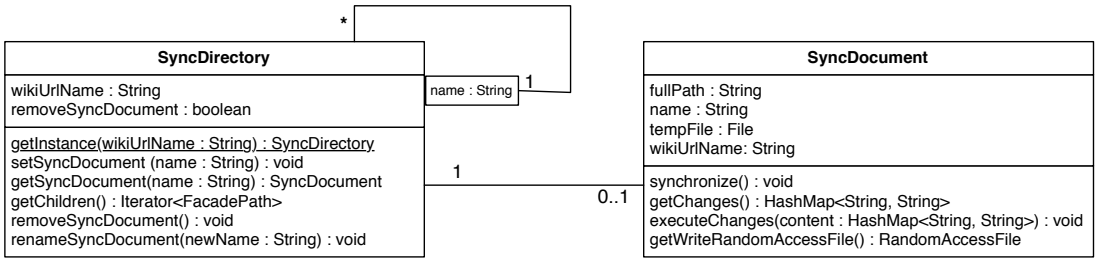


Abbildung 4.14: Klassendiagramm von `SyncDocument` und `SyncDirectory`

Die Klasse `SyncDirectory` hat als Attribut eine statische HashMap, welche Instanzen von der Klasse `SyncDirectory` verwaltet. Über die Methode `getInstance()` erhält man die entsprechende Instanz aus dieser HashMap. Wenn die HashMap nicht die gewünschte Instanz enthält, wird `null` zurückgeliefert. Die Methode `getInstance()` in Verbindung mit der HashMap dient dazu, dass es immer nur eine Instanz der Klasse `SyncDirectory` pro Wiki-Instanz-Ordner gibt. Einer solchen Instanz ist höchstens eine Instanz der Klasse `SyncDocument` zugeordnet. D.h., durch die HashMap gibt es immer nur ein Paar von „Sync"-Directory und „Changeset"-Datei. Die Klasse `SyncDocument` besitzt, wie schon die Klasse `VirtualDocument`, ein File-Attribut. In diese temporäre Datei wird beim Synchronisieren durch das Synchronisationsprogramm der Inhalt der „Changeset"-Datei geschrieben. Zum Schreiben dieser temporären Datei gibt es die Methode `getWriteRandomAccessFile()`. Über die Methoden `setSyncDocument()` bzw. `getSyncDocument()` wird das SyncDocument-Attribut in der Klasse `SyncDirectory` gesetzt bzw. abgefragt. Die Methode `removeSyncDocument()` löst die Verbindung zwischen `SyncDirectory` und `SyncDocument` auf. Mit Hilfe der Methode `renameSyncDocument()` kann das String-Attribut `name` der Klasse `SyncDocument` verändert werden. Die Methode `synchronize()` in der Klasse `SyncDocument` wird dann aufgerufen, wenn die „Changeset"-Datei vollständig geschrieben wurde. Die Methode `synchronize()` liest die temporäre Datei aus und schreibt die Offline-Notizen, die in einem JSON-Array in der Datei gespeichert sind, in die Datenbank. Auf diese Methode

wird später in Kapitel 4.6.3 noch genauer eingegangen.

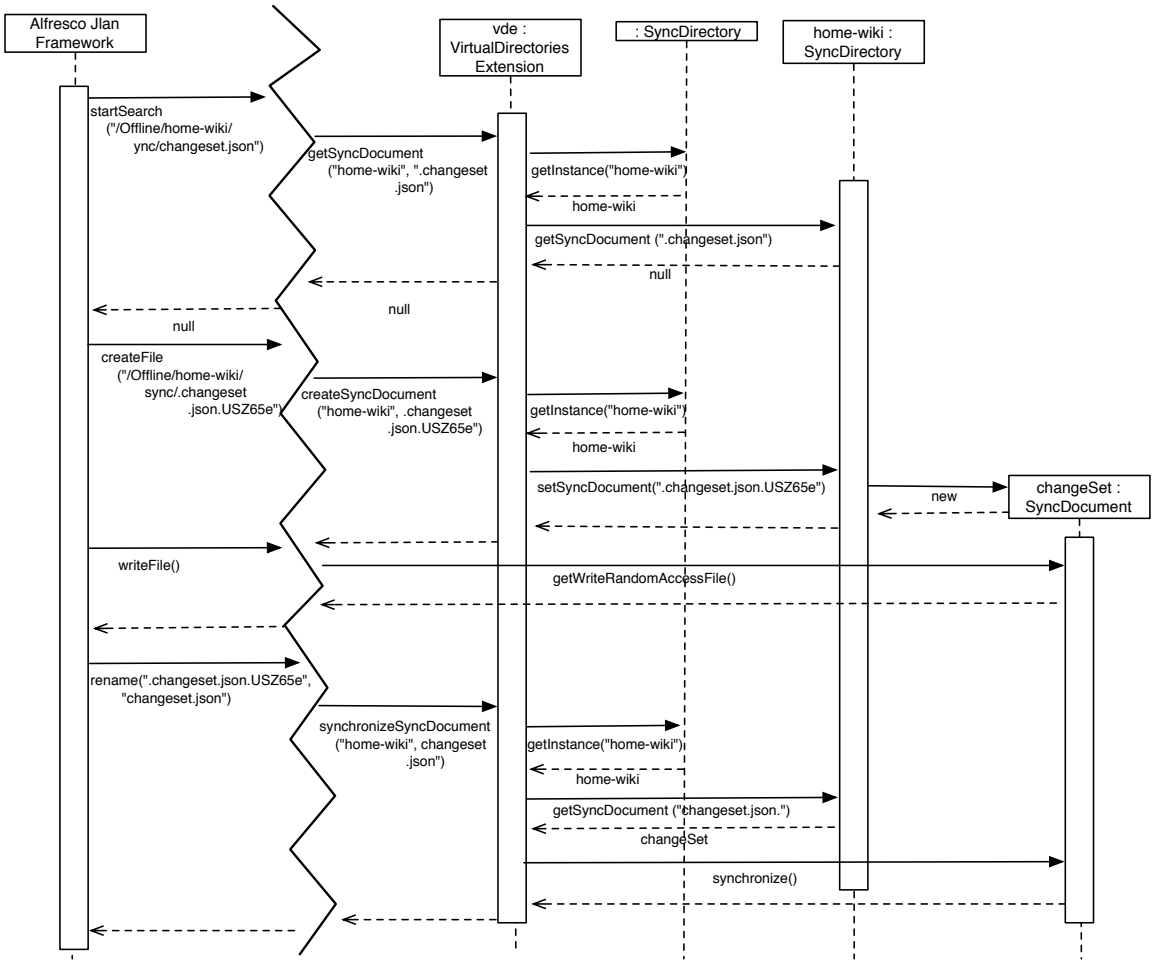


Abbildung 4.15: Sequenzdiagramm der Synchronisation der „Changeset"-Datei

Das Sequenzdiagramm in Abbildung 4.15 zeigt, wie die eben vorgestellten Methoden beim Synchronisieren einer „Changeset"-Datei durch das Synchronisationsprogramm „rsync" ins Spiel kommen. In diesem Sequenzdiagramm wird also der lokale „Sync"-Ordner, in dem sich die Datei „changeset.json" befindet, mit dem „Sync"-Ordner der entsprechenden Wiki-Instanz auf dem SMB-Laufwerk synchronisiert. Zwischen dem Alfresco Jlan Framework und der Klasse `VirtualDirectoriesExtension`, symbolisiert durch den gezackten Trennstrich, befinden sich noch weitere Klassen, die an diesem Prozess teilnehmen. Dies sind u.a. die Klasse `TriciaDiskDriver`, welche die Schnittstelle des Jlan Frameworks implementiert und die Klasse `FacadePath`, deren wichtigste Methode `getFacadePathbyPath()` schon in Kapitel 4.3.2 vorgestellten wurde. Das Zusammenspiel dieser Klassen wurde schon in Kapitel 4.3 erläutert und wird daher nun aus Vereinfachungsgründen weggelassen.

Wie oben beschrieben, prüft das Programm „rsync" zunächst, ob die Datei „changeset.json" schon im „Sync"-Ordner vorhanden ist. Das Jlan Framework führt dazu die Me-

thode `startSearch()` aus. Als Ergebnis wird dem Jlan Framework `null` zurückgeliefert. D.h., es existiert keine Datei „changeset.json“ in dem „Sync“-Ordner. „rsync“ versucht nun, wie oben angesprochen, eine temporäre Datei „changeset.json.USZ65e“ im „Sync“-Ordner anzulegen. Über die Methode `setSyncDocument()` wird in der Instanz „homewiki“ vom Typ `SyncDirectory` das `SyncDocument`-Attribut gesetzt. In die temporäre Datei dieser Instanz der Klasse `SyncDocument` schreibt nun das Jlan Framework den Inhalt der lokalen Datei „changeset.json“. Ist der Schreibvorgang beendet, benennt „rsync“ die temporäre Datei in den Namen der richtigen Datei um. Bei der Umbenennung in den Namen „changeset.json “ wird bemerkt, dass die lokale Datei vollständig in die temporäre Datei der Klasse `SyncDocument` geschrieben wurde. Deshalb wird nun die Methode `synchronize()` auf der Instanz der Klasse `SyncDocument` ausgeführt. Die Implementierung dieser Methode zeigt das nächste Kapitel 4.6.3. Würde das Synchronisationsprogramm „DirSync Pro“ eingesetzt werden, entfällt der letzte Schritt der Umbenennung und die Methode `synchronize()` würde sofort nach Abschluss des Schreibvorganges ausgeführt werden.

4.6.3 Verarbeitung der „Changeset“-Datei

Wurde die „Changeset“-Datei vollständig in die temporäre Datei der entsprechenden Instanz der Klasse `SyncDocument` geschrieben, wird anschließend die Methode `synchronize()` auf dieser Instanz ausgeführt. Die Implementierung der Methode zeigt der Quelltext 4.24.

Quelltext 4.24: Einspielen der „Changeset“-Datei in die Datenbank

```
public void synchronize() {
        if (getName().equals(CHANGESET_FILE_NAME)) {
                HashMap<String, String> offlineNotes = getChanges();
                if (offlineNotes != null) {
                        executeChanges(offlineNotes);
                }
                SyncDirectory.getInstance(wikiUrlName).removeSyncDocument();
        }
}

private HashMap<String, String> getChanges() {
        HashMap<String, String> offlineNotes = new HashMap<String, String>();
        BufferedReader br;
        String content = "";
        try {
                br = new BufferedReader(new InputStreamReader(new FileInputStream(tempFile
                    )));
                StringBuffer contentOfFile = new StringBuffer();
                String line;
                while ((line = br.readLine()) != null) {
                    contentOfFile.append(line);
                }
                content = contentOfFile.toString();
                if(!content.startsWith("[") || !content.endsWith("]")){
                        return null;
                }
                JSONArray a = new JSONArray(content);
                for (int i = 0; i < a.length(); i++) {
                        JSONObject jo = (JSONObject) a.get(i);
                                offlineNotes.remove(jo.get("wikiPageId"));
                                offlineNotes.put(jo.getString("wikiPageId"), jo.getString(
                                    "content"));
```

```
                }
        } catch (Exception e) {
                Loggers.serviceLog.error("Could not parse "+CHANGESET_FILE_NAME, e);
                return null;
        }
        return offlineNotes;
}

private void executeChanges(HashMap<String, String> offlineNotes) {
        Set<String> keySet = offlineNotes.keySet();
        for (String key : keySet) {
                WikiPage wikiPage = WikiPage.SCHEMA.
                    getRegisteredWritableCopyAndCheckMayEdit(key);
                String time = new SimpleDateFormat("MMMM d, yyyy").format(new Timestamp(
                    System.currentTimeMillis()));
                String newContent = wikiPage.content.get()+"<hr><p><b>"+SessionLocal.
                    getUser().getName()+" ("+time+"):</b></p>"+offlineNotes.get(key);
                wikiPage.content.set(newContent);
                wikiPage.persist();
        }
}
```

Zunächst wird noch einmal überprüft, ob der Dateiname der temporären Datei „changeset.json“ ist. Ist dies der Fall, wird über die Methode `getChanges()` die temporäre Datei ausgelesen und als Ergebnis eine HashMap zurückgeliefert. Wie in Kapitel 4.6.1 zu sehen ist, wird in die Datei „changeset.json“ ein Array von JSON-Objekten geschrieben. Über einen `BufferedReader` wird der Inhalt dieser „Changeset“-Datei in den String `content` eingelesen. Anschließend wird ein neues `JSONArray`-Objekt mit Hilfe dieses Strings erzeugt. Die einzelnen JSON-Objekte in diesem Array haben die Attribute „wikiPageId“ und „content“. Aus den JSON-Objekten wird eine HashMap gebildet. Als Schlüssel dient dabei die Wikiseiten-„Id“. Der Wert ist der Inhalt der Offline-Notiz. Haben mehrere JSON-Objekte dieselbe Wikiseiten-„Id“, wird die alte Offline-Notiz aus der HashMap gelöscht und die neueste eingefügt.

Die Methode `executeChanges()` übernimmt als Attribut diese HashMap. Anhand der Schlüssel der HashMap wird die entsprechende Wikiseite gefunden. Der dazugehörige Wert in der HashMap wird nun mit Ergänzung einiger Meta-Daten an das Ende der jeweiligen Wikiseite angehängt und in der Datenbank gespeichert.

Anschließend wird in der Methode `synchronize()` noch die Bindung zwischen den Instanzen der Klassen `SyncDirectory` und `SyncDocument` aufgehoben. Außerdem wird die temporäre Datei wieder gelöscht.

5 Zusammenfassung

Ziel dieser Arbeit war es, ein Konzept für einen Offline-Zugriff auf eine Enterprise 2.0 Plattform zu entwickeln und dieses Konzept anschließend zu implementieren. Bei dieser Enterprise 2.0 Plattform handelt es sich um die Software Tricia, welche von der InfoAsset AG[1] vertrieben wird.

In Kapitel 1 wurde die Software Tricia vorgestellt. Tricia ist eine Webapplikation, die der Benutzer in einem Internetbrowser bedient. Bei Tricia handelt es sich um ein Wikisystem, was auf die Bedürfnisse von Unternehmen ausgerichtet ist. Unternehmen können in Tricia firmeninternes sowie firmenexternes Wissen ablegen und verwalten. Besonderes Augenmerk liegt in Tricia auf dem Konzept der hybriden Wikis. Bei hybriden Wikis kann unstrukturierter Inhalt mit strukturierten Informationen verknüpft werden.

In Kapitel 1 wurde außerdem die allgemeine Motivation für die Implementierung einer Offline-Funktionalität in eine Webanwendung herausgearbeitet. Es wurde kritisch festgestellt, dass die Internetabdeckung in der westlichen Welt schon sehr gut ist und weiter wächst und somit der Bedarf einer Offline-Funktionalität immer weiter abnimmt. Auf der anderen Seite wurde festgestellt, dass es durchaus immer noch Situationen gibt, z.B. auf Geschäftsreisen, bei dem ein Benutzer keine Internetverbindung zur Verfügung hat. Und gerade auf Geschäftsreisen ist ein Zugriff auf das Unternehmens-Wissen essenziell für den Geschäftserfolg. Aufgrund dieser Überlegung wurde die Implementierung einer Offline-Funktionalität in Tricia weiterverfolgt.

In Kapitel 2 wurden dann drei Ansätze vorgestellt, die grundsätzlich für die Implementierung eines Offline-Zugriffs auf eine Webapplikation geeignet sind. Diese drei Ansätze wurden daraufhin hinsichtlich ihrer Eignung für die Implementierung einer Offline-Funktionalität in Tricia untersucht.

Als erstes wurde der Ansatz einer Browser-Plugin basierten Offline-Funktionalität untersucht (siehe Kapitel 2.1). Beispielhaft für die Browser-Plugins wurde das Google Gears Plugin vorgestellt, welches potentiell für Tricia in Frage kommen würde. Als Vorteil dieses Ansatzes wurde herausgearbeitet, dass man als Entwickler eine umfangreiche API für die Implementierung der eigenen Offline-Funktionalität zur Verfügung gestellt bekommt. Als großer Nachteil wurde die Abhängigkeit vom Plugin-Hersteller bemängelt. Als Entwickler ist man abhängig davon, dass der Hersteller des Plugins sein eigenes Produkt weiter pflegt. Als Negativbeispiel dient hierbei wiederum Google Gears, was im Jahr 2011 eingestellt wurde. Da Gears das einzige Plugin war, welches für Tricia in Frage gekommen wäre, wurde von diesem Ansatz Abstand genommen.

Der zweite Ansatz in Kapitel 2.2 hat sich mit den Offline-Features in HTML5 beschäftigt. Dazu wurden die Möglichkeiten des Application Cache und des Offline Storage vorgestellt. Man bekommt als Entwickler, ähnlich wie beim Ansatz der Browser-Plugins, eine JavaScript-API an die Hand. Als Nachteil hat sich erwiesen, dass die Spezifikation der

[1] `http://www.infoasset.de/`, aufgerufen am 9. August 2011

Offline-Möglichkeiten noch nicht fertig gestellt und noch uneinheitlich sind. Die Browserhersteller konnten sich noch nicht auf eine einheitliche Spezifikation der Offline-Möglichkeiten in HTML5 einigen. Da die HTML5 Features nur über JavaScript angesprochen werden können, wurde außerdem festgestellt, dass dadurch die bestehende Trennung von Front-End- und Back-End-Quelltext in Tricia aufgeweicht werden würde. Das Kapitel kam daher zu dem Entschluss, dass HTML5 für die Implementierung einer Offline-Funktionalität in Tricia nicht der richtige Ansatz ist.

Als dritter Ansatz wurde ein dateibasierter Offline-Zugriff vorgestellt (siehe Kapitel 2.3). Beim dateibasierten Ansatz stellt die Webapplikation Dateien für den Benutzer zur Verfügung, die sich dieser auf den lokalen Rechner herunterlädt. Auf die lokal gespeicherten Dateien kann der Benutzer ohne Internetverbindung zugreifen, um deren Inhalte zu betrachten. Außerdem kann der Benutzer Veränderungen an diesen lokalen Dateien vornehmen. Die lokalen veränderten Daten sollen dann, sobald wieder eine Internetverbindung besteht, mit der Webapplikation synchronisiert werden können. Ein Nachteil dieses Ansatzes für die Implementierung in Tricia ist, dass die Daten, die in Tricia in einer relationalen Datenbank gespeichert sind, in eine Dateiform umgewandelt werden und dem Benutzer ausgeliefert werden müssen. Als Vorteil hat sich aber herausgestellt, dass dieser Ansatz ohne große Veränderungen an der schon bestehenden Webapplikation implementiert werden kann. Eine Offline-Funktionalität kann als neues Plugin in die bestehende Architektur in Tricia integriert werden. Für den Dateiaustausch mit dem Benutzer bietet sich das schon implementierte Samba-Laufwerk in Tricia an. Aufgrund der großen Vorteile wurde der dateibasierte Ansatz für die weitere Konzeption und Implementierung eines Offline-Zugriffs auf Tricia gewählt.

Das Kapitel 3 entwickelte ein Konzept, wie der dateibasierte Ansatz für einen Offline-Zugriff in Tricia umgesetzt werden kann.

Das Kapitel 3.1 hat dazu zunächst die Haupfunktionalitäten der Webanwendung aus Benutzersicht herausgearbeitet. Dabei wurde festgestellt, dass der Inhalt der hybriden Wikiseiten der wichtigste Bestandteil von Tricia ist. Auch die Möglichkeit, Dateianhänge an Wikiseiten anzuheften sowie die Suchfunktion wurden als elementare Features der Webanwendung von Tricia identifiziert. Anschließend stellte sich die Frage, welche Features der Benutzer der Webanwendung auch in einer Offline-Version von Tricia benötigt. Für den Offline-Lesezugriff wurde festgestellt, dass für den Benutzer vor allem der Inhalt der hybriden Wikiseiten von Interesse ist, sowie der zusätzlich dazu abgelegten Informationen (Dateianhänge, Tags, etc.). Für den Schreibzugriff wurde festgestellt, dass ein Benutzer im Offline-Betrieb nicht das Bedürfnis hat, ganze Wikiseiten zu ändern oder neu anzulegen. Der Benutzer ist eher daran interessiert, kurze Notizen an Wikiseiten anheften zu können.

Im Kapitel 3.2 wurde ein Grundkonzept erstellt, wie der dateibasierte Ansatz für Tricia aus Benutzersicht funktionieren kann. Benutzer sollen sich die Offline-Dateien über das schon implementierte Samba-Laufwerk von Tricia herunterladen können. Die lokal veränderten Daten sollen dann mit diesem Samba-Laufwerk wieder synchronisiert werden können. Es wurde außerdem festgelegt, dass jede Wikiseite einer Datei auf dem Samba-Laufwerk entsprechen soll.

Anhand der herausgearbeiteten Features für eine Offline-Version von Tricia beschäftigte sich Kapitel 3.3 mit der Frage, welches Format die Offline-Dateien haben sollen. Dabei wurde zwischen HTML-Dateien und strukturierten Dateien abgewogen. Der große Vorteil bei einer Implementierung der Dateien im HTML-Format ist, dass diese relativ einfach zu

implementieren sind und keine Intelligenz auf dem Client, außer einem Internetbrowser, voraussetzen. Die Hauptanforderungen an eine Offline-Version von Tricia können durch HTML-Dateien abgedeckt werden. Strukturierte Dateien haben den Nachteil, dass eine abgespeckte Version der Tricia-Serveranwendung implementiert werden müsste, welche mit den Dateien umgehen kann. Dieser Ansatz wurde auf Grund des zu hohen Aufwands verworfen und sich für den Ansatz der HTML-Dateien entschieden.

Das Kapitel 3.4 hat eine Ordnerstruktur erarbeitet, in welcher die HTML-Dateien auf dem SMB-Laufwerk angeboten werden. Als Ergebnis wurde ein neuer Standardordner eingeführt, der nur für die Offline-Funktionalität zuständig ist. In diesem Ordner wird pro Wiki-Instanz ein Ordner angeboten. In diesen Wiki-Instanz-Ordnern befinden sich neben anderen Ordnern ein „Pages"-Ordner. Dieser „Pages"-Ordner enthält die HTML-Dateien, welche jeweils eine Wikiseite der entsprechenden Wiki-Instanz repräsentieren.

Das nächste Kapitel 3.5 hat ein Konzept entwickelt, wie der Lesezugriff über die HTML-Dateien gestaltet werden kann. Dabei wurde zunächst ein Entwurf erarbeitet, wie die Wikiseiten in den HTML-Dateien dargestellt werden sollen. Es wurde darauf geachtet, dass das Design der HTML-Dateien dem Design der Wikiseiten in der Webapplikation entspricht. Bis auf Kleinigkeiten entspricht das äußerliche Design der HTML-Dateien dem der Webanwendung. Um die fehlende Suchfunktion in der Offline-Version von Tricia einigermaßen zu ersetzen, wurde ein Konzept für eine Index-Datei erarbeitet. Eine Index-Datei soll als eine Art Inhaltsverzeichnis über alle HTML-Dateien der aktuellen Wiki-Instanz dienen. Zum Schluss wurde die Problematik der Referenzierungen in der Offline-Version von Tricia angesprochen. Dabei wurde festgestellt, dass Referenzierungen nur auf Wikis, Wikiseiten, Ordner und Dateien möglich sind.

Anschließend wurde in Kapitel 3.6 ein Konzept für das Anlegen von Notizen auf den HTML-Dateien gezeigt. Das Kapitel hat sich dabei dem Konzept der Wiki-Software TiddlyWiki[2] bedient. Über eine Schaltfläche auf der HTML-Seite soll ein Benutzer ein Textfeld für das Verfassen seiner Notiz öffnen können. Anschließend soll der Benutzer diese Notiz in den HTML-Dateien speichern können.

Das letzte Kapitel 3.7, welches sich mit der Konzeption eines dateibasierten Ansatzes beschäftigte, zeigt einen Entwurf für die Synchronisation der lokal angelegten Notizen mit der Webanwendung. Dazu wurde das Konzept der „Two-Step"-Synchronisation entwickelt. Zunächst wurde dabei herausgestellt, dass für diese Art der Synchronisation eine neue Datei generiert werden muss, die alle im Offline-Modus angefertigten Notizen speichert. Diese Datei wird bei der „Two-Step"-Synchronisation im ersten Schritt auf das SMB-Laufwerk von Tricia synchronisiert. Die vorher generierte Datei wird dabei auf dem Server ausgelesen und die Notizen werden in die Datenbank eingespielt. Im zweiten Schritt wird der Wiki-Instanz-Ordner auf dem SMB-Laufwerk mit dem lokalen Wiki-Instanz-Ordner synchronisiert.

Das letzte große Kapitel 4 beschäftigte sich damit, wie das in Kapitel 3 erarbeitete Konzept implementiert wird.

In Kapitel 4.1 wird die Plugin-Architektur von Tricia vorgestellt. Dabei wurde das neue Plugin „offlineWiki" eingeführt. In diesem Plugin wird der dateibasierte Ansatz implementiert.

Das nächste Kapitel 4.2 zeigte die schon vorhandene Implementierung des SMB-Lauf-

[2] `http://www.tiddlywiki.com/`, aufgerufen am 9. August 2011

werks im Zusammenspiel mit der Dateisicht auf Tricia. Dazu wurden die drei Klassen `Path`, `Directory` und `Document` sowie die dazugehörigen Fassaden-Klassen `FacadePath`, `FacadeDirectory` und `FacadeDocument` vorgestellt. Für die Implementierung des SMB-Laufwerks wird das Alfresco Jlan Server Framework[3] eingesetzt. Das Kapitel erläutert die Implementierung der zur Verfügung gestellten Schnittstelle sowie die Integration der Dateisicht auf Tricia in diese Implementierung.

In Kapitel 4.3 wurde gezeigt, wie die Ordnerstruktur für die Offline-Dateien in die bestehende Implementierung des SMB-Laufwerks integriert wird. Dazu wurden zunächst die oben schon erwähnten Fassaden-Klassen um virtuelle Ordner und Dateien erweitert. Dabei wurde für jeden Ordner bzw. Dateityp eine Subklasse von `FacadeDirectory` bzw. `FacadeDocument` implementiert. Als nächstes wurde auf die Methoden eingegangen, die für die Generierung des Inhalts eines Ordners aus der Offline-Ordner-Struktur notwendig sind.

Kapitel 4.4 zeigt, wie die Generierung der Dateien aus der Ordnerstruktur implementiert ist. Dabei wurde gezeigt, dass für die Implementierung der HTML-Dateien das Template-System von Tricia eingesetzt wird. Durch das Template-System lassen sich Platzhalter in einem HTML-Quelltext deklarieren, welche zur Laufzeit ersetzt werden. Für die HTML-Dateien der Wikiseiten und die HTML-Dateien der Index-Dateien wurden dafür jeweils eigene HTML-Grundgerüste mit Platzhaltern angelegt. Bei den Meta-Dateien wurde auf die schon vorhandenen Dateien der Webapplikation zurückgegriffen.

Die Implementierung des Schreibzugriffs auf die HTML-Dateien wurde in Kapitel 4.5 gezeigt. Dazu wurde im HTML-Template der Wikiseiten der JavaScript Editor TinyMCE[4] integriert. Anschließend wurde gezeigt, wie bei der Speicherung einer Notiz die gesamte HTML-Datei über JavaScript eingelesen, manipuliert und anschließend wieder geschrieben wird.

Das letzte Kapitel 4.6 erläutert die Implementierung der Synchronisation der Offline-Notizen mit der Webanwendung. Dazu wurde die Implementierung einer „Changeset"-Datei gezeigt, welche bei der „Two-Step"-Synchronisation im ersten Schritt auf das SMB-Laufwerk synchronisiert wird. In der „Changeset"-Datei werden alle Offline-Notizen in einem Array aus JSON-Objekten gespeichert. Diese Datei wird bei der Synchronisation auf dem Server ausgelesen und die Notizen werden in die Datenbank eingespielt.

[3] `http://www.alfresco.com/products/aifs/`, aufgerufen am 10. August 2011
[4] `http://tinymce.moxiecode.com/`, aufgerufen am 10. August 2011

Literaturverzeichnis

[Boo11] BOODMAN, Aaron: *Gears API Blog: Stopping the Gears*. http://gearsblog.blogspot.com/2011/03/stopping-gears.html, aufgerufen am 18. März 2011

[CN10] CAMPESATO, Oswald ; NILSON, Kevin: *WEB 2.0 Fundamentals with Ajax, Development Tools, and Mobile Platforms*. 1. 2010

[Hog10] HOGAN, Brian P.: *HTML5 and CSS3: Develop with Tomorrow's Standards Today*. Pragmatic Bookshelf, 2010

[inf11a] INFOASSET: *Tricia - Unternehmsweite Zusammenarbeit und Informationsmanagement mit hybriden Wikis*. http://www.infoasset.de/file/attachments/blogs/infoasset-news/infoasset-spricht-auch-deutsch/infoAsset%20Tricia%20Flyer%20deutsch.pdf, aufgerufen am 21. Mai 2011

[inf11b] INFOASSET: *The Tricia Templating System*. http://www.infoasset.de/wikis/javadoc-import-wiki/platform-documentation-templatedoc, aufgerufen am 23. Juni 2011

[inf11c] INFOASSET: *Tricia Platform Documentation*. http://www.infoasset.de/wikis/javadoc-import-wiki/home, aufgerufen am 26. April 2011

[LASS10] LUBBERS, P. ; ALBERS, B. ; SALIM, F. ; SMITH, R.: *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*. Apress, 2010

[Leb10] LEBLON, Robin: *Building advanced, offline web applications with HTML 5*, BarcelonaTech (UPC), Diplomarbeit, 2010

[Mah11] MAHEMOFF, Michael: *„Offline": What does it mean and why should I care?* http://www.html5rocks.com/tutorials/offline/whats-offline/, aufgerufen am 20. April 2011

[Pil10] PILGRIM, Mark: *HTML5: Up and Running*. O'Reilly Media, 2010

[RW11] RANGANATHAN, Arun ; WILSHER, Shawn: *Firefox 4: An early walk-through of IndexedDB*. http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase, aufgerufen am 9. Mai 2011

[VVVE09] VELTE, T. ; VELTE, A. ; VELTE, T.J. ; ELSENPETER, R.C.: *Cloud Computing: A Practical Approach*. McGraw-Hill, 2009

[WHA11] WHATWG: *HTML5 Living Standard: Offline Web applications*. `http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html`, aufgerufen am 22. März 2011

[Wor11] WORLD WIDE WEB CONSORTIUM: *HTML5 Spezifikation*. `http://dev.w3.org/html5/spec/Overview.html`, aufgerufen am 22. April 2011